Applying Processes Rescheduling over Irregular BSP Application

Rodrigo da Rosa Righi¹, Laércio Lima Pilla¹, Alexandre Silva Carissimi¹,
Philippe O.A. Navaux¹, and Hans-Ulrich Heiss²

¹ Instituto de Informática, Federal University of Rio Grande do Sul - Brazil
² Kommunikations- und Betriebssysteme, Technical University Berlin - Germany

Abstract. This paper shows an evaluation of processes rescheduling over an irregular BSP (Bulk Synchronous Parallel) application. Such application is based on dynamic programming and its irregularity is presented through the variation of computation density along the matrix' cells. We are using MigBSP model for processes rescheduling, which combines multiple metrics - Computation, Communication and Memory - to decide about processes migration. The main contribution of this paper includes the viability to use processes migration on irregular BSP applications. Instead to adjust the load of each process by hand, we presented that automatic processes rebalancing is an effortless technique to obtain performance. The results showed gains greater than 10% over our multi-cluster architecture. Moreover, an acceptable overhead from MigBSP was observed when no migrations happen during application execution.

1 Introduction

The dynamism and adaptivity are keywords in the new scenarios of parallel and distributed computing [1]. For example, the dynamism can be seen at the application level as well as a characteristic of the parallel machine architecture. In the first case, processes can change their amount of computation and/or their pattern of communication at any moment during application runtime. The dynamism related to infrastructure is demonstrated, for instance, through the fluctuation of processors load and availability, as well as in bandwidth due to either congestion in the network or modifications on its utilization. This kind of dynamism and uncertainty are challenges found in large distributed systems like grids [1]. Considering both ideas of dynamism, properties established previously may not be proven during runtime making dynamic scheduling pertinent. Dynamic scheduling can deal with the adaptivity issue since decisions managed during runtime can trigger changes in the scheduler behavior itself and adaptations that guarantee an acceptable level of application performance.

In this context, we address the issue of adaptivity through the control of processes migration over grid environments. We have developed a model called **MigBSP** that controls processes rescheduling on BSP (Bulk Synchronous Parallel) applications [2,3]. MigBSP performs an automatic load (processes) rebalancing among the resources without changing the application's code and acts

without previous knowledge about the application's behavior. Our model can be seen as a dynamic rescheduler that treats both cases of dynamicity related above. Especially at application level, it uses two patterns - one for Computation and other for Communication - in order to measure the processes' regularity regarding their performed instructions as well as communicated bytes at each superstep. Besides computation and communication phases of a superstep, our model observes the costs to decide about the processes transferring. Finally, contrary to existing approaches [2], MigBSP performs the rescheduling according to the system state using an adaptable interval between calls for migration.

Aiming to perform an evaluation of MigBSP, we modeled an heterogeneous multi-cluster architecture and an irregular application. The application is based on dynamic programming (DP), which is a popular algorithm design technique for optimization problems [4]. In practice, there are many irregular DP applications where the workload varies across the matrix. Firstly, alternatives to achieve performance on such application include: (i) the mapping of the most loaded cells to faster processors; (ii) to determine appropriate data partitioning scheme on the fly or; (iii) to apply receiver-initiated load balancing algorithms inside the application. However, these techniques require an effort to recognize the target architecture previously and/or change application's code by hand. On the other hand, a possibility is to use automatic processes migration. In this way, our model acts at middleware level attempting to gain performance effortlessly.

This paper presents the impact of MigBSP over an irregular DP application. We will observe the impact of the load computation density on choosing processes for migration and on the regularity of rescheduling calls. As general ideas, migrations take place to faster resources (since the application is CPU-bound) and are considered viable as far as the load increases along the matrix.

2 MigBSP: Processes Rescheduling Model

Figure 1 (a) shows a superstep s in which the processes are not balanced among the resources. The main idea of MigBSP is to reduce the time of each superstep. Figure 1 (b) shows the expected result with processes redistribution after the end of superstep s . The architecture is heterogeneous and composed by clusters, supercomputers and local networks. The model requires that the involved nodes allow all-to-all asynchronous communications. The heterogeneous issue considers the processors' clock (all processors have the same architecture), as well as network speed and level (Fast and Gigabit Ethernet and multi-clusters environment, for instance). This architecture is assembled with Sets (different sites) and Set Managers. Set Managers are responsible for scheduling, capturing data from a specific Set and exchanging it among other managers.

