A First Step to the Evaluation of SimGrid in the Context of a Real Application

Abdou Guermouche
LaBRI and Univ. Bordeaux1
Bordeaux, France
Email: Abdou.Guermouche@labri.fr

Hélène Renard
I3S and Univ. Nice/Sophia-Antipolis
Sophia-Antipolis, France
Email: Helene.Renard@polytech.unice.fr

***Abstract*—Simulation is a "simple" method to experimentally evaluate the behavior of algorithms designed for parallel and distributed platforms. Moreover, the reliability of the evaluation strongly depends on the models used inside the simulator. This paper is devoted to the study and the evaluation of the behavior of the SimGrid simulator through a comparison between a simulated and a real execution of the same target application (i.e. heat propagation). Our target platforms are heterogeneous and their characteristics may dynamically vary during the execution. The obtained results (on a set of various platforms) show that the behavior observed when using the simulated platform is very close to the one obtained on the real one.**

***Keywords*-Data redistribution, simulator, SimGrid, real execution.**

I. Introduction

Distributed computing has become mainstream and is used today in a wide variety of domains for a large range of applications. It is an extremely active research area covering distributed algorithms, distributed resource management, and resource and application scheduling. One major issue in distributed computing is the ability of the designer to evaluate her/his algorithm/software either by some theoretical methods or by experimental approaches. For the latter case, one key issue is to have a reproducible behavior of the underlying resources to be able to do a significant evaluation of the studied problem. Thus, simulation was introduced to try to avoid these difficulties related to real-life experiments while illustrating the behavior of the algorithms. It has been extensively used in many areas like microprocessor and network protocol design. Since few years, many softwares for the simulation of distributed applications running on top distributed systems have appeared like GridSim [1], Optorsim [2], PlanetSim [3], PeerSim [4] or SimGrid [5].

Our main interest is the study of large scale distributed computing experiments. From the simulation point of view, they are two types of simulators : packet-level simulators (like ns-2, ...) and application-level simulators. Packet-level simulators track the movement of each packet in data flows which makes them less adapted to application simulations where effects (i.e., communication times) are more important than causes (e.g, packets movements). The class of application-level simulators is divided into two main parts : simulators for P2P systems, and simulators for "grid" environments. The P2P simulation tools are by nature extremely scalable (in the sense that they are able to simulate thousands of peers without significantly increasing execution time) and do usually not model the CPU since the classical metric in the P2P area is message count. Some simulators like PlanetSim assume constant communication duration for each node while other simulators like PeerSim completely ignore communications cost. This makes them not adapted to our context. Finally, grid simulation tools try to find a tradeoff between execution speed and accuracy of the simulation. Their main metric is the application runtime. We can distinguish 4 simulators ChicSim [6], OptorSim, GridSim and SimGrid. The first 2 of them designed to study data-replication on grids, are not active since 2004, and their network model is either not scalable (ChicSim) or flawed (OptorSim). GridSim was initially designed for grid economy and then been extended to model large scale applications. It provides a fine-grained model of the network with latency, seek time and max transfer rate and mimics the flow fragmentation, but does not take TCP flow management mechanisms into account. This approach makes it potentially not scalable when the size of the simulated platform increases. Finally, only the SimGrid network model has been extensively validated by its authors (see [7], [8]). For all these reasons and regarding the size of our target platforms, we have chosen to compare our real-life execution to SimGrid.

In this paper, we are interested in evaluating the behavior of SimGrid against a real-life execution in the context of a complex application. This will illustrate the fact that SimGrid is able to simulate a behavior very close to the one observed in real-life experiments. Our target application (i.e. *heat propagation*) requires both load-balancing and data redistribution techniques to run efficiently on a heterogeneous platform where processors are organized through a virtual ring. Because of variations in the resources performance (CPU speed, communication bandwidth) or in the system/application requirements (completed tasks, new tasks, migrated tasks, etc.), data have to be redistributed between participating processors so that the current (estimated) load is better balanced.

To be able to compare the simulated execution with a

real-life one, we have to use some emulation tools to be able to control the characteristics of the platform. This is particularly important to illustrate the quality of the models used by `SimGrid` to simulate computations and network communications. Thus we used the `wrekavoc` [9] tool to dynamically change the characteristics of the platform. We then study the response of the application/algorithms to these variations in both simulated and real-life environments. Our goal is to illustrate through this case-study how the simulator behaves in comparison to a real-life execution.

We present in Section II our target application together with the data redistribution algorithm used. After that, in Section III, we show how we implemented our application and how we dealt with the modification of the characteristics of the platform. Then, we give in Section IV experimental results where we present a comparison of simulated executions and their corresponding real-life executions. We show the interest of the `SimGrid` simulator. Some related works are then described in Section V. Finally we give some concluding remarks and future work in Section VI.

II. Framework

We present in this section our target application together with the redistribution algorithm we use when imbalance occurs (mainly because of the modification of the characteristics of the platform). Our application : *heat propagation* follows an iterative scheme where communications and computations take place. The algorithm iteratively operates on a wide array of rectangular sample data. This data matrix is split in vertical slices that are allocated to resources (see Figure 2 in Subsection II-B where dashed arrows represent communication). This geometric constraint recommends that processors must be organized as a virtual ring (there are cyclic boundary conditions on our grid). Each processor only communicates twice, once with his predecessor (virtual) in the ring and the other time with his successor. The ring layout will also be used for redistributing data. We do not discuss in this paper the load-balancing algorithm needed to build the virtual ring. Finally, it is important to note that even if the *heat propagation* is a simple iterative application, the corresponding model (i.e. the ring organization of the processors) can be used for more complex ones (see Section II-B for more details). Finally, note that we assume for the rest of the paper that communications follow a 1-port model.

