Validating Wrekavoc: a Tool for Heterogeneity Emulation

Olivier Dubuisson
Felix Informatique
Laxou, France
olivier.dubuisson@free.fr

Jens Gustedt
INRIA, Nancy – Grand Est
Villers lès Nancy, France
Jens.Gustedt@loria.fr

Emmanuel Jeannot
INRIA, Nancy – Grand Est
Villers lès Nancy, France
Emmanuel.Jeannot@loria.fr

Abstract

Experimental validation and testing of solutions designed for heterogeneous environment is a challenging issue. Wrekavoc is a tool for performing such validation. It runs unmodified applications on emulated multisite heterogeneous platforms. Therefore it downgrades the performance of the nodes (CPU and memory) and the interconnection network in a prescribed way. We report on new strategies to improve the accuracy of the network and memory models. Then, we present an experimental validation of the tool that compares executions of a variety of application code. The comparison of a real heterogeneous platform is done against the emulation of that platform with Wrekavoc. The measurements show that our approach allows for a close reproduction of the real measurements in the emulator.

1. Introduction

Distributed computing and distributed systems is a branch of computer science that has recently gained very large attention. Grids, clusters of clusters, peer-to-peer systems, desktop environments, are examples of successful environments on which applications (scientific, data managements, etc.) are executed routinely.

However, such environments, are composed of different elements that make them more and more complex. The hardware (from the core to the interconnected clusters) is hierarchical and heterogeneous. Programs that are executed on these infrastructures can be composite and extremely elaborate. Huge amounts of data, possibly scattered on different sites, are processed. Numerous protocols are used to interoperate the different parts of these environments. Networks that interconnect the different hardware are also heterogeneous and multi-protocol.

The consequence is that applications (and the algorithms implemented by them) are also very complex and very hard to validate. However, validation is of key importance: it helps to assess the correctness and the efficiency of the proposed solution, and, allows comparing a given solution to other already existing ones. Analytic validation consists in modeling the problem space, the environment and the solution. Its goal is then to gather knowledge about the modeled behavior using mathematics as a tool. Unfortunately, this approach is intractable in our case due to the complexity and partial unpredictability of the studied objects.

In our case, it is therefore necessary to switch to experimental validation. This approach consists in executing the application (or a model of it), observe its behavior on different cases and compare it with other solutions. This necessity for experiments truly makes this field of computer science an *experimental science*.

As in every experimental science, experiments are made through the means of tools and instruments. In computer science one can distinguish three different methodologies for performing experiments, namely, real-scale, simulation and emulation [1].

Emulation will be the concern of this paper: we show how our emulation tool called Wrekavoc [2] is able to help in experimentally validating a solution designed for a distributed environment. The goal of Wrekavoc is to transform a homogeneous cluster into a multi-site distributed heterogeneous environment. This is achieved by degrading the perceived performance of the hardware by means of software run at user level. Then, using this emulated environment a real unmodified program can be executed to test and compare it with other solutions.

The contribution of this paper is twofold. First, we present two extension of Wrekavoc with respect to the previous version described in [2], namely degradation of memory and an enhancement of the routing model. Then, in an intensive experimental campaign we demonstrate the *realism* of Wrekavoc. In order to do that we compare the execution of different applications on a heterogeneous platform with the execution on a homogeneous cluster and Wrekavoc. This is done by using many different parallel programming paradigms and by executing exactly the same applications on the real platform and in the emulator.

2. Related work

Here, we review some tools described in the literature that allow to perform large scale grid experiments. None of these tools allows to execute an unrestricted and unmodified application under precise and reproducible experimental conditions that would correspond to a given heterogeneous environment.

2.1. Real-scale Experimental Testbeds

GRID5000 [3] and *Das-3 [4]* (The Distributed ASCI Supercomputer 3) are distributed testbeds that interconect clusters of different sites. *Planet-lab [5]* is a globally distributed platform of about 500 nodes, all completely virtualized. It allows to deploy services on a planetary scale. With these platform, the control of the experimental conditions is often quite limited. In addition, the management of experimental campaigns is still a tedious and time-consuming task.

2.2. Emulators

The main technique of *Microgrid [6]* is to intercept major system calls of the application. Unfortunately, Microgrid is not supported anymore and does not work with the recent compiler versions. *eWAN [7]* is a tool that is designed for the accurate emulation of high speed networks. It does not take CPU and memory capacities of the hosts into account. *ModelNet [8]* also is a tool that is principally designed to emulate the network component. *Virtualization* is complementary to this work but it does not generate heterogeneity by itself. The RAMP (Research Accelerator for Multiple Processors) project [9] aims at emulating low level characteristics of an architecture (cache, memory bus, etc.) using FPGA.

2.3. Simulators

Bricks [10], SimGrid [11] and GridSim [12] are simulators that allow for the experimentation of distributed algorithms (in particular scheduling) and the impact of platforms and their topology. *GridNet [13], [14]* is specialized on data replication strategies. Others, focused on network simulations are NS2 [15], OPNetModeler [16] and OMNet++ [17]. A general disadvantage of all these tools is that there are only few studies concerning their realism.

3. Wrekavoc

Wrekavoc addresses the problem of controlling the heterogeneity of a cluster. Our objective is to have a configurable environment that allows for reproducible experiments of real applications on large sets of configurations. This is achieved without emulating any of the code of the application but by degrading hardware characteristics.Four characteristics of a node are degraded: CPU speed, memory size, network bandwidth and network latency.

Wrekavoc's notion to break a homogeneous cluster down into an heterogeneous one is called *islet*. An islet is a set of nodes that share similar limitations. Two islets are linked together by a virtual network which can also be limited. For the details see [2].

3.1. Use case

We describe here a use-case to show what Wrekavoc can help to achieve.

Suppose, a computer scientists has invented a new distributed algorithm for solving a computational problem on a distributed environments. He/she wants to test this new algorithm and compare it with other existing solutions. He/she also wants as little experimental bias as possible and therefore discards simulation. However, heterogeneous environments (such as Grid'5000 [3], [1] or Planet Lab [5]) that allow for well defined experimental conditions are not very common. Also they have a fixed topology and hardware setting, hence they do not cover a sufficiently large range of cases.