MigBSP answers the following issues: (i) "When" to launch the migration; (ii) "Which" processes are candidates for migration; (iii) "Where" to put an elected process. In [3] we described the ideas to treat these questions in details. However, such work does not deal with irregular applications.

Fig. 1. Supersteps in different situations

The decision for processes remapping is taken at the end of a superstep (after the barrier). This migration point was chosen because in this moment it is possible to analyze data from all BSP processes at their computation and communication phases. Aiming to generate the least intrusiveness in application as possible, we applied two adaptations that control the value of α . α is updated at each rescheduling call and will indicate the interval to the next one. Aiming 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. α is filled with α' value in the moment of rescheduling call. The adaptations' ideas are: (i) to postpone the rescheduling call if the system is stable (processes are balanced) or to turn it more frequent, otherwise; (ii) to delay this call if a pattern without migrations on ω past calls for rescheduling is observed. Aiming to analyze the system stability, the times of the slowest and fastest processes at each superstep are captured as well as the average time is computed. Only the processes that perform any computation activity enters to observe system stability. A variable 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.

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 idea consists in performing a subset of the processes-resources tests at the rescheduling moment. $PM(i, j)$ is found using Computation, Communication and Memory metrics. The relation among them is based on the notion of force from physics. Computation and Communication act in favour of migration, while Memory works in an opposite direction. 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 based on all computation phases between two activations of processes rescheduling. For this prediction it is used the Aging concept. 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 considering data between two rebalancing activations. Memory metric - $Mem(i, j)$ - considers process memory, transferring rate between considered process and the manager of target Set, as well as migration costs.

$$PM(i, j) = Comp(i, j) + Comm(i, j) - Mem(i, j) \quad (1)$$

$PM(i, j)$ selects the candidate processes for migration (see Equation 1). A high $PM(i, j)$ means that process i has high computation time, high communication with processes that belong to Set j and presents low migration cost. 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 among other managers. There are two heuristics to choose the candidates, both based on a decreasing ordered list of PM s. The heuristics are: (i) processes that have PM higher than a $MAX(PM) \times x$ are candidates, where x is a percentage; (ii) choose just one process.

$PM(i, j)$ of a candidate process i is associated to a Set j . 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. Concerning this, we computed two times: $t1$ and $t2$. $t1$ means the local execution of process i , while $t2$ encompasses its execution on the other processor and includes the migration costs. For each candidate is chosen a new resource (if $t1 > t2$) or its migration is canceled.

3 Irregular DP Application

Dynamic programming is a method of solving problems exhibiting the properties of overlapping subproblems and optimal substructure [4]. DP algorithms can be classified according to the matrix size and the dependency relationship of each matrix cell. An algorithm for a problem of size n is called tD/eD if its matrix size is $O(n^t)$ and each matrix cell depends on $O(n^e)$ other cells. In this paper we work with 2D/1D DP algorithms, which are all irregular with load changes along the matrix's cells. In particular, we observed the functioning of Smith-Waterman algorithm [5] that is a well-known algorithm for local sequence alignment.

Smith-Waterman algorithm proceeds in a series of wavefronts diagonally across the matrix. Figure 2 (a) illustrates a 4×4 matrix with a column-based processes allocation. The more intense the shading, the greater is the load computation density of the cell. Each wavefront corresponds to a superstep. For instance, Figure 2 (b) shows a 4×4 matrix with 7 supersteps. This organization

Fig. 2. Irregular BSP dynamic application organization

brings the following conclusions: (i) $2n - 1$ supersteps are computed in a square matrix with order n and; (ii) each process will be involved on n supersteps.

At each superstep, each process computes a block of data and sends it to other process. Figure 2 (b) shows the communication among the processes. Considering that cell a, b (a means a matrix' line, while b is a matrix' column) needs data from the $a, b - 1$ and $a - 1, b$ other ones, we have an interaction from process pb to $pb + 1$. We do not have communication inside the same column.

