Wrekavoc: a Tool for Emulating Heterogeneity

Louis-Claude Canon[1]   Emmanuel Jeannot[2]

[1]ESEO
4 rue Merlet de la Boulaye
BP 30926
49009 Angers cedex 01, France
louis-claude.canon@eseo.fr

[2]Loria INRIA-Lorraine
Campus scientifique
54506 Vandœuvre les Nancy
France
emmanuel.jeannot@loria.fr

Abstract

Computer science and especially heterogeneous distributed computing is an experimental science. Simulation, emulation, or in-situ implementation are complementary methodologies to conduct experiments in this context. In this paper we address the problem of defining and controlling the heterogeneity of a platform. We evaluate the proposed solution, called Wrekavoc, with micro-benchmark and by implementing algorithms of the literature.

1. Introduction

Research on algorithms for heterogeneous platforms is a very active domain. It encompasses the fields of scheduling [12], load balancing [1], linear algebra [8], data redistribution [3], etc. Unfortunately heterogeneity makes problems harder to solve. In few cases polynomial algorithms are found, sometimes only approximation algorithms are proposed while in many cases no theoretical results are available. In the later case an experimental approach can be used to test or compare heuristics. In large-scale distributed heterogeneous systems, numerous parameters and complex interactions between resources make models mostly intractable. Moreover, even if in some cases theoretical results are found, an experimental approach on a real platform can also help in validating both the modeling and the algorithm. Since experimental evaluation is mandatory in algorithmic (for heterogeneous platform) research, several complementary methodologies have been proposed.

Simulators focus on a certain part of the platform and abstract the remaining of the system. Simulations enable reproducible experiments and allow to test a large set of platforms and experimental conditions. Examples of simulators designed to test and compare scheduling algorithms in large-scale distributed systems are Bricks [16], GridSim [4] or Simgrid [10]. It is out of the scope of this paper to compare the relative merits of these simulators and the reader is referred to [15] for further details and other simulators. Surprisingly very few studies on the comparison between simulation and real experiments have been conducted. The validation of Bricks was performed by incorporating NWS [18] into Bricks. Then, they run an applications within Bricks and a real environment and compare the behavior of NWS under both environments. Simgrid validation was done by comparing simulation and analytical results on a tractable scheduling problem.

In some situations complex behaviors and interactions between the distributed resources cannot be simulated. This is due to the difficulty to capture and extract all the factors that play a role during the execution of a given application (such as OS specific features: for instance process scheduling, or hardware special capabilities: for instance hyperthreading, cache memory, multi-core processors or runtime performance: for instance the different versions or flavors of MPI). In-situ experiments solve this problem by running a real software on a realistic platform. Typically, experiments on real-life heterogeneous platforms are made using the available workstations of a laboratory. However, these machine are often shared with other users making reproducibility of experiments hard to achieve. Recently experimental dedicated platforms have been proposed to tackle this problem (Das-2 [5], Grid-explorer [7], Grid'5000 [6], or Planet-Lab [14]). However, the degree of heterogeneity of these platforms is very low and fixed. This makes the evaluation of algorithms designed for heterogeneous environment very hard to con-

1-4244-0054-6/06/$20.00 ©2006 IEEE

duct and results hard to extrapolate for other heterogeneous cases. In order to tackle this problem, Latsovetsky et al. have proposed in [9] a new approach that consists in comparing the efficiency of the heterogeneous solution of a problem with the homogeneous one. In this case, the homogeneous setting have the same aggregate performance of of the heterogeneous one. But, still the heterogeneity is not controllable.

Between simulation and real-life experiments stands emulation (*e.g.*, Microgrid [19]) which goal is to test real applications (as in real-life experiments) with less abstraction than with simulators. However, in Microgrid each program have to be linked against the Microgrid library that interprets system call leading to an increase of the execution time.

In this paper we address the problem of controlling the heterogeneity of a cluster. Our objective is to have a configurable environment that allows for reproducible experiments on large set of configurations using real applications with no emulation of the code. Given an homogeneous cluster, our proposed solution (called 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. This paper describes this tool, and its evaluation with micro-benchmark and by implementing algorithms for heterogeneous environments.

The remaining of this paper is organized as follow. In section 2 wrekavoc goals, model and implementation are presented. In section 3 describes how to define and control the heterogeneity of a cluster. Experimental results and validation of wrekavoc are given in Section 4. We compare our approach with other solution in section 5. Finally we conclude the paper section 6

2. Wrekavoc

2.1. Design Goals

Given a homogeneous cluster[1] we want to transform it into an heterogeneous one. We also want heterogeneity to be controlled and reproducible. One way of transforming an homogeneous cluster into an heterogeneous one is to update the hardware with more powerful devices (upgrading the CPU, adding some memory, etc.) . However, in this case, the heterogeneity is fixed and not controllable. An other way is to degrades its performances. This is the approach taken in this paper because it leads to a controllable layout. We target the degradation of the following characteristics:

- CPU power,
- network bandwidth,
- network latency and
- memory.