Now, thanks to Wrekavoc, he/she can take a homogeneous cluster (running under Linux) and transform it into a multi-site heterogeneous environment by defining and the topology, the interconnections characteristic, the CPUs speed and the memory capacity, Therefore, while only one homogeneous cluster is required, the possible configurations are numerous. The only restriction is that every emulated node must correspond to a real node of the cluster. Then, he/she has simply to run an application implementing the algorithm to test it and compare it with other existing solutions.

3.2. Design goals

Wrekavoc was designed with the following goals in mind.

1) Transform a homogeneous cluster into a heterogeneous multi-site environment. This means that we want to be able to define and control the

heterogeneity at a very low level (CPU, network, memory) as well as the topology of the interconnected nodes.
2) Ensure reproducibility. Reproducibility is a principal requirement for any scientific experiment. The same configuration with the same input must have the same behavior. Therefore, external disturbance must be reduced to the minimum or must be monitored so as to be incorporated into the experiment.
3) We want the user to be able to define and control the heterogeneity of the environment using simple commands and interfaces.
4) There are two ways for changing a homogeneous cluster into an heterogeneous one. The first way consists in partially upgrading (or downgrading) the hardware (CPU, network, memory). However, with this approach, the heterogeneity is fixed and the control is very low. The second approach consists in degrading the performance of the hardware by means of software. We have chosen this approach as it ensures a higher flexibility and control of the heterogeneity.
5) As we are going to degrade the different characteristics of a given node (CPU, memory, network), we want this degradation to be independent. This means that, for instance, we want to be able to degrade CPU without degrading the bandwidth and *vice versa*.
6) The last, but not least, wanted feature is *realism*. This means that we want Wrekavoc to provide a behavior as close as possible to the reality. Ensuring realism is necessary to assess the quality of the experiments and the confidence in the results.

3.3. Implementation details

The implementation follows the client-server model. On each node for which we want to degrade the performance a daemon runs and waits for orders from the client. The client sends a configuration file that describes the heterogeneity settings to this daemon. When a server receives a configuration order it degrades the node characteristics accordingly. The client can also order to recover the non-degraded state.

Four characteristics of a node are degraded: CPU speed, memory size, network bandwidth and network latency. We now explain briefly here how each of these degradations is performed. For the details see [2].

CPU Degradation. We have implemented several methods for degrading CPU performance. The first approach consists in managing the frequency of the CPU through the kernel CPU-Freq interface. As this interface is not always available or has a too coarse granularity, we propose two other solutions. One is based on CPU burning. A program that runs under real-time scheduling policy burns a constant portion of the CPU cycles, regardless how many processes are currently running. The drawback of this approach is that burning CPU cycles degrades also the network performance (processing the TCP stack requires some processing) and thus breaches the independence requirement described in the above section. The third approach is based on user-level process scheduling called CPU-lim. A CPU limiter is a program that supervises processes of a given user. On Linux, using the `/proc` pseudo-filesystem, it suspends the processes (using signal `SIGSTOP`) when they have used more than the required fraction of the CPU. It reactivates processes when necessary to achieve a precise degradation (using signal `SIGCONT`).

Network Limitation. Limiting latency and bandwidth is done using *tc* (traffic controller) based on *Iproute2* a program that allows advanced IP routing. With this tool it is possible to control both incoming and outgoing traffic. This needs versions of *tc* 2.6.8.1 or above to allow to control the latency of the network interface together with the bandwidth.

Memory Limitation. Up to the present version, memory degradation was done by limiting the largest `malloc` a user can make. This is possible through the security module PAM. Limiting the whole memory usable by all the processes is a new feature that will be described in Section 4.

3.4. Configuring and Controlling Nodes and Links

Wrekavoc's main notion to break a homogeneous cluster down into a heterogeneous one is called *islet*. An islet is a set of nodes that share similar limitations. Two islets are linked together by a virtual network which can also be limited (see Fig. 1).

An islet is defined as a union of IP address intervals or a list of machine names. All islet configurations are stored in a configuration file. In a second part of this file the network connection (bandwidth and latency) between each pair of islet is specified.

The common characteristics of the nodes in an islet are described by a random variable. We provide two types of distributions for describing this random variable: Gaussian or uniform. For instance, we may specify that each node in a given islet has a network bandwidth chosen uniformly in the interval [100,500] Mbit/s and also that the latency is chosen using a Gaussian distribution with a mean of 10 ms and a standard deviation

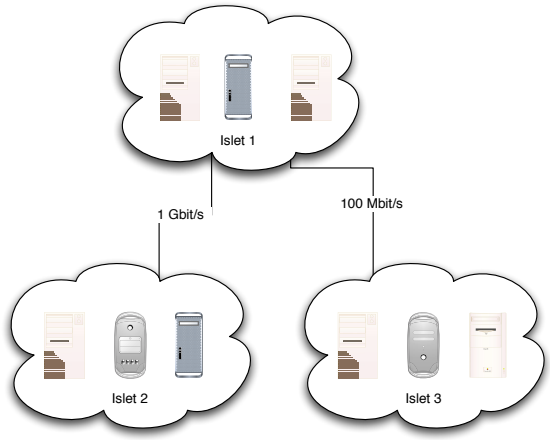


Figure 1. Islets logical view

of 5 ms. In order to ensure the reproducibility of a given setting, it is possible to use a fixed seed for drawing the random numbers in the configuration of each islet.

4. New Features

In the first version of Wrekavoc [2] there was a gap between the intended semantic of the configuration file and the actual implementation. Two main problems needed to be addressed. First, the memory limitation was only limited to the per process usage and the network regulation only worked well between nodes within the same islet.

We now show how we have addressed these different points.

4.1. Real memory degradation

The memory limitation was done by restricting the maximum memory each process can allocate. Therefore, the total amount of memory on a given host was not restrained since several processes could allocate the maximum amount up to the physical memory size.

In this new version, in order to limit the total size physical memory to a given target size S now uses the POSIX system calls `mlock` and `munlock` to pin physical pages to memory. These pages are then inaccessible to the application and thus constrain the physical memory.

4.2. Adding gateways for improved network regulation and routing

The network degradation within an islet has successfully been evaluated and validated in [2]. However between two islets the degradation was not correct when several communication streams use the same logical inter-islet link. In this case the degradation was performed by the sending and receiving node. Hence, each pair of such nodes were not aware of other communications between different pairs. The emulation worked as if there were as many inter-islet links than pairs of processors between the islets.

