Toward an Experiment Engine for Lightweight Grids

Brice Videau, Corinne Touati and Olivier Richard
Laboratoire d'Informatique de Grenoble (LIG) - INRIA, MESCAL Project
ZIRST 51, av. J. Kuntzmann, 38330 Montbonnot St Martin
{brice.videau, corinne.touati, olivier.richard}@imag.fr

ABSTRACT

This paper presents a case study conducted on the Grid'5000 platform, a lightweight grid. The goal was to make a rather simple experiment, and study how difficult it was to carry out correctly. This means it had to be correct, reproducible and efficient.

The paper shows that despite the precautions taken, many parameters that could have an effect on the result were at first overlooked. It also shows that benchmarking plays a key role on making an experiment correct and reproducible. The process is in the end extremely tedious, and stresses the need for new tools to help users.

The contribution of this work is to present a methodology to get correct results on grid architecture, to identify relevant problems and to propose an infrastructure that answers part of the problems encountered during experiments. Additionally, pieces of this infrastructure have been built and are also presented.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

Experiment methodology, experiment framework, lightweight grid, performance study

1. INTRODUCTION

Needs in computing power steadily increase and grow faster than the power of single machines. Hence New computer clusters and grids are built every year, and their computing power has increased a thousandfold in ten years. Nowadays, a cluster of several hundred nodes is common, and grids of several thousand nodes have been built, while others are on the making. For instance the CERN is building one for the large hadron collider and TeraGrid of the National Science Foundation is another large scale grid. Other references can be found in the top 500 list [1], gathering the 500 more power powerful computers.

GridNets 2007 October 17-19, 2007, Lyon, France.
 ISBN 978-963-9799-07-3.
DOI 10.4108/gridnets.2007.2245

To successfully harness this computing power, software infrastructures have been developed. Experiments have to be conducted in order to understand these distributed architectures [21]. The Grid'5000 project, a network of clusters distributed among France, was created to study these large systems [10]. Running experiments on such a large and distributed machine is a challenge. The number of nodes, the different hardware and software configurations, and the different network topologies are as many parameters that can impact the results of a measurement. These parameters add to those of the software tested. In the end the number of variables becomes too large, and cannot all be taken into account.

To identify the problems an experimenter can encounter, a case study was conducted. The methodology followed tried to guarantee the correctness and reproducibility of the results. The case study chosen is the performance evaluation of a file broadcasting tool. It was chosen because it is a well known program whose behavior can be almost predicted, and that would help spotting abnormal results. This experiment proved rather difficult, as many problems appeared.

At first, the Grid'5000 platform will be introduced, followed by a more precise description of the problematic and by an overview of related works. The case study will then be detailed as well as its results. The solution proposed to solve the problems encountered will be presented, as well as the first software modules created.

2. THE GRID'5000 PLATFORM

Grid'5000 is a large scale experimental platform, with deep reconfiguration capability, a controlled level of heterogeneity and a strong control and monitoring infrastructure. It can be presented as a lightweight grid. A lightweight grid can be seen as a simplified version of a grid. It is highly homogeneous in its administration process and software. For instance, accounts are managed with LDAP for the whole platform and users have a single account. Each cluster has its own NFS sever, but mount points are unique for a user. A lightweight grid is like a cluster of clusters but on a larger scale, and involves many different sites and organizations.

Grid'5000 aims to reach 5000 processors distributed among

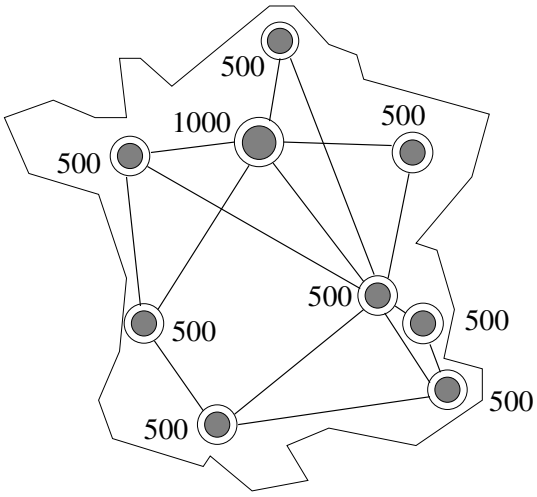

Figure 1: An overview of Grid'5000, showing the number of processors in each site

France (cf figure 1). Sites are interconnected by a 10 Gib/s link. Every node of a cluster is using a 1Gib/s Ethernet network. Other networks like Infiniband, or Myrinet can be provided on a per cluster basis.

Grid'5000 seeks to cover nearly all aspects of software involved in Grid computing : networking protocols, operating systems, middlewares and applications.

