Large-Scale Experiment for Topology-Aware Resource Management

Yiannis Georgiou[1], Guillaume Mercier[2](✉), and Adèle Villiermet[3]

[1] Atos–Bull, Grenoble, France
yiannis.georgiou@atos.net
[2] Bordeaux INP, Talence, France
guillaume.mercier@bordeaux-inp.fr
[3] Inria Bordeaux Sud-Ouest, Talence, France
adele.villiermet@inria.fr

Abstract. A Resource and Job Management System (RJMS) is a crucial system software part of the HPC stack. It is responsible for efficiently delivering computing power to applications in supercomputing environments and its main intelligence relies on resource selection techniques to find the most adapted resources to schedule the users' jobs. In [8], we introduced a new topology-aware resource selection algorithm to determine the best choice among the available nodes of the platform based on their position in the network and on application behaviour (expressed as a communication matrix). We did integrate this algorithm as a plugin in Slurm and validated it with several optimization schemes by making comparisons with the default Slurm algorithm. This paper presents further experiments with regard to this selection process.

Keywords: Resource management · Job allocation
Topology-aware placement · Scheduling · Slurm

1 Introduction

Computer science is more than ever a cornerstone of scientific development, as more and more scientific fields resort to simulations in order to help refine the theories or conduct experiments that cannot be carried out in reality because of their scale or their prohibitive cost. Currently, such computing power can be delivered only by parallel architectures. However, harnessing the power of a large parallel computer is no easy task, because of several factors. It features most of the time a huge amount of computing nodes, and this scale has to be taken into account when developing applications. Then, the nodes architecture has become more and more complex, as the number of cores per node is in constant increase from one generation of CPU to the next. One way of dealing with this complexity would be to take into account the application behavior (e.g. its communication pattern, or its memory access pattern) and to deploy it on the computer accordingly by mapping processes to cores depending on their affinity [9]. However, since a parallel machine can be very large, it is often shared by many users running their applications at the same time. In such a case, an application execution will depend on a nodes allocation that has been determined by the Resources and Jobs Management System (RJMS). Most of the time, a RJMS works in a best-effort fashion, which can lead to suboptimal allocations. As a consequence, we did investigate in [8] the idea of taking into account an application behaviour directly in the RJMS, in its process of allocation and reservation of computing resources (nodes). We carried out experimental validation on small scales and did conduct simulations for larger scales. In this paper, we shall present larger experiments (not simulations) to confirm our simulations results. This paper is organized as follows: Sect. 2 gives an overview of the context and background of this work. It introduces the software elements leveraged by this work before giving some technical insights about the integration of TreeMatch into Slurm. Then Sect. 3 shows and explains the results obtained. We discuss the comparison between simulation and emulation in Sect. 4 while some related works are listed in Sect. 5. Finally, Sect. 6 concludes this paper.

2 Context and Background

A substantial part of this work deals with the integration of a new resource allocation and reservation policy within the Slurm [14] RJMS. This policy takes into consideration application behaviour and a matching between the needed resources and the behaviour is determined, thanks to a dedicated algorithm called TreeMatch [10]. We now describe both software elements in this section.

2.1 SLURM

Simple Linux Utility Resource Management (a.k.a SLURM) is a RJMS used and deployed on a large number of parallel machines.

Its resource selection process takes place as part of the global job scheduling procedure. In particular, this procedure makes use of the `plugin/select`, which is responsible for allocating the computing resources to the jobs. There are various resource selection plugins in Slurm that can take into account the specificities of the underlying platforms' architecture such as `linear` and `cons_res`. The `select/cons_res` plugin is ideal for multicore and manycore architectures where nodes are viewed as a collection of consumable resources (such as cores and memory). In this plugin, nodes can be used exclusively or in a shared mode where a job may allocate its own resources differently than the other jobs sharing the same node [1].