To solve this problem, we have dedicated, for each islet, a node that acts as a gateway. This gateway is responsible to forward TCP packets from one islet to another by sending these packets to the corresponding gateway. Gateways regulate bandwidth and latency using TC in the same way as for regular nodes. This allows complex topologies between islets where some islets are directly connected and some are not. We have implemented a standard routing protocol[1] for forwarding packets between islets.

5. Validation

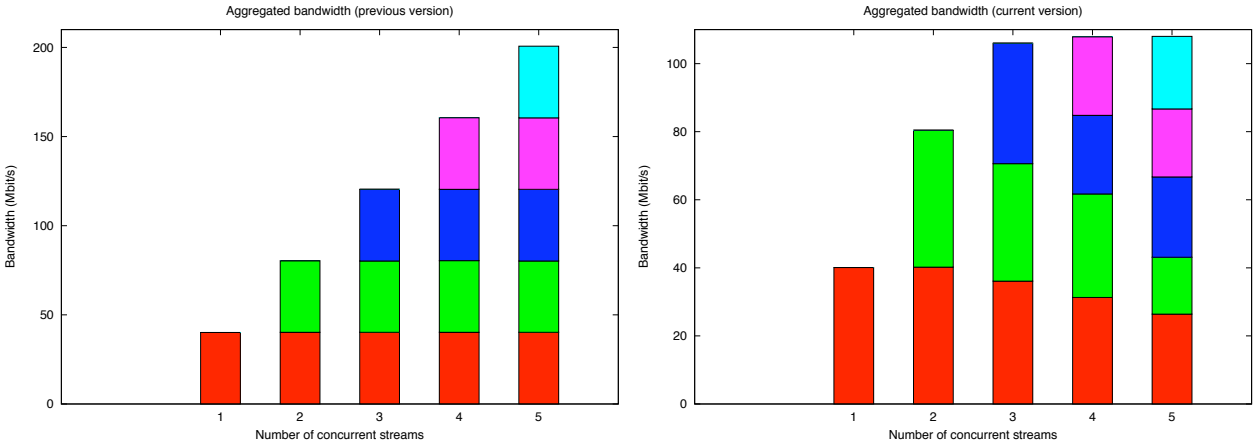
5.1. The realism issue

An experimental tool such as a simulator, an emulator or even a large-scale environments always provides an abstraction of the reality: to some extent, experimental conditions are always synthetic. Therefore, the question of realism of the tools and the accuracy of the measurements is of extreme importance. Indeed, the confidence in the conclusions drawn from the experiments greatly depends on this realism. Hence, a good precision of the tools is mandatory to perform high quality experiments, to be able to compare with other results, for reproducibility, for calibration to a given environment, for possible extrapolation to larger settings than the given testbed, etc. However, a brief look at the literature shows that concerning simulators [11], [12] or emulators [6], [8], the validation of the proposed tools as a whole is seldomly addressed. Concerning Wrekavoc, we perform this validation here.

5.2. Inter-islet network

The test consists in measuring the bandwidth that is observed for network streams between two nodes

1. We use the RIP (Routing Information Protocol) that sets up routes by minimizing the number of hops



(a) Behavior of the previous version: aggregated bandwidth is linearly increases with the number of streams

(b) Behavior of the new version: aggregated bandwidth is correctly limited to 100Mbit/s

Figure 2. Inter-islet aggregated bandwidth when adding streams (shown in different colors). Node bandwidth: 40 Mbit/s, Inter-islet link: 100 Mbit/s.

of different islets. The expected behavior is that the aggregated bandwidth should increase linearly until it reaches the bandwidth of the inter-islet link. Beyond that point, the inter-islet bandwidth is shared between the different communicating streams. Thanks to TCP this sharing must be fair. This means that the amount of allocated bandwidth must roughly be the same for every stream. In Fig. 2, we present one such test. Here each individual node can send data at 40 Mbit/s and the inter-islet link has a bandwidth of 100 Mbit/s.

We see the difference between the previous version of Wrekavoc. In Fig. 2(a), corresponding to the previous version, each stream behaves independently without considering the limitation of the inter-islet link (as described in section 4.2). With the current version (Fig. 2(b)) the behavior respects the experimental hypothesis with a margin of error less than 10% and thus we may conclude that Wrekavoc enables a fair sharing of the bandwidth.

The second set of benchmarks (not shown as a graph) consists in measuring the bandwidth on a chain of islets. The goal is to see what happens in the presence of a bottleneck. To evaluate that, we have linked 4 islets in a chain. Within each islet one node can send data at 100 Mbit/s. The link between Islets 1 and 2 has a bandwidth of 20 Mbit/s the other links have higher bandwidth. We have measured the bandwidth used on each link when a node communicates from Islet 1 to Islet 4. We have seen that on each link the bandwidth does not exceed 20 Mbit/s with less than 5% of error. Here again, Wrekavoc behaves as desired.

The last set of benchmarks we have performed consists in measuring what happen if streams originating from different islets arrive at the same islet. Here again we observe that independently from the number of incoming streams the sum of the bandwidths for each stream never exceeds the configured bandwidth of the middle islet.

5.3. Validation with real applications

We validate the realism of Wrekavoc by comparing the behavior of the execution of a real application on a real heterogeneous environments and the same application using Wrekavoc. Such validation uses all the features of Wrekavoc (Network, CPU and memory degradation).

The heterogeneous platform we used was composed of 12 PC linked together by a Gbit switch. The characteristics of the nodes are described in Table 1. All nodes have the same Linux distribution and kernel version and the same version of MPI (OpenMPI 1.2.2). We have huge heterogeneity in terms of RAM, MIPS, clock frequency, network card, type and architecture of processors.