Experimenters studying these different aspects may have very different needs. For instance, some will need Linux, others Solaris or FreeBSD, or Windows. Grid'5000 provides a reconfiguration mechanism allowing researchers to deploy, install, boot and run their software image. Default environments are provided and users can use them as a base for their own. Typically users reserve some nodes (that can be distributed among several clusters), deploy their images, conduct their experiment, collect results and release the resources. The deployment tool currently used in Grid'5000 is kadeploy [14].

In order to protect Grid'5000 from outside attacks and to protect the outside from attacks launched using Grid'5000, the grid is isolated from the Internet. Inside Grid'5000 packets can freely move from on site to the other thanks to the dedicated network provided by Renater, the french academic network. This network can only be accessed from gateways located in each participating laboratory.

As homogeneity greatly simplifies studies, many nodes of Grid'5000 are identical. These nodes are dual CPU 1U racks equipped with 2 AMD Opteron running at 2 Ghz, 2 GiB of memory and two 1Gib/s Ethernet adapter. But to allow studies of heterogeneity influence, 1/3 of the nodes are heterogeneous.

Reservation of resources is done via the OAR batch scheduler [9], which works at the cluster level. It can enforce admission rules, interface with the reconfiguration tool and allow precise resource description. A higher level tool called OARGRID allows users to make reservations among different clusters simultaneously.

Other large scale experimental platforms exist: The most well known are Emulab [22] and PlanetLab [8]. Emulab is a network testbed built with emulation in mind. PlanetLab is a distributed system connecting real machines by the Internet, at the planet scale. PlanetLab is used for network studies as well as for distributed systems research. Grid'5000 is more homogeneous than PlanetLab whereas it is more heterogeneous and distributed than Emulab that has a more cluster approach. It has its own dedicated network while PlanetLab is connected via Internet. Let us finally mention DAS2, [2] a cluster of clusters with a dedicated network but which lacks deployment facilities.

3. PROBLEMATIC

Many aspects of Grid experimentation can prove difficult to achieve correctly:

- A lot of parameters exists, some that users are not aware of, and they can be overlooked. So experiment designs have to account for this diversity, while remaining efficient.
- The status of the platform can of course be very difficult to grasp because the status of a single node can have a tremendous effect on the results. But interactions from other users via the network can also be disastrous.
- The experimenter can easily do some things automatically or without thorough checking, so experiments become very difficult to reproduce.
- Last but not least, running and supervising an experiment distributed among several clusters is a difficult task.

3.1 Efficient Experimental Design

As the platform is very complex, an experimental design has to be able to study many parameters simultaneously. But the cost in time of this tedious process can be very high. On a platform like Grid'5000 which is extensively used, reserving many nodes for an extended period of time is limited by admission rules. An experiment design has thus to be efficient and easily split apart in independent measurements.

Breaking the studies in several parts can also prove interesting. First, a design studying the rough influence of many parameters can be used, then a second design studying more carefully the parameters having the most influence can be elaborated.

3.2 Platform Status

The state of the machine running the experiment strongly impacts the results. Two kind of states can be distinguished: the global state of the platform and the state of each node.

The global state is affected mainly by the network load due to other experiments, or by problems of the network infrastructure. If the experiments are not completely separated, by running on different switches and not using the same intercluster links for instance, results cannot be guaranteed

accurate. Hence evaluating the impact of the load on an experiment is a problem by itself, and seems hard to achieve. So, either the confinement of experiments can be asserted or they have to be ran when no one else is using the resources, by making a reservation for one (or several) whole cluster(s) at the same time.

The state of a single node can also affect the outcome of an experiment. When measuring the performance of an algorithm, everything must be set as close as possible to the ideal case. Otherwise unexpected parameters might interfere with the experiment and skew the results, and the experimenter will draw erroneous conclusions. For instance, let us imagine a distributed tool which performance can be limited by the hard drive of each node. If a node has a slow disk, because of a failure or a model difference, the measurements using this nodes will show poor performance. Part or all the measurements will be skewed by this node. In this example an unknown variable has been introduced, while not being taken into account by the experimental design. This scenario can accur with many different kinds of resources. Those *slow* nodes have to be identified beforehands and removed from the experiment. During the experiment, nodes have to be monitored to prevent *slow* nodes from appearing.

3.3 Experiment Supervision

The supervision of a running experiment is tedious. Since one has to work at night to obtain a large share of resources, people are reluctant to make experiments. Hence, everything that can be used to automate the process will be appreciated by experimenters.