A. Data redistribution algorithms

Let us describe the redistribution algorithm we will use. We assume for the rest of the paper that the communications follow a 1-port model. We present only the case where we consider a bidirectional heterogeneous ring. The case of homogeneous rings is out of scope of this paper and has already been studied in [10]. Moreover, the algorithm presented in this section has already been fully described in [11], we just remind the main idea behind it for the sake of clarity.

Throughout this section, we suppose that we have a *light* redistribution: we assume that the number of data items sent by any processor throughout the redistribution algorithm is less than or equal to its original load. There are two reasons for a processor P_i to send data: (i) because it is overloaded ($\delta_i > 0$); (ii) because it has to forward some data to another processor located further in the ring. If P_i initially holds at least as many data items as it will send during the whole execution, then P_i can send at once all these data items. Otherwise, in the general case, some processors may wait to have received data items from a neighbor before being able to forward them to another neighbor.

Let $\mathcal{S}$ be a solution, and denote by $\mathcal{S}_{i,i+1}$ the number of data items that processor P_i sends to processor P_{i+1}. Similarly, $\mathcal{S}_{i,i-1}$ is the number of data items that P_i sends to processor P_{i-1}. In order to ease the writing of the equations, we impose in the first two equations of System 1 that $\mathcal{S}_{i,i+1}$ and $\mathcal{S}_{i,i-1}$ are non-negative for all i, which imposes to use other variables $\mathcal{S}_{i+1,i}$ and $\mathcal{S}_{i-1,i}$ for the symmetric communications. The third equation states that after the redistribution, there is no more imbalance. We denote by τ the execution time of the redistribution. For any processor P_i, due to the one-port constraints, τ must be greater than the time spent by P_i to send data items (fourth equation) or spent by P_i to receive data items (fifth equation). Our aim is to minimize τ, hence the system:

$$\text{MINIMIZE } \tau \text{ SUBJECT TO}$$
$$\begin{cases} \mathcal{S}_{i,i+1} \geq 0 & 1 \leq i \leq n \\ \mathcal{S}_{i,i-1} \geq 0 & 1 \leq i \leq n \\ \mathcal{S}_{i,i+1} + \mathcal{S}_{i,i-1} - \mathcal{S}_{i+1,i} - \mathcal{S}_{i-1,i} = \delta_i & 1 \leq i \leq n \\ \mathcal{S}_{i,i+1} c_{i,i+1} + \mathcal{S}_{i,i-1} c_{i,i-1} \leq \tau & 1 \leq i \leq n \\ \mathcal{S}_{i+1,i} c_{i+1,i} + \mathcal{S}_{i-1,i} c_{i-1,i} \leq \tau & 1 \leq i \leq n \end{cases} \quad (1)$$

We use System 1 to find an optimal solution to the problem. If, in this optimal solution, for any processor P_i, the total number of data items sent is less than or equal to the initial load ($\mathcal{S}_{i,i+1} + \mathcal{S}_{i,i-1} \leq L_i$), we are under the "light redistribution" hypothesis and we can use the solution of System 1 safely. A proof of the correctness of what has been stated in this section is given in [11].

B. Heat propagation

To study the behavior of our algorithms, we focus on a simple application: the heat propagation. Heat propagation is one application among many others which fits into our model. Indeed, the results presented in this paper is valid for more complex applications. To be more precise, this study applies to applications having a pattern of communication equivalent to the one of a ring (eddy simulation, simulation of turbulence in aerodynamics, etc.).

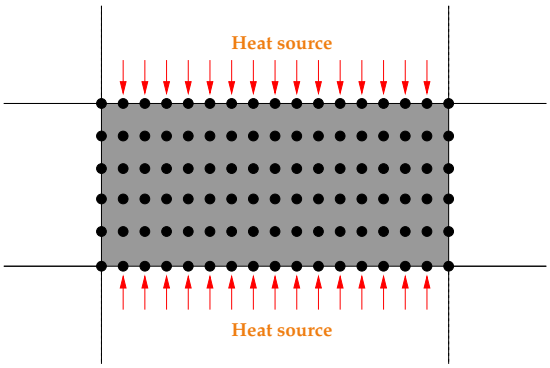


Figure 1. Example of heat propagation.

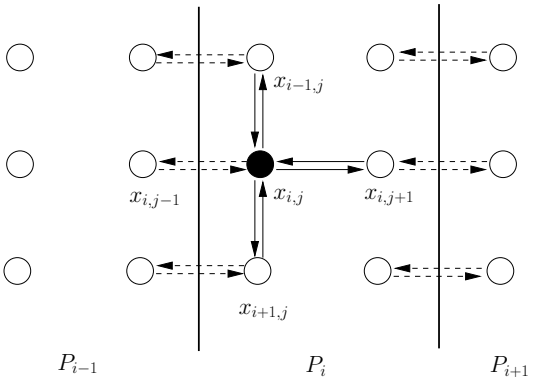


Figure 2. Communication scheme.

Let us imagine a metal plate (see Figure 1) on which is applied a heat source to the edges. Heat spread within the plate. The temperature at the edges is kept constant, the heat distribution in the plate tends to a steady state. This problem can be solved firstly using finite difference methods to discretize the two dimensional plate into a two dimensional grid. Then, an iterative method, like Jacobi is used to solve the discretized equation. For explanatory purposes, we consider the communication requirements associated with a single numerical computation, a finite difference method of Jacobi. In this class of numerical methods, a multidimensional grid is updated several times by replacing the value at each point with an operation on the values of a fixed number of neighboring points (see Figure 2).