Unfortunately, it has not been possible to perform experiments on a larger heterogeneous cluster. To the best of our knowledge, a heterogeneous environment

ID	Proc	RAM (MiB)	System	Freq (MHz)	HDD type	HDD (GiB)	Network card (Mbit/s)	MIPS
1	P. IV	256	Debian 2.6.18-4-686	1695	IDE	20	100	3393
2	P. IV	512	Debian 2.6.18-4-686	2794	IDE	40	1000	5590
3	P. IV	512	Debian 2.6.18-4-686	2794	IDE	40	1000	5590
4	P. III	512	Debian 2.6.18-4-686	864	IDE	12	100	1729
5	P. III	128	Debian 2.6.18-4-686	996	IDE	20	100	1995
6	P. III	1024	Debian 2.6.18-4-686	498	SCSI	8	1000	997
7	P. II	128	Debian 2.6.18-4-686	299	SCSI	4	1000	599
8	P. II	128	Debian 2.6.18-4-686	299	SCSI	4	100	599
9	P. II	128	Debian 2.6.18-4-686	298	SCSI	4	100	596
10	P. II	64	Debian 2.6.18-4-686	398	IDE	20	100	798
11	P. IV	512	Debian 2.6.18-4-686	2593	IDE	40	1000	5191
12	Dual Opteron 240	2048	Debian 2.6.18-4-amd64	1604	IDE	22	1000	3211 and 3207

Table 1. Description of the heterogeneous environment

with the desired characteristics (heterogeneity, scale, reproducible experimental conditions) that could be used to calibrate Wrekavoc against it is not available. For instance experiments on Planet-Lab are usually not reproducible and Grid'5000 is not heterogeneous enough.

We have compared the execution of different applications on the heterogeneous cluster described above and on Grid'5000 clusters *heterogeneized* with Wrekavoc.

5.3.1. Calibration. Due to the fact that the modeling of CPU in Wrekavoc uses frequency scaling (with a small set of possible frequencies), it has not always been easy to calibrate Wrekavoc to obtain a behavior similar to the above heterogeneous environment. Sometimes it has required some fine tuning of the configuration depending of the tested application. The calibration issue is a well known problem of emulation. We think that despite the good results presented here that we can improve this aspect of Wrekavoc. However, since the goal of this paper is to asses the realism of Wrekavoc, calibration problems are left for future work.

5.3.2. Fine grain without load balancing. The first set of experiment we performed aims at showing what happen when no load balancing is achieved on a heterogeneous cluster. In this case, on the real environment, faster nodes will finish their computation earlier and will be idle during some part of the computation. We wanted to see how Wrekavoc is able to reproduce this behavior.

The application we used is a parallel sort algorithm implemented within the parXXL [18] library. The algorithm used is based on Gerbessiotis' and Valiant's sample sort algorithm [19].

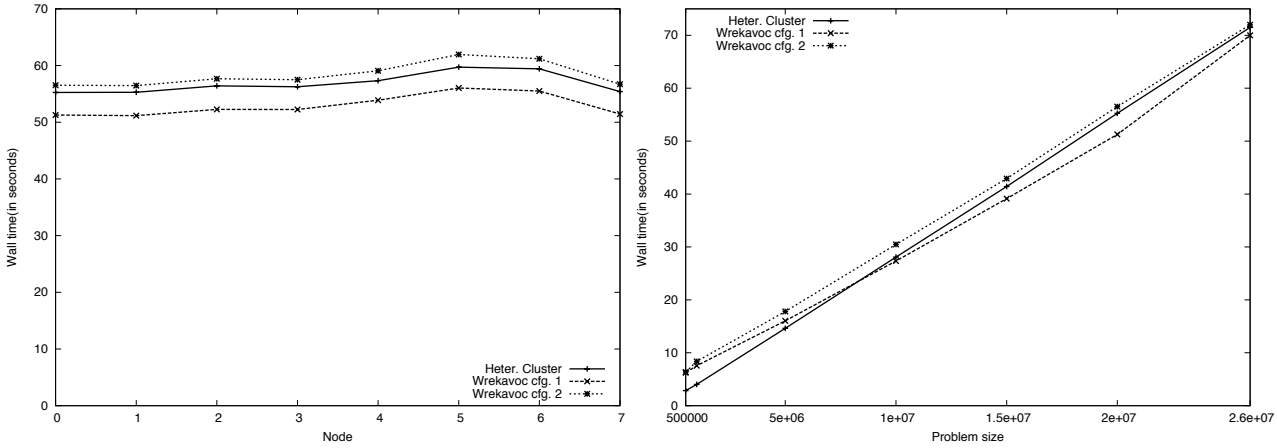
Fig. 3 shows the wall-time of each node for the heterogeneous cluster and two configurations.

In order to see the difference we performed 5 sorts of 20,000,000 `doubles` (64 bits) in a row. Results show that Wrekavoc is able to reproduce the reality with a good accuracy as both configurations are close upper (or under) approximation. In the worst case, the Wrekavoc configuration results have a difference of 8% from the real platform.

In Fig. 4, we show the wall-time for the first node when varying the problem size from 500,000 to 20,000,000 of `doubles`. We see here, that Wrekavoc has some difficulties in correctly emulating the reality for small size problem. However, for large sizes, when the running time is above 20 seconds, the estimation is very realistic with an error margin below 10%.

5.3.3. Static load balancing. Here we have tested applications that perform a static load balancing at the beginning of the application according to the speed of the processors and the amount of work to perform. We have implemented two algorithms that perform parallel matrix multiplication on heterogeneous environments. The first algorithm from Beaumont *et al.* [20] is based on a geometric partition of the columns on the processors. The second from Lastovetsky *et al.* [21] uses a data partitioning based on a performance model of the environment.

In Fig. 5(a), we show the comparison between the CPU, communication and synchronization time for the Beaumont *et al.* algorithm for matrix sizes of 1000. Nodes are sorted by CPU time. We see that the Wrekavoc behavior is very close to the behavior when using the heterogeneous cluster. Also, the proportions of the timings (CPU, communication and synchronization) are conserved. In order to discuss the differences between the two graphs of Fig. 5(a) quantitatively, we plot the relative error in Fig. 7(a). More precisely, in this graph we show the sum of the relative errors of all the timings (CPU, communication



(a) Times of each node of the heterogeneous cluster with 20,000,000 `doubles`.

(b) Times of machine "*ID 2*" with varying problem size.

Figure 3. Execution walltime for the sort application on the real and emulated heterogeneous cluster. Averages over 10 runs.

and synchronization) of the graphs of Fig. 5(a): for each x value i and each timing (CPU, communication and synchronization) we compute the relative error $100 \times \text{abs}(c_h(i) - c_w(i))/C_h(i)$ where $c_h(i)$ (resp. $c_w(i)$) is the value of the timing for the heterogeneous (resp. Wrekavoc) case and $C_h(i)$ is the sum of the value of all the timings on the heterogeneous cluster. In the figure, the different colors then represent the contributions of the three resources.