Yet, a truly automated process must account for the problems seen in the previous section, namely managing machines, and identifying measurement failures in order to reproduce them after fixing the problems and killing locked measurements. Many users code ad hoc scripts to manage their experiments, but usually a failure during the experiment provokes its abortion.

Providing users with a framework to run and automate experiments could be the way to go. It could allow them to run only short parts of an experiment, enabling their execution during the day. It could also automatically test the nodes and platform status.

3.4 Experiment Reproducibility

The reproducibility of an experiment is tied with the problematics mentioned above. In order to reproduce an experiment, every bit of information is capital. Indeed, the results of an experiment are meaningless if the software and hardware configurations are unknown, or if there are hidden variables like slow nodes. Unfortunately experimenter do not always log everything they do during an experiment, and can forget important steps they took.

To be able to reproduce an experiment, all these information have to be made available. But logging the output of every command and managing the files produced is hard to achieve. The experiment supervision framework mentioned above could help the experimenters with this aspect of experimentation also.

4. RELATED WORKS

This section presents the experimental designs used for the case study, and the benchmarking tools required to monitor the platform, and ends with a review of experiment supervision tools.

4.1 Experimental Designs

The case study objective was to use an experimental design studying a lot of parameters simultaneously, without being too costly. Designs that correspond to this objective are full and fractional factorial designs, which are used in many science fields. Their application to computer science is described in [17], chapters 16-23. Fractional factorial designs study the influence of many parameters but not their interactions. Full factorial designs study the influence of fewer parameters (usually three or four), but take into account the interactions between them.

Experiments based on these designs are found in [15] and [16]. These designs offer lots of information in a minimum number of measurements, and enable experimenters to distinguish between parameters that have a high impact, and those that are negligible. They can save experimenters a lot of time, and can easily be broken in independent measurements.

4.2 Monitoring and Benchmarking

To determine if the results of an experiment are valid, the state of the platform it is ran on must be known. Yet, the global state of a cluster or a grid is hard to grasp because of the number of parameters. Software like Network Weather Service [23] have been developed to reach this goal, but they are intrusive and can skew the measurements. They also require infrastructures to be set up on the grid.

The state of a single node can be obtained using benchmarking tools. The result of the benchmarks are compared to reference values before a measurement. If nodes present a significant performance difference they are discarded. Benchmarks exist for all kind of resource, like SpecINT and SpecFP [3] for processors, IOzone [4] for disks or netperf [5] for network interfaces.

4.3 Experiment Supervision

Experiment supervision has seen a strong development along with grid architecture. Indeed, doing experiments on a platform as complex and volatile without human intervention is difficult, but doing it *by hand* is becoming too complex. That is why each category of grid, aimed at production or experimentation, has its own automated experimentation project. For instance, Emulab platform [22] has DART [11], PlanetLab [8] has PluSH [7] and Globus [13] has ZENTURIO [20]. But it is too early to tell if these managers are fit for the lightweight grid model of Grid'5000.

ZENTURIO is very complex, with dozens of modules and would be hard to install on the numerous software environments of Grid'5000. In order to use it we would also have to install Globus. Nevertheless ZENTURIO offers a complete infrastructure and a language to describe complex experiments.

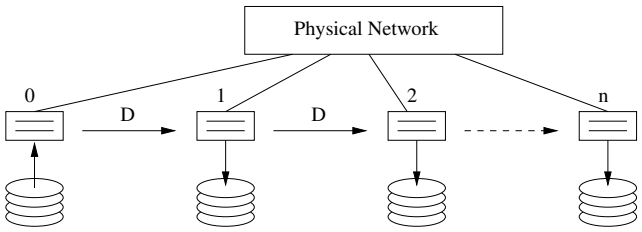

Figure 2: The mput pipeline: computers are interconnected by a high speed network, the root node (number 0) reads files on the disk and sends them to the next node which writes to the disk and sends them to the next node, and so on.

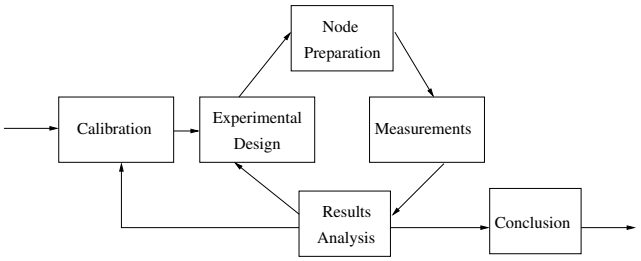

Figure 3: The experimental process followed is a 5 steps iterative process. When new problems are identified, a new benchmark is created and added to the pool and the platform is calibrated again.