This simple application fully corresponds to our model. Indeed, since the communications are in a local neighboring, if we give to each process an vertical bloc of columns in the discretized grid, the communication between processors follow our ring assumption. Indeed, each processor P_i communicate only with its neighbors P_{i-1} and P_{i+1}. This is illustrated in Figure 2 where communication take place on the boundaries of the block assigned to processor P_i (dashed arrows). Finally, note that only $x_{i,j}$ has all its operations illustrated on the figure for the sake of clarity.

Our algorithm applies perfectly to the discretized two-dimensional grid where each processor P_i knows and needs to exchange data only with its neighbors P_{i-1} and P_{i+1}.

III. Real-life and simulation

The experiments presented in the next section have the objective to compare the behavior of the same application running on two different platforms : a simulated one using `SimGrid`, and a real one. `SimGrid` is a toolkit that provides core functionalities for the simulation of distributed applications in heterogeneous distributed environments. The main interest of such an approach is to confront the studied algorithm to a completely tunable platform. In addition, the fact that experiences can be completely reproducible represents a major advantage. This corresponds to a high-level implementation of our application without all the technical details needed by a real-life implementation. In the other hand, we have implemented the corresponding real-life application. The corresponding code is written in the C language using standard UNIX sockets for communication and the XDR layer for interoperable communications between heterogeneous machines. It is important to note that MPI hasn't been selected for the communication layer because we want our application to be able to run on top of a platform composed by a set of distant clusters containing heterogeneous processors.

Our iterative application follows the scheme described in Algorithm 1. In the following paragraph process is used to refer to a process running "alone" on a given node (i.e. it is not possible to have more than one process running on the node). The processes taking part to the execution are organized through a virtual ring. The *master* process (See Figure 3) does not participate to the computations, it is only in charge of : i) gathering state information from its workers when needed, ii) calling the redistribution algorithm described in Section II-A. Worker processors are the ones in charge of doing the computation and exchanging data at the end of each iteration. In addition, they periodically have to determine their characteristics (i.e. speed of the processor and network links latency and bandwidth) and to redistribute data according to the result of the algorithm applied by the master.

This organization of the processes is used in both the simulated and real-life context. Note that apart from the processes described above, there is a "monitor" which supervises the execution of the application. Its main role is to periodically modify the characteristics of the platform on the fly. It consists in an external process in the context of the real life implementation (see Figure 3). The "monitor" has not been simulated in the `SimGrid` version since `SimGrid` offers this functionality.

The monitor uses the `wrekavoc` [9] tool to control the

```
1:  while end not detected do
2:    if I am master then
3:      if modulo (current iteration number, interval) == 0 then
4:        wait for state information from all workers
5:        use the algorithm presented in Section II-A to build
          redistribution information
6:        send redistribution information to each worker
7:    else
8:      exchange data with neighbors.
9:      update local data and process current iteration.
10:     if modulo (current iteration number, interval) == 0 then
11:       perform benchmarks to get the new characteristics of
          my processor and my network links.
12:       send my new state to master
13:       wait for redistribution information from master
14:       apply the redistribution algorithm according to the
          decision of master
```

Algorithm 1: Iterative scheme.

heterogeneity of the underlying platform. Indeed, this toolkit allows the user to degrade (to only cite them) CPU and network capabilities of each node of the platform remotely. The idea here, is to modify periodically some characteristics of randomly chosen resources in the platform and to compare the behavior observed in both platforms (simulated and real-life). Moreover, during the real-life execution once a variation has to be applied, a corresponding simulated platform is generated (this platform will be used during the simulated execution). The variation concerns a random number of resources (either processors or network links) and assigns to each selected resource a random percentage of its full capacity (the percentage is picked from the interval $[40; 100]$). The `wrekavoc` tool is then used to apply the modification. Thus, the monitor follows the scheme described in Algorithm 2. Finally, note that the modification of the characteristics of the platform in the simulated context is done by giving to `SimGrid` a platform with modified characteristics. This modified platform is generated by the monitor each time the properties of the platform have to change.

```
1: while end not detected do
2:   if I need to modify the platform then
3:     Pick a random number of resources to degrade
4:     for each selected resource do
5:       Pick a random degradation factor from the interval
         [40; 100]
6:     Apply the modification of the characteristics of the plat-
       form using wrekavoc
7:     Generate the corresponding simulated platform
```

Algorithm 2: Monitor scheme.

Note that to be able to compare the behavior of all the phases of the execution in the two contexts (real-life and simulated one), we implemented in the simulated version the phase were a process discovers the characteristics of the resources it uses (see line 11 in Algorithm 1). This could be avoided since `SimGrid` is able to give to each simulated process all the information it needs about the underlying platform. Thus, even if in the simulated version, we provide to `SimGrid` the modified platform, we let our application (either in the simulated or in the real-life context) discover the modification of the characteristics of the platform. To be more precise, each time we want to redistribute data, we gather information concerning the state of each host taking part to the execution (its processor speed, its network link bandwidth and latency, etc ...). Everything in this case is done dynamically by measuring the characteristics of the platform (see line 11 in Algorithm 1).