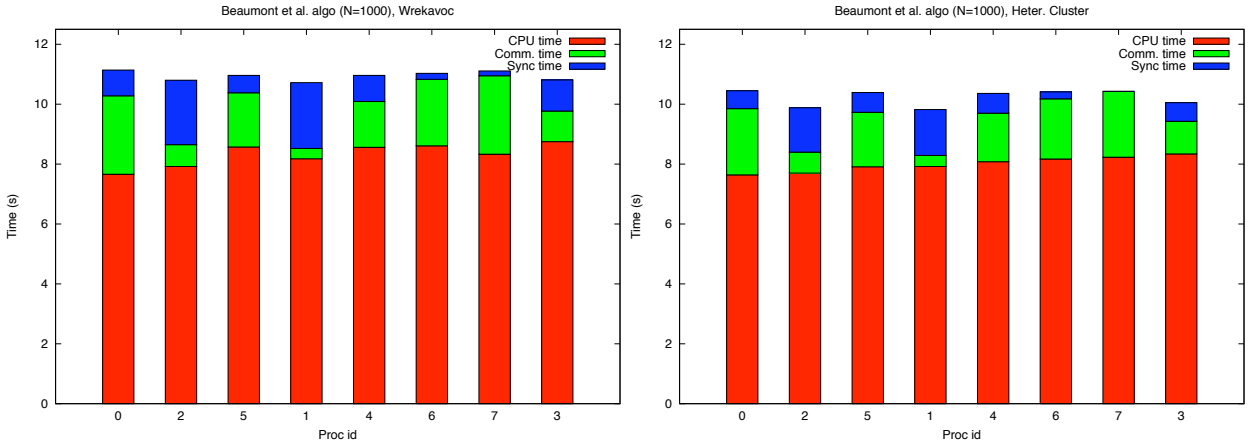
We see that the overall relative error is always below 10%. The worst case is for processor 5 for which the error is mainly due to the synchronization time while in absolute, this timing is the very small (less than 0.6 seconds).

In Fig. 5(b), we show the comparison between the CPU, communication and synchronization time for the Beaumont *et al.* algorithm for varying matrix size on a fixed node. We see that the Wrekavoc behavior is very close to the behavior when using the heterogeneous cluster. The only problem concerns an increasing shift of the timings when matrix size increases. More precisely, in Fig 7(b), we stack the contribution of the relative error between Wrekavoc and the heterogeneous cluster of all the timings (CPU, communication and synchronization) of the graphs of Fig. 5(b). We see that the overall relative error is always lower than 10%. Moreover, we see that the communication contribution to the error is marginal (less than 0.6 %). This means that here, Wrekavoc was able to emulate the communication with a great precision.

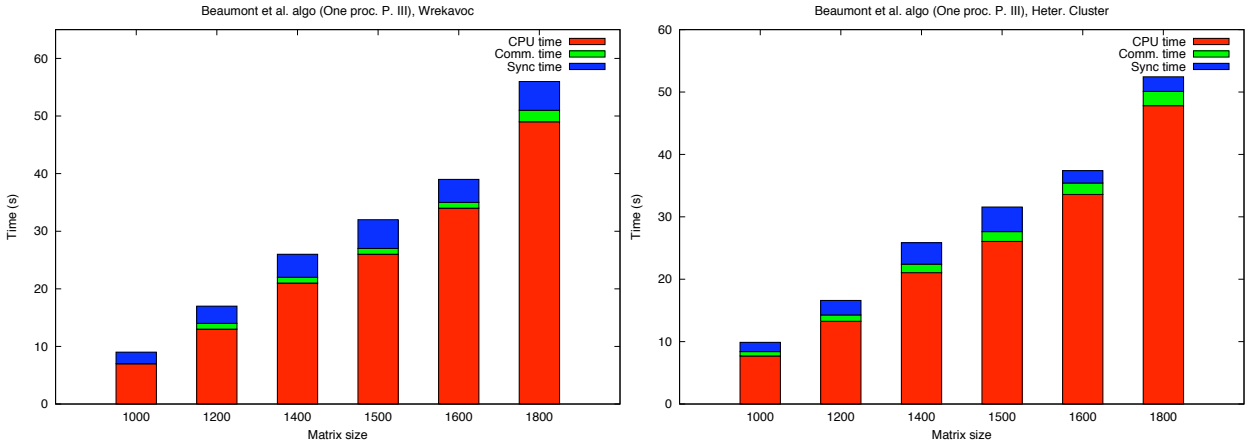
In Fig. 6(a) and 6(b), we present the same measurements as in Fig. 5(a) and 5(b) but for the Lastovetsky *et al.* algorithm. Here again we see that the Wrekavoc is very realistic with again a small shift in execution time when the matrix size increases. In Fig 7(c) we stack the contribution of the relative error between Wrekavoc and the heterogeneous cluster of all the timings (CPU, communication and synchronization) of the graphs of Fig. 6(a). From this figure, we can see that the relative error is very low in general (lower than 6%). There is one exception for processor 5 where the error is a little bit larger than 10%. The relative error of the graphs of Fig. 6(b) are shown in Fig. 7(d). Results show that these errors are always lower than 8% and that the CPU and communication time do not contribute a lot to these errors showing that Wrekavoc is doing a good job for emulating the heterogeneous hardware.

In conclusion to this section, we see that Wrekavoc precisely emulates the heterogeneous cluster (within 10%) in the general case. Moreover, most of the relative error is caused by synchronization. The problem might lay in the emulation of the idle times of processes. This is linked to the way they are scheduled by Wrekavoc.

