INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Multi-Site Emulation using Wrekavoc: Validating Distributed Algorithms and Applications

Olivier Dubuisson
FELIX Informatique

Jens Gustedt — Emmanuel Jeannot
INRIA Nancy – Grand Est

N° 6696

Octobre 2008

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--6696--FR+ENG

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche NANCY - GRAND EST

Multi-Site Emulation using Wrekavoc: Validating Distributed Algorithms and Applications

Olivier Dubuisson
FELIX Informatique

Jens Gustedt — Emmanuel Jeannot
INRIA Nancy – Grand Est

Thème NUM — Systèmes numériques
Équipe-Projet AlGorille

Rapport de recherche n° 6696 — Octobre 2008 — 27 pages

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 an unmodified applications on emulated multisite heterogeneous platforms. 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 is done between a real heterogeneous platform 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.

Key-words: Tool for experimentation; performance modeling; emulation; heterogeneous systems

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Émulation multi-site en utilisant Wrekavoc : validation d'algorithmes et d'applications distribuées

Résumé : La validation et le test expérimental de solutions conçues pour des environnements hétérogènes est un problème difficile. Wrekavoc est un outil pour accomplir une telle validation. Il permet d'exécuter une application non modifiée sur une plate-forme hétérogène et multi-site émulée. Il dégrade les performances des nœuds (CPU et mémoire) et le réseau d'interconnexion comme spécifié par l'utilisateur. Nous reportons de nouvelles stratégies pour améliorer la précision des modèles réseau et mémoire. Puis, nous présentons une validation expérimentale de l'outil en comparant les exécutions de divers codes d'applications. La comparaison est faite entre une plate-forme hétérogène réelle et l'émulation de cette plate-forme avec Wrekavoc. Les mesures montrent que notre approche permet, dans l'émulateur, une reproduction proche des mesures réelles.

Mots-clés : Outil pour l'expérimentation ; modélisation de performance ; émulation ; système hétérogènes

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]. *Real-scale* (also called *in situ*) experiments consist in executing and benchmarking the real solution on a real platform. The goal is to have as few experimental bias as possible. This is ensured by observing the interactions at every level of the execution stack (from the hardware to the system and the application). The drawbacks of this methodology are that performing an experiment at real-scale is labor-intensive and that the environment on which the solution is tested is not fully controllable.

Simulations allow to overcome these drawbacks by executing a model of the application on a model of the environment. Therefore, this methodology ensures

reproducibility and control at a low cost in terms of development and execution time. The main drawback is that it has more experimental bias: for instance, it is very hard to simulate low level interaction such as cache effects, process scheduling, etc.

Finally, *emulation* is an intermediate approach between real-scale and simulation. Emulation consists in executing a real application on a model of the target platform. Hence, as for simulation, it ensures a good reproducibility and control. The experimental cost is higher than in simulation but lower than for real-scale. Moreover, since a real application is executed it is expected to have less experimental bias than simulation. Being situated between real-scale experiments and simulation, emulation also provides another gain. It allows for a validation of parts of the models that are needed in simulation, in particular the modeling of the compute platform.

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 program can be executed to test and compare it with other solutions. Building such a tool is a scientific challenge: it requires to establish links between reality and models. Such models need to be validated in order to understand their limits and to asses their realism. However, a brief look at the literature shows that concerning simulators [3, 4] or emulators [5, 6], the validation of the proposed tools as a whole is seldomly addressed. For instance, the SimGrid validation was done by comparing simulation and analytical results on a tractable scheduling problem.

The contribution of this paper is the following. We present the extension of Wrekavoc with respect to the previous version described in [2]. The goal of these extensions is mainly to provide a better realism of the tool concerning network and memory emulation. Another goal is to ease its usability. The second contribution of this paper is to demonstrate how Wrekavoc can help in experimentally validating distributed algorithms and applications. In order to do that we have compared the execution of different applications on a heterogeneous setting using many different parallel programming paradigms with the execution of the same application on a homogeneous cluster and Wrekavoc. The different paradigms we have tested are: fine grain computation without load balancing, master-worker, static load balancing, dynamic load balancing and work stealing.

The organization of this paper is the following. First, related work of existing experimental tools is presented in Section 2. Then, in Section 3 we present Wrekavoc. The new features are described in Section 4. The experimental validation of Wrekavoc is demonstrated in Section 5. Finally, we conclude and give some perspectives in Section 6

2 Related work

In the following we will briefly review the existing tools in the three different approaches described above. We will see that none of them allows to execute an unrestricted and unmodified application under precise and reproducible experimental conditions that would correspond to a heterogeneous environment.