2.2 TreeMatch

TreeMatch [10], is a library for performing process placement based on the topology of the underlying machine and the behaviour of the application. This behaviour can be expressed in several ways: communication scheme, memory accesses pattern, etc. As for the target architectures, TreeMatch is able to deal with multicore, shared memory machines as well as distributed memory machines. It computes a permutation of the processes to the processors/cores in order to minimize some cost function (e.g. communication costs). To be more specific, it takes as input a tree topology (where the leaves stand for computing resources and internal nodes correspond to switches or cache levels) and a matrix describing the affinity graph between processes. Such a matrix can be obtained using an application monitoring tool [2]. A hierarchy is extracted from this graph that matches the topology tree hierarchy. The outcome is therefore a mapping of the processes onto the underlying computing resources. The objective function optimized by TreeMatch is the Hop-Byte [15], that is, the number of hops weighted by the communication cost: Hop-Byte$(\sigma) = \sum_{1 \leq i<j \leq n} \omega(i,j) \times d(\sigma(i), \sigma(j))$, where n is the number of processes to map, σ is the process permutation produced by TreeMatch (process i is mapped on computing resource $\sigma(i)$), $A = (\omega_{i,j})$ $1 \leq i \leq n$, $1 \leq j \leq n$ is the affinity matrix between these entities and hence $\omega(i,j)$ is the amount of data exchanged between process i and process j and $d(p_1, p_2)$ is the distance, in number of hops, between computing resources p_1 and p_2. An important feature of TreeMatch lies in its ability to take *constraints* into account. When not all leaves are available for mapping (because some of them are already used by other applications as it is the case in this paper), there is a possibility to restrict the leaves onto which processes can be mapped so that only a subset of nodes is used for the mapping.

2.3 TreeMatch Integration Within SLURM

We have implemented a new selection option for the Slurm cons_res plugin. In this case the regular best-fit algorithm used for nodes selection is replaced by our TreeMatch variant. To this end, we need to provide three pieces of information: the job affinity matrix, the hardware topology but also the constraints due to other jobs allocations. The communication matrix is provided at job submission time through a distribution option available in the `srun` command. As for the global cluster topology, it is provided to the controller by a new parameter in a Slurm configuration file. Whenever a job allocation is computed, this topology is completed by the constraints information. These constraints are provided by the nodes and cores bitmaps used by the Slurm controller to describe the cluster utilization. TreeMatch then utilizes all these pieces of information to compute the allocation of resources tailored for the submitted job. However, as the TreeMatch overhead increases with the size of the hardware topology (in terms of nodes count), we improve the computation time by restricting the search in a fitting subtree in the global architecture.

3 Experimental Validation

We presented some preliminary results in [8] that we completed with new experiments described in this section. We carried out experiments on a larger scale than previously and we also make comparisons between these real-world results and the simulations of large-scale experiments shown in [8] to demonstrate the accuracy of the simulator used in our work. Our experiments have been carried out on the Edel cluster from the Grid'5000 Grenoble site. Edel is composed of 72 nodes featuring 2 Intel Xeon E5520 CPUs (2.27 GHz, 4 cores/CPU) and 24 GB of memory. We use the Edel cluster to emulate Curie (a TGCC cluster with 5040 nodes and 80640 cores[1]) using an Slurm internal emulation technique called `multiple-slurmd` initially described and used in [6]. We base our experiments on a Curie workload trace taken from the Parallel Workload Archive[2]. Two sets of jobs are considered: the first one fills the cluster, and the jobs belonging to this set are always scheduled using Slurm in order to have the same starting point for all the experiments. The second set, called the *workload*, is the one we actually use to compare the different strategies. All the measurements are done through the Slurm login system which gives us workload traces similar to the ones obtained from Curie. Finally, we need to provide each job with a communication matrix in order to use TreeMatch. For these experiments we use randomly generated matrices featuring various sparsity rates. Since we do not know the real nature of the jobs executed on Curie, creating random matrices is acceptable as the only available data from the original workload is a job duration. However, in a real setting, we will need the user to provide the communication matrix of its application. This can be done through monitoring in the MPI library with [2]. Moreover, in the real case, it may happen that not every application can provide their communication matrix. We have studied this in simulation in [7] and show that the whole system can benefit from this approach even though only a fraction of the applications provide their communication matrix.

We made comparisons between three cases: the classical topology-aware Slurm selection mechanism (SLURM), the same mechanism but using TreeMatch for process placement after the allocation process and just before the execution starts (TM-A) and last when TreeMatch is used both for the allocation process *and* for the process placement using the subtree technique to reduce the overhead (TM-Isub).

