



**HAL**  
open science

## Communication models insights meet simulations

Pierre-François Dutot, Millian Poquet, Denis Trystram

► **To cite this version:**

Pierre-François Dutot, Millian Poquet, Denis Trystram. Communication models insights meet simulations. 13th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar'2015), Aug 2015, Vienne, Austria. pp.258-269, 10.1007/978-3-319-27308-2\_22 . hal-01230288

**HAL Id: hal-01230288**

**<https://hal.science/hal-01230288>**

Submitted on 21 Jan 2017

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Communication Models Insights Meet Simulations

Pierre-François Dutot<sup>1</sup>, Millian Poquet<sup>1</sup>, and Denis Trystram<sup>1,2</sup>

<sup>1</sup> Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France  
CNRS, LIG, F-38000 Grenoble, France  
Inria

<sup>2</sup> Institut Universitaire de France, France

**Abstract.** It is well-known that taking into account communications while scheduling jobs in large scale parallel computing platforms is a crucial issue. In modern hierarchical platforms, communication times are highly different when occurring inside a cluster or between clusters. Thus, allocating the jobs taking into account locality constraints is a key factor for reaching good performances. However, several theoretical results prove that imposing such constraints reduces the solution space and thus, possibly degrades the performances. In practice, such constraints simplify implementations and most often lead to better results.

Our aim in this work is to bridge theoretical and practical intuitions, and check the differences between constrained and unconstrained schedules (namely with respect to locality and node contiguity) through simulations. We have developed a generic tool, using *SimGrid* as the base simulator, enabling interactions with external batch schedulers to evaluate their scheduling policies. The results confirm that insights gained through theoretical models are ill-suited to current architectures and should be reevaluated.

**Keywords:** FCFS with backfilling, simulations, heterogeneity

## 1 Introduction

Large scale high performance computing platforms are becoming increasingly more complex. Determining efficient allocation and scheduling strategies that can adapt to their evolutions is a strategic and difficult challenge. We are interested here in the problem of scheduling jobs in hierarchical and heterogeneous large scale platforms. The application developers submit their jobs in a centralized waiting queue. The job management system aims at determining a suitable allocation for the jobs, which all compete against each other for the available computing resources. The performances are measured by some objectives like the maximum completion times or the slowdown. The most common scheduling policy is First Come First Served (FCFS) which takes the jobs one after the other according to their arrival times with backfilling (BF), which is an improvement mechanism that allows to fill idle spaces with smaller jobs while keeping the original order of FCFS.

In practice the job execution times depend on their allocation (due to communication interferences and heterogeneity in both computation and communication), while theoretical models of parallel jobs are usually considering jobs as black boxes with a fixed execution time. Existing communications models do not fully reflect the network complexity and thus, simulations are required to take into account the impact of allocations.

Our goal within this work is to test existing heuristics dealing with allocation constraints, namely contiguity and locality. Contiguity forces jobs to be allotted on resources with a contiguous index (assuming that system administrators numbered their resources by proximity) while locality is a stronger constraint imposing some knowledge of the cluster structure to use allocations restricted to clusters whenever possible.

**Contributions** We show in this paper that insights gained while studying theoretical models are sometimes at odd with the practical results due to shortcomings in the models. Moreover, this was done through the development of a framework as a layer above SimGrid. This framework is generic enough to enable a large range of tests and is scheduled for a public release as an open source git project, as soon as the basic documentation is completed.

More precisely, we ran wide range simulations on FCFS/BF focusing on the impact of communications under several scenarios of locality constraints. The main result is to show that taking communications into account matters, but contrary to the intuition given by theoretical models, the most constrained scenarios are the best! In other words, the constrained policies allow greater gains in performances than the overhead due to the cost of the locality constraint. Moreover, this work opens the possibility to study new functionalities in SimGrid with our open access framework (like energy trade-off between speed scaling and shutdown policies or considering release dates).

## 2 Related Work

Modeling the modern High Performance Computing platforms is a constantly renewed challenge, as the technology evolves and quickly renders obsolete the models developed for the previous generation. While interesting and powerful, the synchronous PRAM model, delay model, LogP model and their variants (such as hierarchical delay, see [2] for a description of these models) are ill-suited to large scale parallelism on hierarchical and heterogeneous platforms.