2.1 Real-scale Experimental Testbeds

GRID5000 [7] is a national French initiative to acquire and interconnect clusters on 9 different sites into a large testbed. It allows for experiments on all levels from the network protocols to the applications. This testbed includes Grid Explorer [8], a designated scientific instrument with more than 500 processors. *Das-3 [9]* The Distributed ASCI Supercomputer 3 is a Dutch testbed that links together 5 clusters at 5 different sites. Its goal is to provide infrastructure for research in distributed and grid computing. GRID5000 and Das 3 have very similar goals and collaborate closely. They are connected by a dedicated network link. *Planet-lab [10]* is a globally distributed platform of about 500 nodes, all completely virtualized. It allows to deploy services on a planetary scale. Unfortunately, its dynamic architecture makes the controlled reproduction of experiments difficult. These platforms allow to test any type of application. Nevertheless, each platform by itself is quite homogeneous and thus the control and the extrapolation of experimental observations to real distributed production environments is often quite limited. In addition, the management of experimental campaigns is still a tedious and time-consuming task.

2.2 Simulators

Bricks [11], SimGrid [3] and GridSim [4] are simulators that allow for the experimentation of distributed algorithms (in particular scheduling) and the impact of platforms and their topology. Generally they use interfaces that are specific to the

simulator to specify an algorithm that is to be investigated. *GridNet [12, 13]* is specialized on data replication strategies. Others, focused on network simulations are NS2 [14], OPNetModeler [15] and OMNet++ [16].

A general disadvantage of all these tools is that there are only few studies concerning their realism.

2.3 Emulators

Microgrid [5] allows an execution of unmodified applications that are written for the Globus toolkit. Its main technique is to intercept major system calls such as `gethostbyname`, `bind`, `send`, `receive` of the application. Thereby the performance can be degraded to emulate a heterogeneous platform. This technique is invasive and limited to applications that are integrated into Globus. The measured resource utilization seems to be relatively inaccurate, in particular due to the used round-robin scheduler. Moreover and unfortunately, Microgrid is not supported anymore and does not work with the recent gcc 4 version. *eWAN [17]* 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 and does thus not permit to perform benchmarks for an application as a whole. *ModelNet [6]* also is a tool that is principally designed to emulate the network component. Virtual machine technology allows several guest to be executed on the same physical architecture. Hence, virtualization does not generate heterogeneity by itself but is clearly a complementary approach to the one proposed here. The RAMP (Research Accelerator for Multiple Processors) project [18] aims at emulating low level characteristics of an architecture (cache, memory bus, etc.) using FPGA. Even, if in this paper we show that we are already able to correctly emulate this features at the application level, such a project is complementary to this one and could be use to further improve the realism of Wrekavoc.

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: given a homogeneous cluster Wrekavoc degrades the performance of nodes and network links independently

in order to build a new heterogeneous cluster. Then, any application can be run on this new cluster without modifications.

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 [7, 1] or Planet Lab [10]) 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, possible configurations are numerous. The only restriction being 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 detail 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

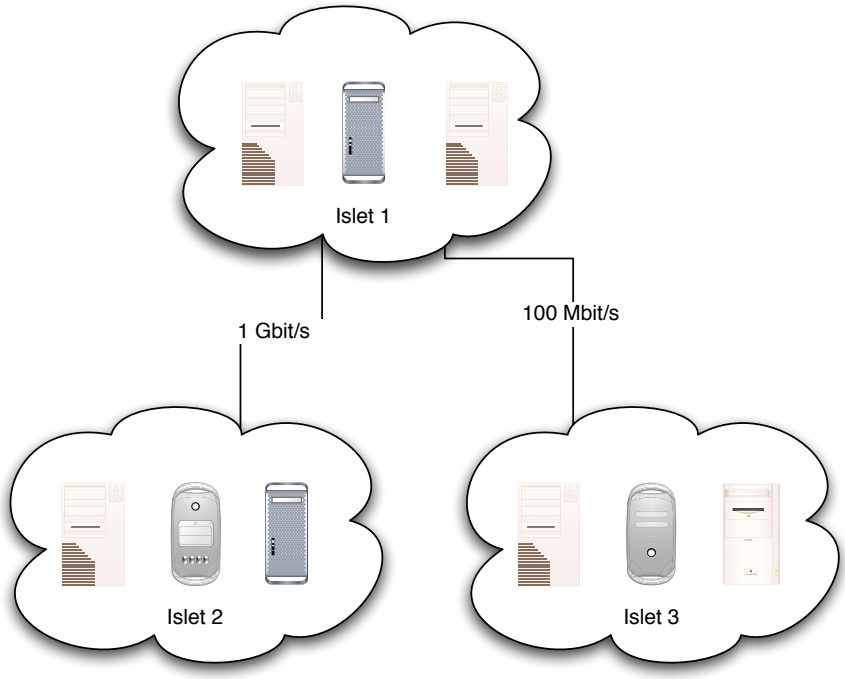


Figure 1: Islets logical view

Wrekavoc's notion to break up a homogeneous cluster 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 (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 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 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

The first version of Wrekavoc [2] matched some of the design goals presented above.

However, 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.

