Noname manuscript No.
(will be inserted by the editor)

Performance and energy task migration model for heterogeneous clusters

Esteban Stafford · José Luis Bosque

Received: DD Month YEAR / Accepted: DD Month YEAR

Abstract This article presents a set of linear regression models to predict the impact of task migration on different objectives, like performance and energy consumption. It allows to establish if at a given moment the migration of a task is profitable in terms of performance or energy consumption. Also, it can be used to determine the best node to migrate a task depending on the objective. The model uses a small set of parameters that are easily measurable. It has been validated against a small heterogeneous cluster using the Slurm resource manager. The model captures the tendencies observed in the results of the experiments, with average relative errors below 3.5% in execution time and 2.5% in energy consumption.

Keywords Task Migration · Performance estimation · Energy consumption · Heterogeneous Clusters

1 Introduction

Task migration is not a novel idea and it has been pursued many times over the last decades [1, 2]. The process of migration consists on interrupting the execution of a task that is running in a compute node, capturing a snapshot of its current state and transferring it to another node. This node can then restart the task and resume its execution from where it was interrupted, without losing any of the work done [3].

E. Stafford
Department of Computer Science and Electronics
University of Cantabria
E-mail: esteban.stafford@unican.es

J. L. Bosque
Department of Computer Science and Electronics
University of Cantabria
E-mail: joseluis.bosque@unican.es

In recent years, developments in HPC clusters make it appropriate to revisit this concept, because these are becoming increasingly heterogeneous [4, 5, 6, 7, 8, 9]. Currently, HPC clusters are composed of computation nodes with great differences in terms of the number and architecture of the processors, memory technologies and interconnection networks. Furthermore, operating these clusters is guided by new objectives, like energy consumption and efficiency. Correctly managing the heterogeneity through scheduling and load balancing techniques is essential to optimise the performance and energy [10, 11, 12, 13]. Another important challenge these clusters face is maintenance and resilience. The number of computer nodes is constantly growing, and there are already machines with millions of cores. For future exascale systems, it is the considered critical to develop strategies that make software resilient against failures. Finally, the fact that the completion times of scientific and big-data applications currently executed in HPC data-centres are well in the range of hours, or even days, makes the overhead of migration less relevant [14, 15].

This paper proposes a set of linear regression models to evaluate the impact of task migration in heterogeneous clusters. It is composed of several mathematical equations that predict the performance and energy consumption of a task that is migrated from one node to another. These equations can be used to determine if a migration is profitable or not, or to select the most adequate receiver from a set of available nodes. The model can be the basis for the implementation of preemptive scheduling and load balancing strategies, the improvement of fault resilience and more efficient system administration.

The model has been validated through experimentation in a small heterogeneous cluster with the well known resource manager Slurm [16]. This validation consist on performing a number of migrations of different benchmarks in various scenarios, where the execution time and energy consumption were measured. These values were compared to those predicted by the model, obtaining an average relative error below 3.5% in the performance estimation and 2.5% in the energy consumption.

The remainder of this article is structured as follows. Section 2 explains the migration concept in detail and presents the proposed models. Section 3 outlines the methodology employed and the experimental results, while Section 4 offers an empirical evaluation of the results. Section 5 deals with related work found in the literature. Finally, Section 6 summarises some concluding thoughts and future lines of work.

2 Task Migration Models

2.1 Migration Process

Migration heavily relies on task checkpointing tools. These implement two operations, *checkpoint* and *restart*. The first is capable of suspending the execution of a given task, storing its state in a *checkpoint file*. Conversely they are also able to read this file, recreating the state of the task and allowing

it to resume execution. With these tools, task migration is implemented by checkpointing in one node, transferring the checkpoint file to another node and resuming there the execution.

These steps have a given time duration that depends on the size of the memory footprint of the task, which add an overhead to the execution time of the task [17]. For a migration to be beneficial, the overhead must be compensated by the performance gain of the receiving node. Also, since the overhead is constant for each application, longer executions will make it less relevant. And since current HPC applications have execution times well in the range of hours, or even days, then task migration techniques are well suited for this environment.

In addition to reducing execution time of tasks, nowadays there is a great concern about energy consumption and efficiency in large HPC centres. In this regard, task migration can be used to relocate tasks to energy efficient nodes, thus improving the efficiency of the centre as a whole.