More recent studies [12] are still refining these models to take into account contentions accurately while remaining tractable enough to provide a useful tool for algorithm design. Even with these models, all but the simplest problems are difficult and polynomial approximations algorithms have mixed results [11].

With millions of processing cores, even polynomial algorithms are impractical when every process and communication have to be individually scheduled. The model of parallel tasks simplifies this problem in a way, by bundling many threads and communications into single boxes, either rigid, rectangular or malleable (see [6], chapters 25 and 26). However, these models are again ill-adapted to

hierarchical and heterogeneous platforms, as the running time depends on more than simply the number of allotted resources. Furthermore, these models hardly match the reality when actual applications are used [3], as some of the basic underlying assumptions on the speed-up functions (such as concavity) are not often valid in practice.

With these limitations in mind, we decided to use simulations to really take into account the communications taking place within the jobs on large scale platforms. While writing a simple simulator is always possible, it appeared more interesting to use a detailed simulator to open our work to a larger set of platforms and job characteristics. Among the likely candidates, Simgrid [1] fulfills all our needs. In particular, the communications can be modeled either with a TCP-flow level model as used in this article or at the packet level for a fine grained simulation. While simulation is not always perfect [4], the results we present here are hopefully giving a better insight in the practical behavior of heuristics than the theoretical models.

A complementary approach to ours is to take into account the communications within the jobs themselves by migrating processes depending on their communication affinity [5]. This approach is rooted in the application, while we are positioning ourselves at the resource and job management system level.

Most available open-source and commercial job management systems use an heuristic approach inspired by FCFS with backfilling algorithms [9]. The job priority is determined according to the arrival times of the jobs. Then, BF (the backfilling mechanism) allows a job to run before another job with a highest priority only if it does not delay it. There exist several variants of this algorithm, like conservative backfilling [9] and EASY backfilling [7]. In the former, the job allocation is completely recomputed at each new event (job arrival or job completion) while in the second, the process is purely on-line avoiding costly recomputations. More sophisticated algorithms have been proposed that consider the routing schemes of the data (like topology aware backfilling introduced in [10]). In our article, we consider that the scheduler has a very limited knowledge of the platform, which is insufficient for topology-aware algorithms.

### 3 Problem Description

Our problem of interest in this paper is the problem of scheduling a set  $\mathcal{J}$  of independent and parallel jobs on a computing platform composed by a set  $M$  of computational resources (nodes or processors).

Each job  $j \in \mathcal{J}$  is characterized by a rigid number  $size_j$  of required resources, a walltime  $w_j$  (which bounds the execution time:  $j$  is killed after  $w_j$  seconds), a release date  $r_j$ , a computation row matrix  $c_j$  where each  $c_{j_k}$  represents the amount of computation on the  $k^{th}$  resource of job  $j$ , a square communication matrix  $C_j$  of size  $size_j \times size_j$  in which each element  $C_j[r, c]$  represents the amount of communication from the  $r^{th}$  resource to the  $c^{th}$  resource of job  $j$ .

Each resource  $m \in M$  has a computational power  $p_m$ . The resources are connected via a network. The network links have both a bandwidth and a la-

tency. Each resource  $m$  has a unique identification number  $id_m$  between 0 and  $card(M) - 1$ .

Since we are interested in the online version of this problem, the scheduler only knows that job  $j$  exists once it is released. Two jobs cannot be processed at the same time on the same computational resource. Each job must be computed exactly once. The jobs cannot be preempted. The scheduling algorithms are considered as oblivious about the jobs inner settings  $c_j$  and  $C_j$ . Furthermore, the algorithms know little about the platform *i.e.* they only know the number of computational resources and their identification numbers.

## 4 Simulation Framework

As stated in section 2, we turned to simulations to evaluate many batch scheduling algorithms to check if theoretical models match the practical experience. The added benefit over real experiments is that simulation enables reproducibility, and can easily be extended to test a very large number of parameters. The founding principle of our work is to use an existing platform simulation framework and to add a scheduling layer on the top of it. This approach allows us to take advantage of the simulation accuracy and the scalability of recognized software and allows separation of concerns since we are not simulation experts.

The survey [1] compares state-of-the-art simulators that could interest us. We chose to use SimGrid because it allows heterogeneity in both computational power and in network links latency and bandwidth, has a good TCP flow network model, can be used easily (thanks to a good documentation and a lot of examples), is fast, has little chance of becoming unmaintained (still actively developed after 10 years of existence) and comprises features that we may use in the future *e.g.* MPI applications simulation.