The measurement of the characteristics of the platform is done before the use of the redistribution algorithm. Each worker process performs a small benchmark (based on the same computation kernel as regular iterations) operation to compute the speed of the underlying processor. Concerning network link characteristics, they are built thanks to the use of the `iperf`[1] tool. This tool allows us to get both latency and bandwidth between each consecutive nodes in the logical ring of processes. The measurements are done in a 1-port like model (i.e. each process does only one measurement at a time). The detailed scheme for this phase is described in Algorithm 3. From a practical point of view, each communication consists in an `iperf` measurement of 4 seconds which results on a bandwidth and latency. The choice of this kind of measurements is motivated by the fact that our target platforms has very heterogeneous network links. Finally, note that we could have used frameworks like NWS [12] for gathering information about the state of the resources of our platform but we are not interested in the forecasting mechanisms that make this kind of frameworks very useful.

```
1: Perform a small benchmark for getting the current speed of
   the underlying processor
2: if logical id in the ring is even then
3:   Receive from left neighbor
4:   Receive from right neighbor
5:   Send to left neighbor
6:   Send to right neighbor
7: else
8:   Send to right neighbor
9:   Send to left neighbor
10:  Receive from right neighbor
11:  Receive from left neighbor
```

Algorithm 3: Benchmarking scheme.

IV. Experimental results

In this paper, we base our platform model on a real-world multi-cluster platform, `Grid'5000` [2] [13]. The goal of Grid'5000 is to build a highly reconfigurable, controllable and monitorable experimental platform to allow

[1] `http://www.noc.ucf.edu/Tools/Iperf`
[2] `http://www.grid5000.fr/`

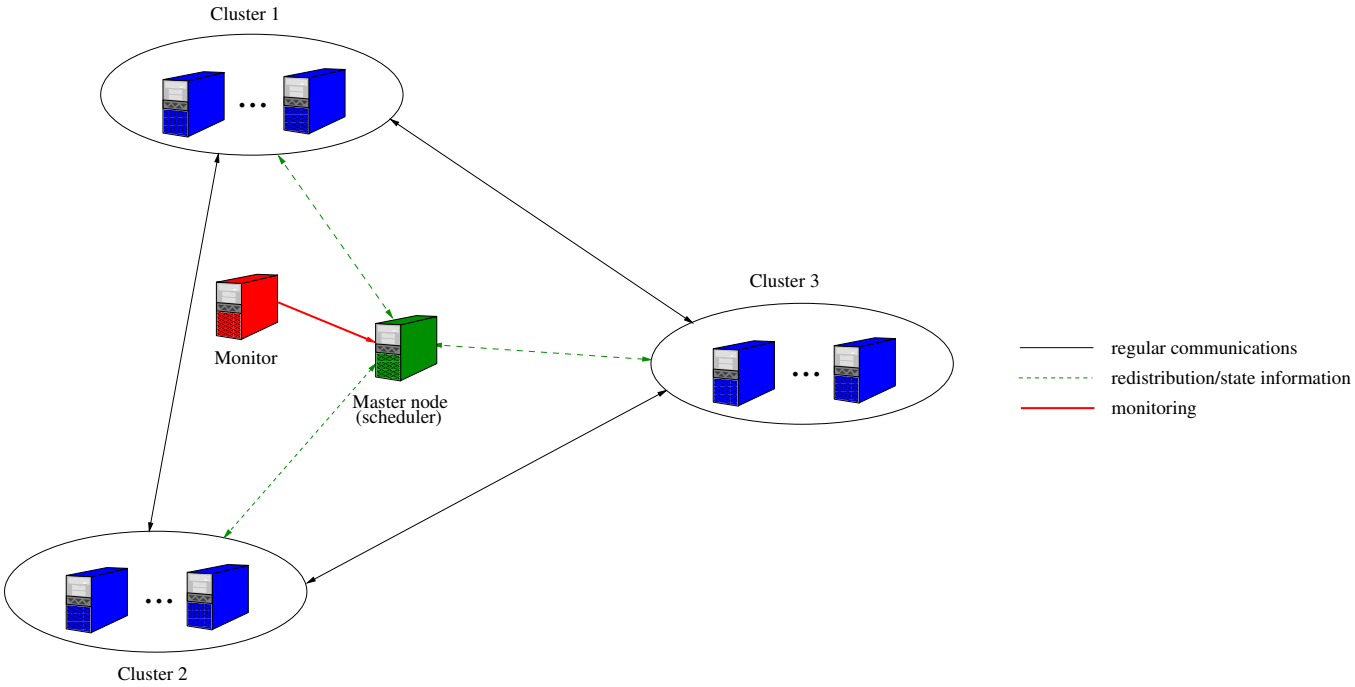


Figure 3. Experimental scheme.

experimental parallel and distributed computing research. The platform consists of nine geographically distributed sites, aggregating a total of 5,000 CPUs, and is funded by the French ACI Grid initiative of the French Ministry of Research and Education. Each of the nine sites hosts at least one commodity cluster, and the number of processors per cluster ranges from around 100 to around 1,000. The architectures of these processors are AMD Opteron, Intel Xeon, Intel Itanium 2, or PowerPC. Each cluster uses a Gigabit interconnect (GigaEthernet or Myrinet) internally, and all clusters are interconnected together by the wide-area RENATER Education and Research Network, an up to 10 Gigabit/sec network. The experiments presented in this section correspond to the platforms described in Table I. We selected three different set of results from a large amount of experiments. Each result has been reproduced at least five times before it is "validated". Note also that, in the platforms presented in Table I, neither the master nor the monitor nodes are counted. Finally, the intrinsic speed of the processors can vary with a factor near to 2 between the slowest processors (i.e. Bordeaux) and the fastest one (i.e. Lille). From the simulation point of view, the latest stable release of `SimGrid` have been used (i.e. version 3.3.2). In addition, we used the `simulacrum` [3] tool to generate the `XML` description of the platform which is used by `SimGrid`. Note that, we modified the platform so that the instant processor speed corresponds to the processor speed obtained when using our computing kernels. Note, that the platform produced by `simulacrum` provides the theoretical characteristics of the platform, we thus modify the description according to our measurements.

