1

Process Rescheduling in High Performance Computing Environments

Rodrigo da Rosa Righi and Lucas Graebin
Programa Interdisciplinar de Pós-Graduação em Computação Aplicada
Universidade do Vale do Rio dos Sinos
Brazil

1. Introduction

Scheduling is an important tool for manufacturing and engineering, where it can have a major impact on the productivity of a process (Min et al., 2009). In manufacturing, the purpose of production scheduling is to minimize the production time and costs, by informing a production facility when to make, with which staff, and on which equipment (Zhu et al., 2011). Nowadays, it is possible to observe the use of specific computational tools for production scheduling in which greatly outperform older manual scheduling methods. These tools implement mathematical programming methods that model the problem as an optimization issue where some objective, *e.g*. total duration, must be minimized (or yet maximized) (Yao & Zhu, 2010).

The concepts behind manufacturing and production scheduling can be employed in serie of contexts where optimization field takes place (Delias et al., 2011; Fan, 2011; Wang et al., 2011). In a common meaning, scheduling consists in formulating plans in which organize objects for operating efficiently in a specific context. Especially, this chapter discusses about a rescheduling method for high-performance environments (HPC) like Computational Grids, or just Grids (Yu & Buyya, 2005). The presented method deals with processes, in which represent an execution entity of the operating system. Here, we named scheduling as the first mapping of processes to resources in the Grid. Thus, the method is responsible for process rescheduling, so changing the first process-processors assignment in the distribute system by offering efficient scheduling plans along the application's lifetime. As an optimization problem , its main purpose focuses on minimizing the execution time of the application as a whole.

As grid computing emerged and got widely used, resources of multiple clusters became the dominant computing nodes of the grid (El Kabbany et al., 2011; Qin et al., 2010; Xhafa & Abraham, 2010). Applications containing routines for solving linear systems and fast Fourier transform (FFT) are typical examples of tightly-coupled parallel applications that may take profit from the power of cluster-of-clusters (Sanjay & Vadhiyar, 2009). Given that these clusters can be heterogeneous and the links among them are normally not fast, scheduling and load balancing are two key techniques that must be organized to reach high performance in this architecture. An alternative for offering this treatment focuses on scheduling using process migration. Therefore, we can reshape the process-resource matching at runtime

in accordance with both the behavior of the processes and the resources (processors and network).

Generally, process migration is implemented within the application, resulting in a close coupling between the application and the algorithms' data structures. Such an implementation is not extensible, due to the specificity of the shared data structure. Even more, some initiatives use explicit calls in the application code (Bhandarkar et al., 2000) and obligate extra executions to get tuned scheduling data (Silva et al., 2005; Yang & Chou, 2009). 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 (da Rosa Righi et al., 2010). It was designed to work with phases-based applications with BSP behavior (Bulk Synchronous Parallel) and acts over cluster-of-clusters architectures. MigBSP extensively uses heuristics to adapt the interval between migration calls, to analyze the behavior regularity of each process as well as to select the candidates for migration. Heuristics were employed since the problem of finding the optimum scheduling in heterogeneous system is in general NP-hard (Xhafa & Abraham, 2010).

Concerning the choosing of the processes, MigBSP creates a priority list based on the highest Potential of Migration (*PM*) of each process (da Rosa Righi et al., 2010). *PM* combines the migration costs with data from both computation and communication phases in order to create an unified scheduling metric. Using a hierarchy notion based on two levels (Goldchleger et al., 2004), each *PM* element concentrates a target process and a specific destination site. Figure 1 goes through the *PM* approach. The process denoted in the top of the mentioned list was always selected to be inspected for migration viability. Although we achieved good results when using this approach, we agree that an optimized one can deal with multiple processes when rescheduling verification takes place. A possibility could concern the selection of a percentage of processes based on the highest *PM*. Nevertheless, a question arises: How can one reach an optimized percentage value for dynamic applications and heterogeneous environments? A solution could involve the testing of several hand-tuned parameter instances and the comparison of the results. Certainly, this idea is time consuming and new applications and resources require a new series of tests.

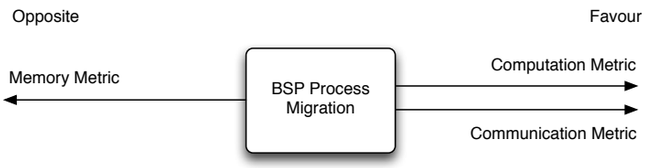

Fig. 1. Potential of Migration (*PM*) combines Computation, Communication and Memory data in order to offer an unified scheduling metric. The rationale of creating *PM* consists in evaluating migrations of processes to different sites, reducing the number of tests in the rescheduling moment.

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 complexity and optimization decisions from users (Ding et al., 2009; Nascimento et al., 2007; Sanjay & Vadhiyar, 2009). 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 (Duselis et al., 2009) and Backtracking (Baritompa et al., 2007) concepts to

evaluate the migration impact on each element of the *PM* list in an autonomous fashion. In addition, other AutoMig's strength comprises the needlessness to complete an additional MigBSP parameter 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 (Bonorden, 2007). The lowest forecast value indicates the most profitable plan for process rescheduling.