The degradation of each characteristic has to be independent (one can degrades CPU power without modifying network performance) and by software means (without modifying each node or rebooting the cluster). Lastly, we want to be able to configure a large cluster easily and rapidly.

2.2. Implementation

Our solution (called Wrekavoc) is implemented using the client-server model. A server, with administrator privilege, is deployed on each node one wants to configure and run as daemon. The client reads a configuration file and sends orders to each node in the configuration. The client can also order the nodes to recover the original state. The overall software stack is described in Fig. 1.

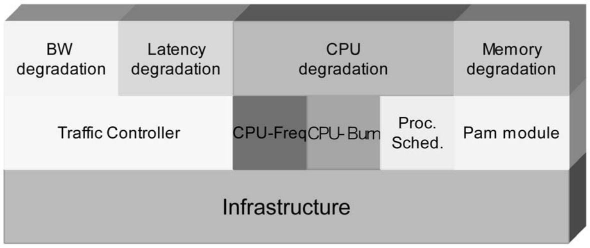

Figure 1. Software stack of Wrekavoc.

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. This interface was designed to limit CPU frequency in order to save the power on laptops. It is based on several CPU technologies (such as *powernow*) which are not always available on cluster nodes. However, only 10 different frequencies are available through CPU-Freq. Therefore, if the required CPU technologies are not available on the nodes or if the discretization is too coarse 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

[1] As a first approach we target Unix (preferably Linux) clusters.

CPU, whatever the number of processes currently running. More precisely, a CPU-burn sets the scheduler to a FIFO policy and gives itself the maximum priority. It then compute the time it needs to make a small computation. This computation is blocking and therefore no other program can use the CPU. After the computation the CPU burner then sleeps for the corresponding amount of time. It then restarts the whole process. A small tuning time is needed to make sure sleeping and calculation time are longer than 5ms, in order to be executed by the scheduler. The system call used to set the scheduler is `sched_setscheduler`. Thanks to the Unix `sched_setaffinity` system call, each CPU burner is tight to a given processor on a multi-processor node. The main drawbacks of this approach is that the CPU limitation occurs for every processes whatever its mode (kernel or user) and therefore, the network bandwidth is limited by the same fraction than the CPU. When an independent limitation of the CPU and the network is required, we propose a third alternative based on user-level process scheduling called CPU-lim. A CPU limiter is a program that supervises processes of a given user. Using the `/proc` pseudo-filesystem, it suspends the processes when they have used more than the required fraction of the CPU using the `SIGSTOP` and `SIGCONT`. This alternative is used for the experiments of Section 4.

Network Limitation. Limiting latency and bandwidth is done using *tc* (traffic controller) [17] based on *Iproute2* a program that allows advanced IP routing. With this tools it is possible to control both incoming and outgoing traffic. Furthermore, the latest versions (above 2.6.8.1) allows to control the latency of the network interface. An important aspect of tc is that it can alter the traffic using numerous and complicated rules based on IP adresses, ports, etc. We use tc to define a network policy between each pair of nodes. It raises scalability issue as each nodes as to implement $n-1$ rules with a configuration with n nodes. This issue will be discussed in the experimental section. Degradation of network latency and bandwidth is implemented using Class Based Queueing (CBQ): incoming or outgoing packets are stored into a queue according to the given quality of service before being transmitted to the TCP/IP stack. In order to work a kernel version above 2.6.8 is required and needs to be compiled with the `CONFIG_NET_SCH_NETEM=m` option.

Memory Limitation. Wrekavoc is able to limit the largest malloc a user can make. This is possible through the security module PAM. However, we have not been able to limit the whole memory usable by all the processes yet.

3. Configuring and Controlling Nodes and Links

The configuration of a homogeneous cluster is made through the notion of islet. An islet is a set of nodes that share similar limitation. Two islets are linked together by a virtual network which can also be limited (see figure 2).

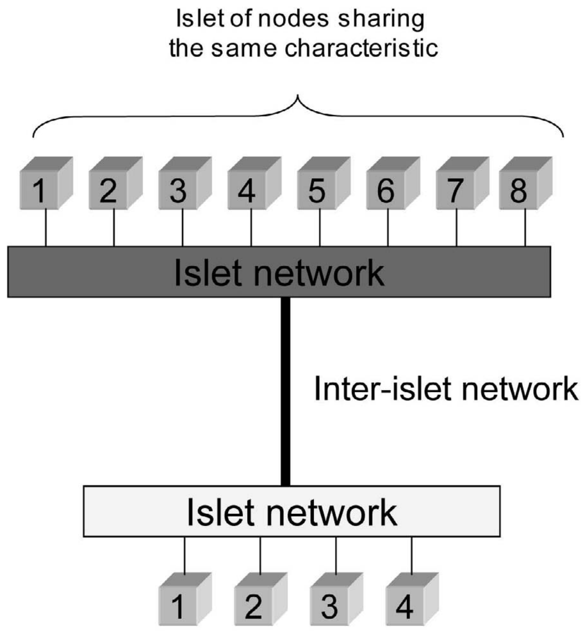

Figure 2. Islets logical view.