Three metrics have been used in this performance assessment: two of them regard the whole workload while the last one concerns each individual job:

– *makespan* measures the time taken between the submission of the first job and the completion of the last job of the *workload*.
– *utilization* represents the ratio between the CPUs used and the total number of CPUs in the cluster during the execution of the *workload*.
– *job flowtime (or turnaround time)* represents the time taken between the submission and the completion of a given job.

In our previous work, the *workload* comprised about 60 jobs. In this paper we used a much larger *workload* of 1500 jobs. To keep jobs duration reasonable

[1] http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm.
[2] http://www.cs.huji.ac.il/labs/parallel/workload/.

Com	SLURM	TM-A	TM-Isub
50%	51002	38252	37230
33%	50997	45897	41817

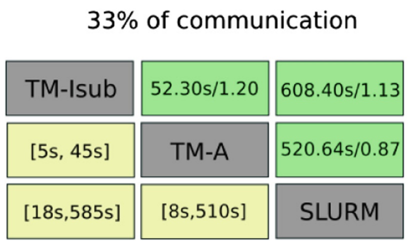
(a) Makespan

Com	SLURM	TM-A	TM-Isub
50%	34%	44%	46%
33%	34%	38%	42%

(b) Utilization

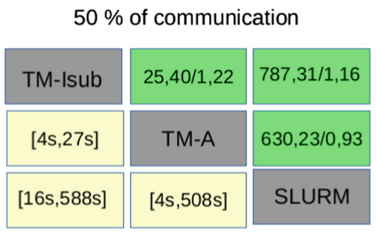
Fig. 1. Workload metrics for various strategies and different amounts of communication ratio. Emulation of the Curie cluster with the Edel cluster (Grid'5000)

we decreased their runtime by a 50% factor. This reduction factor impacts the flowtime and was used to keep our experiments under the 48 h time limit for each case. Figure 1 describes the results obtained for this workload and two different communication ratios of the jobs. The communication represents the ratio between the communication time over the whole runtime of the application. These ratios are fixed to 33% and 50% to illustrate the case of a communication bound application (50% of communication ratio) and more compute-bound cases (1/3 of communication time). However, as they are not an input of the algorithm, we will not need to measure it in a real setting. Here, we use these ratios to see their impact on the performance of the algorithm. Figure 1a shows that using TreeMatch to reorder the process ranks of the jobs reduces the makespan, but using it inside Slurm to allocate nodes decreases it even more. We can also see that the larger the communication ratio, the greater the gain. This is an expected outcome, as TreeMatch reduces the communication times. Figure 1b also shows that for the same submission workload, TreeMatch improved the resource utilization.


33% of communication

TM-Isub	52.30s/1.20	608.40s/1.13
[5s, 45s]	TM-A	520.64s/0.87
[18s,585s]	[8s,510s]	SLURM

(a) 33% of communication


50 % of communication

TM-Isub	25,40/1,22	787,31/1,16
[4s,27s]	TM-A	630,23/0,93
[16s,588s]	[4s,508s]	SLURM

(b) 50% of communication

Fig. 2. Statistical comparison of selection methods: flow time. Emulation of the Curie cluster with the Edel cluster (Grid'5000)

In Fig. 2, we use paired comparisons between different strategies for jobs flowtime. In this case, we considered job-wise metrics, as we want to understand if, when we average all the jobs, a strategy turns out to be better than another. Each strategy is displayed on the diagonal. On the upper right, we have the average difference between the strategy on the column and the one on the row and the geometric mean of the ratios. For instance, we see that on average the job flowtime is 608.40 s faster with TM-Isub than with SLURM and the average ratio is 1.13. On the lower left part, we plot the 90% confidence interval of the corresponding mean. The interpretation is the following: if the interval is positive, then the strategy on the row is better than the strategy on the column with a 90% confidence. In this case, the corresponding mean is highlighted in green. If the interval is negative, the strategy on the line is better than the one on row and the corresponding mean is highlighted in red. Otherwise, we cannot statistically conclude with a 90% confidence on which strategy is the best and we do not highlight the corresponding mean. For example, we can see that using TreeMatch in SLURM is better than not using it.