One of our main objectives was to be able to use already-developed scheduling algorithms without modifying their programming code a lot and to be able to simply create new algorithms. For this purpose, we base our simulation framework on two separate components: the simulating core and the scheduling module. These two components communicate via an event-based synchronous network protocol. When an event that may imply a scheduling decision occurs in the simulating core, the simulating core tells the scheduling module what happened and waits for its decision. The main events that will interest us in the scope of this paper are when jobs are released and when they complete their execution. In the scope of this paper, scheduling decisions are either to allocate some resources to some jobs or to do nothing.

The simulating core is fully written in C and is based on SimGrid, which is in charge of simulating what happens on the computational resources and on the network. All SimGrid platforms may be used by our simulating core as long as the user specifies which resources are used for the scheduling processes and which ones are used to compute jobs. Since SimGrid allows to create a wide range of simulators, it cannot be used directly to simulate a complex batch system. The purpose of the simulating core is thus to make the use of SimGrid easier

in conjunction with event-based batch scheduling algorithms. Our core basically handles the input jobs, asks the scheduling module for decisions, ensures that jobs are simulated by SimGrid according to the topology and produce some output traces and statistics. Since this is a work in progress and not the main point of this paper, it will not be further described in the present paper. It will be published and put at the disposal of the community once mature enough.

The scheduling module can be developed in any programming language that allows network programming via *Unix Domain Sockets*. This component can be seen as an iterative process which consists in waiting an event from the simulating core, updating some data structures then making a scheduling decision. Therefore, existing algorithms which are based on events like job releases or completions can easily be plugged with the simulating core.

## 5 Evaluation

### 5.1 Platform and Jobs Description

Since we would like to know how the algorithms presented in [8] behave within realistic simulations, we use the same kind of platforms that the paper described. Our platforms include sets of closely located computational resources called *clusters*. Each cluster  $c$  is a tree formed by a switch  $s_c$  and a set of computational resources which are all directly connected to it. The cluster switch  $s_c$  has an internal bandwidth  $bw_{s_c}$  and an internal latency  $lat_{s_c}$ . All resources within the cluster  $c$  have the same computational power  $cp_c$ , the same bandwidth  $bw_c$  and latency  $lat_c$  between the resource and  $s_c$ . The clusters are connected together via a unique switch  $b$  whose shared bandwidth is  $bw_b$  and whose latency is  $lat_b$ . The implementation of the algorithms presented in [8] constrain all the clusters to have the same size. We chose to keep the parameters they used which are 8 clusters of 16 computational resources each, leading to a total of 128 resources per platform.

In the following experiments, each run instance consists of a platform, a workload and a scheduling algorithm. Every generated workload consists of 300 jobs extracted from the cleaned trace (in the SWF format) of the CEA-Curie supercomputer. Our job selection criteria were to remove jobs that cannot fit entirely in one cluster and, in order to obtain interesting workloads, to ensure the resulting schedule makespan is not fixed by the longest job. Tiny jobs fit easily in the backfilling and very big ones are usually in specific queues, we then decided to only keep jobs whose execution time  $t_j$  is between two bounds  $l_t \leq t_j \leq u_t$ . Typical values for the bounds are  $l_t = 1 \text{ hour}$  and  $u_t = 1 \text{ day}$ . The method used to select the jobs is to first remove every job that does not fit our criteria then to randomly pick 300 jobs depending on a given random seed.

Since the trace only contains the execution time, without any detail of actual computations or communications patterns, we chose to use basic homogeneous patterns and to create the amounts from the real execution time of the jobs. Let  $ret_j$  be the real execution time of job  $j$  in the trace file. Let  $rw_j$  be the user-given