4 Evaluation Methodology

We are using simulation in three scenarios: (i) Application execution simply; (ii) Application execution with scheduler without applying migrations; (iii) Application execution with scheduler allowing migrations. Scenario ii indicates the overhead associated with scheduling calculus and message passing between processes and Set Managers, as well as among Set Managers properly. The analysis of scenarios i and iii shows the performance when migrations are applied.

The configuration of scenarios ii and iii depends on the Computation Pattern $P_{comp}(i)$ of each process i . $P_{comp}(i)$ increases or decreases depending on the prediction of the amount of performed instructions at each superstep. $PI_t(i)$ represents this prediction for superstep t and process i . It is based on the Aging concept, in which uses the idea that the prediction is more strongly influenced by recent values. The formula to compute $PI_t(i)$ is shown below. k means superstep 1 or the superstep after the last call for processes rescheduling. $I_t(i)$ represents the amount of instructions executed by process i during superstep t .

$$PI_t(i) = \begin{cases} I_t(i) & \text{if } t = k \\ \frac{1}{2}PI_{t-1}(i) + \frac{1}{2}I_t(i) & \text{if } k < t \leq k + \alpha - 1 \end{cases}$$

$P_{comp}(i)$ is updated at each superstep following the Algorithm 1. We consider a specific process as regular if the forecast is within a δ margin of fluctuation from the amount of instructions performed. In our experiments, we are using 10^6 as the amount of instructions for the first superstep and 10^9 for the last one. The increase of load computational density among the supersteps is uniform. Considering this, we applied δ equal to 0.01 (1%) and 0.50 (50%) to scenarios ii and iii, respectively. This last value was used because $I_2(1)$ is 565.10^5 and $PI_2(1)$ is 287.10^5 when a 10×10 matrix is tested (19 supersteps). The percentage of 50% enforces instruction regularity in the system. Both values of δ will influence the Computation metric, and consequently the choosing of candidates for migration. Scenario ii tends to obtain negatives values for PM since the Computation Metric will be close to 0. Consequently, no migrations will happen on this scenario. Following with the parameters, we tested the behavior of square matrixes of order 10, 25, 50, 100 and 200. In addition, we performed a column-based mapping, where process pi is responsible for column i of the matrix. Each cell of a 10×10 matrix needs to send 500 Kbytes and each process occupies 1.2 Mbyte in memory (700 Kbytes for other data). The cell of 25×25 matrix communicates 200 Kbytes and each process occupies 900 Kbytes in memory and so on.

Algorithm 1. Computation Pattern $P_{comp}(i)$ of the process i

```

1: for  $t$  from superstep  $k$  to superstep  $k + \alpha - 1$  do
2:   if  $PI_t(i) \geq I_t(i) \cdot (1 - \delta)$  and  $PI_t(i) \leq I_t(i) \cdot (1 + \delta)$  then
3:     Increases  $P_{comp}(i)$  by  $\frac{1}{\alpha}$  up to 1
4:   else
5:     Decreases  $P_{comp}(i)$  by  $\frac{1}{\alpha}$  down to 0
6:   end if
7: end for

```

Fig. 3. Testbed infrastructure and the initial processes-resources mappings

We are using the Simgrid [6] (MSG module). This simulator is deterministic, where a specific input always results in the same output. We assembled an infrastructure with five Sets, as we can see in Figure 3. Each node has a single processor. These Sets represent a real infrastructure with five clusters located at Federal University of Rio Grande do Sul, Brazil. For sake of simplicity, we hide the network of each cluster. Clusters Labtec, Corisco and Frontal have their nodes linked by Fast Ethernet, while ICE and Aquario use Gigabit connection. The migration costs are based on executions with AMPI [7] on our clusters.

Figure 3 also presents the initial processes-recourses mappings. Finally, initial tests were executed using α equal to 2, 4, 8 and 16. Furthermore, we employed ω equal to 3, initial D equal to 0.5 and used heuristic one to choose the candidates for migration with x equal to 80%. We observed that our testbed environment prioritized the heterogeneity issue. Future works include the execution of BSP applications and MigBSP over dynamic environments. Simgrid allows to write files informing the variation in time of bandwidth, latency and CPU capacities.