DART is a bit special because its aim is to run non regression tests on distributed softwares. For now, DART is only implemented on Emulab, which renders it hard to evaluate its use on other platforms. It is also dedicated to clusters. A new experiment workbench for the Emulab platform is presented in [12], but it is still strongly tied to Emulab and networking research.

PluSH seems to be the most promising. Unfortunately it is still strongly tied to PlanetLab's architecture. Nevertheless the recent developments of PluSH in experiment description languages, their deployment management and their portability objectives are promising. It is still too early to know if their language will fit Grid'5000 needs.

Lightweight grids can be used to study grids, clusters, middlewares, network, applications, etc... In order to adapt to the many faces of lightweight grids, a more generic toolkit could prove necessary.

5. CASE STUDY

So as to be representative of an experiment on the Grid'5000 platform, the case study had to be carefully selected. The criteria it had to fulfill were: it had to be distributed over enough nodes, it had to very intensive on various parts of the platform and it was a plus if it was used on Grid'5000. The mput program was found to be a perfect candidate.

5.1 The mput Program

This program is a high performance file broadcaster. It uses a chain of TCP connections to broadcast the files, that are at first present only on the source node [18]. At initiation it creates an instance of itself on each node using an integrated parallel launcher [19]. The launcher can use a tree of fixed or dynamic arity. The dynamic arity is created by a work stealing algorithm. Once every nodes are launched they chain each other in the order of their network addresses. The program can then be seen as a pipeline having the depth of the number of nodes minus the emitter (cf Figure 2). The pipeline is fed at one end (the emitter) with the files, while every other nodes write the files to the disk. The program is a multithreaded application written in the C++ language and is about 2000 lines long.

As can be seen, this program is very sensitive, as a slow element will affect the whole pipeline. At first we can identify two possible bottlenecks: the throughput of the disks on each machine and the bandwidth of the network. But we cannot rule out other parameters like machine load, or the size of the file to be transferred.

5.2 Methodology

The methodology we used is represented on Figure 3. The first thing to do is obtain reference performance values of the different kinds of machines available on the grid (calibration). This benchmarking phase will help determine if a machine is in a *standard* state or needs to be fixed, and thus has to be removed from an experiment. An experiment design is then selected, according to the available resources and platform architecture. A larger pool of machines than the one needed is then selected, in which every node is checked for defects, and non conforming ones are eliminated. The experimental design is then executed on this pool and the results are analyzed a first time to see if everything went well. If they are not satisfactory, it might be for two reasons. First, the experimental design might not be good enough, and does not show anything interesting, but the data is still meaningful and another plan is designed. Second, if something was forgotten during the calibration phase some nodes may not be in a correct state: culprits need to be identified. Toward that end a new benchmark is created that identifies such failures and added to the benchmark suite. All or part of the results obtained are invalid.

The first thing to notice is that at the present time a tool to monitor accurately the global state of Grid'5000 is not available. Because of this, all the experiments of the present work were made at night when alone on the cluster, so as to avoid network interferences with other users. An advantage of Grid'5000 is that 2/3 of the nodes are homogeneous and, as the impact of hardware was not to be investigated, only nodes with the same configuration were selected. Those were IBM eServer 325, with the same hardware setup, running the same Linux distribution, the same kernel version and the same compiler version (gcc). Two clusters were used for the experiment, 400 km apart, one at the Orsay site (216 nodes at the time), the other at the Lyon site (56 nodes at the time), interconnected by a Gigabit link (Renater 3 at the time). Inside the clusters nodes were connected by a Gigabit Ethernet network.

Before each measurement, nodes were tested using two simple benchmarks. The first stressed the disk by writing a 1 GiB file. The second stressed the CPU by computing the

	I	A	B	C	D	E	F	G
1	+1	+1	+1	+1	+1	+1	+1	+1
2	+1	−1	+1	+1	−1	+1	−1	−1
3	+1	+1	−1	+1	−1	−1	+1	−1
4	+1	−1	−1	+1	+1	−1	−1	+1
5	+1	+1	+1	−1	+1	−1	−1	−1
6	+1	−1	+1	−1	−1	−1	+1	+1
7	+1	+1	−1	−1	−1	+1	−1	+1
8	+1	−1	−1	−1	+1	+1	+1	−1

Figure 4: Fractional factorial design used, measurements (1..8) and parameters (A..G) levels (-1,+1).