This book chapter aims to describe AutoMig in details. Particularly, we evaluated it by using a BSP application that computes image compression based on the Fractal method (Guo et al., 2009). Considering that the programmer does not need to change his/her application nor add a parameter on rescheduling model, the results with migration were satisfactory and totaled a mean gain of 7.9%. Furthermore, this classification is due to fact that AutoMig does not know any application and resource descriptions in advance. The results showed a serie of situations where AutoMig outperforms the heuristic that elects only one process. Next section shows MigBSP briefly and serves as the basis for understanding the proposed heuristic.

2. MigBSP: Rescheduling model

MigBSP answers the following issues regarding load balancing: (i) "When" to launch the migration; (ii) "Which" processes are candidates for migration; (iii) "Where" to put an elected process. In a previous paper we described the ideas to treat these questions in details (da Rosa Righi et al., 2010). The model requires both unicast and asynchronous communications among the processes. The target architecture is heterogeneous and composed by clusters, supercomputers and/or local networks. The heterogeneity issue considers the processors' clock speed (all processors have the same machine architecture), as well as network speed and level (Fast and Gigabit Ethernet and cluster-of-clusters environments, for instance). Such an architecture is assembled with abstractions of Sets (different sites) and Set Managers. As an example, a specific Set could be composed by the nodes from a cluster. Set Managers are responsible for scheduling, capturing data from a specific Set and exchanging it among other managers.

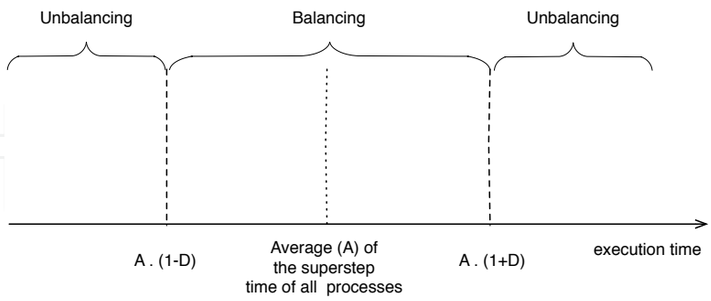

Fig. 2. Analysis of balancing and unbalancing situations which depend on the distance D from the average time A.

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 one. To store the variations on system state, a temporary variable called α' is used and updated at each superstep through the increment or decrement of one unit. 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 denoted 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 balanced. In summary, the higher the value of α, the lower the model's impact on application runtime.

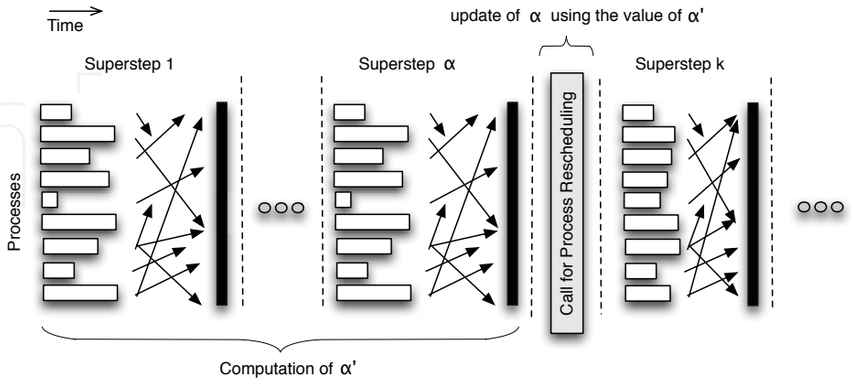

Fig. 3. Overview of an application execution with MigBSP: α parameter indicates the next interval for process rescheduling and depends on the value of α', which is updated at each BSP superstep.

The balanced state is based on the superstep time of each BSP process. Figure 2 depicts how both balanced and unbalanced situations can be reached. In implementation view, the processes save their superstep time in a vector and pass it to their Set Managers when rescheduling is activated. Following this, all Set Managers exchange their information. Set Managers have the times of each BSP process and compute the balancing situation. Therefore, each manager knows the α' variation locally. Figure 3 illustrates an example of the interaction between a BSP application and MigBSP. As we can observe, this figure presents the expected result when calling rescheduling actions since a reduction in time can be verified in the remaining BSP supersteps.

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. Considering that Grid scheduling is multi-objective in its general formulation (Xhafa & Abraham, 2010), $PM(i,j)$ is found using Computation, Communication and Memory metrics as we can see in Equation 1. The relation among them is based on the notion of force from Physics. Computation and Communication act in favor of migration, while Memory works in an opposite direction. The greater the value of $PM(i,j)$, the more prone the BSP processes will be to migrate.

$$PM(i,j) = Comp(i,j) + Comm(i,j) - Mem(i,j) \tag{1}$$

The Computation metric — $Comp(i,j)$ — uses a Computation Pattern $P_{\text{comp}}(i)$ that measures the stability of a process i regarding the number of instructions at each superstep (see Equation 2). This value is close to 1 if the process is regular and close to 0 otherwise. Other element in $Comp(i,j)$ is a computation time prediction $CTP_{k+\alpha-1}(i)$ of the process i at superstep $k+\alpha-1$ (last superstep executed before process rescheduling). Supposing that $CT_t(i)$ is the

computation time of the process i during superstep t, then the prediction $CTP_{k+\alpha-1}(i)$ uses the Aging concept as follows(da Rosa Righi et al., 2010; Tanenbaum, 2003).