2.2 Performance Model

The construction of this model is based on computing the time a task spends in the three phases that it undergoes when it is migrated. These are, the time executed in the sender node T_s, followed by the time overhead of the migration process T_m, and finally the time of the execution performed in the receiver node T_r. Then the execution time of a migrated task T can be modelled as follows:

$$\begin{aligned} T &= T_s + T_m + T_r \\ &= T_b\alpha + T_m + T_b\frac{1-\alpha}{S} \\ &= T_b\left(\alpha + \frac{1-\alpha}{S}\right) + T_m \end{aligned} \tag{1}$$

Where the parameters of the model are the following:

- The *base time* T_b is the execution time of the task in the sender node without migration. It is common practice in resource managers where the users provide an expected execution time of the tasks they submit. For instance, one of the most popular scheduling algorithms, backfill, relies on this value.
- The *migration point* α specifies the portion of the execution of the task that runs in the sender node. This can be computed as the fraction of the current execution time divided by the base time, yielding values between 0 and 1. Values close to 0 indicate that the task is migrated at the beginning of the execution and higher values tell that the migration occurs later in the execution. When $\alpha = 1$, it means that the whole task runs in the sender node and a migration does not take place.

- The *speedup* S is an estimation of the performance improvement of the receiver node relative to the sender one. Values of speedup greater than 1 mean better performance of the receiver node, that will bring an improvement of the execution time. Conversely, the receiver node can also be slower than the sender, then giving values of S lower than 1.
- The *migration time* T_m is the time overhead introduced by the migration process, including the checkpoint, transfer and restart operations.

With the above expression, a necessary condition can be extracted to determine if the migration of a task between two nodes is advantageous at a given point in its execution.

$$\alpha + \frac{(1-\alpha)}{S} + \frac{T_m}{T_b} < 1 \tag{2}$$

2.3 Energy Model

Traditionally, migration has been carried out to reduce the execution time of a task, thus targeting nodes with more computational capacity but higher energy consumption. Nowadays there is a growing concern about the energy consumption of HPC clusters, therefore a model is provided to predict the total energy consumed by a task undergoing migration. It can be used to decide where to migrate task in order to achieve an energy saving objective.

This model is based on the same three phases described in the previous section, considering the energy as the product of time and power consumption. Thus, the total energy consumed by a migrated task can be modelled as:

$$\begin{aligned} E &= E_s + E_m + E_r \\ &= \alpha T_b P_s + E_m + \frac{1-\alpha}{S} T_b P_r \\ &= T_b \left(\alpha P_s + \frac{1-\alpha}{S} P_r \right) + E_m \end{aligned} \tag{3}$$

In addition to the parameters used in the performance model, this one includes the following:

- The power of the sender P_s and receiver P_r nodes.
- The migration energy E_m is the energy attributed to the three steps of the migration process.

Similar to the performance model it is also possible to obtain a necessary condition to determine if a migration is advantageous in terms of energy consumption.

$$\left(\alpha P_s + \frac{1-\alpha}{S} P_r \right) + \frac{E_m}{T_b} < 1 \tag{4}$$

Table 1 Summary of experiment details.

Experiment	Benchmark	Parameters	T_b	Memory
MD	Molecular Dynamics	Part=2500 Cycles=10	100s	7MB
MDL	Molecular Dynamics	Part=2500 Cycles=40	375s	7MB
NN	Mandelbrot	Depth=20000 Side=2000	100s	68MB
MNB	Mandelbrot	Depth=500 Side=12000	105s	2.15GB

It is worth pointing out that reducing the energy consumption can be achieved in two ways, by shortening the execution time or by consuming less power. It can happen that the receiver node is much faster than the sender and does not consume significantly more power, and due to the execution time reduction it provides an energy saving. Or on the contrary, the receiver node might not be much faster, so the execution time does not change significantly, but it is far more energy efficient, meaning that in the same time it consumes less energy than the sender.

3 Evaluation

The empirical evaluation of the models proposed in this article has been carried out in a cluster with 20 computational nodes. Each one has Kaby Lake Intel Core i5-7500 CPU with 4 cores and 8GB of DDR4 memory. This processor is operated in a range of clock frequencies between 1.7 GHz and 3.4 GHz. Therefore, the heterogeneity of the system is achieved by assigning different frequencies to different nodes, in such a way that they provide different computational power.