4 Comparison Between Simulation and Emulation

As explained in the previous section, we have emulated the Slurm execution of the Curie machine using the Edel cluster of Grid'5000. As a matter of fact, it is not possible to experiment new scheduling strategies for a batch scheduler on a production machine. As the Slurm engine is unmodified, this emulator is, in any cases, very close to the real behavior of Slurm. On the other hand in [7,8], we have used a simulator to perform extensive tests on different settings. Here, we present early results to validate the simulator using emulation measurements. In Fig. 3 we present the comparison between the simulation and the emulation flowtime for the 1500 jobs used in experiments of the previous sections. Measuring the average flowtime is very important as this metric assesses each job independently and is less affected by the last submitted job than the makespan.

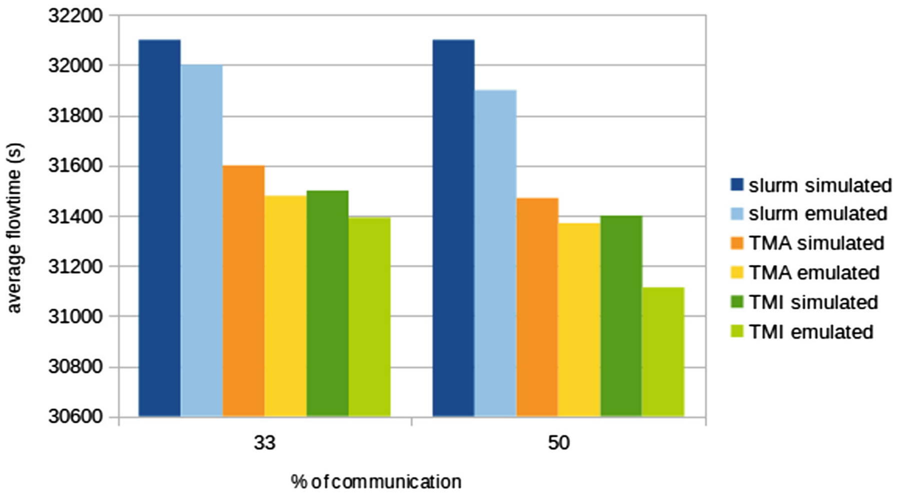


Fig. 3. Comparison of the emulated flowtime vs. the simulated one for 1500 jobs of the Curie trace. Remark that the Y-axis does not start at 0.

We plot the results for the different strategies (plain Slurm, TreeMatch after (TMA) or TreeMatch inside Slurm (TMI)). We see that the simulator keeps the order of the emulator concerning this metric: in both cases, TMI is better than TMA that is better than plain Slurm. Moreover, we see that simulation results are very close to emulation results (be aware that the y-axis does not start at 0). In all cases, the simulator has at least 6% of accuracy.

Having a simulator that is very close to the emulator is very important. This justifies the use of simulations and hence saves a lot of experimental time and allows for testing many different settings.

5 Related Works

Many RJMS take advantage of the hardware topology to provide compact and contiguous allocations (Slurm [14], PBS Pro [12], Grid Engine [11], and LSF [13] or Fujitsu [5]) so as to reduce the communication costs during the application execution (switches that are the deeper in the topology tree are supposed to be cheaper communication-wise than upper ones). Some other RJMS offer task placement options that can enforce a clever placement of the application processes. It is the case of Torque [4] which proposes a NUMA-aware job task placement. OAR [3] uses a flexible hierarchical representation of resources which offers the possibility to place the application processes upon the memory/cores hierarchy within the computing node. However, to the best of our knowledge there is no work that considers the application communication pattern to optimize the HPC resource selection and mapping.

6 Conclusion and Future Work

Job scheduling plays a crucial role in a cluster administration and utilization, enabling both a better response time and a better resource usage. In this paper, we have presented the results of large-scale experiments using our allocation policy that allocates and maps at the same time application processes onto the computing resources, based on the behaviour (a communication matrix in our case) of the considered application. This strategy has been implemented in Slurm. We have tested this strategy on emulation and compared it with the standard Slurm topology-aware policy and the method consisting in mapping processes after the allocation is determined.