Another set of issues concerned the ease of use of Wrekavoc that should allow to configure an environment in an efficient and simple way.

We now show how we have addressed these different points.

4.1 Real memory degradation

As said above, 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 $M - S$ physical pages to memory (where M is the size of the physical memory). These pages are then inaccessible to the application and thus constrain the physical

memory that is available for it to S. Therefore, if the sum of memory allocated by application processes exceeds S the system will start swapping, just as it would on the real platform.

4.2 Adding gateways for improved network regulation

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. A consequence of adding a gateway to each islets is that it uses a node that cannot be used for computation. However, this allows complex topologies between islets where some islets are directly connected and some are not.

If two nodes in different islets that are not directly connected (such as nodes in Islet 2 and Islet 3 in Fig. 1) need to communicate, packets are forwarded from islet to islet. For this purpose, we have implemented a routing protocol[1]. In Fig. 1, the gateway of Islet 1 forwards packets that go from Islet 2 to Islet 3.

4.3 Enhanced configuration

When a user defines an islet, he/she has to give the IP addresses of the machines that compose this islet. These IP addresses are stored in the configuration file that is then read by the Wrekavoc client. However, if the user wants to use this configuration file again, nothing guarantees that he/she will have the same usable nodes. For instance, on the Grid'5000 platform the user reserves nodes using the OAR batch scheduler, and it is possible that from one experiment to another not all the same nodes are available. Usually this would imply that the user would have to edit the configuration file and change the IP address of the islet's nodes

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

according to the nodes of the cluster he/she has been given. In order to avoid this painful and error-prone process of editing the configuration file, we have added the possibilities of defining islets using name aliases or by simply giving the number of nodes of each islet. In this case, the user provides the configuration file and the list of nodes he/she wants to configure. The Wrekavoc client then maps the list of nodes with the configuration file to setup each islet accordingly. When the user comes back with a new set of nodes he/she can still use the same configuration file and just have to provide the new list of usable machines.

Building the configuration file can be troublesome at first. It requires to define all the characteristics of all the islets while respecting the syntax of the file. Moreover, understanding the interaction between islets is not easy when just reading the file. In order to ease the definition of islets and their inter-relationship we have designed a graphical user interface. With this interface a user can define an islet topology and the characteristics of the whole emulated environment as shown in Fig. 2.

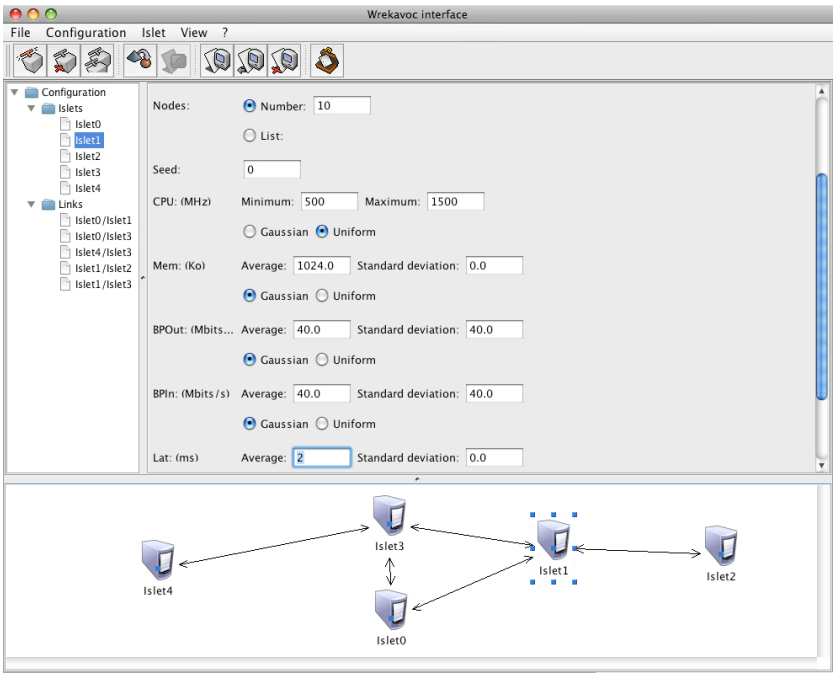


Figure 2: Wrekavoc's graphical interface for configuring the environments

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.

Usually, 3 levels of realism are considered:

1. An experiment is said *qualitatively* correct if it respects the *ordering* of measurements. For instance, if an experimental comparison implies that solution A_1 is better than solution A_2, then this should be true for the real setting.

2. An experiment is said *quantitatively* correct if it is able to tell *how much* something is better/worst than something else. For instance, if an experimental comparison says that A_1 is k times better than solution A_2, then it should be the same in the reality.