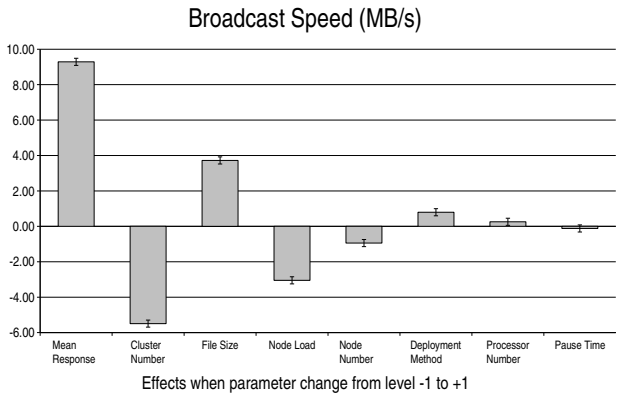

Figure 5: Case study results: effects sorted in order of their influence on the broadcast speed in MiB/s.

Ackermann function for parameters 4 and 1. The time to complete each benchmark was used to remove slow nodes from the valid pool of machines. Computers were later investigated in order to know why they were slow. This process can drastically reduce the number of available nodes for the experiment: in the case study more than 50% of the nodes were deemed unsuitable.

5.3 Experimental Design

So as to make a non trivial experiment, with multiple parameters, a factorial fractional design with two levels for each parameter has been used. The design used is represented on Figure 4. Each line of the table corresponds to a measurement that was repeated eight times. Detailed analysis of fractional factorial designs with replications can be found in [17], chapter 18. For each measurement, parameters are set according to the levels in the table. Parameters and their possible values (with the corresponding levels in parenthesis) are:

- A: size of the file to broadcast, 10MiB (-1) or 1GiB (+1),
- B: number of nodes, 4 (-1) or 32 (+1),
- C: number of clusters, 1 (-1) or 2 (+1),
- D: number of processors used by node, 1 (-1) or 2 (+1),
- E: arity of the deployment tree, fixed (-1) or dynamic (+1),
- F: load of the nodes, no load (-1) or loaded (+1),
- G: pause time between measurements, 0s (-1) or 60s (+1).

I is not a parameter, it is the average response of the system studied. Machine load (parameter E) is created by a loading program (an infinite loop) that just takes some processor time. A node that is loaded is running 1 loop on each available processor. Let y be the system response, the influence of the parameters can be modeled as follow:

$$y = q_0 + q_A x_A + q_B x_B + q_C x_C + q_D x_D + q_E x_E + q_F x_F + q_G x_G + e,$$

where e is the error, q_0 is the average response of the system, q_i is the influence of the parameter i when the level changes from -1 to +1. The x_i's are the levels of the parameters, -1 or +1. The q_i's are obtained by solving a system of linear equations given by the different measurements.

5.4 Results

Results of the case study are presented on Figure 5. Three main effects can be identified: the number of clusters (C), the size of the file (A) and the load of the nodes (F). In the best case, transferring a 1 GiB file on a single cluster with no load, the throughput measured reaches 20 MiB/s whereas the model predicts 21MiB/s = 9 (Mean response) + 5.5 ($x_C = -1$) + 3.5 ($x_A = +1$) + 3 ($x_F = -1$). In the worst case, transferring a 10 MiB file on two clusters with loaded nodes, the throughput falls to 3 MiB/s whereas the model predicts -2MiB/s. The differences measured between the values and the model predictions are due to the fact that the later does not account for the interactions between parameters. The impact of the file size is easy to explain: with a 10 MiB file most of the time is spent deploying the application. The impact of the load shows that the program needs some computing power to manage the 60 MiB/s flow of data that goes through a node: 20 MiB in, 20 MiB out, and 20 MiB to write on the disk. It was not at first an expected limiting factor, so it is good it was not ruled out. The impact of the number of clusters is not as easy to understand, and will be explained in Section 5.4.3. In fact the objective of the study was not the results but to point out the problems encountered during the experiment.

5.4.1 Experimental Problems

First of all, measurements proved difficult to realize without human intervention. The parallel launcher used at the time was not mature enough and sometimes deadlocked. The experimenter had to be in front of the screen to kill those locked tasks. Sometimes the performance was very low, due

to hardware failures and misconfiguration of nodes. The culprits had to be identified and the measurements redone.

This process could not be made automatic. In order to achieve this, one must be able to interpret the results of measurements in real-time, and to determine if a measurement failed or was skewed. Then, node performance has to be evaluated again and a new pool of working nodes created. The script has to be able to run and analyze benchmarks.

5.4.2 Heterogeneity Problems

During the benchmark phase many problems were encountered. The first problem found was that the nodes could be divided in two groups regarding their disk performance. Some nodes presented abnormally low performance, and apparently in a random fashion. The benchmark could run well three times in a row and the fourth it would take twice as long to run. A careful study of the hardware revealed two different brands of disks. Some disks of the second brand sometimes had low performance. So half of the nodes had to be ruled out of the experiment. A short while after the manufacturer changed the disks of the second brand for ones of the first.

