Future Generation Computer Systems 78 (2018) 272–286

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

MigPF: Towards on self-organizing process rescheduling of Bulk-Synchronous Parallel applications

Rodrigo da Rosa Righi [a,*], Roberto de Quadros Gomes [a], Vinicius Facco Rodrigues [a], Cristiano André da Costa [a], Antonio Marcos Alberti [b], Laércio Lima Pilla [c], Philippe Olivier Alexandre Navaux [c]

[a] Universidade do Vale do Rio dos Sinos, Rio Grande do Sul, Brazil
[b] Instituto Nacional de Telecomunicações, Minas Gerais, Brazil
[c] Universidade Federal do Rio Grande do Sul, Rio Grande do Sul, Brazil

HIGHLIGHTS

- We are proposing MigPF, a migration model for Bulk-Synchronous Parallel programs on heterogeneous environments.
- Using ideas from the TCP protocol, MigPF adapts the interval between migration calls.
- Scientific contribution includes a new algorithm that self-organizes process migration.
- We implemented a prototype running over AMPI and evaluated against other proposals.
- Shear-sort and Image Compression applications were tested in a heterogeneous cluster.

ARTICLE INFO

Article history:
Received 10 November 2015
Received in revised form
19 April 2016
Accepted 5 May 2016
Available online 18 May 2016

Keywords:
Bulk Synchronous Parallel
Process migration
Load balancing
Self-organizing

ABSTRACT

Migration is a known technique to offer process rescheduling, being especially pertinent for Bulk Synchronous Parallel (BSP) programs. Such programs are organized in a set of supersteps, in which the slowest process always determines the synchronization time. This approach motivated us to develop a first model called MigBSP, which combines computation, communication, and migration costs metrics for process rescheduling decisions. In this paper, a new model named MigPF enhances our previous work in three aspects: (i) a different algorithm for detecting imbalance situations, which considers the performance of each processor, including its enclosed processes, instead of individual times of each process; (ii) an improvement on the rescheduling reactivity through shortening the interval for the next migration call when imbalance situations arise; (iii) a new algorithm for self-organizing the migratable processes and their destinations. Particularly, this third item represents our main scientific contribution, not only in terms of the MigBSP context, but also in a broader one that covers the entire BSP landscape. The algorithm assembles n possible rescheduling plans (where n is the number of processes) and provides a prediction function (pf) that chooses the most profitable rescheduling plan when compared to the current mapping. We developed a MigPF prototype with the Adaptive MPI (AMPI) library, which offers a standard framework for implementing migration-based load balancing policies. We tested this prototype against other built-in AMPI rescheduling policies with two scientific applications: shear sorting and fractal image compression. The results revealed performance gains up to 41% and an overhead limited to 5% when migrations do not take place.

1. Introduction

Process scheduling is a key technique for providing high performance to parallel applications [1–3]. It concerns a first mapping of processes to resources in order to optimize an adopted metric (or a set of them), normally the application's total execution time (or makespan). Nevertheless, the power of using a parallel machine can be minimized, or yet, nullified, if the following scenarios occur [4,5]: (i) bad use of heterogeneous resources, mainly when considering round-based applications[1]; (ii) shared

* Corresponding author.
E-mail address: rrrighi@unisinos.br (R. da Rosa Righi).

http://dx.doi.org/10.1016/j.future.2016.05.004
0167-739X/

[1] Here, round-based or iterative applications refer to BSP (Bulk-Synchronous Parallel) applications [6,7].

infrastructures with fluctuations on network bandwidth and CPU load; (iii) irregular applications in which computation and communication patterns are not predictable at either editing or compilation time. An alternative should concern the rewriting of application source code for providing either sender- or receiver-initiated load balancing algorithms [8–10]. This style of load balancing requires detailed knowledge of the application code, and the same can be extended for the parallel machine. Another less intrusive alternative considers the use of rescheduling enabled by process migration [11,12]. This technique can be implemented by using migration directives at predetermined places according to a parallel programming model (Master–Slave, Bulk Synchronous Parallel, Divide-and-Conquer and Pipeline) [2,13–15].

Process migration is especially important on Bulk Synchronous Parallel (BSP) [6,7] applications, in which the slowest processor always limits the application's performance, since the others must wait on barrier synchronization. BSP represents a common model for writing successful parallel programs that exhibit a phase-based computational behavior [14,16]. Thus, migration can benefit a large set of applications in production worldwide. Technically, a BSP application is composed by a set of rounds or supersteps, each one composed by parallel computation on each process, arbitrary communication among them, and a barrier synchronization. In particular, the barrier can be seen as a pertinent place to do the process rescheduling whereas there is a complete knowledge of the system (processes and parallel machine) and there is no communication in transit [17]. Aiming to cover process migration on BSP applications efficiently, we developed a rescheduling model called MigBSP [17]. This model combines computation, communication and migration costs metrics to answer the three traditional questions regarding load balancing: (i) When to activate it? (ii) Which processes will be involved? and (iii) Where to put the processes selected for migration?