Defining an Islet. An islet is defined as a union of IP addresses (or machine names) intervals. The limitation parameters of each node of the islet take a value that can be defined in two ways. It can follow a Gaussian distribution[2] of the form [*mean*;*std. dev.*] or can follow a uniform distribution of the form [*min*-*max*]. For each islet we define several parameters. `SEED` is an integer that is used for the random distributions (-1 means a random seed). `CPU` is a distributed value of the CPU frequency in MHz of the noes of the islet. `BPOUT` (resp. `BPIN`) is a distributed value of the outgoing (resp. incoming) bandwidth in kB/s. `LAT` is the distributed value of network latency in ms. `USER` is the

[2] each node can have exactly the same value if we set 0 for the standard deviation.

```
islet1 :  [192.168.1.1-192.168.1.10] {
   SEED: 1
   CPU : [1000;0]
   BPOUT : [125000;0]
   BPIN : [125000;0]
   LAT : [0.05;0]
   USER : user1
   MEM : [80000;0]
}
islet2 :  [192.168.2.1-192.168.2.10]-[192.168.3.1-192.168.3.10] {
   SEED : -1
   CPU :[100-2000]
   BPOUT : [12500;0]
   BPIN : [12500;0]
   LAT : [0.05;0]
   USER : user1
   MEM :[80000;0]
}
!INTER : [islet1;islet2] [1250;0] [2500;0] [120;0] 1
```

Figure 3. Configuring two islets

name of the user for which the limitation are made. `MEM` is the distributed value of the maximum `malloc` in kB.

Linking Islets Together. Each islet configuration is stored into a configuration file. At the end of this file is described the network connection (bandwidth and latency) between each islet using the `!INTER` keyword. The last number of the `!INTER` line is the seed used for the random distribution.

Example. Figure 3 shows how to emulate two clusters within two islets. Of course, this configuration have to be executed on a cluster having at most the required performance. In this example, *Islet1*, is made of 10 nodes at 1 GHz with giga-ethernet interconnect (1Gb/s (125000 kB/s) bandwidth and 50 μs latency) and *Islet2* comprises 20 heterogeneous nodes with frequency between 100 MHz and 2GHz and a fast ethernet network (bandwidth: 100 Mbit/s, latency: 50 μs). The two clusters are linked by a network with a bandwidth of 10 Mbit/s and a latency of 120 ms from islet1 to islet2 and a bandwidth of 20 Mbit/s from islet2 to islet1. Remark that using -1 as seed for islet2 means that each time we configure the nodes we will get a different configuration of the nodes (reproducibility of the nodes configuration is done by using positive seeds).

4. Validation and Experimentation

All the experiments performed in this section were done using the Grid-explorer [7] cluster with 216 nodes. Each node has two 2 GHz AMD Opteron 246 with 2 GB of RAM. It runs under Linux Debian 3.1 with kernel 2.6.8.

4.1. Deployment Time.

The client reads the configuration file, parses the file and builds an XML file for each nodes. Then, it sends this sub-configuration to each node. When a node receives a configuration file it configures its own characteristics according to this file (see figure 4)

Figure 5 shows the configuration time against the number of nodes. 4 curves are shown: one islet, two islets, two nodes per islet and one node per islet. Results show that the configuration time increases with the number of nodes. The worst case occurs when we have one node per islet (the same number of islets as nodes). Even in this case configuring 130 nodes takes only 22 seconds, while with two nodes per islets it takes less than 10 seconds.

4.2. Micro-benchmark.

We have tested separately each kind of degradation using micro-benchmark.

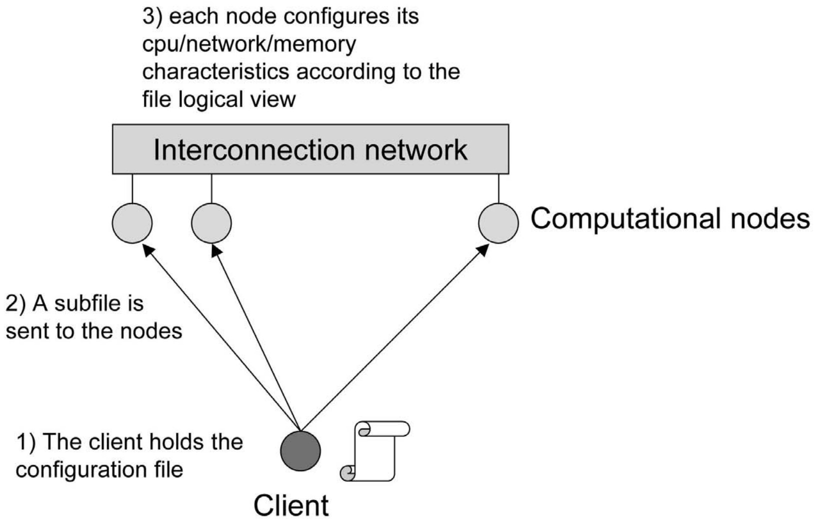
3) each node configures its cpu/network/memory characteristics according to the file logical view
Interconnection network
Computational nodes
2) A subfile is sent to the nodes
1) The client holds the configuration file
Client