walltime of job  $j$ . Let  $F_{comp}$ ,  $F_{comm}$  and  $F_w$  be respectively the computation factor, the communication factor and the walltime factor. For each job  $j$ , the computation row matrix  $c_j$  is computed via  $c_j = R_{size_j}^1 \times ret_j \times F_{comp}$  where  $R_{size_j}^1$  is a row matrix of  $size_j$  columns of 1. For each job  $j$ , the communication square matrix  $C_j$  is obtained with the following formula  $C_j = S_{size_j}^1 \times ret_j \times F_{comm}$  where  $S_{size_j}^1$  is a square matrix of size  $size_j \times size_j$  of 1. For example,  $R_3^1 = (1 \ 1 \ 1)$  and  $S_2^1 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ . For each job  $j$ ,  $size_j$  is read from the trace and  $w_j$  is chosen big enough to ensure the job won't be killed via the following formula  $w_j = \max(rw_j, ret_j \times F_w)$ . With small walltimes, the jobs allocations would not matter since jobs would not be allowed to complete and would simply be killed after the same amount of time in any placement. Finally, the release date of each job  $j$  is set to 0 to remain in the same experimental setting as in [8], which will allow us to analyse the difference between our results and the previous ones.

## 5.2 Competing Heuristics

The scheduling algorithms we will compare are variants of the well-known conservative backfilling algorithm [9] targeting the minimization of *makespan* (completion time of the last running job). This algorithm maintains two data structures. The first one is a list of queued jobs and the times at which they are guaranteed to start execution. The other is a profile which stores the expected future processor usage. This profile is usually a list of consecutive time slices which store the resources status for each period. When a new job  $j_n$  is submitted, the profile is traversed in order to find a *hole* in the resource usage in which  $j_n$  may fit (depending on the job width  $w_{j_n}$  and height  $size_{j_n}$ ). Let us suppose the profile traversal is done by ascending date and that this procedure will return different holes in which  $j_n$  may fit. When a fitting hole is found, it may either be accepted or rejected. If accepted, the scheduling algorithm must select which resources within the hole will be allocated to  $j_n$ . Otherwise, if the hole is rejected, the profile traversal continues and future fitting holes will be found until one is accepted. The algorithms that will be studied in the present paper differ in their last phase, which consists in accepting or rejecting the current hole and selecting which resources are allocated to  $j_n$  in case of acceptance. A detailed description of these variants and their pseudo-code is given in [8]. In the remaining of this section, let  $j_n$  be the newly submitted job,  $H \subseteq M$  the set of available resources in the current hole and  $S \subseteq H$  the selection of resources within  $H$  on which the job  $j_n$  will be executed.

The **basic** variant always accepts the first fitting hole and selects the first resources *i.e.*  $S \subseteq H$  such that  $card(S) = size_{j_n}$  and  $\sum_{s \in S} id_s$  is minimal. The **best effort contiguous** always accepts the first fitting hole and selects a continuous block of resources if possible. In this context, the contiguity of the set of resources  $S$  means that there exist resources with contiguous indexes. If there is no contiguous set of resources of size  $size_{j_n}$  in  $H$ , this variant selects the first resources as the basic variant would do. The **best effort local** variant always

accepts the first fitting hole and selects a local set of resources if possible. Otherwise, it returns the first resources as the basic variant would do. In the context of this paper,  $S$  is said to be a local set of resources if all the resources in  $S$  are located in the same cluster. The **contiguous** variant forces the contiguity constraint on  $S$ . Consequently, this variant may reject the first fitting holes if they do not match the constraint. The **local** variant forces the locality constraint on  $S$ . Consequently, just as in the case of the contiguous variant, the local variant may reject the first fitting holes if they do not match the locality constraint. Thanks to the authors of the article [8], we were able to directly use their algorithms implementation in conjunction with our simulator which avoided us to reimplement them.