The Slurm Workload Manager [16] is used to manage the tasks in the cluster and the Distributed MultiThreaded CheckPointing (DMTCP) [18] has been selected as the migration tool. The DMTCP Slurm plug-in described in [19] has been used, with some implementation adjustments.

A set of benchmarks were used to represent the behaviour of common HPC workloads. These are Matrix Multiplication, N-Body, Molecular Dynamics, Mandelbrot, LU and Heated Plate. Although the conclusions of this article are based on the results of all these benchmarks, due to space restrictions the graphs shown are restricted to two of them that were deemed representative of the set: Mandelbrot and Molecular Dynamics.

For each of the two benchmarks two sets of execution parameters were used, as shown in Table 1. The Molecular Dynamics experiments have the same footprint and different execution time, to evaluate the impact of task duration in the model predictions. Similarly the Mandelbrot experiments have practically the same execution time but different footprint, to observe how the model behaves with the memory size of the task.

The metrics used to validate the models are the execution time and the energy consumption. The latter was obtained through the Slurm accounting

plugin, that uses RALP to give the addition of the package and DRAM energies.

Performance model evaluation The model requires the definition of two parameters, the migration time and the speedup of the second node. The former was estimated by measuring the time difference between executions of the benchmark with and without migration. It includes the checkpoint, wait and restart time. Checkpoint and restart times are fairly deterministic and depend on the memory footprint of the benchmark, [17]. However the most part of the migration time is the wait time introduced by the periodic nature of the scheduling algorithm of Slurm. On average the migration time is 6.15 seconds, with standard deviation of 2.3. The second parameter is the relative speedup between the nodes involved in the migration. Since all the nodes in the cluster have the same architecture, the speedup can be estimated by dividing the clock frequencies of the CPUs: $S = \frac{F_1}{F_2}$.

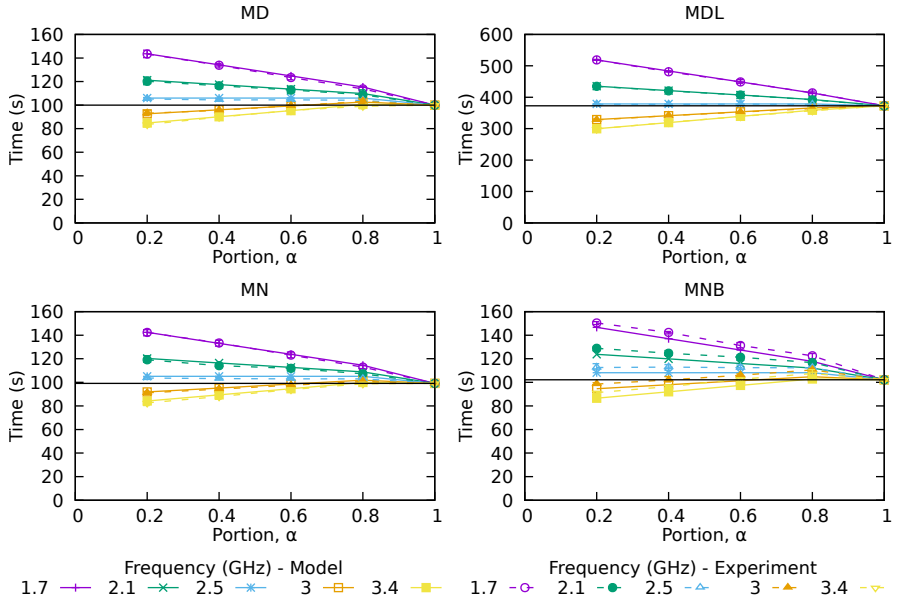

Fig. 1 Comparison of performance model with experiments.

Figure 1 shows the execution times of the benchmarks in different scenarios as described in Table 1. The horizontal axis represents the point in the execution when the migration takes place (α). Points on the left side mean early migrations, and advancing on the axis means delaying the migration. The last point ($\alpha = 1$) means no migration at whatsoever. All the executions start in a node running at 2.5 GHz, the different lines in the graphs show the frequency of the node that the benchmark is migrated to. In addition Figure 2 shows the relative error between the model and the measured execution times.