$$CTP_t(i) = \begin{cases} CT_t(i) & if\ t = k \\ \frac{1}{2}CTP_{t-1}(i) + \frac{1}{2}CT_t(i) & if k < t \leq k + \alpha - 1 \end{cases}$$

$Comp(i,j)$ also presents an index $ISet_{k+\alpha-1}$(j). This index informs the average capacity of Set j at the $k + \alpha - 1^{th}$ superstep. For each processor in a Set, its load is multiplied by its theoretical capacity. Concerning this, the Set Managers compute a performance average of their Sets and exchange this value. Each manager calculates $ISet(j)$ for each Set normalizing their performance average by its own average. In the sequence, all Set Managers pass $ISet(j)$ index to the BSP processes under their jurisdiction.

$$Comp(i,j) = P_{comp}(i)\ .\ CTP_{k+\alpha-1}(i)\ .\ ISet_{k+\alpha-1}(j) \tag{2}$$

In the same way, the Communication metric — $Comm(i,j)$ — computes the Communication Pattern $P_{\text{comm}}(i,j)$ between processes and Sets (see Equation 3). Furthermore, this metric uses a communication time prediction $BTP_{k+\alpha-1}(i,j)$ involving the process i and Set j between two rebalancing activations. This last parameter employs the same idea used to compute $CTP\_t(i)$, where the prediction value is more strongly influenced by recent supersteps. The result of Equation 3 increases if the process i has a regularity considering the received bytes from processes of Set j and performs slower communication actions to this Set.

$$Comm(i,j) = P_{comm}(i,j)\ .\ BTP_{k+\alpha-1} \tag{3}$$

The Memory metric — $Mem(i,j)$ — composition can be seen in Equation 4. Firstly, the memory space in bytes of considered process is captured through $M(i)$. After that, the transfer time of 1 byte to the destination Set is calculated by $T(i,j)$ function. The communication involving process i is established with the Set Manager of each considered Set. Finally, the time spent on migration operations of process i to Set j is calculated through $Mig(i,j)$ function. These operations are dependent of operating system, as well as the tool used for providing process migration.

$$Mem(i,j) = M(i)\ .\ T(i,j)\ +\ Mig(i,j) \tag{4}$$

BSP processes calculate $PM(i,j)$ locally. At each rescheduling call, each process passes its highest $PM(i,j)$ to its Set Manager. This last entity exchanges the PM of its processes with other managers. As mentioned earlier, there is a heuristic to choose the candidate for migration which is based on a decreasing-sorted list composed by the highest PM value of each process. This heuristic chooses the head of the list. $PM(i,j)$ of a candidate process i is associated to a Set j intrinsically. The manager of this Set will select the most suitable processor to receive the process i.

Before any migration, its viability is verified considering the following data: (i) the external load on source and destination processors; (ii) the BSP processes that both processors are executing; (iii) the simulation of considered process running on destination processor; (iv) the time of communication actions considering local and destination processors; (v) migration

costs. Then, 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 costs. For each candidate, a new resource is chosen if $t_1 > t_2$.

3. 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 rescheduling moment. Especially, AutoMig's proposal solves the problem described below.

- **Problem Statement** - Given n BSP processes and a list of the highest PM (Potential of Migration) of each one at the rescheduling 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 4 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" in the third PM considers its own migration and the fact that "E" and "B" were migrated previously. Algorithm 1 presents AutoMig's approach in details.

Decreasing-sorted list based on the highest PM of each process	Value of the Scheduling prediction pf	Emulated migrations at each evaluation level
1st PM (Process E, Set 2) = 3.21	1st Scheduling = 2.34	(E)
2nd PM (Process B, Set 1) = 3.14	2nd Scheduling = 2.14	(E)(B)
3rd PM (Process A, Set 2) = 3.13	3rd Scheduling = 1.34	(E)(B)(A)
4th PM (Process C, Set 2) = 2.57	4th Scheduling = 1.87	(E)(B)(A)(C)
5th PM (Process G, Set 2) = 2.45	5th Scheduling = 1.21	(E)(B)(A)(C)(G)
6th PM (Process D, Set 1) = 2.33	6th Scheduling = 2.18	(E)(B)(A)(C)(G)(D)
7th PM (Process F, Set 1) = 2.02	7st Scheduling = 4.15	(E)(B)(A)(C)(G)(D)(F)

Fig. 4. Example of the AutoMig's approach. Only one process is migrated at each level of the PM list. A migration of a process on level l presents an impact in $l + 1$ and so on.

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 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 scheduling of the current mapping.

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 4. 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 their vectors and compute *pf* for each level of the list 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 (Moreno-Vozmediano & Alonso-Conde, 2005). 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.