The opposite problem was also found, namely a node had better disk performance than the others. The partition table was different, and the benchmark was writing on a more performant part of the disk. This node was reinstalled.

During measurements, performance was sometimes lower than those expected when many nodes were used. At boot some nodes (5%) were randomly configuring their network interface at 100 Mib/s instead of the expected 1 Gib/s. Those nodes were of course slowing down the whole pipeline. A new firmware for network cards solved the problem, but it was not out when measurements were done. Hence nodes had to be checked at each reboot, and our benchmark suite gained a network test.

5.4.3 Low Performance Using Two Clusters

As shown on Figure 5 column "Cluster Number", when two clusters are used performance collapses. This was surprising at first because usually the tool is disk limited and not network limited. In fact bandwidth was limited by the TCP windowing. As the clusters used were 400 km apart latency wass important and TCP windows had to be enlarged accordingly. To complete the study, measurements would have to be redone. Figure 4 column C shows that only half of them (1,2,3 and 4) concern 2 clusters (+1 level), so only these have to be redone.

6. TOWARD AN EXPERIMENT ENGINE

The problems encountered when running a rather small experiment (only 32 nodes on two clusters), will get worst as the size of experiments grow. To address these problems, tools need to be developed. They have to be able to run an experiment without human interaction, manage resources, check the platform status and take care of the logs generated. The following subsection presents the model of automatic experiment we designed. The next subsection introduces the execution engine developed to process experiments. And in the last subsection the remaining work is summarized.

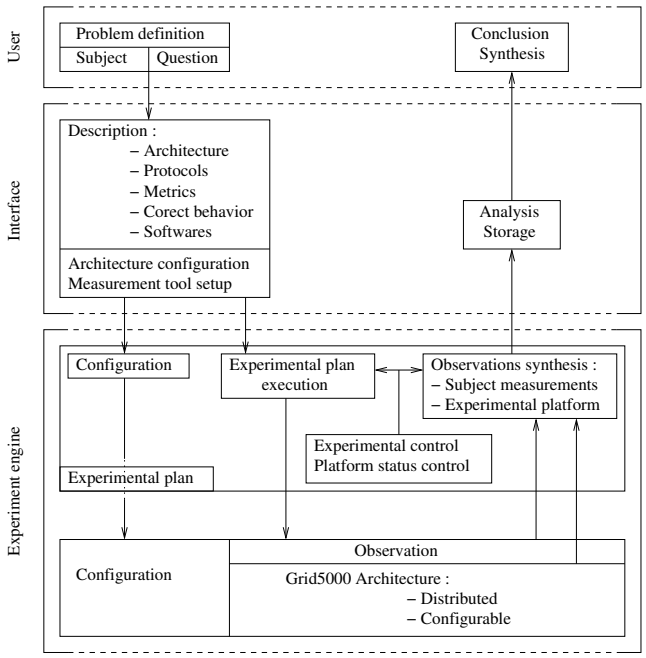

Figure 6: Experiment controlled by an engine.

6.1 Engine Controlled Experiments

A Grid'5000 experiment that is ran on an engine can be modeled as shown on Figure 6.

At first, the experimenter defines his problem, namely specify his study subject, and the question asked.

Then he has to formulate his experiment in a language describing several aspects of the experimental process. For instance what files are produced by which command, the expected return values of commands, timeouts, the environment required by the user. The descrition includes the flow of commands, and depending on commands results, the path to be taken in the flow. This program constitutes the experimental plan that will be ran in the experiment engine.

At this time the engine takes care of the configuration of the platform and tests its good behavior. If enough resources are found to meet the experiment requirements the engine processes the plan. It constantly checks that the commands behave as specified by the experimenter, and logs outputs. It can periodically check the status of the platform to ensure the best experimental results can be expected. If an error happens, it either tries to run the command again, exits or proceeds with the next commands, according to the experimenter specifications.

Finally the engine returns the platform to its default state, and backs up every file generated and every output logged. Some analysis on the data can be performed.