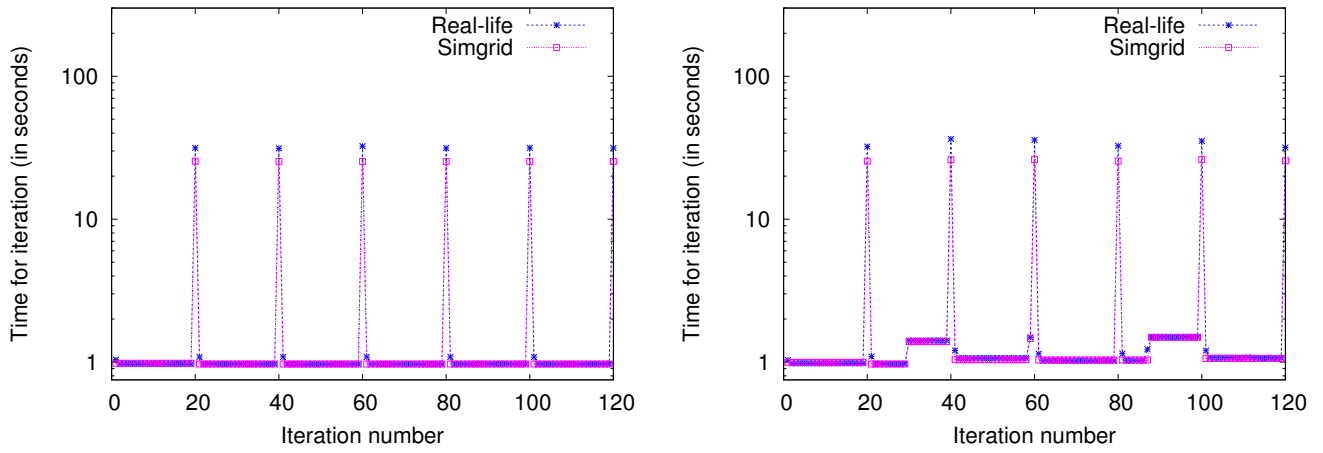
Concerning the parameters of the application, the size of the rectangular matrix is computed in such a way that each processor initially has a slice of 40000 by 1000 entries. Note also that instead of applying the load balancing described in [14] for building the initial ring of processes, we use an arbitrary distribution where all the processes appear in the ring with a number of columns of 1000. This is mainly motivated by the fact that the cost of the load balancing algorithm is prohibitive with platforms having more than 20 nodes. This explains why initially the load-balancing is not optimal in the results we present below. In addition, when not stated explicitly the redistribution is done once every 20 iterations. Furthermore, we limited our number of iterations to 120 even if the convergence hasn't been reached. This is motivated by the fact that it is too costly to wait for the convergence and by the fact that it is sufficient for our study to focus on 120 iterations (we are more interested in this paper by the behavior of the application than the results it produces). All the following figures are presented in log scale. Finally, when redistribution occurs, its time is included in the time of the previous iteration (the redistribution is done at the end of a given iteration).

We report in Figure 4, a comparison of the behavior of our application on the two contexts over a single site platform. First of all, if we consider Figure 4(a), we can observe that

[3] `http://pda.gforge.inria.fr/`

	Bordeaux	Grenoble	Lille	Lyon	Nancy	Orsay	Rennes	Sophia	Toulouse		total
1 site platform	-	-	-	-	-	-	-	24	-		24
2 sites platform	20	-	18	-	-	-	-	-	-		38
5 sites platform	24	-	20	-	8	30	-	18	-		100

Table I
DESCRIPTION OF THE EXPERIMENTAL PLATFORMS.



(a) No platform variation.

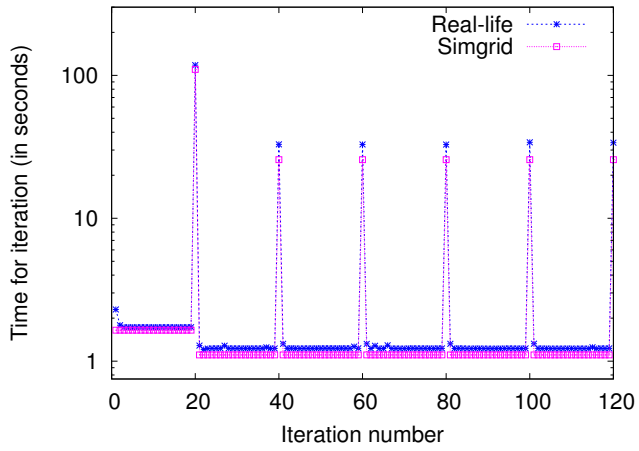
(b) With platform variation (3 platform variations, once every 29 iterations).

Figure 4. Time needed (in seconds) for each iteration on the real-life and the simulated platform: one site platform.