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

$$Comm_p(i,j) = Max_k \; (\; \forall \; k \in Sets \; (Bytes(i,k) \; . \; T(k,j)) \;) \tag{6}$$

Algorithm 1 AutoMig's approach for selecting the processes

1: Each process computes *PM* locally (see Equation 1).
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 Equations 5 and 6 for their processes. The results will be used later for measuring the performance of 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 Equations 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**

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

Considering Equation 7, it is important to 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. Finally, AutoMig's selection approach uses a global strategy, where data from all processes are considered in the calculus. Normally, this strategy provides better results but requires synchronization points for capturing data. However, we take profit from the barriers of the BSP model for exchanging scheduling information, not paying an additional cost for that.

Kowk and Cheung (Kwok & Cheung, 2004) arranged the load balancing topic in four classes: (i) location policy; (ii) information policy; (iii) transfer policy and; (iv) selection policy. AutoMig answers the last issue by using a global strategy (Zaki et al., 1997). In this type of scheme, the decisions are made using a global knowledge, *i.e.*, data from all processes take

part in the synchronization operation for processes replacement. The list of the highest *PM* of all BSP processes is known by Set Managers when attempting for migration. Therefore, the main advantage of global schemes comprises the better quality of load balancing decisions since the entire studied objects are considered. On the other hand, the synchronization is the most expensive part of this approach (El Kabbany et al., 2011; Zaki et al., 1997). However, we take profit from BSP model organization, which already imposes a barrier synchronization among the processes. Therefore, we do not need to pay an addition cost to use the global idea.

4. Evaluation methodology

We are simulating the functioning of a BSP-based Fractal Image Compression (FIC) application. FIC has generated much interest in the image compression community as competitor with well established techniques like JPEG and Wavelets (Guo et al., 2009). One of the main drawbacks of conventional FIC is the high encoding complexity whereas decoding time is much lower (Xing, 2008). Nevertheless, fractal coding offers promising performance in terms of image quality and compression ratios. Basically, FIC exploits similarities within images. These similarities are described by a contractive transformation of the image whose fixed point is close to the image itself. The image transformation consists of block transformation 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.

For each range an appropriate domain must be found. A root mean-square-error (rms) distance is calculated in order to judge the quality of a single map. The encoding time depends on the number of domains whose each range must be compared to. 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 must be compared with $8(t-d+1)^2$ domains. The greater the number of domains, the better will be the compression quality. In addition, 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. Moreover, each process sends $\frac{t}{r}$ ranges before calling the barrier, which must be multiplied by 8 to find the number of bytes (we considered a range with 8 bytes in memory).

The main aim of the experimental evaluation is to observe the performance of MigBSP when working with AutoMig heuristic. Considering this, we applied 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. Scenario ii consists in performing all scheduling calculus and decisions, but it does not comprise any migrations actually. Scenario iii adds the migrations costs on those processes that migrate from one processor to another. Both the BSP application and the model were developed using the SimGrid Simulator (MSG Module) (Casanova et al., 2008). It makes possible application modeling and processes migration. In addition, it is deterministic, where a specific input always results in the same output.

Algorithm 2 Modeling of a single superstep for the Fractal Image Compression problem

1: Considering a range-pool rp of the image ($0 \leq rp \leq \frac{t}{r} - 1$), where t and r mean the sides of the $t \times t$ image and $r \times r$ 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 ($0 \leq i \leq n - 1$) sends data to its right-neighbor $i + 1$. Process $n - 1$ sends data to process 0 (where n is the total number of processes)
10: Call for synchronization barrier

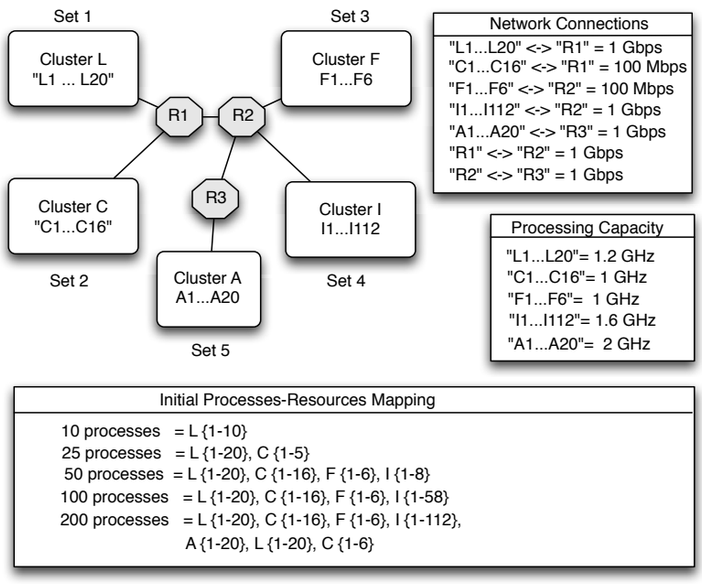

Fig. 5. Multiple Clusters-based topology, processing and network resources description and the initial processes-resources scheduling.

