Analyzing the Frequency of Processes Migration Calls on BSP Application Execution[*]

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

Abstract

This paper presents an analysis regarding the impact of the frequency of calls for processes migration on a specific BSP application execution. For that, we are using our model of processes rescheduling that combines three metrics - Memory, Computation and Communication - in order to measure the potential of migration of each BSP process.

1. Introduction

The use of dynamic resources and irregular applications are more and more present in distributed computing. In these situations, the initial mapping of processes to resources may not remain efficient during application runtime. This occurs because both computing and network resources can suffer modifications in their states. In addition, the amount of processing as well as network interaction among the processes can vary in application execution. Concerning this, processes rescheduling (migration) to new processors becomes relevant in order to improve resources usage and minimize the waiting time for results[1, 4].

In this context, this paper describes a **processes rescheduling model**[5] that acts over **BSP (Bulk Synchronous Parallelism)**[2] applications. Besides computation and communication phases of a BSP superstep, our model observes migration operation costs and memory to decide transferring viability of a process. In addition, this paper presents some experimental results, emphasizing the impact of the frequency for processes relocation in the application execution time as a whole.

2. BSP Processes Rescheduling Model

Our BSP Processes Rescheduling Model is presented in details in [5]. Here, we will explain its main ideas, as well as the issues that it addresses in processes migration. Processes relocation is made using dynamic process rescheduling, where data is captured during application runtime. Figure 1 (a) shows a superstep k of a application in which the load (processes) is not balanced among the resources. The idea of our processes rescheduling model is to reduce each superstep time. Figure 1 (b) shows the expected result with processes redistribution at the end of superstep k, which will influence the execution of the next supersteps, (including k+1 and so on). The target architecture is heterogeneous and composed by clusters, supercomputers and local networks. The model requires that the involved nodes must have all-to-all connections. The heterogeneous issue considers the processors' capacity (all processors have the same machine architecture), as well as network speed and level (Fast and Gigabit Ethernet and multi-clusters environment, for instance). This architecture is assembled with the notion of hierarchy, with abstractions of Sets (division by sites) and Set Managers. Set Managers are responsible for scheduling, capturing data of a specific Set and exchanging it among other managers.

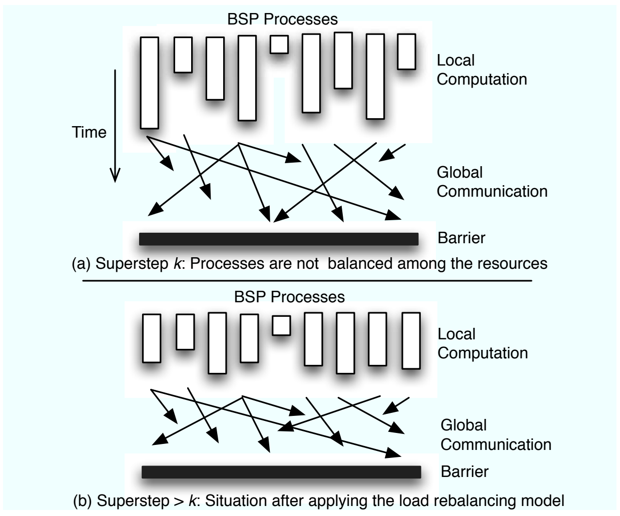

BSP Processes
Local Computation
Time
Global Communication
Barrier
(a) Superstep k: Processes are not balanced among the resources
BSP Processes
Local Computation
Global Communication
Barrier
(b) Superstep > k: Situation after applying the load rebalancing model

Figure 1: Supersteps in different situations

The final result of the model is a mathematical formalism that answers the following issues regarding processes migration: (i) "When" to launch the processes migration; (ii) "Which" processes are candidates for migration and; (iii) "Where" to put an elected process from the candidates ones.

[*] This work is partially supported by Capes. Process: 1476/08-4

The decision for processes remapping is taken at the end of a BSP superstep (after barrier and before the next superstep). 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, it is used a variable called α. It informs the interval of supersteps for the next processes rescheduling call. Up to the next call, at each superstep a temporary value α' is updated. α' is loaded to α when load rebalancing is activated. α increases if the system tends to stability in conclusion time of each superstep and decreases case opposite.

The answer for “Which” is solved through Potential of Migration computation. Each BSP process i computes n functions $PM(i, j)$, where n is the number of Sets and j means a specific Set. $PM(i, j)$ is found out using Computation, Communication and Memory metrics. The relation among them is based on the notion of force from physics area. Computation and Communication are metrics that acts in favour of migration, while Memory one represents an idea of cost and acts in an opposite direction.