5.3.4. Dynamic load balancing for iterative computation. Here, the dynamic load balancing strategy consists in exchanging some workload at execution time in function of the progress in the previous iteration. The program we have used solves an advection-diffusion problem (kinetic chemistry) described in [22]. Here, the



(a) Beaumont *et al.* algorithm, $N = 1000$



(b) Beaumont *et al.* algorithm for the same processor

Figure 4. Comparison of node runtime (CPU, communication and synchronization) for the static load balancing application (matrix multiplication – Beaumont *et al.* algorithm). For each subfigure, Wrekavoc is on the left and the heterogeneous cluster is on the right

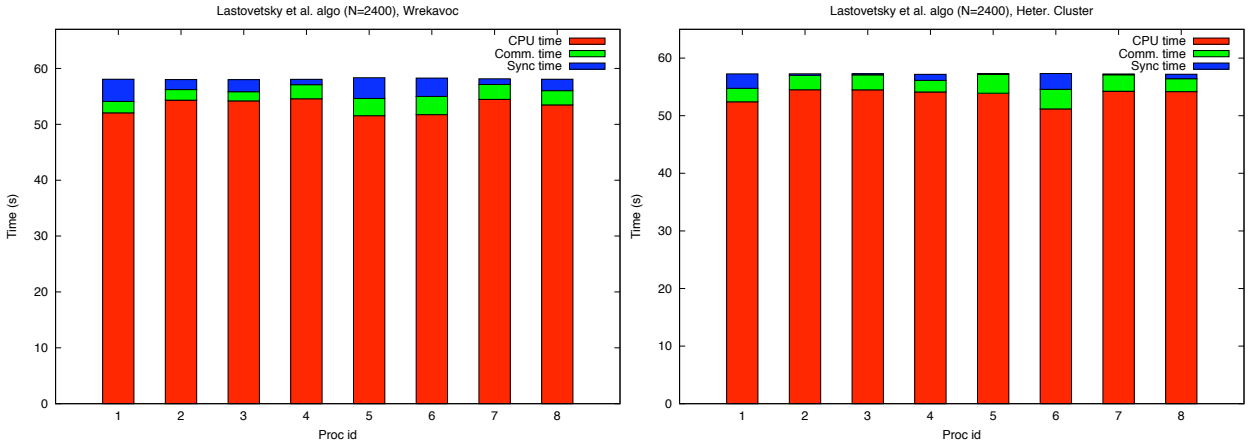
load corresponds to the number of lines of the input matrix that is given to a particular process.

In Fig. 8 we show the evolution of the load balancing of this application. At each iteration, we have monitored the number of lines hold by each processor. We plot this number during the whole execution of the application for a problem on a surface of 300 columns and 400 lines. There are 40 iterations. The results show that the evolution of the load balancing using Wrekavoc (right) or the heterogeneous cluster (left) are extremely similar. Processor 1 (the fastest), holds an increasing amount of load in both situation. More interestingly, processor 2 starts to have a very low load and then its load increases. Moreover, processor 7, the slowest one, is the least loaded at the end.

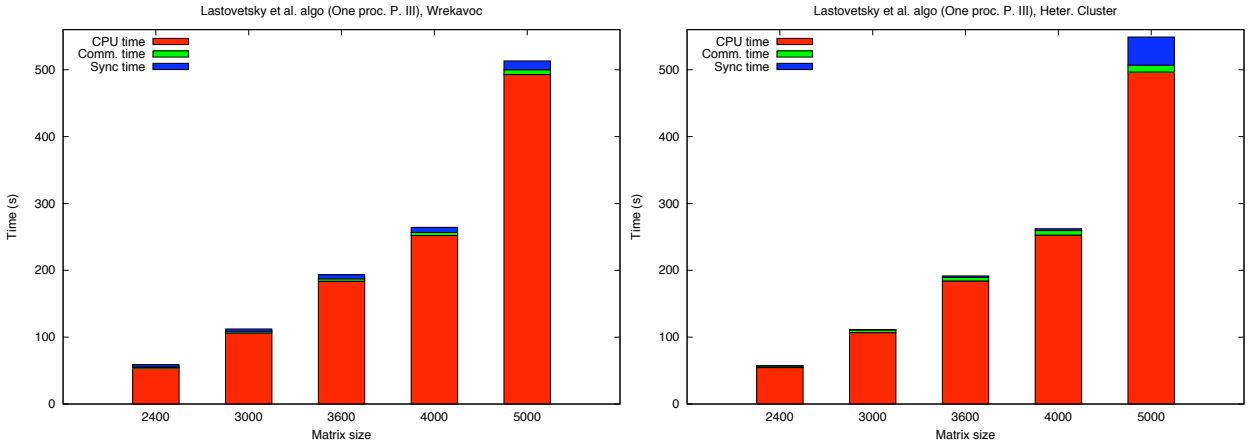
In Fig. 9, we show the relative error between the two graphs of Fig. 8. More precisely, for the eight processors involved in these experiments we computed the relative error as:

$$100 \times \text{abs}(n_w(i) - n_h(i))/n_h(i) \text{ for } i \in [1, 40]$$

(where $n_h(i)$ (resp. $n_w(i)$)is the number of lines treated by the processor for the heterogeneous cluster (resp. Wrekavoc case) at iteration i. Then, for each iteration, we plot the maximum the maximum, minimum and average value. We see that the error never exceeds 15%



(a) Lastovetsky *et al.* algorithm, N=2400



(b) Lastovetsky *et al.* algorithm for the same processor

Figure 5. Comparison of node runtime (CPU, communication and synchronization) for the static load balancing application (matrix multiplication – Lastovetsky *et al/* algorithm). For each subfigure, Wrekavoc is on the left and the heterogeneous cluster is on the right

on maximum and 7% on average. Moreover the average error relatively stable after 30 iterations.

In summary, the predictability of the behavior of the emulated application is very high.

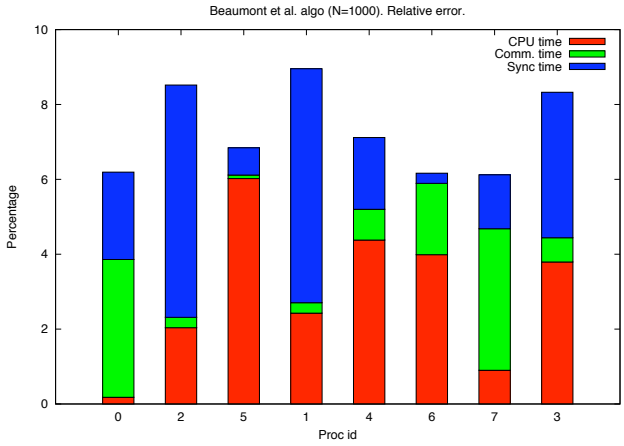
6. Conclusion

Nowadays computing environments are more and more complex. Analytic validation of solutions for these environments are not always possible or not always sufficient.