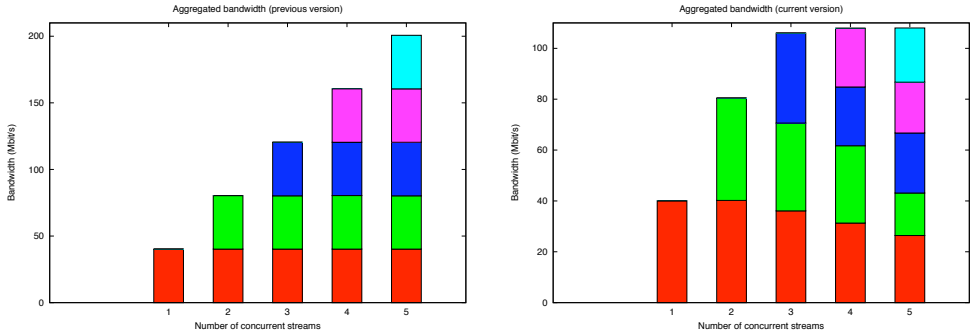
3. An experiment is said *predictive* if it is able to predict the *absolute value* of the behavior. For instance, if an experiments says that solution A will run in k seconds it should be the same in the reality.

Here we assess the realism of Wrekavoc by performing micro-benchmark tests for measuring the inter-islet network regulation (intra-islet benchmarking has been done in [2]) and by comparing the behavior of real applications on Wrekavoc and on real heterogeneous cluster.

5.2 Inter-islet network

We have performed several tests to validate inter-islet communication using benchmarks.

The first set of tests, consists in measuring the bandwidth that is observed for network streams between two nodes of different islets. The expected behavior is



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

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

Figure 3: Inter-islet aggregated bandwidth when adding streams (shown in different colors). Node bandwidth: 40 Mbit/s, Inter-islet link: 100 Mbit/s.

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. 3, 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. Bandwidth measurements are done using iPerf [19].

We see the difference between the previous version of Wrekavoc. In Fig. 3(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. 3(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.

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.

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

Table 1: Description of the heterogeneous environment. All nodes were running a Debian distribution with Linux kernel 2.6.18 for the corresponding system (686 or amd64).

5.3.1 Description of the heterogeneous cluster

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 (between 128 MiB and 2 GiB), MIPS (given by the *bogomips* feature of `/proc`, between 596 and 5590) and clock frequency. Concerning the network card we have two types of Ethernet cards (100 Mbit/s and 1 Git/s). Moreover and very importantly we are using different kind of processors (Pentium II, Pentium III and Pentium IV as well as Opteron) that have different architecture, memory systems, instruction sets, cache size, etc.

The goal of the following experiment is to see if, at the application level, Wrekavoc is able to realistically emulate all these features.

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

5.3.2 Wrekavoc on Grid'5000

We have compared the execution of different applications on the heterogeneous cluster described above and on clusters *heterogeneized* with Wrekavoc. We have used the clusters of Grid'5000 [7]. For a given applications we have used the same cluster for performing all the tests. However, due to problem of availability, we have not used the same cluster for all the different applications.

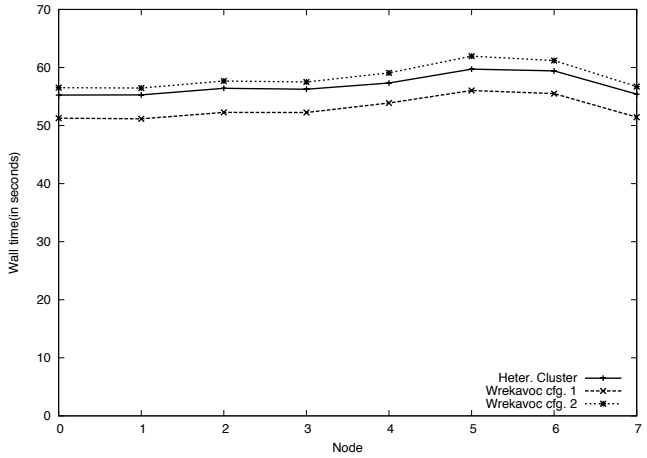


Figure 4: Execution walltime for the sort application of each node of the heterogeneous cluster with 20,000,000 `doubles`. Average of 10 runs.

5.3.3 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.

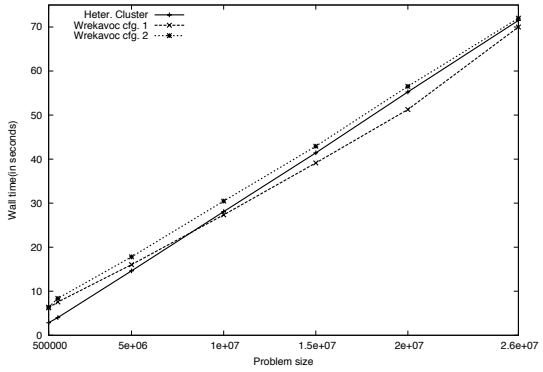


Figure 5: Execution walltime for the sort application of one node (machine with ID 2) of the heterogeneous cluster with varying problem size. Average of 10 runs.