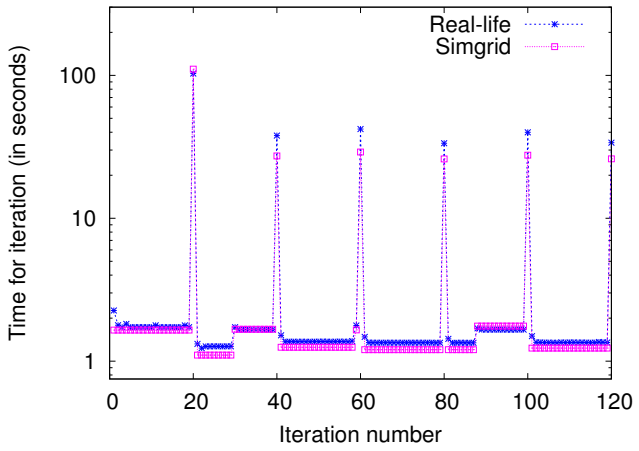
if we do not dynamically modify the characteristics of the platform, SimGrid is able to predict (almost perfectly) the execution time for each iteration. Note also that the peaks happening each 20 iterations are almost perfectly predicted. These peaks correspond to both the phase where we build the state of the platform and the time needed by data redistribution. However we can also observe that the time needed to perform and get the results of the measurements is slightly greater in the real-life execution. This is mainly due to the fact that we use external programs to perform the benchmarks (iperf, etc.). Thus it is difficult to predict the time needed for the invocation of the external tool. Another interesting point concerns the time needed to build the state information. Indeed, discovering the current state of the platform is very costly in comparison to the time needed by an iteration. However, the time needed for measuring the characteristics of the platform is independent from the number of processes and from the size of the problem to be solved. Thus, this cost can be amortized. Considering the case where the characteristics of the platform change during the execution, we report in Figure 4(b), the results obtained on the one site platform. We can see that once again SimGrid was able to reproduce the behavior observed in the real-life experiment. Finally, concerning the moment where the characteristics of the platform change (i.e. iteration 29, 58 and 87). We can see that from these iterations the execution time increases since the data distribution is no more optimal in the case of the modified platform. Then after the redistribution phase, the execution time decreases to its optimal value.

We report in Figure 5 the results obtained with the platform composed by two distant clusters : the first one in Bordeaux and the second one in Lille. This time, if we consider the case where no platform modification occurs (see Figure 5(a)), we can see that SimGrid was able once again to reproduce the behavior of the real-life execution. We can see also that this time, the curves do not fit perfectly and there is a slight difference between the real-life and the simulated execution. This is mainly due to the use of the long range network links which connect the two sites (these links are shared among the users of Grid'5000). Concerning the case where the platform is dynamically modified, we can see in Figure 5(b) that the curves are close and that SimGrid is able to simulate the behavior of the platform after a modification (see for example the timings between the 87^{th} iteration and 100^{th} iteration).

Concerning the results on the largest platform (i.e. the five sites one), the results are presented in Figure 6. We can see that when no platform modification is applied (see Figure 6(a)), the SimGrid curve is a "smooth" version of the one corresponding to the real-life execution. We can see also that this time, the time per iteration for the real-life

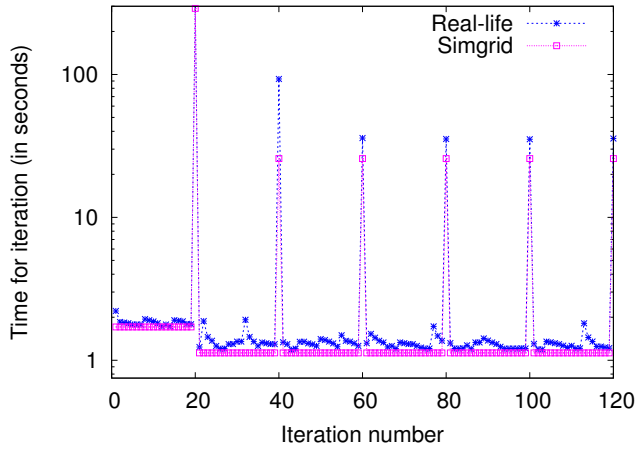


(a) No platform variation.

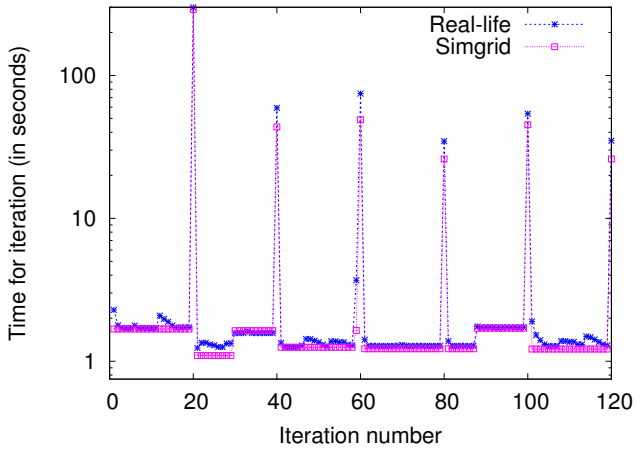


(b) With platform variation (3 platform variations, once every 29 iterations).

Figure 5. Time needed (in seconds) for each iteration on the real-life and the simulated platform: two sites platform.



(a) No platform variation.

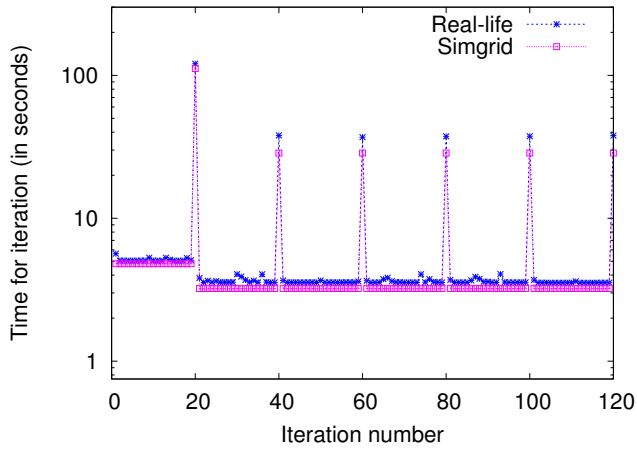


(b) With platform variation (3 platform variations, once every 29 iterations).