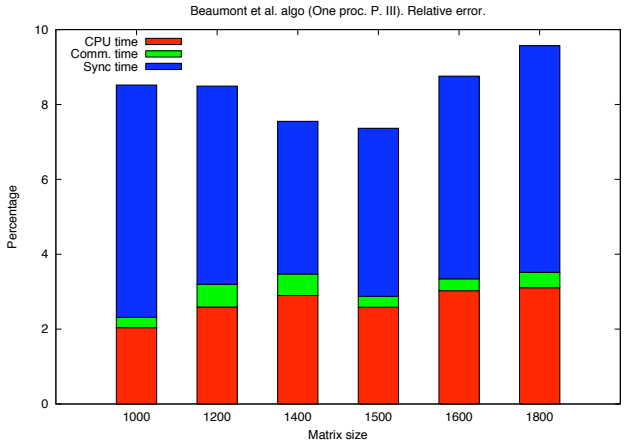
In this paper, we present the enhancement and the validation of Wrekavoc, a tool for performing experimental test, benchmarks and comparison of solutions designed for distributed and heterogeneous environments.

Other than parts of our community, we take validation and calibration of models and tools as real scientific challenge. We are convinced that the confidence in the results that are found by means of such tools (simulators, emulators or real-scale platforms), is very low without an extensive study of their realism.

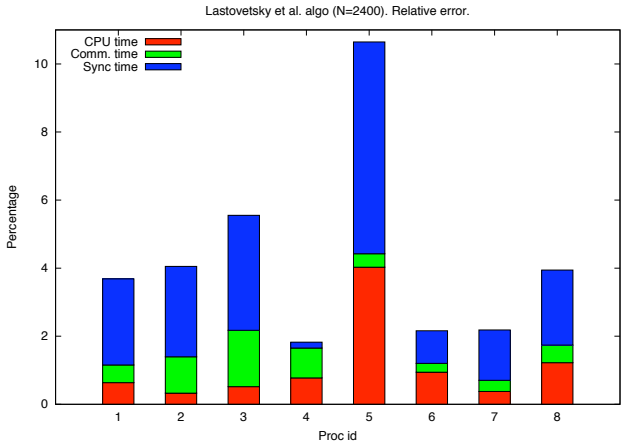
We have validated Wrekavoc using micro-benchmarks (for inter-site communication) and by



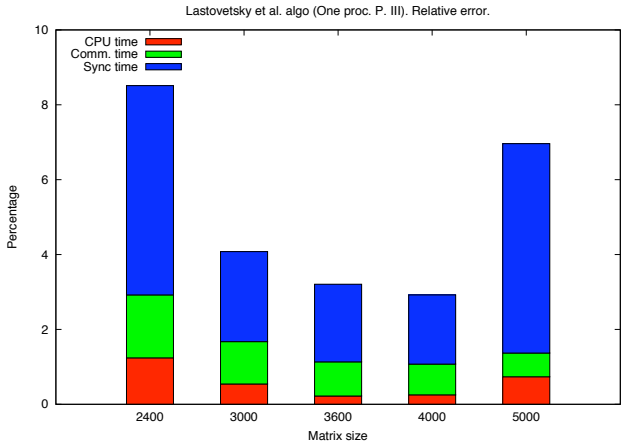
(a) Between the graphs of Fig. 5(a)



(b) Between the graphs of Fig. 5(b)



(c) Between the graphs of Fig. 6(a)



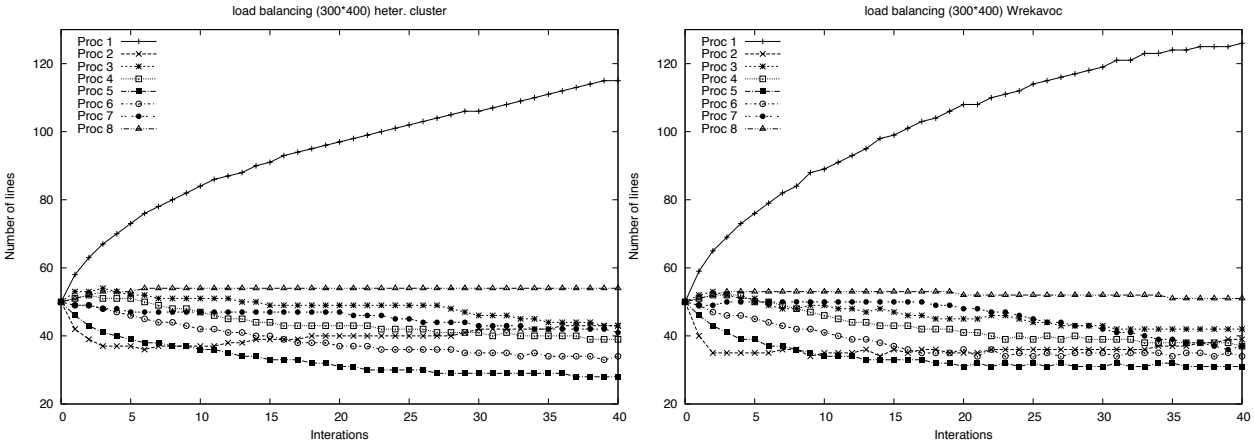
(d) Between the graphs of Fig. 6(b)

Figure 6. Relative errors. Contribution of each of the timings (CPU, communication and synchronization.)

comparing the execution of several real applications on a real environments to a cluster running Wrekavoc. The results obtained in the experiments concern all the features of Wrekavoc (network regulation, memory limitation and CPU degradation). Results show that Wrekavoc is realistic and has a very good reproducibility. Moreover, the tool provides emulations that are not tied to the real host platform: at the application/user level, different architectural features (*e.g.*processor architecture, memory bandwidth, cache, instruction sets, etc.) are correctly emulated. But evidently, still it could be improved.

Furthermore, despite the fact that the experiment has been done on an average-scale environment we are confident that the similar results would have been possible on a larger setting. Unfortunately, to the best of our knowledge, such large-scale heterogeneous environments with reproducible experimental conditions do not exist.

For us it was extremely important to first be able to emulate a static heterogeneous platform accurately, before even trying to emulate dynamic features. Future work is directed towards the ease of the calibration of the environment, a better emulation of multi-threaded programs and to new modeling features such as node volatility and dynamic load.