The application we used is a parallel sort algorithm implemented within the parXXL [20] library. The algorithm used is based on Gerbessiotis' and Valiant's sample sort algorithm [21].

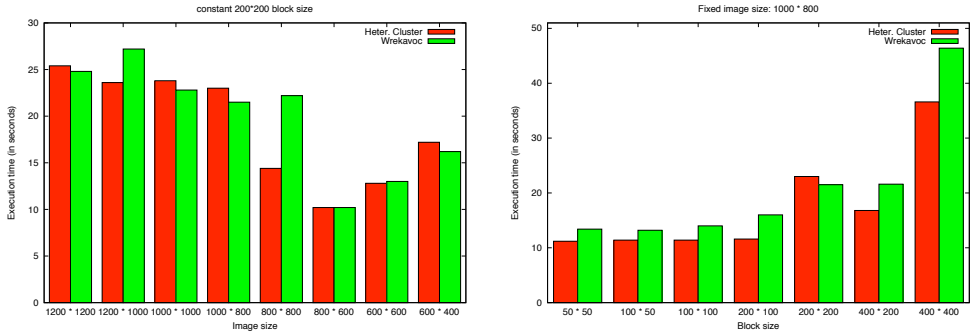
Here, we have found two realistic configurations and compared their behavior. Fig. 4 shows the wall-time of each node for the heterogeneous cluster and the two configurations. In order to see the difference we performed 5 sorts of 20,000,000 `doubles` (64 bit) in a row.

Results show that Wrekavoc is able to reproduce the reality with a reasonable accuracy as both configurations are close upper (or under) approximation.

In Fig. 5, 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%.

5.3.4 Master-worker

The second sets of experiments concerns the master-worker paradigm. We have a master that holds some data and a set of workers that are able to process the data. When a worker is idle it asks the master for some data to process, performs the computation and sends the results back. Such paradigm allows for a dynamic load balancing. As an application of this paradigm, we have chosen parallel image rendering with the Povray [22] ray-tracer. Here, the master holds a synthetic



(a) block size of 200×200 and different image sizes

(b) image size 1000×800 with different block sizes

Figure 6: Parallel rendering times of the master-worker application

description to a scene. This scene is decomposed into blocks such that each part can be processed by a worker.

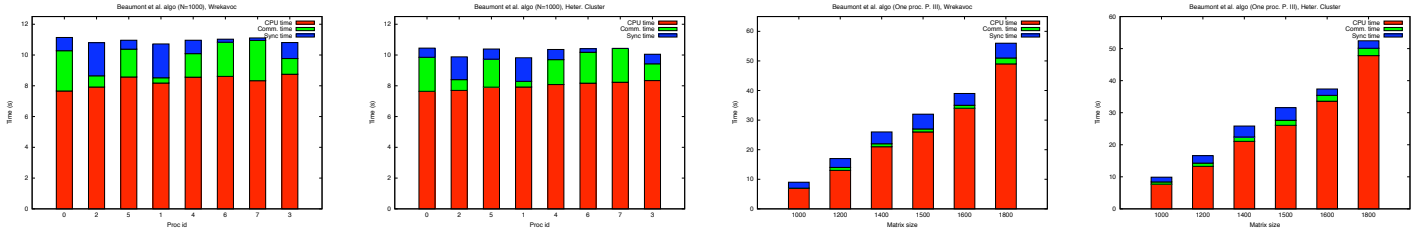
In Fig. 6(a) (resp. 6(b)), we present the comparison between the running time of the application when the block size is fixed (resp. the image size is fixed) and the image size varies (resp. the block size varies).

We see that most of the time Wrekavoc is able to match reality. Most of the results are within an error margin of 10%. The worst case is shown in Fig. 6(a) for image size of 800 with an error of 35%. However, it is able to guess that, for this block size, the smallest processing time is for (800×600).

5.3.5 Static load balancing

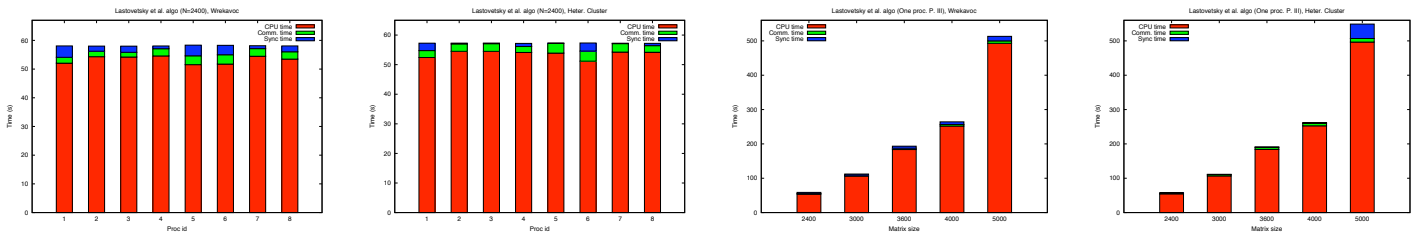
Here we have tested algorithms and applications that perform a static load balancing. In this case the load balancing is fixed at the beginning of the application according to the speed of the processors and the amount of work to perform. We have implemented two algorithms that perform parallel matrix multiplication on heterogeneous environments. The first algorithm from Beaumont *et al.* [23] is based on a geometric partition of the columns on the processors. The second from Lastovetsky *et al.* [24] uses a data partitioning based on a performance model of the environment.