Aiming to test the scenarios, we assembled an infrastructure with five Sets which is depicted in Figure 5. 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. Moreover, it represents the current resources at Unisinos University, Brazil. 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 5. Since the application proceeds in communications from process i to $i + 1$, we opted by using the contiguous approach in which a cluster is filled before passing to another one (Pascual et al., 2009). Furthermore, the values of 40, 20 and 10 were used for the side (d) of a square domain. The range value (r) is obtained by $\frac{d}{2}$. The considered figure is a square with side 1000. The lower the d variable, the greater the number of domains to be tested by each process. These parameters turn possible the verification of the AutoMig's overhead and

situations where process rescheduling is applicable. Finally, the migration costs are based on previous executions with AMPI (Huang et al., 2006) on our clusters.

5. Analyzing AutoMig's overhead and decisions

Table 1 presents the initial tests when dealing with 40 and 20 for both domain and range sizes, respectively. This configuration enables a short number of domains to be computed by each process. Thus, the processes have a small computation grain and their 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. Therefore, AutoMig figures out the lowest *pf* in which in reached through the current scheduling. Consequently, both times for scenario ii and iii are higher than the time spent in scenario i. In this context, a large overhead is imposed by MigBSP since the normal application execution is close to 1 second in average.

Processes	Scenario i	Scenario ii (MigBSP and AutoMig without Migrations)	Scenario iii (MigBSP and AutoMig enabling Migrations)
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

Table 1. Results when using 40 and 20 for domain and range, respectively (time in seconds)

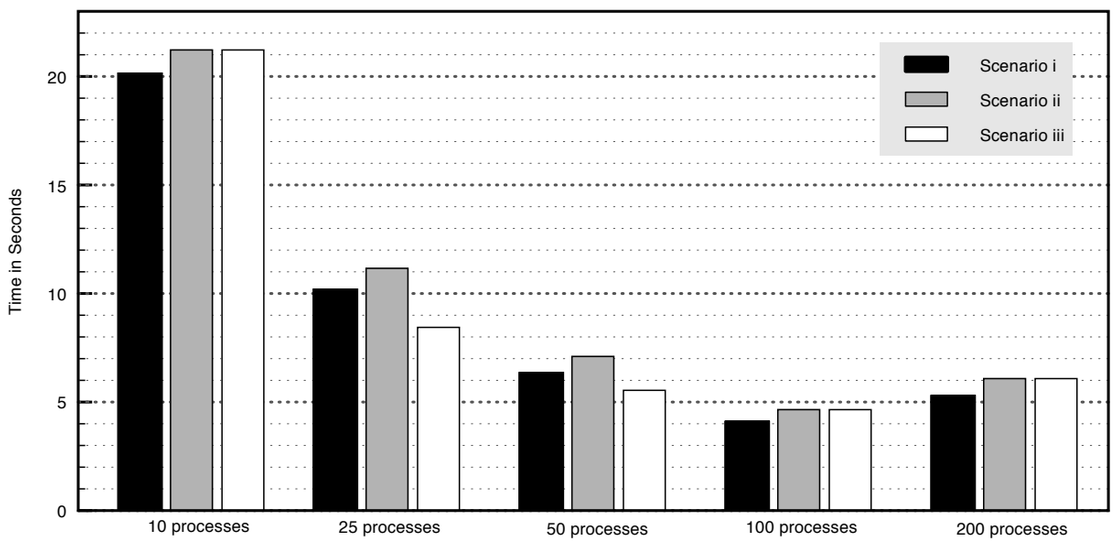

Fig. 6. Evaluating the migration model with AutoMig when using 20 and 10 for domain and range, respectively.

We increase the number of domains when dealing with 20 for the domain's side. This context generated the results presented in Figure 6. As presented in the previous execution, migrations did not take place with 10 processes. They are balanced and their reorganization to the fastest cluster imposed costs larger than the benefits. *pf* of 0.21 was obtained for the current processes-resources mapping by using 20 for domain and 10 processes. All predictions in the

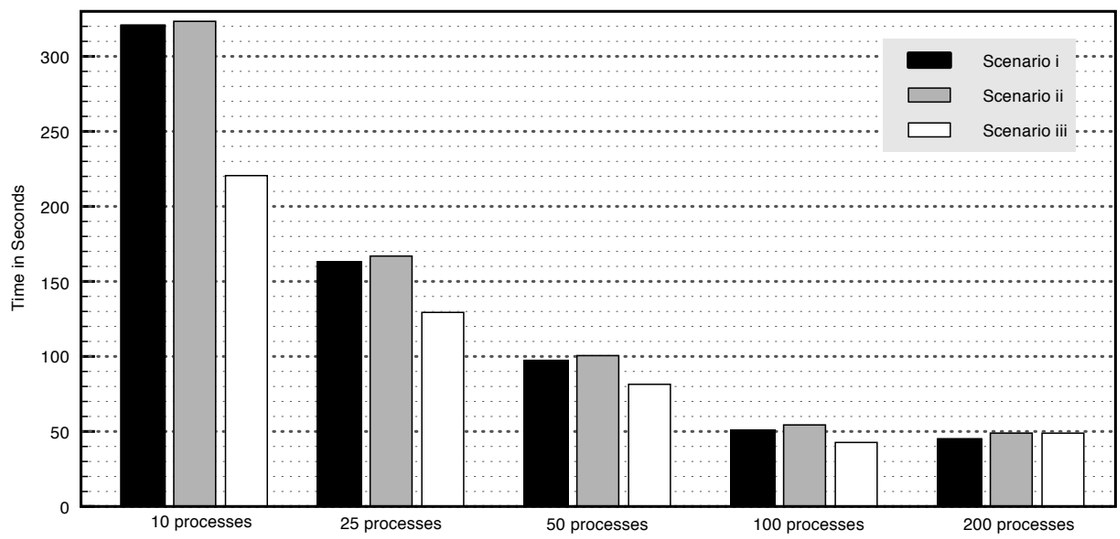

Fig. 7. AutoMig's results when enlarging the work per process at each superstep. This graph illustrates experiments with domain 10 and range 5.