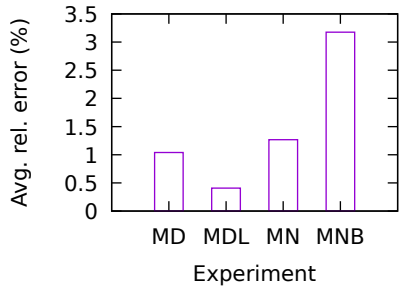

Fig. 2 Error of performance model.

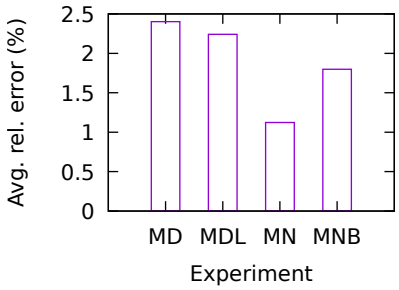

Fig. 3 Error of energy model.

Energy model evaluation The energy model requires some parameters in addition to those of the performance model. These are the power consumption of the sender and receiver nodes. This power is heavily dependant on the nodes and the benchmarks. Even with nodes with the same hardware and performance, there are small differences in the power, that over long periods of time can amount to large differences in the total energy consumption. These small differences can be seen in Figure 4, that shows the power consumption of several benchmarks in two different nodes.

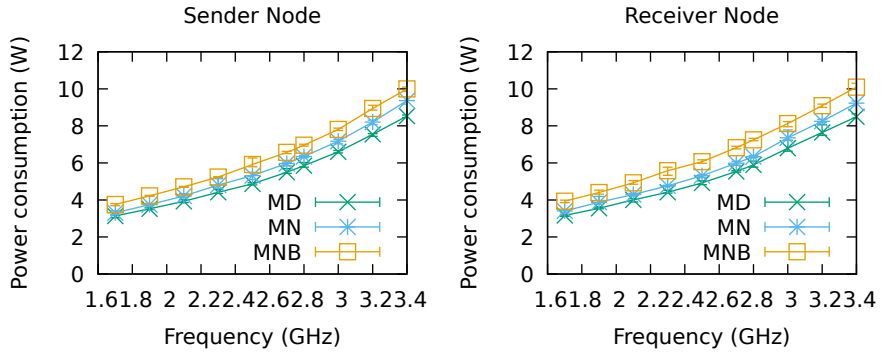

Fig. 4 Power consumption of two nodes with different benchmarks.

Similar to the performance experiments, Figure 5 shows the energy consumed by each execution varying the frequency of the receiving node and the point (α) at which migration takes place. Finally, Figure 3 shows the average relative error commited by the energy model.

4 Discussion

In light of the experimental results described in the previous section it is possible to discuss the validity of the models presented in this article.

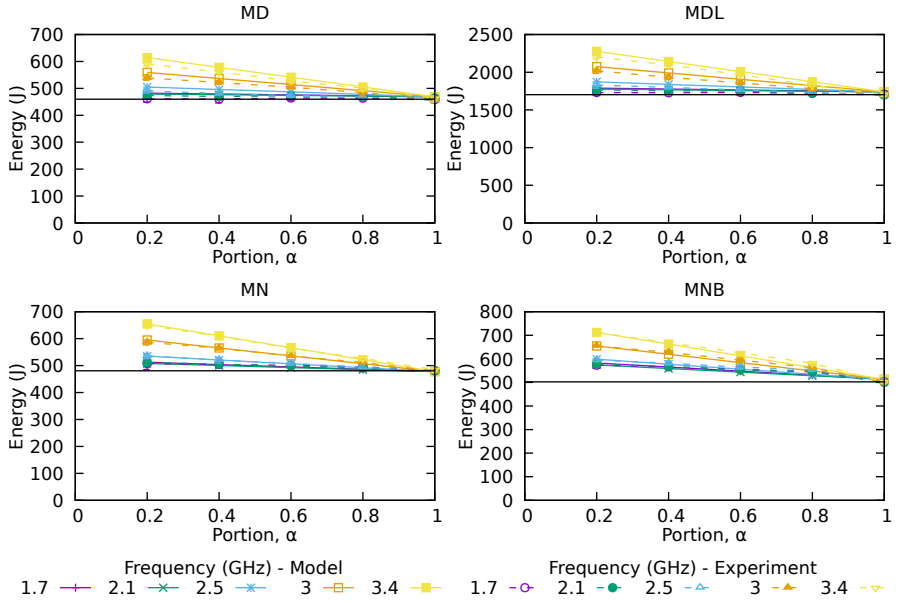

Fig. 5 Comparison of energy model with experiments.