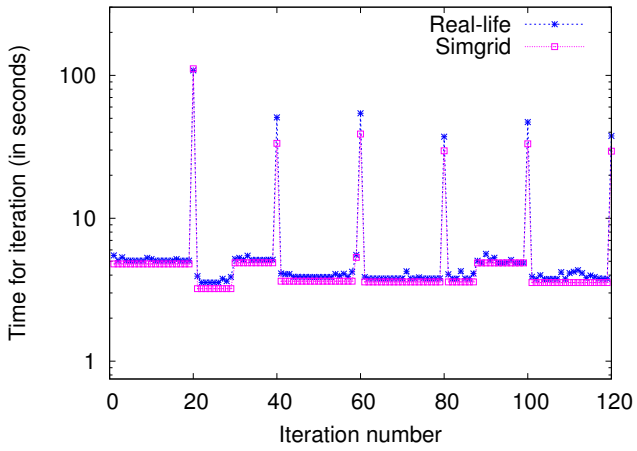
Figure 6. Time needed (in seconds) for each iteration on the real-life and the simulated platform: five sites platform.

execution is more irregular. This is mainly due to the use of numerous long range shared network links. An illustration of the complex behavior of the real-life execution is the point corresponding to iteration 40 in the real-life curve. As we can see, there is a big difference between the two curves (30 seconds for `SimGrid` against 92 seconds for the real-life execution). This difference was observed during the benchmarking phase where a single network test lasted 70 seconds. This was not the normal behavior and was due to a network incident (DNS failure) at that moment. We have chosen to show this result to illustrate the fact that in a real-life context, the experiments can be perturbed by external phenomena that cannot be managed unless by reperforming the experiment. The results corresponding to the case where the platform is dynamically modified are presented in Figure 6(b). We can see, that once again, `SimGrid` is able to predict the behavior of the application with the difference that in the real-life context, the results are more irregular. These tests on our most distributed experimental platform shows the interest of the `SimGrid` environment which succeeds to simulate the behavior of the application without being perturbed by all the "external noise" related to a real-life execution.

Finally, we report in Figure 7 experimental results where



(a) No platform variation.



(b) With platform variation (3 platform variations, once every 29 iterations).

Figure 7. Time needed (in seconds) for each iteration on the real-life and the simulated platform: two sites platform. Each iteration is three time more costly than a regular one.

slightly modified our application to simulate more computations and communication per iteration. This can correspond to other iterative schemes that solve more complex physical problems like eddy simulation. This time an iteration is three times more costly than in the regular experiments. In addition, during an iteration, each process has to exchange the three columns on the boundary with each neighbor in the logical ring. The results show that the behavior is the same as the one observed before : The shape of the two curves is very close. We can also observe in Figure 7(b) that `SimGrid` is able to reproduce the behavior of the real-life execution even when the characteristics of the platform are modified. One interesting point, is that in comparison with the results shown in Figure 5, these results are slightly more irregular for the real-life execution. This illustrates the fact that the more we have to deal with large data, the more we can be perturbed by external phenomena. These perturbations can make the results more difficult to analyze. This last point illustrates the interest of the simulated approach especially when using a tool that is able, as `SimGrid` does, to catch almost all the behavior observed during the real-life experiment.

V. Related work

The study of scheduling algorithms for distributed iterative applications has been an active area of research. Given the more and more complex nature of distributed platforms, it is often impossible to obtain theoretical or analytical results to compare the performance of algorithms targeting such systems in [15], [16]. This last statement is especially true when we consider the fact that it is very difficult to have to reproducible executions for an experimental evaluation of the algorithms. Thus, in the context of the experimental evaluation of scheduling algorithms resort to effectively evaluate and compare their efficacy over a wide range of scenarios.

Simulations not only enables repeatable results but also make it possible to explore wide ranges of platform and application scenarios in [17]. Moreover from `SimGrid`'s point of view, simulations are configurable, repeatable and fast. Furthermore, its models have clear limitations (e.g. for short transfers [18]).

Comparing the result of a simulation with a real-life experiment has been studied especially in the context of processor design. In [19], [20] authors try to compare different simulation schemes with the results obtained with a real chip. There are also many works that try to answer the question of how to evaluate the simulation [21] or what are the parameters that can be used to evaluate performance [22]. Concerning `SimGrid`, some studies have been done to evaluate the accuracy of the underlying models [18], [23].

VI. Conclusion

In this paper, we have presented a study where we compare the behavior of a given application on two different contexts: a simulated one using `SimGrid`, and a real-life implementation. Our application (*heat propagation*) requires the processes to be organized as a virtual ring and data redistribution may be needed when the resources performance varies. For this study, we implemented two versions of our target application. The first one was built on top of the `SimGrid` simulation kit. This represented a high-level implementation well-suited for studying algorithms. The other one was a real-life implementation on top of

standard C sockets. One major issue was the fact that we wanted to control the characteristics of the platform. Thus, we used an emulation tool, `wrekavoc`, to externally control and dynamically modify the performance of the resources. It is important to mention that the most consuming part of this work was to design a monitoring tool on top of `wrekavoc` which had to as less intrusive as possible. The monitor was in charge of applying the modifications and to generate the corresponding simulated platforms. Once again, the most difficult issue was to make sure that the characteristics of the simulated platform correspond to the real-life ones. The experimental study showed the interest of the `SimGrid` framework in the sense that it was able to mimic the behavior observed on real-life executions on several platforms composed from nodes of the `Grid'5000` environment. One interesting remark to point out is that the kind of experiments carried in this paper require some administrator rights on the computational nodes (especially to control network latency and bandwidth). This represents a major "brake" to perform such experiments in a larger scale. Results have also shown that when increasing the size of the platform, the simulated execution is still very close to the real one. However, there are some irregularities due to the use of long range network links. Finally, something that has not been mentioned in the paper is related to the time needed to perform the experiment. Whereas the time needed to perform the real-life experiments require near to one hour per execution, the time needed for the simulation does not exceed thirty seconds. This represents a major positive point for the simulation especially since it is able (in our context) to mimic the real-life execution.