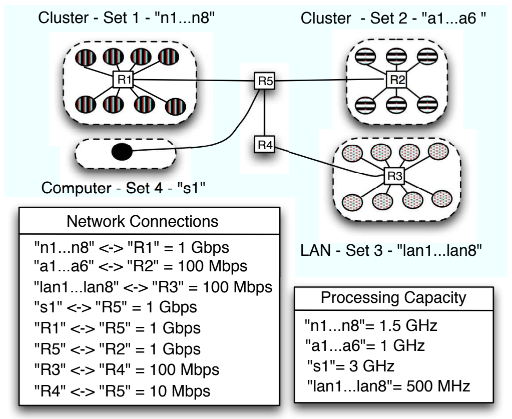
Computation metric considers a Computation Pattern ($P_{comp}(i)$) that measures the stability of a process i regarding the amount of instructions performed at each superstep. This value is close to 1 if the process presents regularity (considering an error interval) and close to 0, otherwise. Besides $P_{comp}(i)$, this metric also performs a computation time prediction based on all computation times of each superstep between two activations of processes rescheduling. In the same way, Communication Metric computes $P_{com}(i, j)$ that means Communication pattern of process i and Set j. Moreover, this metric uses communication time prediction of process i and Set j considering data between two rebalancing activations. Memory metric takes into account process memory, transferring rate between considered process and the manager of target Set, as well as migration costs.

$PM(i, j)$ is calculates as follows: $PM(i, j) = Comp(i) + Comm(i, j) - Mem(i, j)$. 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. There are two heuristics to choose the candidates for migration, all of them based on a decreasing list composed by the highest PM value from each BSP process. They are: (i) select a percentage of processes; (ii) choose just one process.

$PM(i, j)$ of a candidate process i is associated to a set j. Therefore, the manager of this set will select the most suitable processor under its control to receive the process i and this strategy answers “Where”. Before to perform the process migration, its viability is verified. This operation takes into account the external load on source and destination processors, the simulation of considered process running in destination processor, the BSP processes that they are executing, as well as the migration cost of considered process. Finally, for each candidate process is chosen a new resource or its migration is canceled.

3. Experimental Evaluation

The main aim of this experimental evaluation is to observe the impact of different values of α of our processes rescheduling model on a specific BSP application. Considering this, we applied simulation in three scenarios: (i) Application execution simply; (ii) Application execution with scheduler without applying migrations; (iii) Application execution with scheduler allowing migrations. We are using the **Simgrid Simulator**[3] (MSG module), which makes possible application modeling and processes migration. This simulator is deterministic, where a specific input always results in the same output. In addition, a time equal to $Mem(i, j)$ is paid for each migration of process i to Set j (this value will determine the migration costs). We assembled an infrastructure with four Sets, which is depicted in Figure 2. Each node has a single processor. Moreover, we modeled the BSP implementation of Lattice Boltzmann Method to Simgrid using vertical domain decomposition. At each superstep, each process i is responsible for a sublattice computation. After that, this process sends boundary data to its neighbor $i + 1$.


Cluster - Set 1 - "n1...n8"
Cluster - Set 2 - "a1...a6 "
R1
R5
R2
R4
R3
Computer - Set 4 - "s1"
LAN - Set 3 - "lan1...lan8"

Network Connections
"n1...n8" <-> "R1" = 1 Gbps
"a1...a6" <-> "R2" = 100 Mbps
"lan1...lan8" <-> "R3" = 100 Mbps
"s1" <-> "R5" = 1 Gbps
"R1" <-> "R5" = 1 Gbps
"R5" <-> "R2" = 1 Gbps
"R3" <-> "R4" = 100 Mbps
"R4" <-> "R5" = 10 Mbps

Processing Capacity
"n1...n8"= 1.5 GHz
"a1...a6"= 1 GHz
"s1"= 3 GHz
"lan1...lan8"= 500 MHz

Figure 2: Infrastructure for simulations

Initial tests were executed using α equal to 4, 8 and 16. Furthermore, we observed the behavior of 10 BSP processes, used heuristic two to choose the candidate process for migration (see Section 2 for details) and applied two different initial processes-resources mappings.

- The first mapping: {(p1,n1), (p2,n2), (p3,n3), (p4,n4), (p5,n5), (p6,n6), (p7,n7), (p8,n8), (p9,a1), (p10,a2)};
- The second mapping puts one process per Set circularly: { (p1,n1), (p2,a1), (p3,lan1), (p4,s1) (p5,n2), (p6,a2), (p7,lan2), (p8,s1), (p9,n3), (p10,a3)}.