MigBSP was demonstrated as a viable alternative for the rescheduling of scientific applications on multi-cluster environments as attested by our previous works [17,18]. Despite the encouraging results, we envisioned three weak points of the MigBSP's approach, as depicted in Fig. 1. First of all, the mechanism used for detecting load imbalance monitors the times of the fastest and slowest processes, which can lead to premature decisions when there is more than one process per core (or processor entity). Fig. 1(a) presents a false-positive (top) and a false-negative (bottom) analysis on imbalance detection. For example, an imbalanced state can trigger useless migrations and imposes more overhead in the application execution, since the periodicity of rescheduling attempts can become shorter to address this situation quickly. Fig. 1(b) illustrates the current MigBSP's adaptation to regulate α parameter, which is responsible for controlling the next interval for the process rescheduling. Here, the main problem consists in a low reactivity[2] to reorganize the processes when imbalance situations take place. For example, after crossing superstep 30 in Fig. 1(b), the system is diagnosed as imbalanced, but the next attempt to reorganize the process will only happen 8 supersteps ahead from this point. Finally, MigBSP selects the migratable processes using list scheduling and a metric named Potential of Migration (PM) [18]. After ordering the list, it is possible to use one of two heuristics as depicted in Fig. 1(c): (i) select the process on the top; (ii) select a number of processes based on the value of the process on the top. However, the approach (i) can suffer again from lack of reactivity. And the approach (ii), the heuristic needs a parameter which is not tangible for users and it must be replaced each time that some changes to the application source code are done and/or resources changes are perceived.

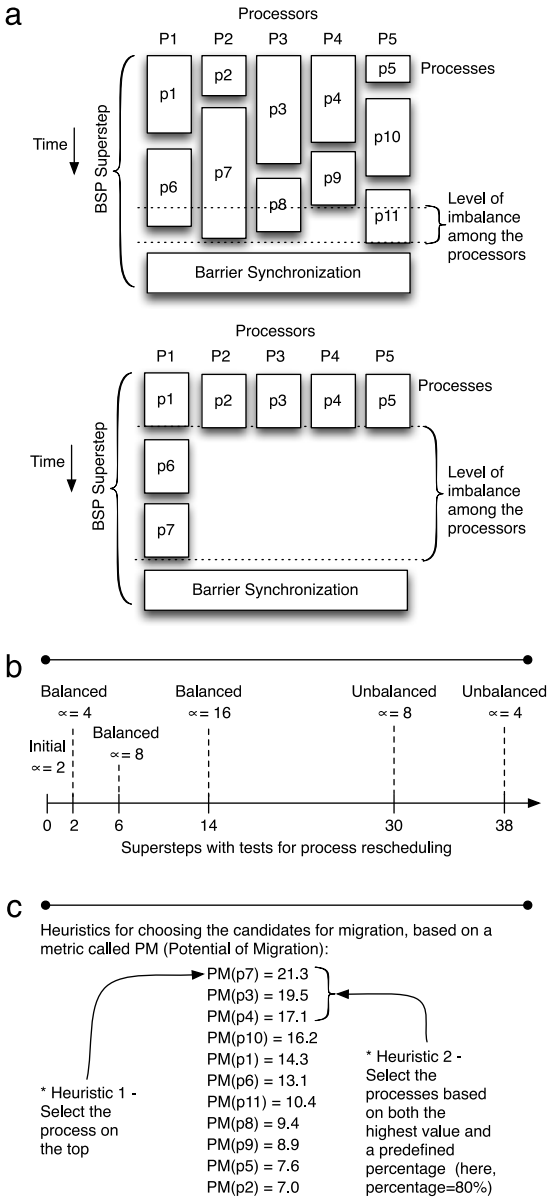

Fig. 1. Some drawbacks of MigBSP [17] that can be further explored: (a) False-positive (top) and false negative (bottom) imbalance detection situations caused when MigBSP monitors the time of the fastest and slowest processes instead of the time of processors; (b) MigBSP uses an index called α that can generate low reactivity to reorganize the processes when load imbalance takes place; (c) The user must specify a migration heuristic by hand, which cannot be optimized or until now can only be functional for a particular set of resources-application.

In this context, we propose MigPF which redesigns MigBSP with the keywords transparency, performance and automatism in mind. More precisely, MigPF addresses three aspects as denoted below:

(i) We redesigned the algorithm that measures whether the system is balanced or not, taking into account the behavior of both processes and processors;
(ii) We made the adaptable interval between supersteps more reactive in order to address imbalance situations in a better way;
(iii) We are providing *pf*: a prediction function to address process migration and their respective destinations, self-organizing these two load balancing decisions (*Which and Where*) without user intervention.

At each migration call, we are using *pf* and the largest PM of each process to on-the-fly create up to *n* new rescheduling

[2] Reactivity on process migration refers to speed to reach a balanced stated when an imbalanced state takes place [18].