(a) heterogeneous cluster

(b) emulation with Wrekavoc

Figure 7. Comparison of the evolution of the load-balancing for executing the advection diffusion application

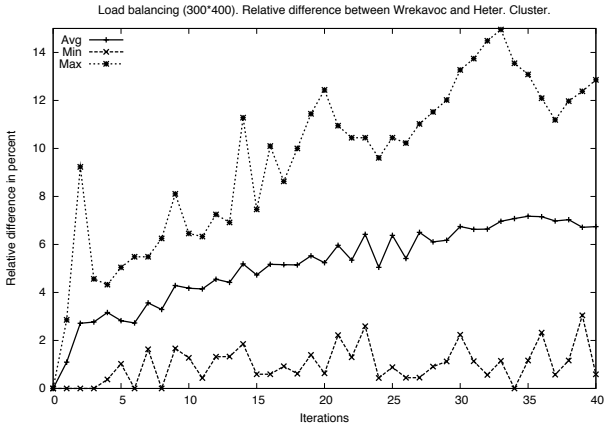


Figure 8. Relative error in percent between graphs of Fig. 8

Acknowledgment

This work was supported by a one year internship grant for the first author by FONGECIF to INRIA Nancy – Grand Est.

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] 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 I. Touche, Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed, International Journal of High Performance Computing Applications, **20**(4), 481–494 (2006).

[2] L.-C. Canon and E. Jeannot, Wrekavoc a tool for emulating heterogeneity, *15th IEEE Heterogeneous Computing Workshop (HCW'06)* (Island of Rhodes, Greece, 2006).

[3] The Grid'5000 experimental testbed, URL https://www.grid5000.fr.

[4] The DAS-3 project, URL http://www.cs.vu.nl/das3/.

[5] Planet lab, URL http://www.planet-lab.org/.

[6] H. Xia, H. Dail, H. Casanova, and A. A. Chien, The microgrid: Using online simulation to predict application performance in diverse grid network environments, *CLADE* (2004).

[7] P. Vicat-Blanc Primet, O. Glück, C. Otal, and F. Echantillac, Emulation d'un nuage réseau de grilles de calcul: eWAN, Tech. Rep. RR 2004-59, LIP ENS, Lyon, France (2004).

[8] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker, Scalability and accuracy in a large-scale network emulator, SIGOPS Oper. Syst. Rev., **36**(SI), 271–284 (2002).

[9] The RAMP project: Research Accelerator for Multiple Processors, URL http://ramp.eecs.berkeley.edu/.

[10] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima, Overview of a performance evaluation system for global computing scheduling algorithms, *HPDC '99* (1999).
[11] H. Casanova, A. Legrand, and M. Quinson, SimGrid: a Generic Framework for Large-Scale Distributed Experiments, *10th IEEE International Conference on Computer Modeling and Simulation* (2008).
[12] R. Buyya and M. M. Murshed, GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Journal of Concurrency and Computation: Practice and Experience, **14**(13–15) (2002).
[13] H. Lamehamedi, Z. S. andB. Szymanskia, , and E. Deelman, Simulation of dynamic data replication strategies in data grids, *Proc. 12th Heterogeneous Computing Workshop (HCW)* (2003).
[14] H. Lamehamedi, B. Szymanski, Z. Shentu, , and E. Deelman, Data replication strategies in grid environments, *Proc. Of the Fifth International Conference on Algorithms and Architectures for Parallel Processing* (2002).
[15] The network simulator NS2, URL http://www.isi.edu/nsnam/ns.
[16] J. Prokkola, Opnet - network simulator, URL http://www.telecomlab.oulu.fi/kurssit/521365A_tietoliikennetekniikan_simuloinnit_ja_tyokalut/Opnet_esittely_07.pdf (2007).
[17] A. Varga, The omnet++ discrete event simulation system, URL http://www.omnetpp.org/download/docs/papers/esm2001-meth48.pdf (2001).
[18] parXXL: A fine grained development environment on coarse grained architectures, URL http://parxxl.gforge.inria.fr/.
[19] A. V. Gerbessiotis and C. J. Siniolakis, A new randomized sorting algorithm on the BSP model, Tech. rep., New Jersey Institute of Technology, URL http://www.cs.njit.edu/~alexg/pubs/papers/rsort.ps.gz (2001).
[20] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, Matrix multiplication on heterogeneous platforms, IEEE Trans. Parallel Distributed Systems, **12**(10), 1033–1051 (2001).
[21] A. Lastovetsky and R. Reddy, Data partitioning with a realistic performance model of networks of heterogeneous computers with task size limits, *Proceedings of the Third International Symposium on Parallel and Distributed Computing / Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (ISPDC/HeteroPar'04)*, pp. 133–140 (IEEE Computer Society Press, 2004).
[22] F. Vernier, *Algorithmique itérative pour l'équilibrage de charge dans les réseaux dynamiques*, Ph.D. thesis, univ. de Franche-Conté (2004).

Olivier Dubuisson received an engineering master from CNAM (Conservatoire National des Arts et Métiers) in 2008. The present work was achieved as part of these studies during a one year internship at the LORIA (Laboratoire Lorrain de Recherche en Informatique et ses Applications) in Nancy, France. Previously, he obtained a University Diploma of Technology in telecommunications and networks at the Université Henry Poincaré, Nancy 1, in 1998 and a CNAM DEST Conception and development in 2004. Since 2001, he works for FELIX Informatique (http://www.felix.fr) on various industrial software (metrology and industrial weighing). He is interested by parallel and distributed processing and emulation.

Jens Gustedt obtained a PhD in mathematics from the Technical University Berlin, Germany, in 1992. Starting from the algorithmic aspects of Discrete Mathematics, his research interests expanded towards efficient algorithms in general and data structures. Since he entered the French National Institute for Computer Science and Control (INRIA) as a senior researcher in 1998 he is mainly involved in research on parallel algorithms and Grid computing with a strong emphasis on experimental validation. Today, he serves as Editor-in-Chief of the journal *Discrete Mathematics and Theoretical Computer Science* and heads the INRIA project team *AlGorille*, Algorithms for the Grid, http://www.loria.fr/equipes/algorille/.

Emmanuel Jeannot is a full-time researcher at INRIA (The French National Institute for Computer Science and Control) and is doing his research at the LORIA (Laboratoire Lorrain de Recherche en Informatique et ses Applications) in Nancy, France. From sept. 1999 to sept. 2005 he was associate professor at the Université Henry Poincaré, Nancy 1. He got his PhD and Master degree of computer science in 1996 and 1999, respectively, both from École Normale Supérieure de Lyon. His main research interests are scheduling for heterogeneous environments and grids, data redistribution, grid computing software, adaptive online compression, programming models, experimental computer science and managing uncertainties in large-scale environment. More information is available at http://www.loria.fr/~ejeannot