At first glance, the empirical data matches the general spirit predicted of the models. In terms of performance, regarding the situations in which migration is profitable, the experiments satisfying equation 2 lie below the black horizontal line (Fig. 1). In more detail, comparing the Molecular Dynamics experiments (MD and MDL), the error of the prediction is substantially decreased when the execution time grows (MDL). This is explained by considering the absolute error, that in these two experiments is very similar, meaning that it does not depend on the length of the application (Fig. 2). These errors will be negligible if real HPC applications are used, that have execution times in the range of hours or even days. Also, longer executions make that the migration is profitable closer to the end of the execution, because the impact of the migration time is comparably less relevant.

Regarding the Mandelbrot experiments (MN and MNB), the error in the predictions of the model doubles when the memory footprint is increased up to 2GB. This is a consequence of two facts, first the migration time of the MNB experiment being slightly higher, due to the large size of the checkpoint file. Second, MNB also has a higher memory access rate, which leads to lower speedup than what the frequency ratio estimates.

Regarding the energy consumption, Figure 5 shows that in this instance, there is no energy saving in task migration. The reason behind this is that the power consumption of the second node is slightly higher than the first (Fig. 4). Note that if the blue line (2.5GHz) were horizontal, it would mean that both nodes consume the same power. The model correctly predicts that, in Molecular Dynamics (MD and MDL), migrating to a slower node reduces the

power consumption but makes the execution time longer, having no impact in the total energy consumption. On the other hand migrating to the fastest node reduces the execution time but at a higher power consumption, which can be seen that increases the consumed energy up to 35% with the earliest migration. As for the precision of the model, Figure 3 presents the average relative error in relation to the experiments. It can be seen that this error is under 2.5% in all cases.

5 Related Work

Task migration is not a novel concept, and several proposals have been presented both, to improve the migration mechanism itself, as well as to be used in scheduling, load balancing and resiliency. On the one hand, there are a series of papers that focus on the development of Checkpoint/Restart (C/R) techniques and tools, to migrate running tasks between compute nodes. In this area [20] presents a simple and transparent method of task migration based on coordinated checkpointing of MPI applications, although it introduces a not negligible overhead because all the processes need to be checkpointed.

A more modern migration mechanism for HPC environments is analysed in [4]. The authors propose a protocol guaranteeing local consistency on a per-connection basis. In contrast to classical C/R approaches, where all processes of a job have to synchronize, this condition is sufficient resulting in a reduced complexity. [9] provides an extensive analysis of the performance, energy and I/O costs associated with a wide array of checkpointing policies. [21] presents a C/R scheme to balance the workload of computational nodes as well as for fault-tolerance algorithms in HPC clusters. The rise of heterogeneous GPU-based systems has led several authors to propose C/R mechanisms for theses systems. For instance, CudaCR [22] presents an optimized schedule strategy for memory corruptions in the device. CLPKM is a framework that provides an abstraction layer between OpenCL applications and the underlying OpenCL runtime to enable preemption of a kernel execution instance based on a software checkpointing mechanism [23]. CRState inserts primitives into OpenCL programs at compile time, so the checkpoint/restart can happen at predetermined places and the major components of the computation state will be extracted at runtime [24].

Distributed Multithreaded Checkpointing (DMTCP)[1] [18] is the checkpointing tool that was selected to evaluate the models proposed in this article. It executes exclusively in userspace, meaning that it does not require modifying the kernel or the configuration of the operating system. It was selected because of its use in HPC environments and its support for MPI task checkpointing.

Other authors focus on using task migration for scheduling and load balancing in HPC systems [5].

In [25] a load balancing algorithm for clusters of multicore processors is presented and discussed. In this algorithm the Extremal Optimization approach

[1] The code is available at https://github.com/dmtcp

is used to periodically detect the best tasks as candidates for migration and for a guided selection of the best computing nodes to receive the migrating tasks. To decrease the complexity of selection for migration, the embedded EO algorithm assumes a two-step stochastic selection during the solution improvement based on two separate fitness functions.

Later, in [8] the same authors present a multi-objective load balancing algorithm based on extremal optimization. It uses three objectives relevant to load balancing: computational load balance of processors, the volume of interprocessor communication and task migration metrics. Extremal optimization is used to find task migrations which dynamically improve processor load balance in a distributed system.