5 Evaluation and Discussions

Table 1 presents the application evaluation. 19 supersteps were crossed when a 10×10 matrix was tested. Adopting this size of matrix and α 2, 13.34s and 14.15s were obtained for scenarios i and ii which represents a cost of 8%. The higher is the value of α , the lower is the MigBSP overhead on application execution. This occurs because the system is stable (processes are balanced) and α always increases at each rescheduling call. Three calls for processes relocation were

done when testing α 2 (at supersteps 2, 6 and 14). The rescheduling call at superstep 2 does not produce migrations. At this step, the load computational density is not enough to overlap the consider migration costs involved on process transferring operation. The same occurred on the next call at superstep 6. The last call happened at superstep 14, which resulted on 6 migrations: $\{(p5,a1), (p6,a2), (p7,a3), (p8,a4), (p9,a5), (p10,a6)\}$. MigBSP indicated the migration of processes that are responsible to compute the final supersteps. The execution with α equal to 4 implies in a shorter overhead since two calls were done (at supersteps 4 and 12). Observing scenario iii, we do not have migrations in the first call, but eight occurred in the other one. Processes 3 up to 10 migrated in this last call to cluster Aquario. α 4 outperforms α 2 for two reasons: (i) it does lesser rescheduling calls and; (ii) the call that causes processes migration was done at a specific superstep in which MigBSP takes better decisions.

Table 1. Evaluation of scenarios i, ii and iii when varying the matrix size

Scenarios		mat. 10×10	mat. 25×25	mat. 50×50	mat. 100×100	mat. 200×200
Scenario i		13.34s	40.74s	92.59s	162.66s	389.91s
Scen. ii	$\alpha = 2$	14.15s	43.05s	95.70s	166.57s	394.68s
	$\alpha = 4$	14.71s	42.24s	94.84s	165.66s	393.75s
	$\alpha = 8$	13.78s	41.63s	94.03s	164.80s	392.85s
	$\alpha = 16$	13.42s	41.28s	93.36s	164.04s	392.01s
Scen. iii	$\alpha = 2$	13.09s	35.97s	85.95s	150.57	374.62s
	$\alpha = 4$	11.94s	34.82s	84.65s	148.89s	375.53s
	$\alpha = 8$	13.82s	41.64s	83.00s	146.55s	374.38s
	$\alpha = 16$	12.40s	40.64s	85.21s	162.49s	374.40s

The system stays stable when the 25×25 matrix was tested. α 2 produces a gain of 11% in performance when considering 25×25 matrix and scenario iii. This configuration presents four calls for processes rescheduling, where two of them produce migrations. No migrations are indicated at supersteps 2 and 6. Nevertheless, processes 1 up to 12 are migrated at superstep 14 while processes 21 up to 25 are transferred at superstep 30. These transferring operations occurred to the fastest cluster. In this last call, the remaining execution presents 19 supersteps (from 31 to 49) to amortize the migration costs and to get better performance. The execution when considering α 8 and scenario iii brings an overhead if compared with scenario i. Two calls for migrations were done, at supersteps 8 and 24. The first call causes the migration of just one process (number 1) to a1 and the second one produces three migrations: $\{(p21,a2), (p22,a3), (p23,a4)\}$. We observed that processes p24 and p25 stayed on cluster Corisco. Despite performed migrations, these two processes compromise the supersteps that include them. Both are executing on a slower cluster and the barrier synchronization always waits for the slowest process. Maintaining the matrix size and adopting α 16, we have two calls for migration: at supersteps 16 and 48. This last call migrates p24 and p25 to cluster Aquario. Although this movement is pertinent to get performance, just one superstep is executed before finalizing the application.