Figure 4. Steps for configuring a cluster

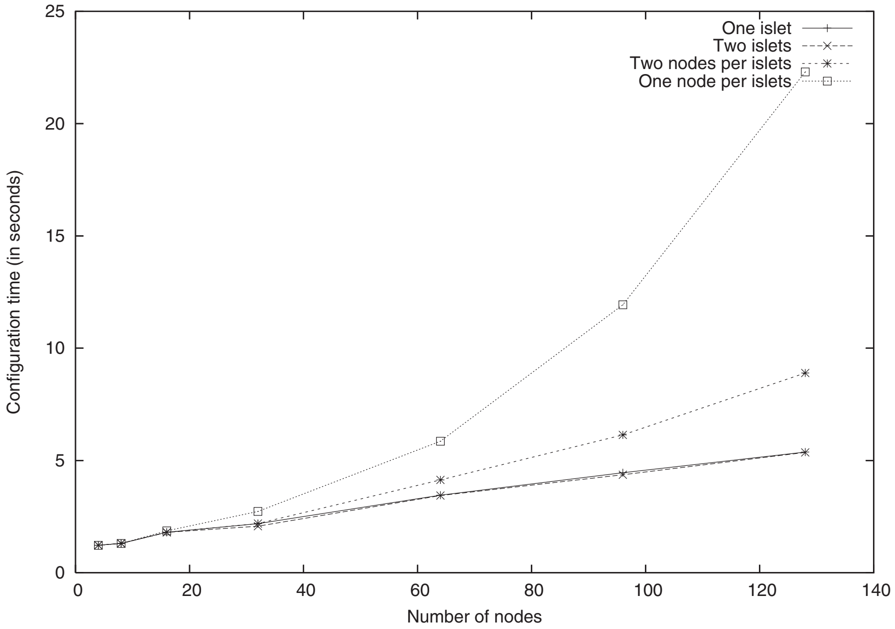

Figure 5. Configuration time for different islet sizes.

set latency	1	5	10	50	100	500	1000
RTT	2.12	10.05	20.12	100.058	200.20	1000.05	1999.75

Table 1. Round-trip-time against desired latency in ms.

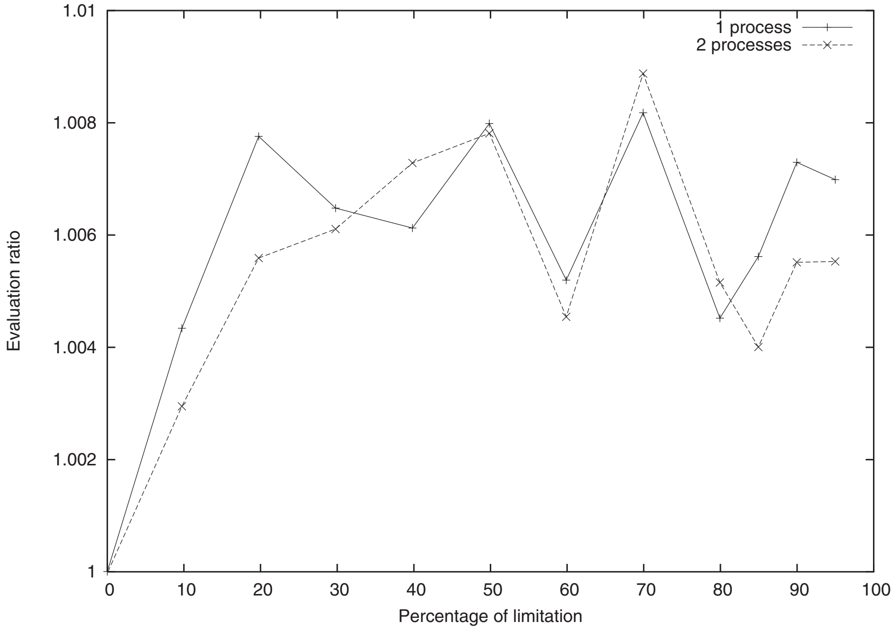

Figure 6. CPU micro-benchmark.