In Fig. 7(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



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

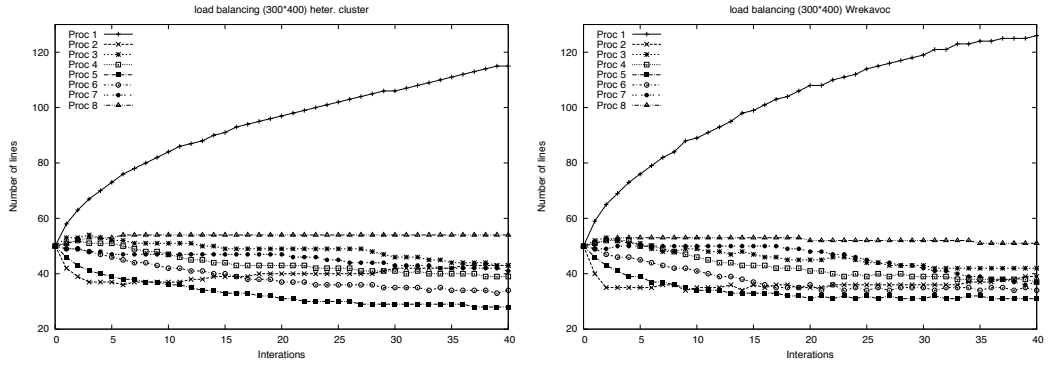
(b) Beaumont *et al.* algorithm for the same processor



(c) Lastovetsky *et al.* algorithm, N=2350

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

Figure 7: Comparison of node runtime (CPU, communication and synchronization) for the static load balancing application



(a) heterogeneous cluster

(b) emulation with Wrekavoc

Figure 8: Comparison of the evolution of the load-balancing for executing the advection diffusion application

to the behavior when using the heterogeneous cluster. We see a little shift in total running time due to the difficulty to calibrate Wrekavoc precisely, but the proportions of the timings is conserved.

In Fig. 7(b), we show the comparison between the CPU, communication and synchronization time for the Beaumont *et al.* algorithm for varying matrix size and a fixed node (Pentium III – ID 4). 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.

In Fig. 7(c) and 7(d), we present the same measurements as in Fig. 7(a) and 7(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.

5.3.6 Dynamic load balancing for iterative computation

The Dynamic load balancing strategy we have investigated here, consists in exchanging some workload at execution time in function of the progress in the previous iteration. The goal is to see if the load balancing is done the same way on the heterogeneous cluster and when using Wrekavoc. The program we have used solves an advection-diffusion problem (kinetic chemistry) described in [25]. In this program the load balancing is automatic. The 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. The results show that the evolution of the load balancing using Wrekavoc (right) or the heterogeneous cluster (left) at 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 summary, the predictability of the behavior of the emulated application is very high.

5.3.7 Work stealing

Here we test the behavior of different work-stealing algorithms. Within this paradigm, an underloaded node chooses another node to ask for some work. Our chosen application for this methodology is the N-queens problem. This problem consists in

Algo 1 : c1	d1	g1(random load stealing, load evenly distributed, granularity 14)			
Algo 2 : c2	d1	g1	**Algo3**: c1	d2	g1
Algo 4 : c1	d3	g1	**Algo5**: c1	d4	g1
Algo 6 : c1	d5	g1	**Algo7**: c1	d1	g2
Algo 8 : c1	d1	g3	**Algo9**: c1	d1	g4
Algo10: c1	d1	g5			

Table 2: The 10 work stealing strategies

placing N queens on a check board of size $N \times N$, such that no queen is blocked by any other one. The goal of the application is to find all the possible solutions for a given N. This problem is irregular: two partial solutions restricted to the same number of rows can have very different number of solutions.

When a node is underloaded different strategies can be applied to choose the node to steal some work.

Strategy c1: chose node at random; **c2**: chose one of two designated neighbors (nodes are arranged according to a virtual ring). The way the load is distributed initially may also have an impact. Here, we use 5 different strategies. **distribution d1**: The load is evenly distributed; **d2**: Place all the load on the fastest node; **d3**: Place all the load on the slowest node; **d4**: Place all the load on an average speed node; **d5**: Place all the load on the second slowest node. When solving the N-queens problem the load is composed of tasks that have a given granularity (*i.e.* the number of rows of the board to be processed sequentially). We have studied different granularities. **Granularity g1:** 14 rows, **g2:** 12 rows, **g3:** 13 rows, **g4:** 15 rows and **g5:** 16 rows.