Besides these mappings, we applied two types of tests: A and B. Superstep time is dominated by computation in test A, while in test B communication among the processes is the more costly part. Each process executes 10^9 instructions and communicates 500 KBytes at each superstep in test A. In test B, the number of instructions changes to 10^6. These values were adopted based on real executions of Lattice Boltzmann method in our clusters at UFRGS, Brazil.

Table 1 shows the results of test A and the first mapping. The system stays stable (α increases at each rescheduling call). This fact causes low intrusion of our model in application's execution comparing scenarios i and ii. One migration occurred {(p9,s1)} when executing 10 supersteps, while two happened between 50 and 2000: {(p9,s1),(p10,s1)}. We observed that migrations occurred to s1, which is the fastest processor. We had a profit of 33% after executing 2000 supersteps in comparison of scenarios i and iii. In this studied table, we can observe that does not exist a large variation when α is changed, since in all conditions it always increases from its initial value. Besides these results, Table 1 also presents that scenarios ii and iii have similar times independent of the employed α.

Figure 3 depicts a graphic in which the amount of computation is reduced when compared to test A. This turns the system unstable, with an α value that never increases. Thus, the call for processes reassignment is launched several times, causing a large overhead (scenario ii). Oppositely, the system begins unstable and becomes stable in scenario iii. The model indicates two migrations {(p9,n7),(p10,n8)} in all executed supersteps, independing on the α used. We verify that the higher is the value of α, the better is the application performance. When 1500 supersteps are executed, 16.99s, 16.03s and 15.10s are achieved with α 4, 8 and 16, respectively.

Figure 4 illustrates a graphic with the second mapping and test A. The system stays stable with this configuration, allowing similar times for scenarios i and ii. We have one migration {(p3,s1)} with 10 supersteps and two {(p3,s1),(p7,s1)} in the remaining supersteps. The more increased is the amount of supersteps, the higher is the obtained gain with migrations. For instance, a reduction of $\approx$ 39% in time is achieved with our rescheduling model when 2000 supersteps are evaluated. Furthermore, we verified that the second mapping provides longer execution times than the first one. This is associated with both the infrastructure heterogeneity and the application's behavior.

The results of test B with the second mapping are shown in Table 2. Analyzing this table, we can observe that scenario iii with α equal to 4 presents execution times higher than scenario i. This occurs because the system is unstable and a call for processes reassignment is done at each α supersteps (α never changes from its initial value with this configuration). Using α equal to 8 and 10 supersteps, one migration occurred: {(p3,l4)}. If we increase the executed supersteps up to 50, we have the following migrations: {(p3,a4),(p7,a5),(p10,n4),(p7,n5),(p2,n6),(p8,a1)}. Maintaining this value of α and varying the number of supersteps from 100 up to 2000, 11 migrations are performed: the same executed with 50 supersteps and {(p9,a2), (p10,a3), (p4,a6), (p5,a1), (p6,a2)}. When using α equal to 16, the system becomes stable and α begins to grow from supersteps higher than 100 (scenario iii). Consequently, the processes rescheduling is postponed and less intrusion in application execution is added.