As far as we know, this study is one of the first confronting `SimGrid` to real-life in the context of a real application. This work represents a first step of a validation through real-life experiments of `SimGrid` against complex applications. A next step could be to compare the behavior of `SimGrid` against a real-life execution in the context of tightly coupled application where network models have to be, in general, accurate.

Acknowledgement: We are grateful to Arnaud Legrand for his help on the experimental part of the paper.

REFERENCES

[1] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and Computation: Practice and Experience (CCPE)*, vol. 14, no. 13, pp. 1175–1220, 2002.

[2] W. H. Bell, D. G. Cameron, A. P. Millar, L. Capozza, K. Stockinger, and F. Zini, "Optorsim: A grid simulator for studying dynamic data replication strategies," *International Journal of High Performance Computing Applications*, vol. 17, no. 4, pp. 403–416, November 2003. [Online]. Available: http://dx.doi.org/10.1177/10943420030174005

[3] P. Garcia, C. Pairot, R. Mondejar, J. Pujol, H. Tejedor, and R. Rallo, "Planetsim: A new overlay network simulation framework," *Software Engineering and Middleware*, pp. 123–136, 2005.

[4] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, "Peersim. `http://peersim.sourceforge.net/`."

[5] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: a Generic Framework for Large-Scale Distributed Experiments," in *10th IEEE International Conference on Computer Modeling and Simulation*, Mar. 2008.

[6] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *11th IEEE International Symposium on High Performance Distributed Computing*, 2002.

[7] K. Fujiwara and H. Casanova, "Speed and accuracy of network simulation in the SimGrid framework," in *2nd international conference on Performance evaluation methodologies and tools*, 2007.

[8] A. Legrand and P. Velho, "Accuracy study and improvement of network simulation in the SimGrid framework," in *International Conference On Simulation Tools And Techniques For Communications, Networks And Systems & Workshops*, 2009.

[9] L.-C. Canon and E. Jeannot, "Wrekavoc: a tool for emulating heterogeneity," in *Parallel and Distributed Processing Symposium, IPDPS 2006*, 2006.

[10] H. Renard, Y. Robert, and F. Vivien, "Data Redistribution Algorithms for Homogeneous and Heterogeneous Processor Rings," in *HiPC'04 11th International Conference On High Performance Computing*, ser. Lecture Notes in Computer Science, 2004.

[11] H. Renard and Y. Robert and F. Vivien, "Data Redistribution Algorithms for Heterogeneous Processor Rings," *International Journal of High Performance Computing Applications*, vol. 20, no. 1, pp. 31–43, 2006.

[12] R. Wolski, N. Spring, and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," *Future Generation Computer Systems*, vol. 15, no. 10, pp. 757–768, 1999.

[13] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and T. Irena, "Grid'5000:a large scale and highly reconfigurable experimental grid testbed," *International Journal of High Performance Computing Applications*, vol. 20, 2006.

[14] A. Legrand, H. Renard, Y. Robert, and F. Vivien, "Mapping and Load-Balancing Iterative Computations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 6, pp. 546–558, 2004.

[15] H. Casanova, "Simgrid: A toolkit for the simulation of application scheduling," in *Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid'01)*. IEEE Computer Society, May 2001.

[16] H. Casanova, K. Fujiwara, A. Legrand, and M. Quinson, "The SIMGRID Project: Simulation and Deployment of Distributed Applications," in *HPDC'06 IEEE Computer Society Press, editor, The 15th IEEE International Symposium on High Performance Distributed Computing*, 2006.

[17] A. Legrand, L. Marchal, and H. Casanova, "Scheduling Distributed Applications: The SIMGRID Simulation Framework," in *Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, May 2003, p. 138.

[18] P. Velho and A. Legrand, "Accuracy study and improvement of network simulation in the simgrid framework," in *Simutools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, 2009, pp. 1–10.

[19] J. Gibson, R. Kunz, D. Ofelt, M. Horowitz, J. Hennessy, and M. Heinrich, "Flash vs. (simulated) flash: Closing the simulation loop," *ACM SIGOPS Operating Systems Review*, vol. 34, no. 5, pp. 49 – 58, 2000.

[20] C. J. Hughes, V. S. Pai, P. Ranganathan, and S. V. Adve, "Rsim: Simulating shared-memory multiprocessors with ilp processors," *Computer*, vol. 35, no. 2, pp. 40–49, 2002.

[21] L. Schaelicke, "Evaluating the impact of the simulation environment on experimentation results," *Perform. Eval.*, vol. 61, no. 4, pp. 329–346, 2005.

[22] L. C. Carrington, M. Laurenzano, A. Snavely, R. L. Campbell, and L. P. Davis, "How well can simple metrics represent the performance of hpc applications?" in *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2005, p. 48.

[23] K. Fujiwara and H. Casanova, "Speed and accuracy of network simulation in the simgrid framework," in *ValueTools '07: Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, 2007, pp. 1–10.