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 BSP model presents a synchronization barrier. For example, 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. Finally, once the work grain decreases when enlarging the processes, the executions with 100 and 200 did not present migrations.

The BSP application demonstrated good performance levels with domain equal to 10 as illustrated in Figure 7. 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 $8((\frac{t}{d})^2.\frac{1}{10})$ 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. The values of *pf* for the *PM* list may be seen as follows:

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

Considering the first up to the ninth *pf*, 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. This situation changes when testing the tenth *pf*. It considers the migration of all 10 processes to the fastest cluster and generates a gain around 31.13% when comparing scenarios iii and i. This analysis is illustrated in Figure 8.

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 consider 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 position and an overhead of 7.64% in application execution was observed comparing both scenarios i and ii.

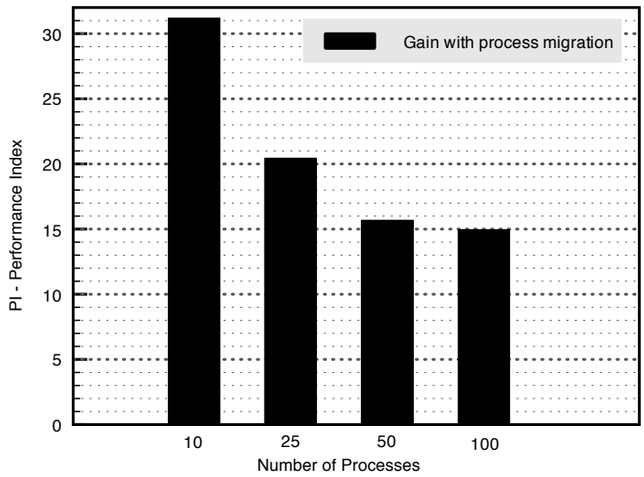

Fig. 8. Migration gains by applying AutoMig with domain 10. Performance Index PI $=(\frac{scen.\ i - scen.\ iii}{scen.\ i} * 100)$.

The previous tests make clear that the higher the computation weight per process, the better will be the gains with process rescheduling. In this way, we tested AutoMig with shorter domain and range values as expressed in Table 2. 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%. Observing the results, we can verify that the more time consuming the application, the lower the AutoMig overhead on that. In addition, we verify 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. There is a limit where the inclusion of processes is not profitable due to the larger number of communication actions and the higher time spent on barriers. Concerning the scalability issue, MigBSP (with AutoMig) shows similar behaviors if compared to those obtained by scenario i (Figures 6 and 8).

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

Table 2. Results when using 4 and 2 for domain and range

Table 2 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.

6. Related work

Vadhiyar and Dongarra presented a migration framework and self adaptivity in GrADS system (Vadhiyar & Dongarra, 2005). The gain with rescheduling is based on the remaining execution time prediction over a new specified resource. Thus, this framework must work with applications in which their parts and durations are known in advance. In addition, the same problem is shown in the following two works. Sanjay and Vadhiyar (Sanjay & Vadhiyar, 2009) 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. The second work comprises the Ding's efforts (Ding et al., 2009). He creates the TPBH (Task Partition-Based Heuristic) heuristic, in which operates with both suffrage and minimum completion time approaches. These two mentioned works treat applications in which the problem size is known in advance. Alternatively, AutoMig just uses data collected at runtime and based on that it takes the performance of different scheduling predictions.

Chen et al. (Chen et al., 2008) proposed processes reassignment with reduced cost for grid load rebalancing. The heuristics permit only movements between the machine with the maximum load level and another machine. Furthermore, this work does not consider the communication issue on selection decisions. Liu et al. (Liu et al., 2009) introduced a novel algorithm for resource selection whose the application reports the Execution Satisfaction Degree (ESD) to the scheduling middleware. Then, this last entity tune the environment by adding/replacing/deleting resources in order to satisfy the user's performance requirements. 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 BSP scope, Jiang, Tong and Zhao presented resource load balancing based on multi-agents in ServiceBSP model (Jiang et al., 2007). Load balancing is launched when a new task is insert in the system and is based on the load rank of nodes. The selection service sends the new task to the current lightest node. Load value is calculated taking such information: CPU, memory resource, number of current tasks, response time and number of network links. Silva et al. (da Silva e Silva et al., 2010) explained the resource management on the InteGrade grid middleware. They presented a grid as a collection of clusters, where each one runs its own Cluster Managers (CM). Analogous to MigBSP, CM is responsible for getting data from a cluster and to exchange it among other CMs.