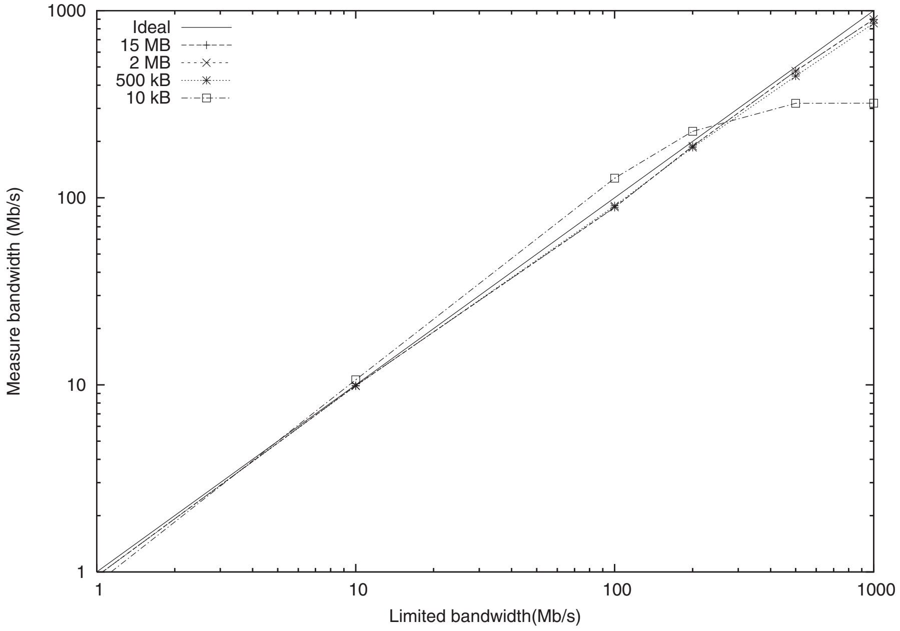

Figure 7. Bandwidth micro-benchmark.

CPU Tests. To measure how performance degradation impacts on the execution of a computation, we use the ratio between the expected and the actual duration times. The expected duration time is computed by applying the percentage of degradation to the time without degradation. When this ratio equals 1 this means that execution times matches the expected time. We can see on Figure 6 that the absolute value of this ra-

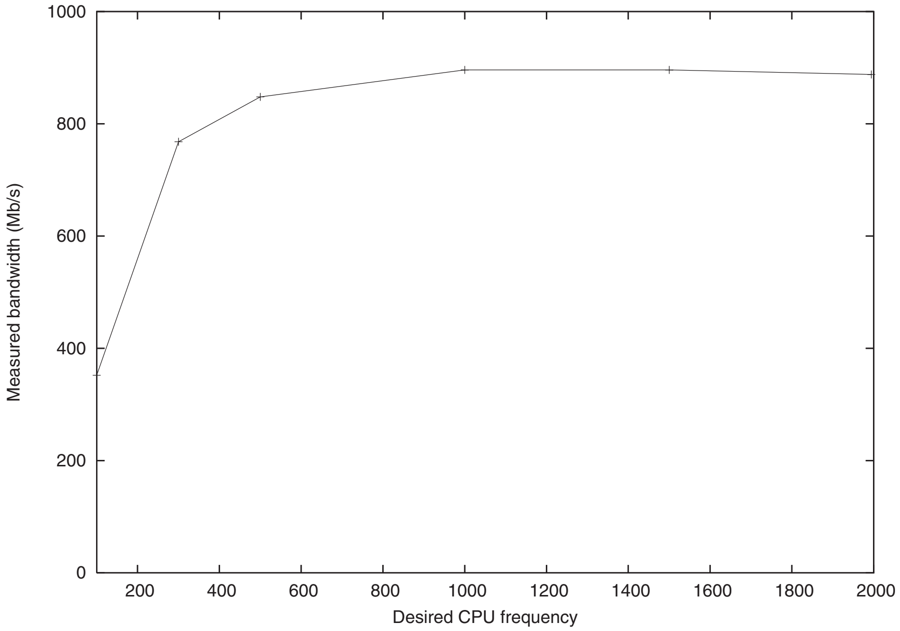

Figure 8. Testing the impact of CPU degradation against available bandwidth between two nodes linked by Gb Ethernet.

tio is never above 1.01 (i.e. less than 1% of difference). Therefore the CPU limitation behaves as expected and is able to handle bi-processors node.

Bandwidth Tests. Figure 7 shows the obtained bandwidth versus the desired bandwidth for different data size (between 15 MB and 10 kB). The ideal line shows what one should obtain theoretically. The results show that the obtained bandwidth is always very close to the desired one. We see that for 10 kB the we obtain a slightly greater bandwidth than the limited bandwidth. This is due to the fact that TC use some bucket to limit the bandwidth. The limitation starts when the bucket is completely filled. The amount of packets to fill the bucket being fixed, we see, for small messages that the real bandwidth is a little bit higher than the desired one. For this same size of data, we see that it is not possible to achieve the peak bandwidth. This is the same phenomena that happen in real network. Indeed, further investigations have shown that we obtain exactly the same bandwidth (320 Mbit/s) for 1 Gbit/s network card without network degradation for 10 kB messages.

Latency Tests. Table 1, shows the average round-trip-time (RTT) obtained by the *ping* command with various degraded latency. Results show that the RTT is exactly twice the value of the desired latency which is exactly what one should obtained theoretically as the latency is paid twice when doing a round trip.

4.3. Independent degradation

We have tested how the bandwidth degradation interfere with the CPU speed: we exchanged a file at different bandwidth speed while running a CPU intensive process. In this case, we have not seen any impact of the variation of the bandwidth with regards to the execution time.