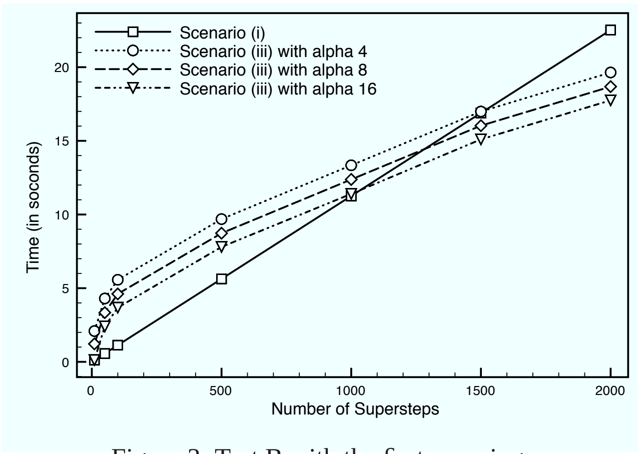


Figure 3: Test B with the first mapping

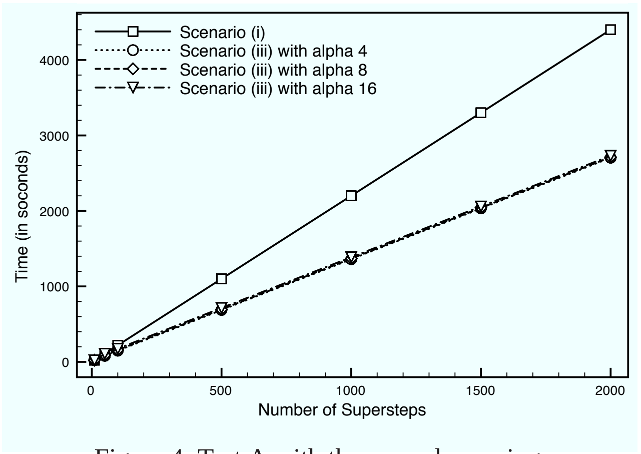


Figure 4: Test A with the second mapping

Concerning the results of Table 2, we can improve the algorithm of processes rescheduling launching in order to reduce the interference of the model in the following way. If the system remains unstable and no migrations are performed during several calls for rescheduling, we can increase α. The higher is this parameter, the better is the result on unstable scenarios. As a general conclusion, we observed that our experiments prioritized the heterogeneity issue. Thus, future works include the execution of BSP applications and our model over dynamic environments. Simgrid allows to write files informing the variation in time of bandwidth, latency as well as CPU capacities.

Table 1: Execution of all situations when dealing with Test A and the first initial mapping

Step	Scen (i)	$\alpha = 4$		$\alpha = 8$		$\alpha = 16$	
		Scen (ii)	Scen (iii)	Scen (ii)	Scen (iii)	Scen (ii)	Scen (iii)
10	10.01	10.15	11.10	10.15	11.10	10.15	10.15
50	50.51	50.58	40.64	50.53	43.70	50.52	51.83
100	101.02	102.34	75.21	101.94	78.27	101.78	85.40
500	505.12	507.14	345.78	506.08	348.84	505.82	356.97
1000	1010.25	1014.27	682.49	1012.27	685.55	1011.27	692.68
1500	1515.37	1522.38	1019.20	1519.40	1022.28	1518.02	1029.39
2000	2020.50	2034.37	1354.90	2030.53	1357.97	2027.48	1365.10

Table 2: Execution of all situations when dealing with Test B and the second initial mapping

Step	Scen (i)	$\alpha = 4$		$\alpha = 8$		$\alpha = 16$	
		Scen (ii)	Scen (iii)	Scen (ii)	Scen (iii)	Scen (ii)	Scen (iii)
10	2.02	3.6	3.2	2.82	2.78	2.02	2.01
50	10.11	19.70	14.48	14.91	9.41	12.51	8.5
100	20.23	40.41	29.07	29.82	17.20	25.03	14.77
500	101.18	201.25	141.72	150.72	83.85	125.95	22.58
1000	202.36	401.30	282.53	302.44	167.65	251.91	35.46
1500	303.55	603.36	423.34	452.96	250.65	377.86	43.35
2000	404.73	802.45	590.02	608.62	330.04	504.82	52.23

4. Conclusion and Future Works

This paper presented the main ideas of our BSP processes rescheduling model. In addition, we discussed the employment of multiple values of initial α and their impact in application execution. α indicates the next interval to launch processes reorganization. Its value is adaptable, varying according to system stability (the higher is α during execution, the lower is the time disparities among the processes per superstep). In general, application performance variations using different contents of initial α are not large in stable scenarios, since α doubles its value at each call for processes rescheduling. On the other hand, the choosing of α is important in unstable scenarios, because it represents the minimum interval for the next call for migration. Our results showed in Figure 3 and Table 2 some values of α where the system starts unstable and becomes stable, achieving high performance with processes migration. Moreover, our results suggests an adaptation of our algorithm if the system is unstable (large variation among the conclusion time of each process per superstep) and no migrations are indicated. The idea in this situation is to increase the value of α in order to minimize the impact of model execution in application performance.

Besides experiments with dynamicity issue, future works include simulation of our model over Grid5000 platform. We intend to develop other BSP applications and test the model's scalability over this platform. Finally, research directions cover two aspects. The first one is the analysis of processes migration costs and their dilution over more than one superstep. The other one deals with self-adjusting of the weights of each considered metric. Initially, this topic will be studied based on the work of Wieczorek et al.[6].

References

[1] G. Aggarwal, R. Motwani, and A. Zhu. The load rebalancing problem. In *SPAA '03: ACM symposium on Parallel algorithms and architectures*, pages 258–265, 2003. ACM Press.

[2] 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.

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

[4] C. Du, X.-H. Sun, and M. Wu. Dynamic scheduling with process migration. In *CCGRID '07: Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 92–99, 2007. IEEE Computer Society.

[5] R. Righi, A. Carissimi, and P. O. A. Navaux. On the dynamic load-rebalancing of bsp application using process miration. In *V Workshop em Processamento Paralelo e Distribudo (WSPPD 2007)*, pages 31–36, 2007.

[6] M. Wieczorek, S. Podlipnig, R. Prodan, and T. Fahringer. Bi-criteria scheduling of scientific workflows for the grid. *International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 9–16, 2008.