6.2 Execution Engine

The execution engine (Figure 7) is the part of the experiment engine that runs the commands and logs the results. It processes every command, from configuration ones to result analysis. The execution module can be separated in three

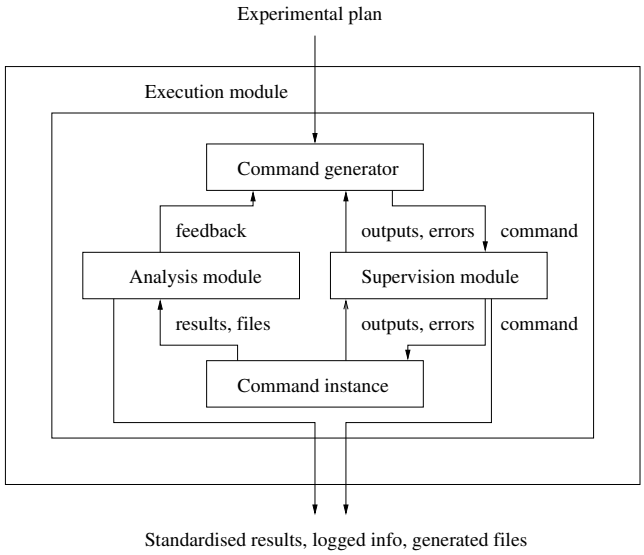

Figure 7: Overview of an execution engine and its key components.

components. The *command generator* that is responsible for translation of the experimental plan into commands. Depending on outputs and errors of previous commands and on feedback from the analysis module, different commands can be issued. The *supervision module* has to execute the commands at the right time and at the right place. It also has to log the outputs and errors of the commands for later archival. The *analysis module* can do run-time analysis of results from commands, in order to influence the unfolding of the experimental plan. At the end the results and the logs are archived.

This part of the engine is already functional, and has ran experiments on several clusters. It issued reservation commands, ran benchmarks, eliminated failing nodes and executed the commands. Every command outputs and status are logged and the files produced are saved. Outputs, status, start and end dates, process identifier and host node can be saved in YAML [6] or XML format for an easy post mortem study. This execution engine is based on a client/server model. It is written in Ruby, an object oriented script language. Communications between clients and servers are based on the SOAP protocol. As a high level language has not yet been developed the client is used via a Ruby script. The engine is recursive as the server can also act as a client to another server. This way, load can be distributed and the grid can be logically divided. The engine makes an extensive use of Taktuk [19], a parallel command launcher that can use SSH to connect to nodes.

To test the execution engine, a token ring (cf Figure 8) has been deployed on Grid'5000. The execution engine was launched on a login machine. At first, 300 nodes located on 5 clusters were reserved by the engine. Using Taktuk, each node was issued the *date* UNIX command. Nodes which were not accepting connections or failed to run the command were removed. Taktuk was then used to launch the token ring on valid nodes. Each instance of the program generated

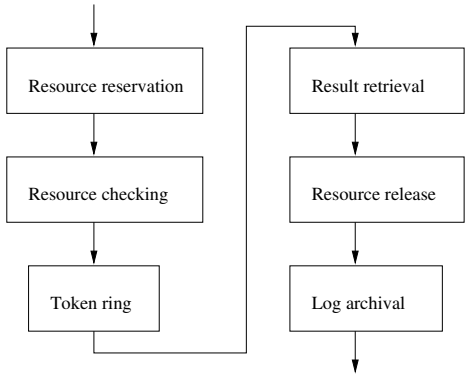

Figure 8: The different steps of the token ring experiment.

an output file on its node. Those files were gathered when the token ring stoped its execution. At this point resources were released, this was the last command issued. Logs of each command and reservation details were then archived.

6.3 Future Work

The execution engine is a first step in the making of an experiment engine. Many parts are still missing, as detailed in this section.

6.3.1 Experimentation Language

Now that the execution engine is almost functional, it needs a language to interface smoothly with the experimenters. But before designing or choosing a language that matches lightweight grid experiments, its specificity have to be taken into account. Nevertheless, it will have some similarities with other workflow languages: resource naming and coupling with the batch scheduler or other resource allocator, precise command description with data flows, outputs, errors and return status, loop and conditional structures with exception support, timeout managements, etc.

Lightweight grid homogeneity and ease of configuration will certainly have an impact on the language chosen. It is too early to draw conclusions, and the problem needs to be studied further. Anyway some specific needs that have to be addressed are: resource set manipulation (iteration, union, split...), special error management (fail-stop, retry, ignore...), time measurements and explicit data placement.

The language developed will certainly be an extension of an existing one. A list of the main workflow languages for grid is presented with their characteristics in [24].

6.3.2 Platform Benchmarks

As shown, finding failing nodes is a difficult task. Test suites must cover every available resources and be fast enough to be frequently used during the experimental process. Developing synthetic tests that detect abnormal behaviors among a group of nodes is necessary. Without them, correctness of experiments and accuracy of the results cannot be asserted.

The token ring evoked in Section 6.2 is an example of tools to monitor the network performance of nodes, and the network

status of the grids. Other tools already exist like processor and disk benchmarks, and must be incorporated in the experimental process.