Other authors propose the Task Packing algorithm, a scheduler policy of the MPI tasks, to re-allocate the tasks within each node based on oversubscription. That is, running an application with a number of OS level tasks larger than the number of available cores [7]. The reallocation is computed using a packing strategy based on a particular case of the well-known Knapsack algorithm.

In [26] an HPC scheduler is presented that applies co-scheduling and utilizes virtual machine migration for a re-orchestration of applications at runtime, based on their main memory bandwidth requirements. The migration model in this case is strongly related with the total main memory bandwidth utilization.

Related to energy consumption [27] presents H-ENERGYLB a heterogeneous energy-aware load balancer which reduces the average power demand of systems with heterogeneous processors and saves energy when applied over iterative applications with imbalanced loads. The decision of task migration is based on a threshold on the load imbalance of the nodes, thus no performance or energy considerations are taken.

Migration has also been used to improve resiliency. The authors of [28] promote process-level live migration combined with health monitoring for a proactive fault tolerance (FT) approach that complements existing C/R schemes with self-healing. This work differs from ours in that the models used here focus on analysing the health of the computation node rather than its performance or energy consumption.

All these articles show that the topic addressed is relevant and up-to-date. However, none of them presents mathematical models capable of predicting the improvement, both in performance and energy consumption, of task migration between two of nodes of a cluster. In addition, none of them has been incorporated to such a widely used workload manager as SLURM, which would allow many datacenters and HPC systems to benefit from it.

6 Conclusions

This article presents a set of models that predict the execution time and energy consumption of a task when it is migrated from one node to another. The

models are adapted to the context of heterogeneous clusters where the performance and power consumption of the nodes may differ from one another. These models allow determining whether a migration is going to be advantageous in terms of performance and/or energy consumption. Furthermore, the models are based on linear regression and rely on a small set of parameters that can be easily obtained.

The models have been evaluated empirically against the results of migrating several benchmarks in different scenarios with nodes of different performance and power consumption. This has shown that the models predict the experimental results with average relative error below 3.5% in performance and 2.5% in energy consumption.

In the future, efforts can be made in order to better estimate the speedup of the migrated application, thus improving the precision of the models. Also, the models can be upgraded to consider more complex scenarios, like parallel applications or hardware accelerators. The presented models can be the heart of preemptive scheduling algorithms that are able to improve performance or energy consumption of heterogeneous clusters.

Acknowledgements This work has been supported by the Spanish Science and Technology Commission under contract PID2019-105660RB-C22 and the European HiPEAC Network of Excellence.

References

1. S. Petri and H. Langendörfer. Load balancing and fault tolerance in workstation clusters migrating groups of communicating processes. *SIGOPS Oper. Syst. Rev.*, 29(4):25–36, October 1995.
2. M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Trans. Comput. Syst.*, 15(3):253–285, August 1997.
3. D. S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou. Process migration. *ACM Comput. Surv.*, 32(3):241–299, September 2000.
4. Simon Pickartz, Stefan Lankes, Antonello Monti, Carsten Clauss, and Jens Breitbart. Application migration in HPC - A driver of the exascale era? *Int. Conference on High Performance Computing and Simulation, HPCS 2016*, pages 318–325, 2016.
5. Y. Jiang. A survey of task allocation and load balancing in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(2):585–599, Feb 2016.
6. J. L. J. Laredo, F. Guinand, D. Olivier, and P. Bouvry. Load balancing at the edge of chaos: How self-organized criticality can lead to energy-efficient computing. *IEEE Transactions on Parallel and Distributed Systems*, 28(2):517–529, Feb 2017.
7. Gladys Utrera, Montse Farreras, and Jordi Fornes. Task Packing: Getting the Best from MPI Unbalanced Applications. *Proceedings - 2017 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2017*, pages 547–550, 2017.
8. Ivanoe De Falco, Eryk Laskowski, Richard Olejnik, Umberto Scafuri, Ernesto Tarantino, and Marek Tudruj. Effective processor load balancing using multi-objective parallel extremal optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '18, pages 1292–1299, New York, NY, USA, 2018. ACM.
9. N. El-Sayed and B. Schroeder. Understanding practical tradeoffs in hpc checkpoint-scheduling policies. *IEEE Transactions on Dependable and Secure Computing*, 15(2):336–350, March 2018.