With these different characteristics out of the 50 possible combinations we have chosen 10 different algorithms[2], as shown in 2.

In Fig. 9, we present the results for the mono-threaded implementation of the 10 algorithms. We see that Wrekavoc is able to reproduce the behavior of the heterogeneous cluster precisely. Timings are almost the same in both situations. The only problematic case is for Algorithm 10. In this case, the granularity is very high and therefore, there are only few tasks to execute. Therefore, in this case, the load balancing is very hard to reproduce.

[2]We favored Strategy c1 since it is known to be the best for this application [26]

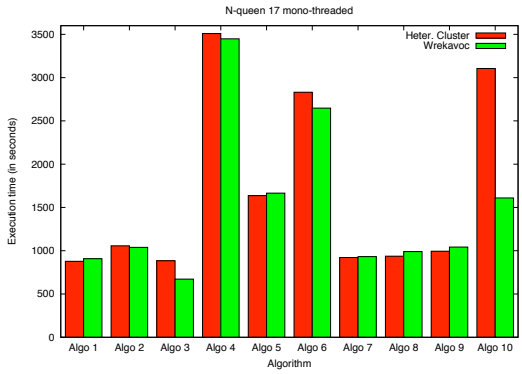


Figure 9: Computation time of the different algorithms for the N-queens problem of size 17, using the heterogeneous cluster or Wrekavoc, kernel 2.6.18

We have implemented a multi-threaded version of the algorithm; one thread for the computation and one thread for the communication. We have tested Wrekavoc with two kernel versions 2.6.18 and 2.6.23. The difference between these two version is a change in the process scheduler. The 2.6.18 version uses the $O(1)$ *scheduler*, the 2.6.23 version of the kernel implements the so-called *Completely Fair Scheduling* (CFS), based on *fair queuing* [27]. Both schedulers are developments of Ingo Molnár.

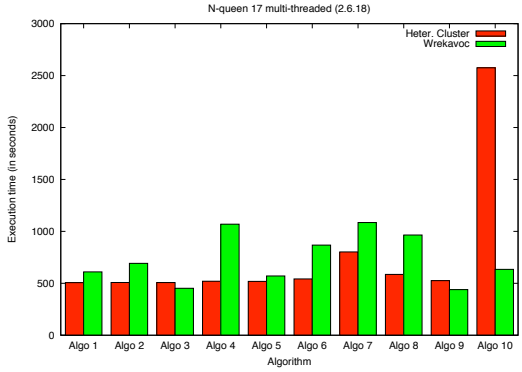


Figure 10: Computation time of the different algorithms for the N-queen problem of size 17, using the heterogeneous cluster or Wrekavoc, multi-threaded version, kernel 2.6.18

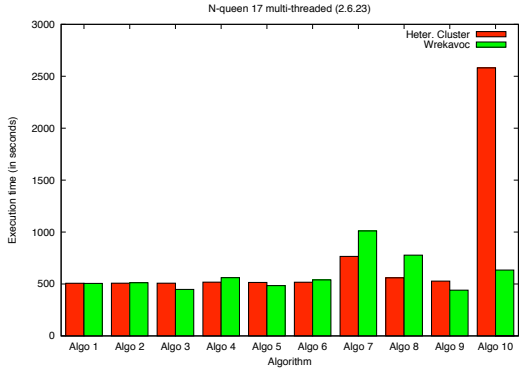
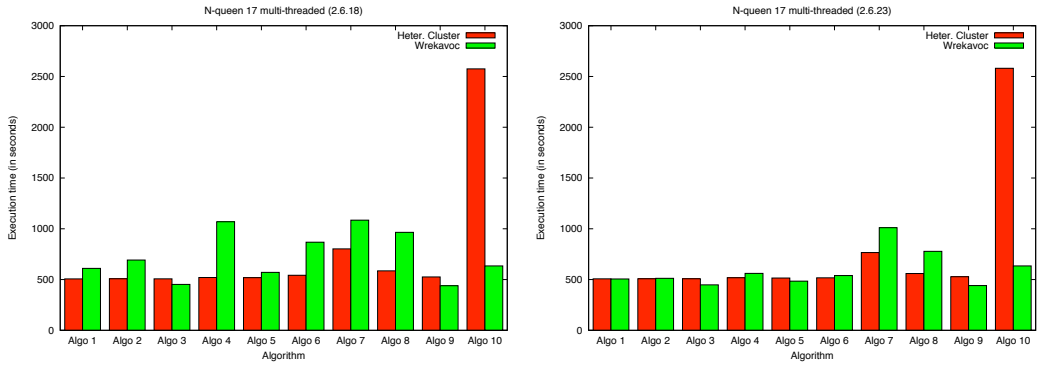