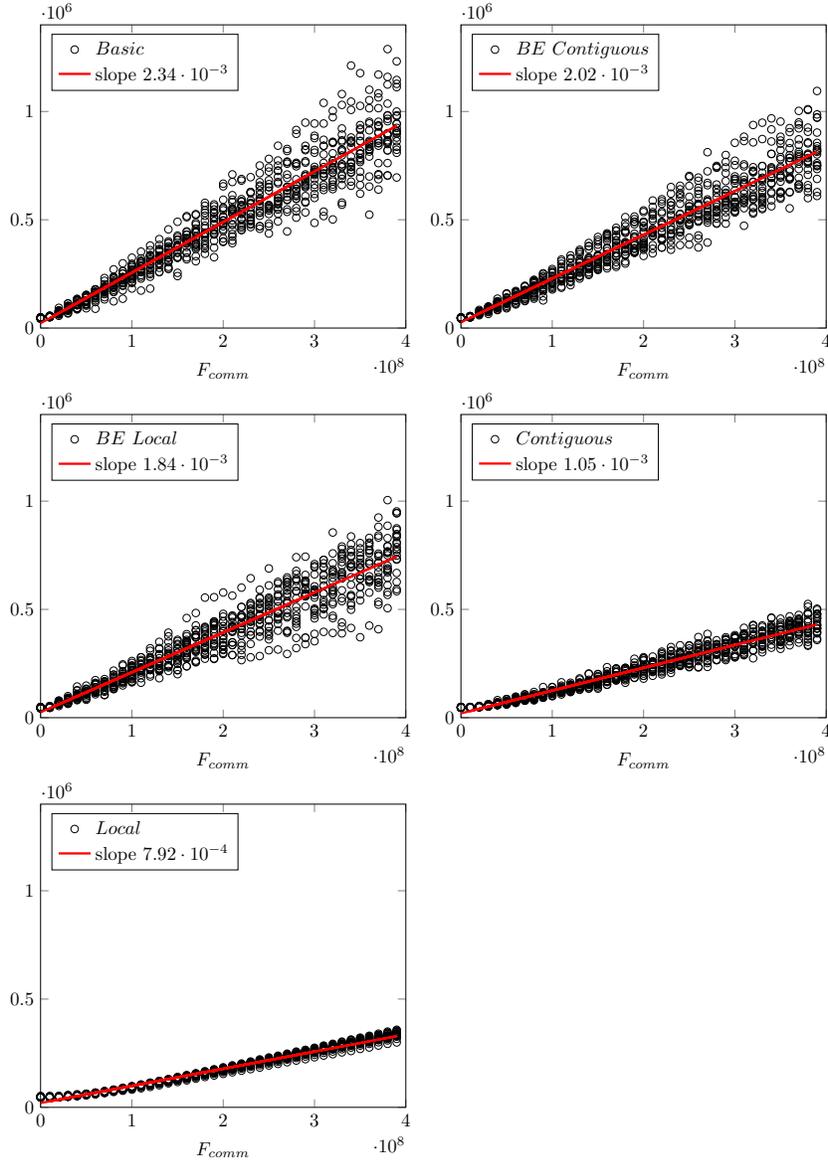
### 5.3 Homogeneous Platform Experiment

The goal of the first experiment was to compare the behaviour of the different scheduling algorithms when the job amount of communication is increased on the same homogeneous platform. The jobs of this experiment were generated with the following parameters: 20 random seeds were used (0 to 19), leading to 20 different base workloads. We picked  $F_{comp} = 10^6$ ,  $F_w = 10^3$ , and 40 different values for the  $F_{comm}$  parameter have been used which correspond to a linear variation starting from 0 with steps of  $10^7$ . The length bounds to pick the jobs were  $l_t = 1 \text{ hour}$  and  $u_t = 4 \text{ hours}$  in order to obtain jobs whose simulated execution time is interesting (*i.e.* the resulting schedule makespan is not only fixed by the biggest job) across the used values of  $F_{comm}$ . All the clusters of the platform used in this experiment are the same and defined by the following parameters.  $bw_{s_c} = 1.25 \cdot 10^9$ ,  $lat_{s_c} = 0$ ,  $bw_c = 1.25 \cdot 10^6$ ,  $lat_c = 24 \cdot 10^{-9}$ . The platform main switch parameters are  $bw_b = 1.25 \cdot 10^9$  and  $lat_b = 24 \cdot 10^{-9}$ . This platform is derived from the existing Grid'5000 Griffon cluster whose platform description was available in the SimGrid examples. The combination of these parameters created 4000 instances (800 per scheduling algorithm variant).