Figure 8 show the obtained bandwidth against the CPU frequency on a Giga-ethernet network. Under 500 MHz the available bandwidth decreases with the CPU frequency. This is due to the fact that Gbit bandwidth is not achievable with slow CPU. In order to confirm that, we have exchanged data between two old PCs (PII at 400 MHz and PIII at 550 MHz) with a Gb PCI ethernet card under linux kernel 2.6.12 at runlevel 1. In this case the best achievable bandwidth was 340 Mb/s with UDP or TCP.

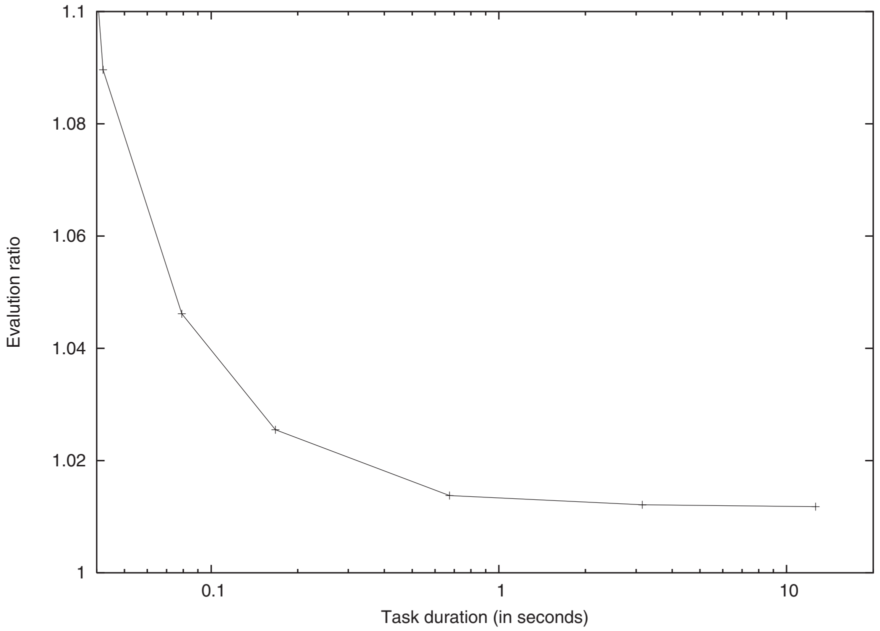

Figure 9. Testing a load-balancing algorithm.

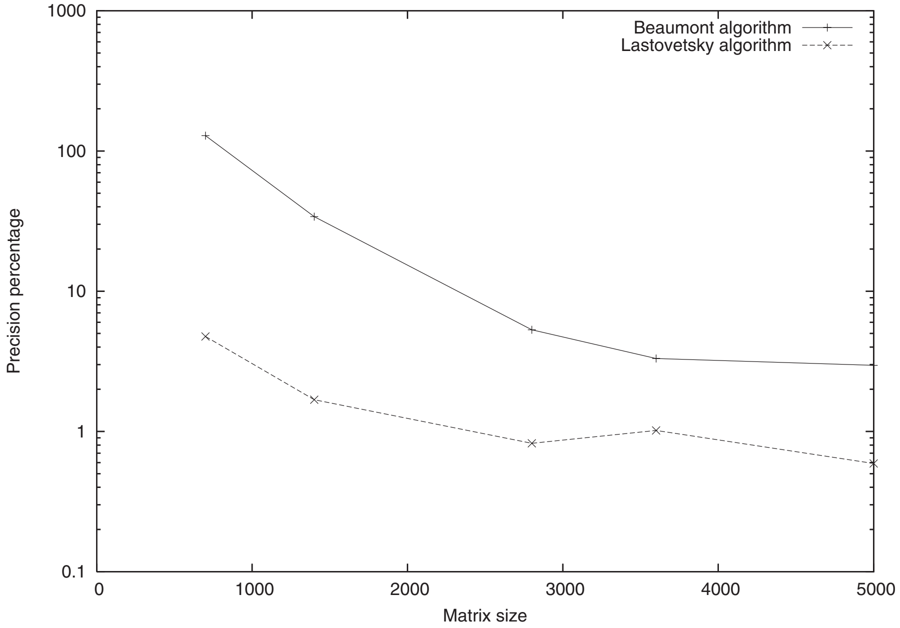

Figure 10. Testing two matrix multiplication algorithms.

4.4. Implementing Algorithms of the Literature.

We have tested Wrekavoc on several algorithms designed for heterogeneous environments.

First a static load-balancing algorithm of [11] page 160 and inspired from [13] is studied. We have chosen to balance a load composed of 500 tasks on 20 nodes when 5 nodes run at 100%, 5 at 75%, 5 at 50% and 5