Figure 11: Computation time of the different algorithms for the N-queens problem of size 17, using the heterogeneous cluster or Wrekavoc, multi-threaded version, kernel 2.6.23



(a) With kernel 2.6.18 (b) With kernel 2.6.23

Figure 12: Computation time of the different algorithms for the N-queens problem of size 17, using the heterogeneous cluster or Wrekavoc, multi-threaded version, kernel 2.6.18 and 2.6.23

We present the results for both kernel versions in Fig. 12. We first see that the multi-threaded version is faster than the mono-threaded (especially when the load is initially placed on slow processors (Algorithms 4, 5 and 6). We see that Wrekavoc is able to reproduce this acceleration from the single threaded version to the multi-threaded. Moreover, we see that using kernel 2.6.23 improves the accuracy of Wrekavoc when experimenting a multi-threaded program. This holds

especially for algorithm using a granularity of 14 rows (Algo 1 to 6). A higher granularity reduces the realism because the number of task to execute lowers when the granularity increases and hence makes it difficult to reproduce the behavior.

We explain the fact that the realism of Wrekavoc is better with kernel 2.6.23 than with kernel 2.6.18 due to the difference of the process scheduler implemented in this two versions. The $O(1)$ scheduler tends to favor I/O restricted tasks. When Wrekavoc suspends a process for controlling the CPU speed and wakes it up, the scheduler increases the task priority (taking it for a I/O task). Therefore, both threads (communication and CPU) of the process have acquired a high priority which favors I/O in our case: the process spends most of its available time to perform the communications. Therefore, the behavior is extremely unrealistic when all the load is on a slow processor at the beginning. The CFS scheduling algorithm solves this problem giving extra priority to suspended tasks such as I/O ones. In conclusion this shows that emulation of multi-threaded program is realistic if running Wrekavoc on a recent kernel version (at least 2.6.23).

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 propose and present Wrekavoc, a tool for performing experimental validation, benchmarks and comparison of solutions designed for distributed and heterogeneous environments.

Wrekavoc uses a homogeneous cluster to define and control a multi-site heterogeneous environment. Therefore, it helps to validate parallel and distributed applications and algorithms. Moreover, on the modeling side, it helps to understand the impact of platform parameters (latency, bandwidth, CPU speed, memory) on application performance.

We have validated Wrekavoc using micro-benchmarks (for inter-site communication) and by comparing the execution of several real applications on a real environments to a cluster running Wrekavoc. 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 level, At the application/user level, different architectural features (*e.g.*processor architecture, memory bandwidth, cache, instruction sets, etc.) are correctly emulated though it could certainly be improved.

Furthermore, despite the fact that the experiment has been done on an average-scale environment we are confident that the same results would have been obtain on larger setting (unfortunately, to the best of our knowledge, such large-scale environment, with reproducibility of the experimental condition does not exist).

Future work is directed towards the ease of the calibration of the environment, a better emulation of multi-threaded programs and to add new modeling features such as node volatility and dynamic load.

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, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see [7]).

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] A. Legrand, L. Marchal, and H. Casanova, Scheduling distributed applications: the simgrid simulation framework, *CCGRID '03*, pp. 138–145 (2003).

[4] 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).

[5] 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).

[6] 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).

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

[8] Grid explorer (GdX), URL `http://www.lri.fr/~fci/GdX/`.

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

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

[11] 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).

[12] 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).

[13] 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).

[14] The network simulator NS2, URL `http://www.isi.edu/nsnam/ns`.

[15] J. Prokkola, Opnet - network simulator, URL `www.telecomlab.oulu.fi/kurssit/521365A_tietoliikennetekniikan_simuloinnit_ja_tyokalut/Opnet_esittely_07.pdf` (2007).

[16] A. Varga, The omnet++ discrete event simulation system, URL `http://www.omnetpp.org/download/docs/papers/esm2001-meth48.pdf` (2001).

[17] 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).

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

[19] Iperf - the tcp/udp bandwidth measurement tool, URL `http://dast.nlanr.net/Projects/Iperf/`.

[20] parXXL: A fine grained development environment on coarse grained architectures, URL `http://parxxl.gforge.inria.fr/`.

[21] 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).

[22] The persistence of vision raytracer, URL `http://www.povray.org/`.

[23] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, Matrix multiplication on heterogeneous platforms, IEEE Trans. Parallel Distributed Systems, **12**(10), 1033–1051 (2001).

[24] 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).

[25] F. Vernier, *Algorithmique itérative pour l'équilibrage de charge dans les réseaux dynamiques*, Ph.D. thesis, univ. de Franche-Conté (2004).

[26] R. D. Blumofe and C. E. Leiserson, Scheduling multithreaded computations by work stealing, J. ACM, **46**(5), 720–748 (1999).

[27] J. Nagle, On packet switches with infinite storage, IETF RFC 970, URL `http://www.rfc-archive.org/getrfc.php?rfc=970` (1985).

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399