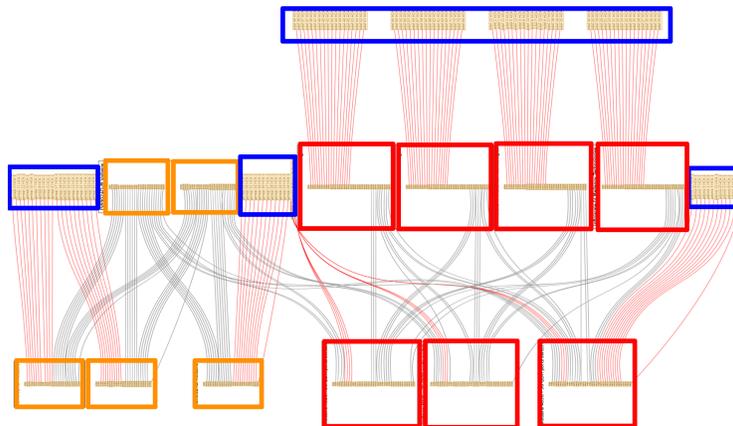
Figure 1 shows the makespan  $C_{max}$  of the resulting schedule of every run instance of the first experiment. Additionally, a linear trendline has been computed for a better comparison of the heuristics. The basic algorithm (as defined in the previous section) depicted in the top left is completely without constraints, and has the worst performance of all competing heuristics. Imposing contiguity without any knowledge of the underlying structure gives better performances than basic, while knowledge of locality further improves the results. More surprisingly, the strict heuristics are outperforming the more relaxed heuristics, even though strict heuristics delay some jobs if the constraints cannot be matched. Furthermore, the makespan induced by the forced constraints are much more stable than their best-effort counterparts.

### 5.4 Heterogeneous Platform Experiments

The goal of the following two experiments is the same as the homogeneous one: seeing how the different scheduling algorithms behave when the amount of com-



**Fig. 1.** The makespan  $C_{max}$  of every run instance in function of the communication factor  $F_{comm}$  for the homogeneous platforms experiment. To each figure corresponds a scheduling algorithm. Each point corresponds to a schedule (800 per scheduling algorithm).



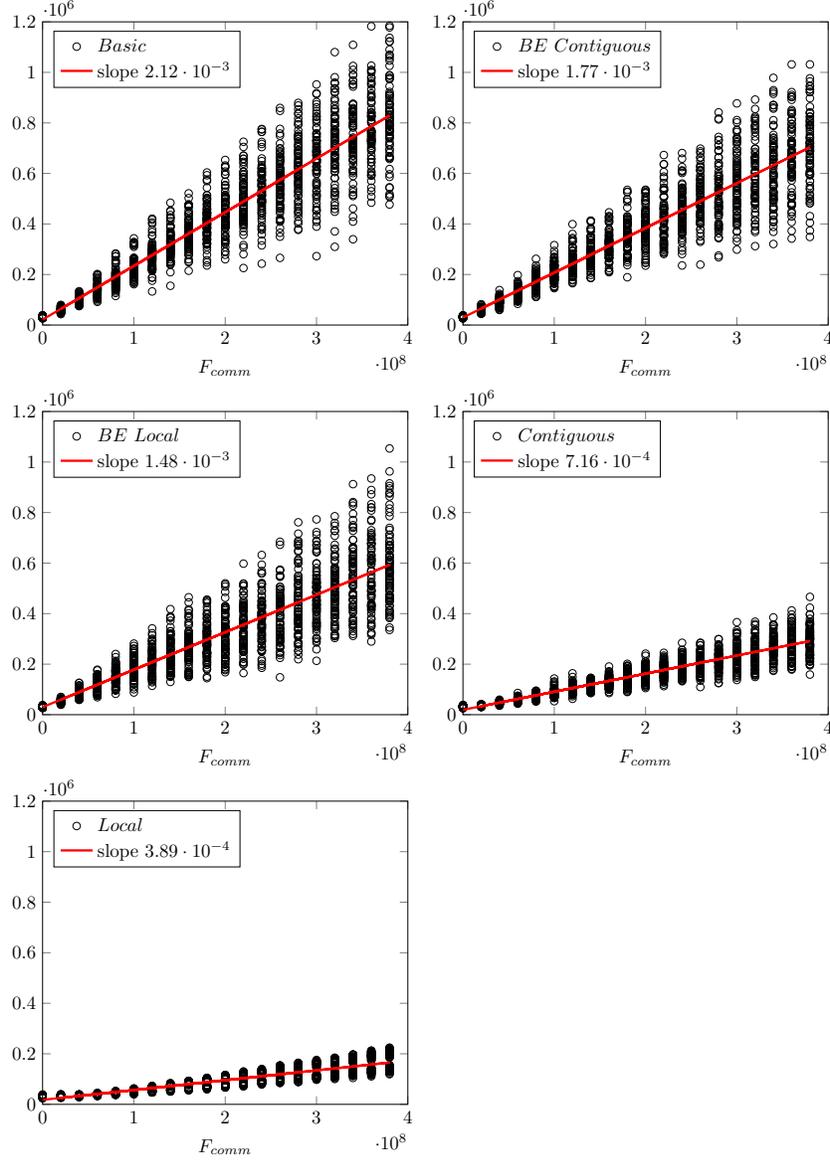
**Fig. 2.** Grid'5000 cluster architecture in Grenoble.