10. J. L. Bosque, P. Toharia, O. D. Robles, and L. Pastor. A load index and load balancing algorithm for heterogeneous clusters. *The Journal of Supercomputing*, 65(3):1104–1113, 2013.
11. M. R. Belgaum, S. Soomro, Z. Alansari, M. Alam, S. Musa, and M. M. Su'ud. Load balancing with preemptive and non-preemptive task scheduling in cloud computing. pages 1–5, Aug 2017.
12. B. Pérez, E. Stafford, J. L. Bosque, and R. Beivide. Energy efficiency of load balancing for data-parallel applications in heterogeneous systems. *J. Supercomput.*, 73(1):330–342, 2017.
13. A. Cabrera, A. Acosta, F. Almeida, and V. Blanco. A dynamic multi–objective approach for dynamic load balancing in heterogeneous systems. *IEEE Transactions on Parallel and Distributed Systems*, 31(10):2421–2434, 2020.
14. Dominik Bartuschat and Ulrich Rüde. Parallel multiphysics simulations of charged particles in microfluidic flows. *J. Comput. Science*, 8:1–19, 2014.
15. O. D. Robles, J. L. Bosque, L. Pastor, and A. Rodriguez. Performance analysis of a cbir system on shared-memory systems and heterogeneous clusters. In *Seventh International Workshop on Computer Architecture for Machine Perception (CAMP'05)*, pages 309–314, 2005.
16. A. B. Yoo, M. A. Jette, and M. Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, pages 44–60, Berlin, Heidelberg, 2003.
17. E. Stafford and J. L. Bosque. Improving utilization of heterogeneous clusters. *The Journal of Supercomputing*, Jan 2020.
18. J. Ansel, K. Arya, and G. Cooperman. Dmtcp: Transparent checkpointing for cluster computations and the desktop. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–12, May 2009.
19. Manuel Rodríguez-Pascual, Jiajun Cao, José A. Moríñigo, Gene Cooperman, and Rafael Mayo-García. Job migration in hpc clusters by means of checkpoint/restart. *The Journal of Supercomputing*, 75(10):6517–6541, Oct 2019.
20. Jiannong Cao, Yinghao Li, and Minyi Guo. Process migration for MPI applications based on coordinated checkpoint. *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, 1:306–312, 2005.
21. Nils Kohl, Johannes Hötzer, Florian Schornbaum, Martin Bauer, Christian Godenschwager, Harald Köstler, Britta Nestler, and Ulrich Rüde. A scalable and extensible checkpointing scheme for massively parallel simulations. *The International Journal of High Performance Computing Applications*, 33(4):571–589, 2019.
22. B. Pourghassemi and A. Chandramowlishwaran. Cudacr: An in-kernel application-level checkpoint/restart scheme for cuda-enabled gpus. In *Int. Conference on Cluster Computing, CLUSTER*, pages 725–732. IEEE Computer Society, 2017.
23. Ming-Tsung Chiu and Yi-Ping You. Clpkm: A checkpoint-based preemptive multitasking framework for opencl kernels. *Journal of Systems Architecture*, 98:53 – 62, 2019.
24. G. Chen, J. Zhang, Z. Zhu, Jiang Q., H Jiang, and C Pang. Crstate: checkpoint/restart of opencl program for in-kernel applications. *The Journal of Supercomputing*, 2020.
25. Ivanoe De Falco, Eryk Laskowski, Richard Olejnik, Umberto Scafuri, Ernesto Tarantino, and Marek Tudruj. Extremal Optimization applied to load balancing in execution of distributed programs. *Applied Soft Computing Journal*, 30:501–513, 2015.
26. Jens Breitbart, Simon Pickartz, Stefan Lankes, and Antonello Monti. Dynamic Co-scheduling Driven by Main Memory Bandwidth Utilization. 2017.
27. E. Padoin, M. Diener, P. Navaux, and J. F. Mehaut. Managing power demand and load imbalance to save energy on systems with heterogeneous CPU speeds. *Symposium on Computer Architecture and High Performance Computing*, pages 72–79, 2019.
28. Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Proactive process-level live migration and back migration in HPC environments. *Journal of Parallel and Distributed Computing*, 72(2):254–267, 2012.