50 processes were evaluated when the 50×50 matrix was considered. In this context, α also increases at each call for processes rescheduling. We observed that an overhead of 3% was found when scenario i and ii were compared (using α 2). In addition, we observed that all values of α achieved a gain of performance in scenario iii. Especially when α 2 was used, five calls for processes rescheduling were done (at supersteps 2, 6, 14, 30 and 62). No migrations are indicated in the first three calls. The greater is the matrix size, the greater is the amount of supersteps needed to make migrations viable. This happens because our total load is fixed (independent of the matrix size) but the load partition increases uniformly along the supersteps (see Section 4 for details). Process 21 up to 29 are migrated to cluster Aquario at superstep 30, while process 37 up to 42 are migrated to this cluster at superstep 62. Using α equal to 4, 84.65s were obtained for scenario iii which results a gain of 9%. This gain is greater than that achieved with α 2 because now the last rescheduling call is done at superstep 60. The same processes were migrated at this point. However, there are two more supersteps to execute using α equal to 4. Three rescheduling calls were done with α 8 (at supersteps 8, 24 and 56). Only the last two produce migration. Three processes are migrated at superstep 24: $\{(p21,a1),(p22,a2),(p23,a3)\}$. Process 37 up to 42 are migrated to cluster Aquario at superstep 56. This last call is efficient since it transfers all processes from cluster Frontal to Aquario.

The execution with a 100×100 matrix shows good results with processes migration. Six rescheduling calls were done when using α 2. Migrations did not occur at the first three supersteps (2, 6 and 14). Process 21 up to 29 are migrated to cluster Aquario after superstep 30. In addition, process 37 to 42 are migrated to cluster Aquario at superstep 62. Finally, superstep 126 indicates 7 migrations, but just 5 occurred: p30 up to p36 to cluster Aquario. These migrations complete one process per node on cluster Aquario. MigBSP selected for migration those processes that belonged to cluster Corisco and Frontal, which are the slowest clusters on our infrastructure testbed. α equal to 16 produced 3 attempts for migration when a 100×100 matrix is evaluated (at supersteps 16, 48 and 112). All of them triggered migrations. In the first call, the 11th first processes are migrated to cluster Aquario. All process from cluster Frontal are migrated to Aquario at superstep 48. Finally, 15 processes are selected as candidates for migration after crossing 112 supersteps. They are: p21 to p36. This spectrum of candidates is equal to the processes that are running on Frontal. Considering this, only 3 processes were migrated actually: $\{(p34,a18),(p35a19),(p36,a20)\}$.

Table 1 also shows the application performance when the 200×200 matrix was tested. The overhead of our model is smaller than 2% for scenario ii. Furthermore, satisfactory results were obtained with processes migration. The system stays stable during all application execution. Despite having more than one process mapped to one processor, sometimes just a portion of them is responsible for computation at a specific moment. This occurs because the processes are mapped to matrix columns, while supersteps comprise the anti-diagonals of the matrix. Using α 2 and considering scenario iii, 8 calls for processes rescheduling were

done. Migrations were not done at supersteps 2, 6 and 14. Processes 21 up to 31 are migrated to cluster Aquario at superstep 30. Moreover, all processes from cluster Frontal are migrated to Aquario at superstep 62. Six processes are candidates for migration at superstep 126: p30 to p36. However, only p31 up to p36 are migrated to cluster Aquario. These migrations happen because the processes initially mapped to cluster Aquario do not collaborate yet with BSP computation. Migrations are not viable at superstep 254. Finally, 12 processes (p189 to p200) are migrated to cluster Aquario when superstep 388 was crossed. At this time, all previous processes allocated to Aquario are inactive and the migrations are viable. However, just 10 remaining supersteps are executed to amortize the processes migration costs.

6 Related Work

GridWay treats with time and cost optimization scheduling and migration [8]. Both migration mechanisms consider only data from CPU, like speed and load. Bhandarkar, Brunner and Kale presented a support for adaptive load balancing in MPI applications [9]. Periodically, the application transfers control to the load balancer using a special call. These authors present a Metis-based strategy that uses the communication graph to remap the processes. AMPI [7] uses Charm++ to offer automatic load balancing. Charm++ collects workload data and the objects communication pattern. At each load balancing time, the load balancer uses such data to redistribute the workload. Vadhiyar and Dongarra presented a migration framework and self adaptivity in GrADS [1]. However, they computed the migration cost as a fixed value. In addition, the gain with rescheduling is based on the remaining execution time prediction over a new resource. Thus, this work must deal with applications in which their parts are known in advance.