munication within jobs is increased. However, these experiments focus on many heterogeneous platforms instead of one homogeneous platform, to more closely reflect the existing clusters in our computing centers. For example, Figure 2 gives an idea of the layout of the Grid'5000 cluster in Grenoble<sup>3</sup>. The red rectangles are 40 Gb/s Infiniband switches, orange rectangles are 20 Gb/s Infiniband switches, while the blue rectangles are three different families of computing nodes.

In order to remain realistic in the kind of platform heterogeneity to simulate, we analyzed the network of several Grid'5000 sites and ran a linear algebra benchmarking tool on many machines to have an idea of how much the node computational power may vary within one site. Our results on the Rennes and Grenoble site showed that the network bandwidth might vary between 1 and 4 and that the node computational power may vary between 1 and 3. More precisely, with our benchmark the computational power in the Rennes site were 1, 2.02 and 2.94 times more powerful than the lowest one. On Grenoble we obtained computational powers of 1.24, 1.61 and 1.72 times the lowest one. We then decided to create a set of lowly heterogeneous platforms and a set of highly heterogeneous platforms and see how the different scheduling variants behave on such platforms.

The two heterogeneous experiments use six clusters whose parameters can be found in Table 1. The first heterogeneous experiment uses four platforms formed by 3 clusters  $c_1$ , 3 clusters  $c_2$  and 2 clusters  $c_3$ . The four platforms differ by the ordering in which the clusters are in the platform. The used orderings are by ascending computational power  $o_1 = (c_1, c_1, c_1, c_2, c_2, c_2, c_3, c_3)$ , by descending computational power  $o_2 = (c_3, c_3, c_3, c_2, c_2, c_2, c_1, c_1)$  and other orderings  $o_3 = (c_1, c_2, c_2, c_3, c_3, c_2, c_1, c_1)$  and  $o_4 = (c_3, c_1, c_2, c_3, c_1, c_2, c_1, c_2)$ . The workloads of this experiment have been generated with the following parameters: 10

<sup>3</sup> For more details, a larger version of the figure is available at:  
<https://www.grid5000.fr/mediawiki/index.php/Grenoble:Network>



**Fig. 3.** The makespan  $C_{max}$  of every run instance in function of the communication factor  $F_{comm}$  for the heterogeneous platforms experiment. To each figure corresponds a scheduling algorithm. Each point corresponds to a schedule (1600 per scheduling algorithm).

**Table 1.** The parameters of the clusters used in heterogeneous experiments. These values are multiplication factors of our base cluster  $b$  whose values are  $bw_{s_b} = 10 \text{ Gbits} \cdot s^{-1}$ ,  $lat_{s_b} = 0 \text{ s}$ ,  $cp_b = 286.097 \cdot 10^3 \text{ flop} \cdot s^{-1}$ ,  $bw_b = 10 \text{ Gbits} \cdot s^{-1}$ ,  $lat_b = 24 \cdot 10^{-9} \text{ s}$ .

$c$	$bw_{s_c}$	$lat_{s_c}$	$cp_c$	$bw_c$	$lat_c$
$c_1$	2	0	1	1	1
$c_2$	4	0	2.02	1	1
$c_3$	1	0	2.94	1	1

$c$	$bw_{s_c}$	$lat_{s_c}$	$cp_c$	$bw_c$	$lat_c$
$c_4$	1	0	1.24	1	1
$c_5$	2	0	1.61	1	1
$c_6$	1	0	1.72	1	1

random seeds have been used (0 to 9). We used  $F_{comp} = 10^6$ ,  $F_w = 10^3$ , and 20 different values for the  $F_{comm}$  parameter have been used which correspond to a linear variation starting from 0 with steps of  $2 \cdot 10^7$ . The length bounds to pick the jobs were  $l_t = 1 \text{ hour}$  and  $u_t = 4 \text{ hours}$ . The second heterogeneous experiment is exactly the same as the first but its platforms use clusters  $c_4$ ,  $c_5$ ,  $c_6$  instead respectively of clusters  $c_1$ ,  $c_2$  and  $c_3$ . We call the first experiment highly heterogeneous because the resource computational power varies from 1 to 3 and the network bandwidth from 1 to 4 within it. We call the second experiment lowly heterogeneous because these amounts doesn't vary as much as in the first experiment. Each experiment consists of 4000 run instances (800 per scheduling algorithm variant).