Results show that our solution yields better makespan, flowtime and utilization compared to these approaches and especially to the standard Slurm policy, which is what we had shown through simulation only in our previous work.

To get further insights we plan to compare these large-scale experiments with the results obtained through simulation in more details.

Acknowledgments. Experiments presented in this paper were carried out using the Grid'5000 testbed (see https://www.grid5000.fr). Part of this work is also supported by the ANR MOEBUS project ANR-13-INFR-0001 and by the ITEA3 COLOC project #13024.

References

1. Balle, S.M., Palermo, D.J.: Enhancing an open source resource manager with multi-core/multi-threaded support. In: Frachtenberg, E., Schwiegelshohn, U. (eds.) JSSPP 2007. LNCS, vol. 4942, pp. 37–50. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78699-3_3
2. Bosilca, G., Foyer, C., Jeannot, E., Mercier, G., Papaure, G.: Online dynamic monitoring of MPI communication. In: 23rd International European Conference on Parallel and Distributed Computing (EuroPar), p. 12. Santiago de Compostella, August 2017. Extended Verion: https://hal.inria.fr/hal-01485243
3. Capit, N., Da Costa, G., Georgiou, Y., Huard, G., Martin, C., Mounié, G., Neyron, P., Richard, O.: A batch scheduler with high level components. In: Cluster Computing and Grid 2005 (CCGrid 2005). IEEE, Cardiff (2005). https://hal.archives-ouvertes.fr/hal-00005106
4. Adaptive Computing: Torque Resource Manager. http://docs.adaptivecomputing.com/torque/6-0-0/Content/topics/torque/2-jobs/monitoringJobs.htm
5. Fujitsu: Interconnect Topology-Aware Resource Assignment. http://www.fujitsu.com/global/Images/technical-computing-suite-bp-sc12.pdf
6. Georgiou, Y., Hautreux, M.: Evaluating scalability and efficiency of the resource and job management system on large HPC clusters. In: Cirne, W., Desai, N., Frachtenberg, E., Schwiegelshohn, U. (eds.) JSSPP 2012. LNCS, vol. 7698, pp. 134–156. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35867-8_8
7. Georgiou, Y., Jeannot, E., Mercier, G., Villiermet, A.: Topology-aware job mapping. Int. J. High Perform. Comput. Appl. **32**(1), 14–27 (2018)
8. Georgiou, Y., Jeannot, E., Mercier, G., Villiermet, A.: Topology-aware resource management for HPC applications. In: Proceedings of 18th International Conference on Distributed Computing and Networking, Hyderabad, India, 5–7 January 2017, p. 17. ACM, Hyderabad, January 2017
9. Jeannot, E., Mercier, G.: Near-optimal placement of MPI processes on hierarchical NUMA architectures. In: D'Ambra, P., Guarracino, M., Talia, D. (eds.) Euro-Par 2010. LNCS, vol. 6272, pp. 199–210. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15291-7_20
10. Jeannot, E., Mercier, G., Tessier, F.: Process placement in multicore clusters: algorithmic issues and practical techniques. IEEE Trans. Parallel Distrib. Syst. **25**(4), 993–1002 (2014). https://doi.org/10.1109/TPDS.2013.104
11. Oracle: Grid Engine. https://blogs.oracle.com/templedf/entry/topology_aware_scheduling
12. PBSWorks: PBS. http://www.pbsworks.com/PBSProduct.aspx?n=PBS-Professional&c=Overview-and-Capabilities
13. Smith, C., McMillan, B., Lumb, I.: Topology aware scheduling in the LSF distributed resource manager. In: Proceedings of Cray User Group Meeting (2001)
14. Yoo, A.B., Jette, M.A., Grondona, M.: SLURM: simple Linux utility for resource management. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS, vol. 2862, pp. 44–60. Springer, Heidelberg (2003). https://doi.org/10.1007/10968987_3
15. Yu, H., Chung, I.H., Moreira, J.: Topology mapping for Blue Gene/L supercomputer. In: Supercomputing 2006. ACM, New York (2006). https://doi.org/10.1145/1188455.1188576