Concerning the migration context, we can cite two works that enable this feature on BSP applications. The first one describes the PUBWCL library which aims to take profit of idle cycles from nodes around the Internet (Bonorden et al., 2005). All proposed algorithms just use data about the computation times from each process as well as from the nodes. Other work comprises an extension of PUB library to support migration (Bonorden, 2007). The

author proposed both a centralized and a distributed strategies for load balancing. In the first one, all nodes send data about their CPU power and load to a master node. The master verifies the least and the most loaded node and selects one process for migration between them. In distributed approach, every node chooses *c* other nodes randomly and asks them for their load. One process is migrated if the minimum load of *c* analyzed nodes is smaller than the load of the node that is performing the test. The drawback of this strategy is that it can create a lot of messages among the nodes. Moreover, both strategies take into consideration neither the communication among the processes, nor the migration costs.

7. Conclusion

Considering that the bulk synchronous style is a common organization on writing successful parallel programs (Bonorden, 2007; De Grande & Boukerche, 2011; Hendrickson, 2009; Hou et al., 2008), 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 activities and adds the transferring overhead of the tested process. 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 (Zaki et al., 1997). 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 SimGrid Simulator. We implemented a BSP version of the Fractal Image Compression algorithm. Besides its real utility in satellite and mobile video areas (Guo et al., 2009), this application was taken because it works with a parameter called domain which turns the creation of different load situations easier. Since the application is CPU-bound, the shorter the size of domain 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. AutoMig demonstrates that migrations do not take place in this situation, owing to the execution rules of a BSP superstep. It has a barrier that always wait for the slowest process (in this case, the process that will remain on the slower cluster).

Finally, future work comprises the use of AutoMig in a HPC middleware for Cloud computing. This middleware will work on self-provisioning the resources for executing parallel applications. Concerning that each application specifies its own SLA (Service Level Agreement) previously, AutoMig appears as the first initiative to reorganize processes-resources shaping on the fly when SLA fails. If the rescheduling does not solve the performance issue, more resources are allocated in a second instance. The final aim is to reduce the costs on both cloud administrator and user levels.

8. References

Baritompa, W., Bulger, D. W. & Wood, G. R. (2007). Generating functions and the performance of backtracking adaptive search, *J. of Global Optimization* 37: 159–175.
URL: *http://portal.acm.org/citation.cfm?id=1196588.1196605*

Bhandarkar, M. A., Brunner, R. & Kale, L. V. (2000). Run-time support for adaptive load balancing, *IPDPS '00: Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, Springer-Verlag, London, UK, pp. 1152–1159.

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

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

Casanova, H., Legrand, A. & Quinson, M. (2008). Simgrid: A generic framework for large-scale distributed experiments, *Tenth International Conference on Computer Modeling and Simulation (uksim)*, IEEE Computer Society, Los Alamitos, CA, USA, pp. 126–131.

Chen, L., Wang, C.-L. & Lau, F. (2008). Process reassignment with reduced migration cost in grid load rebalancing, *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on* pp. 1–13.

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

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

De Grande, R. E. & Boukerche, A. (2011). Dynamic balancing of communication and computation load for hla-based simulations on large-scale distributed systems, *J. Parallel Distrib. Comput.* 71: 40–52.
URL: *http://dx.doi.org/10.1016/j.jpdc.2010.04.001*

Delias, P., Doulamis, A., Doulamis, N. & Matsatsinis, N. (2011). Optimizing resource conflicts in workflow management systems, *Knowledge and Data Engineering, IEEE Transactions on* 23(3): 417 –432.

Ding, D., Luo, S. & Gao, Z. (2009). A dual heuristic scheduling strategy based on task partition in grid environments, *CSO '09: Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization*, IEEE Computer Society, Washington, DC, USA, pp. 63–67.

Duselis, J., Cauich, E., Wang, R. & Scherson, I. (2009). Resource selection and allocation for dynamic adaptive computing in heterogeneous clusters, *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pp. 1 –9.

El Kabbany, G., Wanas, N., Hegazi, N. & Shaheen, S. (2011). A dynamic load balancing framework for real-time applications in message passing systems, *International Journal of Parallel Programming* 39: 143–182. 10.1007/s10766-010-0134-5.
URL: *http://dx.doi.org/10.1007/s10766-010-0134-5*

Fan, K. (2011). A special parallel job shop scheduling problem, *E -Business and E -Government (ICEE), 2011 International Conference on*, pp. 1 –3.

Goldchleger, A., Kon, F., Goldman, A., Finger, M. & Bezerra, G. C. (2004). Integrade object-oriented grid middleware leveraging the idle computing power of desktop machines: Research articles, *Concurr. Comput. : Pract. Exper.* 16(5): 449–459.

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

Hendrickson, B. (2009). Computational science: Emerging opportunities and challenges, *Journal of Physics: Conference Series* 180(1): 012013.
URL: *http://stacks.iop.org/1742-6596/180/i=1/a=012013*

Hou, Q., Zhou, K. & Guo, B. (2008). Bsgp: bulk-synchronous gpu programming, *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, ACM, New York, NY, USA, pp. 1–12.