Figure 3 shows the makespan of the different scheduling algorithm variants when the amount of communication is increased in the heterogeneous experiments. These graphs do not differ greatly from the homogeneous case: for the makespan, the forced constraint variants scale better and are more stable than their best-effort counterparts when the amount of communication within jobs is increased. Furthermore, we did not notice any impact of the cluster ordering within one platform on the resulting schedules makespan. We did not notice a great difference between the slightly heterogeneous platforms and the highly heterogeneous ones neither, that is why the results of the two experiments have been plotted together. The most notable result is that in a heterogeneous setting, the locality knowledge is much more important as the gap between the basic heuristic and the locality aware is greatly increased.

## 6 Conclusion

The purpose of this work was to show through simulations if theoretical models are giving pertinent insight on job scheduling on large scale hierarchical and heterogeneous platforms. The main hypothesis we tested was that enforcing contiguity or locality would not degrade the performance. The results clearly show that the constraints are beneficial to the schedules, by reducing the communication times. More broadly, this shows that models where internal communications are hidden within parallel tasks are very ill-suited to current architectures, and should be reevaluated. The tool we developed is very general and relies on a powerful simulator, which will in the near future enable studies on different network topologies, and assess the impact of scheduling policies on a variety of objectives.

## Acknowledgments.

The work is partially supported by the ANR project MOEBUS. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see <https://www.grid5000.fr>).

## References

1. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing* 74(10), 2899–2917 (Jun 2014)
2. Giroudeau, R., König, J.C.: Scheduling with Communication Delay. In: *Multiprocessor Scheduling: Theory and Applications*, pp. 1–26. ARS Publishing (Dec 2007)
3. Hunold, S.: One step towards bridging the gap between theory and practice in moldable task scheduling with precedence constraints. *Concurrency and Computation: Practice and Experience* 27(4), 1010–1026 (2015)
4. Hunold, S., Casanova, H., Suter, F.: From Simulation to Experiment: A Case Study on Multiprocessor Task Scheduling. In: *Proceedings of the 13th Workshop on Advances on Parallel and Distributed Processing Symposium (APDCM)* (2011)
5. Jeannot, E., Meneses, E., Mercier, G., Tessier, F., Zheng, G.: Communication and topology-aware load balancing in charm++ with treematch. In: *IEEE Cluster 2013*. IEEE, Indianapolis, United States (Sep 2013)
6. Leung, J.: *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman & Hall/CRC Computer and Information Science Series, CRC Press (2004)
7. Lifka, D.A.: The anl/ibm sp scheduling system. In: *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. pp. 295–303. IPPS '95, Springer-Verlag, London, UK, UK (1995)
8. Lucarelli, G., Mendonca, F., Trystram, D., Wagner, F.: Contiguity and locality in backfilling scheduling. In: *Cluster, Cloud and Grid Computing (CCGrid)*, 2015 15th IEEE/ACM International Symposium on (May 2015)
9. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *Parallel and Distributed Systems, IEEE Transactions on* 12(6), 529–543 (2001)
10. Pascual, J.A., Navaridas, J., Miguel-Alonso, J.: Effects of topology-aware allocation policies on scheduling performance. In: *Job Scheduling Strategies for Parallel Processing, 14th International Workshop, JSSPP 2009, Rome, Italy, May 29, 2009. Revised Papers*. pp. 138–156 (2009)
11. Sinnen, O.: *Task Scheduling for Parallel Systems*. Wiley Series on Parallel and Distributed Computing, Wiley (2007)
12. Sinnen, O., Sousa, L.A., Eika Sandnes, F.: Toward a realistic task scheduling model. *IEEE Trans. Parallel Distrib. Syst.* 17(3), 263–275 (Mar 2006)