7. CONCLUSIONS

This paper tried to demonstrate the extreme caution needed when making experiments on a large distributed lightweight grid. Grid'5000 is a perfect example of a grid of almost homogeneous clusters, that can give the experimenter a feeling of security. It appears in fact that many parameters can be overlooked, skewing experimental results.

To ensure that an experiment can be reproduced at different times, or conducted during different time slots on the grid, many precautions have to be taken. The platform has to be thoroughly benchmarked to ensure it is in a similar state. Testing every involved node is also necessary to account for machine failures.

We believe that the case study presented shows that it is a difficult task to achieve *by hand*. Instead of forcing users to follow this tedious methodology, giving them tools to help them in this task must be a better way to go. Logging tools are already functional, and the first benchmarking modules have been developed. Assembling these modules, and linking them with the experiment still needs a bit of coding on the user part. The future objectives are to provide them with a higher level interface to assemble these blocks.

Our contribution is the identification of the problems encountered by the experimenter working on a lightweight grid, a methodology to help him/her in his/her work and the first blocks of tool to help him/her achieve correctness and reproducibility of his/her experiments.

8. ACKNOWLEDGMENTS

This work has been done within the LIG laboratory jointly supported by CNRS, INPG, INRIA, and UJF. Computer resources are provided by the Grid5000 platform (further information at http://www.grid5000.fr/).

9. REFERENCES

[1] http://www.top500.org.
[2] http://www.cs.vu.nl/das2/.
[3] http://www.spec.org.
[4] http://www.iozone.org.
[5] http://www.netperf.org.
[6] http://www.yaml.org.
[7] J. Albrecht, C. Tuttle, A. C. Snoeren, and A. Vahdat. Planetlab application management using plush. *SIGOPS Oper. Syst. Rev.*, 40(1):33–40, 2006.
[8] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale network services. In *First Symposium on Networked Systems Design and Implementation (MSDI)*, pages 253–266, Mar. 2004.
[9] N. Capit, G. D. Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, and O. Richard. A batch scheduler with high level components. In *Cluster computing and Grid 2005 (CCGrid05)*, 2005.
[10] F. Cappello, F. Desprez, M. Dayde, E. Jeannot, Y. Jegou, S. Lanteri, N. Melab, R. Namyst, P. Primet, O. Richard, E. Caron, J. Leduc, and G. Mornet. Grid'5000: A large scale, reconfigurable, controlable and monitorable grid platform. In *Grid2005 6th IEEE/ACM International Workshop on Grid Computing*, 2005.
[11] Chun. DART: Distributed automated regression testing for large-scale network applications. In *International Conference on Principles of Distributed Systems (OPODIS), LNCS*, volume 8, 2004.
[12] E. Eide, L. Stoller, and J. Lepreau. An experimentation workbench for replayable networking research. Technical Report FTN–2006–03, University of Utah, Dec. 2006.
[13] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997. ftp://ftp.globus.org/pub/globus/papers/globus.pdf.
[14] Y. Georgiou, J. Leduc, B. Videau, J. Peyrard, and O. Richard. A tool for environment deployment in clusters and light grids. In *Second Workshop on System Management Tools for Large-Scale Parallel Systems (SMTPS'06)*, Rhodes Island, Greece, April 2006.
[15] C. Jacqmot. *Load Management in Distributed Computing Systems: Toward Adaptative Strategies*. PhD thesis, Université catholique de Louvain, Jan. 1996.
[16] C. Jacqmot and E. Milgrom. Évaluation empirique des performances d'un système informatique : application à l'équilibrage de charge. In *Placement dynamique et répartition de charge: application aux systèmes répartis et parallèles*, pages 231–250, Dec. 1996.
[17] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc, 1991.
[18] C. Martin and O. Richard. Parallel launcher for clusters of PC, parallel compting. In *Parco'01 (Parallel Computing), Naples*, 2001.
[19] C. Martin, O. Richard, and G. Huard. Déploiement adaptatif d'applications parallèles. *Technique et Science Informatiques (TSI)*, 2005.
[20] R. Prodan and T. Fahringer. Zenturio: An experiment management system for cluster and grid computing. In *Proceedings of the 4th International Conference on Cluster Computing (CLUSTER 2002)*, Sept. 2002.
[21] W. F. Tichy. Should computer scientists experiment more? *COMPUTER: IEEE Computer*, 31:32–40, 1998.
[22] B. White, S. Guruprasad, M. Newbold, J. Lepreau, L. Stoller, R. Ricci, C. Barb, M. Hibler, and A. Joglekar. Netbed: an integrated experimental environment. *Computer Communication Review*, 32(3):27, 2002.
[23] R. Wolski, N. T. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.
[24] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3-4):171–200, September 2005.