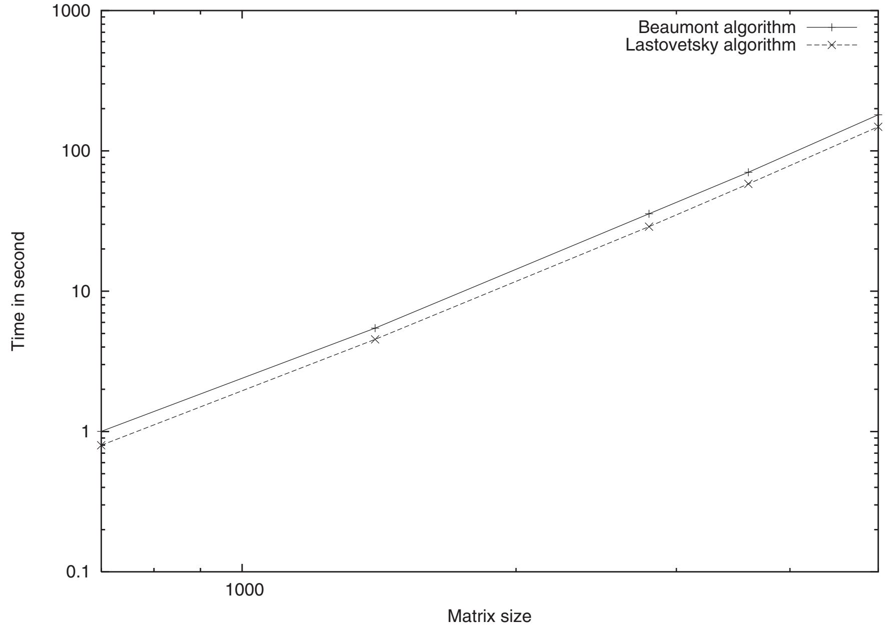

Figure 11. Comparing time of two matrix multiplication algorithms.

Methodology	Simulation (Simgrid)	Emulation (Microgrid)	In-Situ (hom. cluster)	L&R perf. analysis	Wrekavoc
Real app.	No	Yes	Yes	Yes	Yes
Abstraction	Very high	High	No	No	Low
Execution time	Speedup	Slowdown	Same	Same	Same
Proc. folding	Mandatory	Possible	No	No	No
Heterogeneity	Controllable	Controllable	No	Yes	Controllable

Table 2. Comparing different experimental methodologies

at 25%. For this simple case the algorithm balance the load inversely proportional to the speed of the machine (resp. 40 tasks, 30 tasks, 20 tasks and 10 tasks). Since the load balancing is perfect every node should theoretically finish at the same time. In Figure 9, on the y-axis, we plot the ratio $\frac{M}{m}$ where M is the finishing time of the last processor and m the finishing time of the first processor. This ratio is equal to 1 when all the processors finish at the same time. On the x-axis we plot the duration of one task on the 100% processor. Results show that the ratio is always under 1.1 and it decreases as the task duration increases. When task duration is greater than 0.1 second (on the fastest processor) the precision of the emulation is better than 5%.

Second we have run two matrix multiplication algorithms for heterogeneous environments. The first one of Beaumont et al. [2] is based on geometric partition of the column on the processors. The second one of Lastovetsky et al. [8] uses a data partitioning based on a performance model of the environment. Figure 10 shows the percentage $100 \times \frac{M-m}{m}$ (where M is the finishing time of the last processor and m the finishing time of the first processor) against the matrix size for both algorithms. We see that the emulation is correct as this percentage is under 5% for Lastovetsky algorithm. The fact that the percentage is worst for the Beaumont algorithm than for the Lastovetsky one means that the later algorithm provides a better load balancing with regards to the former one. We also provide the execution time of both algorithms for different matrix sizes in Figure 11. We see here that Lastovetsky algorithm outperforms the Beaumont one. However, the fact that these experiments shows that the

Lastovetsky algorithm outperforms the Beaumont algorithm should be consider as a side effect of this paper and not as a real contribution. Further studies such as implementation issues, degree of heterogeneity and so on, are required to assess or not this statement. What is shown here, is that Wrekavoc is a suitable tool for doing such comparison.

5. Related works

Studying and comparing heterogeneous parallel algorithms have been tackled by many researcher. Several tools and methodologies have been proposed to asses the performance of such algorithms. Several characteristics have to be defined in order to compare those approaches :

- the ability to execute a real application or just to simulate its execution.
- the level of abstraction of the target platform. The less abstraction, the better is the confidence in the obtained results.
- the speed of execution. Emulation tends to slow-down the execution while simulation tends to speed-up the execution time.
- the ability to fold several processors onto a single one. This enable the execution of a parallel application on only one processor.
- The management of heterogeneity. It is possible to have heterogeneity? Is it controllable?

In table 2 we compare simulation (i.e. simgrid [10]), emulation (i.e. microgrid [19]), in-situ (GdX [7]), Lastovseky and Reddy approach [9] and wrekavoc. We see that none of the proposed approaches have only advantages. This means that depending of the experimentation goal the methodology have to be chosen carefully.

6. Conclusion

Computer-science and especially heterogeneous distributed computing is an experimental science. Indeed, it is not always possible to obtain theoretical results for heuristics designed in this context. Moreover, it is very hard to derive tractable models of such environment.

Simulation (*e.g.*, Simgrid) emulation(*e.g.*, Microgrid) or in-situ implementation (*e.g.*, Grid 5000) are complementary methodologies that have been proposed to conduct experiments on heterogeneous platforms. They all present advantages and drawbacks. In this work we propose a new approach called Wrekavoc. The goal of Wrekavoc is to define and control the heterogeneity of a given platform by degrading CPU, network or memory capabilities of each node composing this platform. Our current implementation of Wrekavoc have been tested on an homogeneous cluster. We have shown that configuring a set of nodes is very fast. Micro-benchmarks show that we are able to independently degrade CPU, bandwidth and latency to the desired values. Tests on algorithms of the literature (load balancing and matrix multiplication) confirm the previous results and show that Wrekavoc is a suitable tool for developing, studying and comparing algorithms in heterogeneous environments.

Future works are directed toward applying this work to other platforms (such as Windows) or other kind of Unix. If CPU burning works for any cases, the other approaches and the network degradation needs to develop innovative solutions.

7. Acknowledgment

This work is partially funded by the *ACI Masses de données* "Data Grid Explorer" of the french Ministry of Research.

The authors would like to thanks the reviewers for very helpful comments and Marc Thierry for writing the first version of the code.

References

[1] J. M. Bahi, S. Contassot-Vivier, and R. Couturier. Coupling Dynamic Load Balancing with Asynchronism in Iterative Algorithms on the Computational Grid. In *IPDPS*, page 40, 2003.

[2] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert. Matrix Multiplication on Heterogeneous Platforms. *IEEE Trans. on Parallel and distributed Systems*, 12(10):1033–1051, Oct. 20001.

[3] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra. Block-cyclic redistribution over heterogeneous networks. *Cluster Computing*, 3(1):25–34, 2000.

[4] R. Buyya and M. M. Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *CoRR*, cs.DC/0203019, 2002.

[5] The DAS-2 project: `http://www.cs.vu.nl/das2/`.

[6] The Grid 5000 project: `http://www.grid5000.org/`.

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

[8] A. L. Lastovetsky and R. Reddy. Data Partitioning with a Realistic Performance Model of Networks of Heterogeneous Computers with Task Size Limits. In *ISPDC/HeteroPar*, pages 133–140, 2004.

[9] A. L. Lastovetsky and R. Reddy. On performance analysis of heterogeneous parallel algorithms. *Parallel Computing*, 30(11):1195–1216, 2004.

[10] A. Legrand, L. Marchal, and H. Casanova. Scheduling Distributed Applications: the SimGrid Simulation Framework. In *CCGRID*, pages 138–145, 2003.

[11] A. Legrand and Y. Robert. *Algorithmique Parallèle*. Dunod, 2004.

[12] M. Maheswaran, S. Ali, H. J. Siegel, D. A. Hensgen, and R. F. Freund. Dynamic Matching and Scheduling of a class of Independent Tasks onto Heterogeneous Computing System. In *Proceedings of the 8th Heterogeneous Computing Workshop (HCW '99)*, april 1999.

[13] F. Manne and T. Sørevik. Partitioning an Array onto a Mesh of Processors. In *PARA*, pages 467–477, 1996.

[14] Planet lab: `http://www.planet-lab.org/`.

[15] A. Sulistio, C. S. Yeo, and R. Buyya. A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. *Softw., Pract. Exper.*, 34(7):653–673, 2004.

[16] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms. In *HPDC*, 1999.

[17] iproute2+tc notes: `http://snafu.freedom.org/linux2.2/iproute-notes.html`.

[18] R. Wolski. Predicting CPU Availability on the Computational Grid Using the Network Weather Service. *Parallel Processing Letters*, 9(2):227–241, 1999.

[19] H. Xia, H. Dail, H. Casanova, and A. A. Chien. The MicroGrid: Using Online Simulation to Predict Application Performance in Diverse Grid Network Environments. In *CLADE*, page 52, 2004.

8. Biographies

Louis-Claude Canon was born in Angers, France. He received a secondary education diploma (major: mathematics and physics) in 2002. He is studying since 2002 in a French college (five-year curriculum) specialising in electronics and computer engineering (named ESEO). He is planning to complete this master and then to prepare a Ph.D. in computer science probably in a field related to the topic of the University of Angers (namely, artificial intelligence). During two trainings periods (in 2004 and in 2005), he got some experience in an academic environment. He realised his first contact with the research in Poona, in India, where he worked during two months for Dr. D. G. Kanhere on parallel programming and on a project which involved modeling and simulation. More recently, he assisted Pr. E. Jeannot in his research by pursuing the development Wrekavoc.

L.-C. Canon's research interests include parallel and distributed processing, modelling and simulation, language theory, algorithmic and artificial intelligence.

Emmanuel Jeannot is a full-time researcher at INRIA (Institut National de Recherche en Informatique et en Automatique) and is doing its research at the LORIA (Laboratoire Lorrain de Recherche en Informatique et ses Applications) in Nancy. He is currently invited at the Innovative Computing Laboratory of University of Tennesse, Knoxville. 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 (resp. in 1996 and 1999) both from Ecole Normale Supérieure de Lyon. His main research interests are scheduling for heterogeneous environments and grids, data redistribution, grid computing software, adaptive online compression and programming models. More informations are available at `http://www.loria.fr/~ejeannot`