Concerning the BSP scope, Jiang, Tong and Zhao presented resource load balancing based on multi-agents in ServiceBSP [10]. Load balancing is launched when a new task is inserted and is based on the load rank of the nodes. Scheduling service sends a new task to the current lightest node. Load value is calculated taking such information: CPU, memory, number of current tasks, response time and number of network links. In addition, we can cite two works that present migration on BSP applications. The first one describes the PUBWCL library, which aims to take profit of idle cycles from nodes around the Internet [2]. PUBWCL offers migration at each superstep. All proposed algorithms just use data about the nodes and consider the computation times from each process. Other work comprises the PUB library [11]. The author explains that a load balancer decides when to launch the migration, but this issue is not addressed in [11]. He proposed centralized and a distributed strategies for load balancing. 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 the 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 scheduling messages.

7 Conclusion and Future Works

This paper presented MigBSP briefly, as well as the treatment of an irregular application executed using it. MigBSP combines data from multiple metrics to create our decision function PM. PM considers the migration of a process i to a Set j . Thus, we do not test all possibilities at the rescheduling moment. The contribution of this paper is the possibility to use MigBSP over heterogeneous environments in order to get performance on irregular BSP applications. Initially, MigBSP was developed to deal with processes that present a regular behavior. However, it allows to fill parameters that turn possible to adjust its functioning to treat irregularity in an efficient manner. Section 5 presented the results and a discussion about the application execution. The main conclusions are: (i) a simple way to obtain performance is designing α in order to trigger the rescheduling call close to the end of the application, since MigBSP tends to select those processes that have more load; (ii) the greater is the load computational density of the last supersteps, the better will be the results with the migration of the last processes and; (iii) the application behavior implies that the processors can vary their load during the crossing of supersteps, changing their viability to receive processes.

Future works include simulation of MigBSP over the Grid5000. We intend to test MigBSP's scalability over this platform.

References

1. Vadhiyar, S.S., Dongarra, J.J.: Self adaptivity in grid computing: Research articles. *Concurr. Comput.: Pract. Exper.* 17(2-4), 235–257 (2005)
2. Bonorden, O., Gehweiler, J., auf der Heide, F.M.: Load balancing strategies in a web computing environment. In: *Conf. on Parallel Processing and Applied Mathematics*, pp. 839–846 (2005)
3. Righi, R., Pilla, L., Carissimi, A., Navaux, P.O.A.: Controlling processes reassignment in bsp applications. In: *20th International Symposium on Computer Architecture and high Performance Computing*, pp. 37–44. IEEE, Los Alamitos (2008)
4. Low, M.Y.H., Liu, W., Schmidt, B.: A parallel bsp algorithm for irregular dynamic programming. In: Xu, M., Zhan, Y.-W., Cao, J., Liu, Y. (eds.) *APPT 2007*. LNCS, vol. 4847, pp. 151–160. Springer, Heidelberg (2007)
5. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. *Journal of Molecular Biology* 147(1), 195–197 (1981)
6. Casanova, H., Legrand, A., Quinson, M.: Simgrid: A generic framework for large-scale distributed experiments. In: *Conf. Comp. Modeling and Simulation*, pp. 126–131. IEEE, Los Alamitos (2008)
7. Huang, C., Zheng, G., Kale, L., Kumar, S.: Performance evaluation of adaptive mpi. In: *Symp. on Principles and pract. of parallel programming*, pp. 12–21. ACM Press, New York (2006)
8. Moreno-Vozmediano, R., Alonso-Conde, A.B.: Influence of grid economic factors on scheduling and migration. In: Daydé, M., Dongarra, J., Hernández, V., Palma, J.M.L.M. (eds.) *VECPAR 2004*. LNCS, vol. 3402, pp. 274–287. Springer, Heidelberg (2005)

9. Bhandarkar, M.A., Brunner, R., Kale, L.V.: Run-time support for adaptive load balancing. In: IPDPS 2000: Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing, pp. 1152–1159. Springer, London (2000)
10. Jiang, Y., Tong, W., Zhao, W.: Resource load balancing based on multi-agent in servicebsp model. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007 (3). LNCS, vol. 4489, pp. 42–49. Springer, Heidelberg (2007)
11. Bonorden, O.: Load balancing in the bulk-synchronous-parallel setting using process migrations. In: Symp. on Parallel and Distrib. Processing, pp. 1–9. IEEE, Los Alamitos (2007)