Huang, C., Zheng, G., Kale, L. & Kumar, S. (2006). Performance evaluation of adaptive mpi, *PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, ACM Press, New York, NY, USA, pp. 12–21.

Jiang, Y., Tong, W. & Zhao, W. (2007). Resource load balancing based on multi-agent in servicebsp model, *International Conference on Computational Science (3)*, Vol. 4489 of *Lecture Notes in Computer Science*, Springer, pp. 42–49.

Kwok, Y.-K. & Cheung, L.-S. (2004). A new fuzzy-decision based load balancing system for distributed object computing, *J. Parallel Distrib. Comput.* 64(2): 238–253.

Liu, H., Sørensen, S.-A. & Nazir, A. (2009). On-line automatic resource selection in distributed computing, *IEEE International Conference on Cluster Computing*, IEEE, pp. 1–9.

Min, L., Xiao, L. & Ying, C. (2009). An establishment and management system of production planning and scheduling for large-piece okp enterprises, *Industrial Engineering and Engineering Management, 2009. IE EM '09. 16th International Conference on*, pp. 964 –968.

Moreno-Vozmediano, R. & Alonso-Conde, A. B. (2005). Influence of grid economic factors on scheduling and migration., *High Performance Computing for Computational Science - VECPAR*, Vol. 3402 of *Lecture Notes in Computer Science*, Springer, pp. 274–287.

Nascimento, A. P., Sena, A. C., Boeres, C. & Rebello, V. E. F. (2007). Distributed and dynamic self-scheduling of parallel mpi grid applications: Research articles, *Concurr. Comput.: Pract. Exper.* 19(14): 1955–1974.

Pascual, J. A., Navaridas, J. & Miguel-Alonso, J. (2009). Job scheduling strategies for parallel processing, Springer-Verlag, Berlin, Heidelberg, chapter Effects of Topology-Aware Allocation Policies on Scheduling Performance, pp. 138–156.

Qin, X., Jiang, H., Manzanares, A., Ruan, X. & Yin, S. (2010). Communication-aware load balancing for parallel applications on clusters, *IEEE Trans. Comput.* 59(1): 42–52.

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

Silva, R. E., Pezzi, G., Maillard, N. & Diverio, T. (2005). Automatic data-flow graph generation of mpi programs, *SBAC-PAD '05: Proceedings of the 17th International Symposium on Computer Architecture on High Performance Computing*, IEEE Computer Society, Washington, DC, USA, pp. 93–100.

Tanenbaum, A. (2003). *Computer Networks*, 4th edn, Prentice Hall PTR, Upper Saddle River, New Jersey.

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

Wang, Z., Wang, F., Gao, F., Zhai, Q. & Zhou, D. (2011). An electric energy balancing model in a medium enterprise grid, *Power and Energy Engineering Conference (APPEEC), 2011 Asia-Pacific*, pp. 1 –4.

Xhafa, F. & Abraham, A. (2010). Computational models and heuristic methods for grid scheduling problems, *Future Gener. Comput. Syst.* 26(4): 608–621.

Xing, C. (2008). An adaptive domain pool scheme for fractal image compression, *Education Technology and Training and Geoscience and Remote Sensing* 2: 719–722.

Yang, C.-T. & Chou, K.-Y. (2009). An adaptive job allocation strategy for heterogeneous multiple clusters, *Proceedings of the 2009 Ninth IEEE International Conference on Computer and Information Technology - Volume 02*, CIT '09, IEEE Computer Society, Washington, DC, USA, pp. 209–214.
URL: *http://dx.doi.org/10.1109/CIT.2009.138*

Yao, L. & Zhu, W. (2010). Visual simulation framework of iron and steel production scheduling based on flexsim, *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on*, pp. 54 –58.

Yu, J. & Buyya, R. (2005). A taxonomy of scientific workflow systems for grid computing, *SIGMOD Rec.* 34(3): 44–49.

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

Zhu, Z., Chu, F., Sun, L. & Liu, M. (2011). Scheduling with resource allocation and past-sequence-dependent setup times including maintenance, *Networking, Sensing and Control (ICNSC), 2011 IEEE International Conference on*, pp. 383 –387.

Production Scheduling

Edited by Prof. Rodrigo Righi

ISBN 978-953-307-935-6
Hard cover, 242 pages
Publisher InTech
Published online 11, January, 2012
Published in print edition January, 2012

Generally speaking, scheduling is the procedure of mapping a set of tasks or jobs (studied objects) to a set of target resources efficiently. More specifically, as a part of a larger planning and scheduling process, production scheduling is essential for the proper functioning of a manufacturing enterprise. This book presents ten chapters divided into five sections. Section 1 discusses rescheduling strategies, policies, and methods for production scheduling. Section 2 presents two chapters about flow shop scheduling. Section 3 describes heuristic and metaheuristic methods for treating the scheduling problem in an efficient manner. In addition, two test cases are presented in Section 4. The first uses simulation, while the second shows a real implementation of a production scheduling system. Finally, Section 5 presents some modeling strategies for building production scheduling systems. This book will be of interest to those working in the decision-making branches of production, in various operational research areas, as well as computational methods design. People from a diverse background ranging from academia and research to those working in industry, can take advantage of this volume.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen