Grid'5000: a large scale and highly reconfigurable Grid experimental testbed

Franck Cappello, Eddy Caron, Michel Dayde, Frederic Desprez, Yvon Jegou, Pascale Primet, Emmanuel Jeannot,
Stephane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Benjamin Quetier, Olivier Richard,
INRIA, LRI, LIP, IRISA, LORIA, LIFL, LABRI, IMAG
www.grid5000.org Email fci@lri.fr

Abstract

Large scale distributed systems like Grid gather several characteristics making them difficult to study only from theoretical models and simulators. Most of Grid deployed at large scale are production platforms making them inappropriate research tools because of their limited reconfiguration, control and monitoring capabilities. In this paper, we present Grid'5000, a nation wide infrastructure for research in Grid computing. The main design goal of Grid'5000 was to make it a scientific tool like large-scale instruments used by physicists, astronomers and biologists. We describe the motivations, design considerations, architecture, control and monitoring infrastructure of this large-scale instrument. We present configuration examples and performance results about the reconfiguration subsystem.

1 Introduction

Grid is well established as a research domain and proposes technologies that are enough mature to be used for real life applications. Projects like eScience, TeraGrid, DEISA and NAREGI, to cite a few, demonstrate that large scale infrastructures can be deployed to provide scientists fairly easy access to geographically distributed resources belonging to different administration domains. Despite its establishment as a viable computing infrastructure, there are still many issues to be solved and mechanisms to optimize in performance, fault tolerance, QoS, security and fairness.

As large scale distributed systems, Grid software and architecture gather several characteristics making them difficult to study only following a theoretical approach. As a matter of facts, most of the research conducted in Grids are currently performed using simulators, emulators or production platforms. As discussed in the next section, all these tools present limitations making difficult the study of new software and optimizations.

We believe that Grids are so complex and dynamic that there is a need for a new category of tools: highly reconfigurable, controllable and monitorable real life experimental platforms. Such a platform is not a Grid but a large scale tool to study Grid issues in real life scale and conditions.

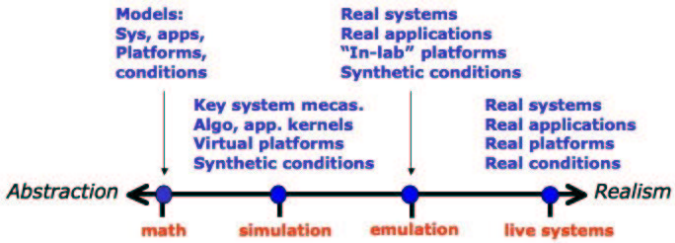

Figure 1. Methodologies used in distributed system studies.

Such kind of tools already exist in other contexts. The closer example is PlanetLab [1], which consists in a set of PCs connected to the internet and forming an experimental distributed system. PlanetLab is used for network study as well as for distributed systems research.

In this paper we present the Grid'5000 project. We first motivate the need for a real size, real life experimental platforms, discussing the limitations of existing tools. In Section 3, we present the principles of Grid'5000, based on the results of Grid researcher interviews. The implementation of Grid'5000 is presented in Section 4. Section 5 present configuration examples, demonstrating the high reconfigurability of the experimental platform. In Section 6 we present performance results concerning one of the key component of Grid'5000: the reconfiguration system.

2 Motivations and related work

Like other scientific domains, research in Grid computing is based on a variety of methodologies and tools. Figure 1 present the spectrum of methodologies used by researchers to study research problem in distributed systems.

In large distributed systems, numerous parameters must be considered and complex interactions between resources makes analytical modelling impractical. Thus simulators, emulators and real platforms are preferred.

Simulators provide a first methodology, focusing on a

specific behavior or mechanism of the distributed system and abstracting the rest of the system. Their fundamental advantage is their independence to the execution platform. Bricks [2] was proposed for studies and comparisons of scheduling algorithms and frameworks, Bricks allows scheduling researches for multi-client, multi-server Grid scenarios. Users can specify network topologies, server architectures, communication models and scheduling framework components. Some Bricks components are replaceable by real software, helping its validation. SimGrid [3, 4] is used to study single-client multi-servers scheduling in the context of complex, distributed, dynamic, heterogeneous environments. SimGrid is based on event driven simulation, providing a set of abstractions and functionalities to build a simulator corresponding to the applications and infrastructures Resources latency and service rate may be set as constants or evolve according to traces. The topology is fully configurable. GridSim [5] is a higher-level simulator, designed to investigate interactions and interferences between scheduling decisions taken by distributed brokers. It focuses on Grid economy research, where the scheduling involves the notions of producers, consumers and brokers. Until recently, GridSim did not consider any network topology. GangSim [6] considers a context where hundreds of institutions and thousands of individuals collectively control and use tens or hundreds thousands of computers and associated storage systems. It models usage policies at the levels of site and VO (Virtual Organization) and can combine simulated components with instances of a VO Ganglia Monitoring toolkit running on real resources. OptorSim [7] focuses on data replication strategies for Data Grids applications processing very large data sets. Its design directly derives from the DataGrid project architecture. OptorSim simulates sites providing computational and/or data-storage resources, resource broker scheduling jobs to computing resources and routers. ChicSim [8] and HyperSim [9] are other simulators close to the presented ones.

Surprisingly, only few results on simulator validation have been presented. The validation of Bricks [2] was performed by incorporating NWS in Bricks and comparing the NWS results obtained on a real Grid with the ones obtained by a Grid simulated by Bricks. SimGrid [3] validation consisted in comparing the simulator results with the ones obtained analytically on a mathematically tractable scheduling problem.

In some situations, complex behaviors and interactions of the distributed system nodes cannot be simulated, because of the difficulty to capture and extract the factors influencing the distributed systems. Emulators can address this limitation by executing the actual software part of the distributed system, in its whole complexity. Emulators are generally run on rather ideal infrastructures (i. e. controlled clusters). MicroGrid [10] allows researchers running Grid applications on virtual Grid resources. Resource virtualization is done by intercepting all direct use of resources. The emulation coordination essentially controls the simulation rate, which is determined by the virtualization ratio for all resources. The emulation time base is controlled by a virtualization library returning adjusted times to the system routines. Accurate processor virtualization relies on specific schedulers and the network virtualization [11] uses the MaSSF system for a scalable online network simulation.

Authors of MicroGrid have conducted a thorough validation [10, 11, 12]. The internal timing of MicroGrid was validated using the AutoPilot system. The capacity of the emulator to enforce memory limitation and to maintain the processing model under CPU and I/O competition was validated using microbenchmark. Emulations results were compared to experimentation ones on real platforms for the NAS benchmark, in order to validate the full emulation engine. Validation with real applications compared the execution times of CACTUS problem solving environment, Jacobi, Scalapack, Fish, Game of life and Fasta on real platforms with the ones obtained by MicroGrid.

Because Emulators use the real software, they cannot scale as well as simulators. Furthermore, there is still a distance between emulators and the reality: even traffic and fault injection techniques, generally based on traces or synthetic generators cannot capture all the dynamic, variety and complexity of real life conditions. Real life experimental platforms solve this problem by running the real software on realistic hardware. DAS2 (http://www.cs.vu.nl/das2/) is basically an idealized Grid, all sites being connected on the Internet. Experiments are run on top of a Grid middleware. managing the classical security and runtime interfaces issues related to Grid platforms. The nodes are voluntarily homogeneous, providing a much simpler management and helping a better environment for performance comparison (speed up of parallel applications) and understanding. PlanetLab [1] is another real life experimental platform, connecting real machines by the Internet, at the planet scale. Some production Grids (TeraGrid, eScience, DataGrid) have also been used as experimental platforms, before being opened to actual users or during some dedicated time slots.

Two strong limitations of real life platforms as experimentation tools are 1) their low software reconfiguration capability and 2) the lack of deep control and monitoring mechanisms for the users. Then next section will show how Grid'5000 addresses these limitations.

3 Designing Grid'5000

The design of Grid'5000 derives from the combination of 1) the limitations observed in simulators, emulators and real platforms and 2) an investigation about the research

topics that the Grid community is conducting. These two elements lead to propose a large scale experimental tool, with deep reconfiguration capability, a controlled level of heterogeneity and a strong control and monitoring infrastructure.

3.1 Experiment diversity

During the preparation of the project, in 2003, we asked researchers in Grid computing the experiment they are willing to conduct on a large scale real life experimental platform. The members of 10 teams in France, involved in different aspects of Grid Computing and well connected to the international Grid community, proposed a set of about 100 experiments. It was surprising to discover that almost all teams require different infrastructure settings for their experiments. The experiment diversity nearly covers all layers of the software stack used in Grid computing: networking protocols (improving point to point and multipoints protocols in the Grid context, etc.), operating systems mechanisms (virtual machines, single system image, etc.), Grid middleware, application runtimes (object oriented, desktop oriented, etc.), applications (in many disciplines: life science, physics, engineering, etc.), problem solving environments. Research in these layers concerns scalability (up to thousands of CPUs), performance, fault tolerance, QoS, security.

3.2 Deep reconfiguration

For researchers involved in protocols, OS and Grid middleware research, the software setting for their experiments often requires specific OS. Some researchers need Linux, while others are interested by Solaris10 or Windows. For networking researches, FreeBSD is preferred because network emulators like Dummynet and Modelnet run only on FreeBSD. Some researches on virtual machine, process checkpoint and migration need the installation of specific OS versions or OS patches that may not be compatible between each others. Even for experiment over the OS layers, researchers have some preferences: for example some prefer Linux kernel 2.4 or 2.6 because their scheduler difference. Researchers needs are quite different in Grid Middleware: some require Globus (in different versions: 3.2, 4, DataGrid version) while others need Unicore, Desktop Grid pr P2P middleware. Some other researchers need to make experiments without any Grid middleware and test applications and mechanisms in a multi-sites, multi-clusters environment before evaluating the Middleware overhead. According to this inquire on researchers needs, Grid'5000 should provide a deep reconfiguration mechanism allowing researchers to deploy, install, boot and run their specific software images, possibly including all the layers of the software stack. In a typical experiment sequence, a researcher reserves a partition of Grid'5000, deploys its software image, reboots all the machines of the partition, runs the experiment, collects results and relieves the machines. This reconfiguration capability allows all researchers to run their experiments in the software environment exactly corresponding to their needs.

3.3 A two levels security approach

Because researchers will be able to boot and run their specific software stack on Grid'5000 sites and machines, we cannot make any assumption on the correct configuration of the security mechanisms. As a consequence, we should consider that Grid'5000 machines are not protected. Two other constraint increases the security issue complexity: 1) all the sites hosting the machines are connected by the Internet, 2) basically inter-site communication should not suffer any platform security restriction and overhead during experiments. From this set of constraints, we decided to use a two levels security design with the following rules: a) Grid'5000 sites are not directly connected to the Internet and b) all communication packets fly without limitation between Grid'5000 sites. The first rule ensures that Grid'5000 will resist to hacker attacks and will not be used as basis of attacks (i. e. massive DoS or other more restricted attacks).

This design rules lead to build a large scale confined cluster of clusters. Users connect to Grid'5000 from the lab where the machines are hosted. Strong authentication and authorization check is done first to enter the lab and then to log in Grid'5000 nodes from the lab.

3.4 2/3 of homogeneous nodes

Performance evaluation in Grid is a complex issue. Speedup evaluation is hard to evaluate with heterogeneous hardware. In addition, the hardware diversity increases the complexity of the deployment, reboot and control subsystem. Moreover, multiplying the hardware configurations directly leads to increase the every day management and maintenance cost. Considering these 3 parameters, we decided that 2/3 of the total machines should be homogeneous. However Grid are heterogeneous by nature and this is an important dimension in the experiment diversity That's the reason why we choose to keep 1/3 of heterogeneous machines.

3.5 Precise control and measurement

Grid'5000 will be used for Grid software evaluation and fair comparisons of alternative algorithms, software, protocols, etc. This implies two elements: first, users should be able to steer their experiments in a reproducible way

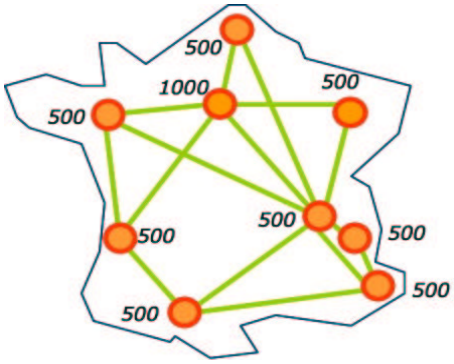

Figure 2. Overview of Grid'5000.

and second, they should be able to access probes providing precise measurements during the experiments. The reproducibility of experiment steering includes the capability to 1) reserve the same set of nodes, 2) deploy and run the same piece of software on the same nodes, 3) synchronize the experiment execution on the all the involved machines, 4) if needed, repeat sequence of operations in a timely and synchronous way, 5) inject the same experimental conditions (synthetic or trace based: fault injection, packet loss, latency increase, bandwidth reduction). As described in the next section, Grid'5000 software set provides a reservation tool (OAR [13]), a deployment tool (Kadeploy [**?**]) and a several tools (FAIL [14], eWAN eWAN experimental condition injectors. Precise and extensive measurement is a fundamental aspect of experimental evaluation on real life platforms. Grid'5000 provides users several sets of probes for measuring network activity during experiments. A first set of probes collects the packet header of all packets crossing the access routers of all sites. These headers are stored in a database during the experiment. A second set of network probes measures the traffic in every tunnel connecting the sites. Local observation of processor memory, disk and network is difficult at the hardware level and since the users may use their proper software configuration, there is no way to provide a built-in and trustable monitoring system for CPU, Memory and Disc. So it is the responsibility of the users to proper install, configure and manage the software observation tools they need for their experiments.

4 Grid'5000 Architecture

The Grid'5000 architecture implements the principles described in the previous section. Based on the researchers requirements, the scalability needs and the number of researchers, we decided to build a platform of 5000 CPUs distributed over 9 sites in France. Figure 2 presents an overview of Grid'5000. Every site hosts a cluster and all sites are connected by high speed network (all links will provide 10Gbps by the end of 2005).

Numbers in the figure give the number of CPUs for every cluster. 2/3 of the nodes are dual CPU 1U racks equipped with 2 AMD Opteron running at 2 Ghz, 2 Go of memory and two 1Gpbs Ethernet Adapter. Clusters are also equipped with high speed networks (Myrinet, Infiniband, etc.). In the rest of this section we present the key architectural elements of Grid'5000.

4.1 A confined system

As discussed earlier, the Grid'5000 architecture should provide an isolated domain where communication fly without restriction between sites and are not possible directly with outside world. Mechanisms based on state-of-the-art technology like public key infrastructures and X509 certificates, produced by the Grid community to secure all resource accessed are not suitable for the Grid'5000. The GSI high level security approach imposes an heavy overhead and impacts the performances, biasing the results of study not directly related to security. Then a private dedicated network (PN) or a virtual private network (VPN) are the only solutions to compose a secure grid backbone and to build such a confined infrastructure. In Grid'5000, we choose to interconnect the sites with a combination of DiffServ and MPLS technology provided by our NREN (service provider). Many VPN implementation solutions are available but they do not provide simultaneously security and QoS guarantees. For security, network layer VPNs may use tunneling or network layer encryption (layer 3 VPN), while in link layer VPNs like MPLS, VPNs are directly provided by network service providers (layer 2-3 VPN). The advantage of the MPLS VPN over IP VPN (Ipsec) is performance. As Grid'5000 sites are connected to the same NREN, the multi-domain issue of the MPLS technology is avoided here. For performance guarantee, a combination of DiffServ and MPLS will be configured for Grid'5000 links. The Premium service will be used for delay and bandwidth guarantees required for reproducible experimental conditions and performance measurements. This MPLS based Grid architecture allows creating a trust context that even enables to experiment new security solutions for IP VPN-based Grids. Figure 3 presents the resulting communication architecture.

Using MPLS in Grid architecture is not an isolated choice. Recently, a Grid VPN research group has born within the GGF, attesting a real interest in developing and using MPLS, G-MPLS or lower level optical switching technologies for the Grid.

4.2 User view and data management

As previously mentioned, communications are done with minimal authentication between Grid'5000 machines. The logical consequence is that a user has a single account

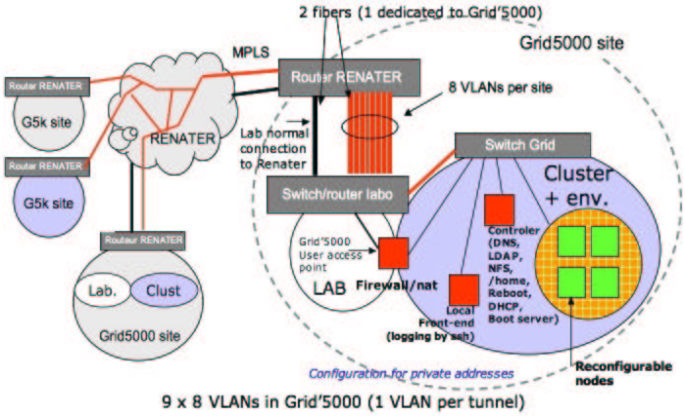

Figure 3. Communication architecture.

across the whole platform. However, each Grid'5000 site manages its own user accounts. Reliability of the authentication system is also critical. A local network outage should not break the authentication process on other sites. These two requirements have been fulfilled by the installation of an LDAP directory. Every site runs an LDAP server containing the same tree : under a common root, a branch is defined for each site. On a given site, the local administrator has read-write access to the branch and can manage its user accounts. The other branches are periodically synchronized from remote servers and are read-only.

From the user point of view, this design is transparent, once the account is created, the user can access any of the Grid'5000 sites or services (monitoring tools, wiki, deployment, etc.). His data, however, are local to every site. They are shared on any given cluster through NFS, but distribution to another remote site is done by the user through classical file transfer tools (rsync, scp, sftp, etc.). Data transfers with the outside of Grid'5000 are restricted to secure tools to prevent identity spoofing and public key authentication is used to prevent brute-force attacks.

4.3 Experiment scheduling

Experiment scheduling and resource allocation is managed by a resource management system called OAR [13] at cluster level and by a simple broker at the grid level. OAR architecture is built from a relational database engine *MySql*. All large-scale operations like parallel tasks launching, nodes probing or monitoring are performed using a specialized parallel launching tool named *Taktuk* [15]. OAR provides most of the important features implemented by other batch schedulers such as priority scheduling by queues, advance reservations, backfilling and resource match making.

At grid level, a simple broker allows co-allocating set of nodes on every selected cluster. The co-allocation process works as following: 1) user submits an experiment which needs several set of nodes on different clusters, 2) in round-robin sequence, the broker submits a reservation to each local batch scheduler. If one reservation is refused, all previous accepted reservations are canceled. When all local reservations are accepted, the user receives an identifier from the broker, allowing the user to retrieve information of allocated set of nodes.

In Grid'5000, resource management system is coupled with node reconfiguration operation at different points. First, a specific queue is defined where users can submit experiments requesting node reconfiguration. Second, there is a dynamic control of deployment rights in the prologue script is executed before starting the experiment. This gives user the capability to deploy system images on the allocated node partition. Rights are revoked in the epilogue script after the experiment. Third, after the completion of experiments involving node reconfiguration, all nodes are rebooted in a default environment. This default environment provides libraries and middleware for experiments without reconfiguration.

4.4 Node reconfiguration

Node reconfiguration operation is based on a deployment tool called *Kadeploy2*. This tool allows users deploying their own software environment on a disk partition of selected nodes. It can be viewed as a variant of system image installation tools, like SIS [**?**]. As previously mentioned, software environment contains all software layers from OS to application needed by users for theirs experiments.

Architecture of *Kadeploy2* is also designed around a database and a set of specialized operating components. The database is used to manage different aspects of the node configuration (disk partition schemes, environment deployed on every partitions), user rights to deploy on nodes, environment description (kernel, initrd, custom kernel parameters, desired filesystem for environment, associated postinstallation) and logging of deployment operations.

Several deployment procedures are available, depending mainly on OS type and filesystem specificity. We only sketch the usual deployment procedure. First, when user initiates a deploy operation, he provides an environment name allowing to retrieve asociated information from the database. The user provides this information at environment registration. Deployment begins by rebooting all nodes on a minimal system through a network booting sequence. This system prepares the targeted disk for deployment (disk partitionning, partition formatting and mounting). The next step in the deployment is the environment broadcast which uses a pipelined transfer between nodes with on the fly image decompression (previous studies have demonstrated that this topology outperforms tree based topologies for deployment system in large clusters). At this point, some ad-

justments must be done on the broadcasted environment in order to be compliant with node and site policies (mounting tables, keys for authentication, information for specific services that cannot support auto-configuration). The last deployment step consists in rebooting the nodes on the deployed system from a network loaded bootloader.

5 Grid'5000 configuration examples

The main objective of Grid'5000 set of software is to ease the deployment, execution and result collection of large scale Grid experiments. In this section, we present 3 examples for Grid'5000 reconfiguration for experiments in networking protocols, Grid middleware and GridRPC environment.

5.1 Testing recent P2P protocols in Grid context

BitTorrent is a popular file distribution system outperforming FTP performance when delivering large and highly demanded files. The key idea of BitTorrent is the cooperation of the downloaders of the same file by uploading chunks of the file to each other. As such, BitTorrent is a nice broadcast protocol for large files in data and computational Grids. BitTorent uses TCP as the transport protocol.

In this part, we describe how we can deploy, run and collect experiment results, when performing simple BitTorrent performance evaluation for a variation of TPC protocol, on homogeneous nodes of Grid'5000. The modification of the TCP stack involves the compilation and deployment of a specific OS kernel. The experiment requires 9 steps: Step 1) : BitTorrent code is instrumented to log reception and emission events (type of communication, sender identifier, receiver identifier, time and chunk identifier). BitTorrent has been instrumented to replay the logged sequence of events. Step 2) the software image is prepared (installing specific libraries and software - Python for BitTorrent), based on a minimal image certified to work on the experimental nodes. The kernel is patched and compiled with alternative TCP versions. The local root file system is then archived and registered on the deploying software database on all sites. Step 3) nodes are reserved possibly from the same selection file, using OAR. Step 4) the archived file system image is deployed on a user specified partition of all nodes, using Kadeploy. Step 5) Kadeploy reboots all the reserved nodes and checks that the machine is responding to ping and ssh. 6) The BitTorrent file to be broadcasted, is stored on the user home directory where the BitTorrent master node will run (the seeder). The list of nodes provided by OAR is stored on the BitTorrent master node. 7) Node clocks are synchronized using NTPdate. 8) A distributed launcher program controls the start of the experiment script on all the nodes. The BitTorrent tracker is started first, then the Torrent file created is registered in the tracker, then the seeder is started on the master and finally, the client (leechers) are started on all the other nodes. The BitTorrent events are recorded locally on all the nodes. 9) all log files are collected and stored in the user home directory of the user site gateway. Reserved nodes are released.

5.2 Deploying a Globus Toolkit

Globus is an open source grid middleware toolkit used for building grid systems and applications. This part describes how we can map a Globus virtual grid on Grid'5000, then how to deploy Globus and run experiments. It focuses on Globus Toolkit 2, the target Globus release used by our experiments.

The topology we choose for our virtual Globus grid is to have one Globus installation on each Grid'5000 site. We consider each site to be a separate cluster that provides services through the Globus Toolkit. Since we are emulating a grid, each cluster manages their own user accounts (i.e. no grid wide user directory). Job execution on clusters is managed by a batch job scheduler (e.g.: OAR, PBS). Each cluster manages user accounts and job scheduling with their software of choice, as we only need homogeneity inside clusters. Each site runs a certification authority (CA) that delivers user certificates for their users, as well as host certificates. We pick a front node on each site, and install Globus services on this front node. These services accept requests from other sites, authenticate and authorize them, then perform an action (e.g.: submit a job) on behalf of the client. Clients authenticate services with a host certificate delivered by the site services run on. The Gatekeeper maps user certificates to the user accounts of each cluster, and executes them with the local job scheduler. Front nodes also run the MDS (Monitoring and Discovery System) service, and GSIFTP (data transfer).

Globus toolkit is deployed by creating a system image that contains a Globus installation tailored for the experiment (since we deploy the whole system image, everything can be customized up to the operating system kernel). We create for each site an image for cluster compute nodes with a batch scheduler, and an image for the front node with the Globus Toolkit services (Gatekeeper, MDS, GSIFTP, and certificates). The virtual Globus grid is deployed on Grid'5000 machines using the Kadeploy tools, thereby turning Grid'5000 into a virtual Globus grid as long as the Kadeploy reservation lasts. While Globus users are running their experiments, log files are saved to the local drives of each node. As soon as the experiment is done, Kadeploy reboots the nodes with their default system image, and users can retrieve their log files and process them.

5.3 A Corba based Grid running DIET and TLSE

The DIET [16] middleware follows the GridRPC paradigm [17] for client-server computing over the Grid. It is designed as a set of hierarchical components (Client, Master and Local Agents, and Server Daemons). It finds an appropriate server according to the information involved in the client request (problem to be solved, size of the data involved), the performance of the target platform (server load, available memory, communication performance), and the availability of data stored during previous computations. The scheduler is distributed using several hierarchies connected either statically (in a Corba fashion) or dynamically (in a peer-to-peer fashion). The main goal of the Grid-TLSE project [19] is to design an expert site that provides an easy access to a number of sparse matrix solver packages allowing their comparative analysis on user-submitted problems, as well as on matrices from collections also available on the site. The site provides user assistance in choosing the right solver for its problems and appropriate values for the solver parameters. A computational Grid managed by DIET is used to deal with all the runs arising from user requests. Our goal in the Grid5000 project is twofold. First we want to validate the scalability of our distributed scheduling architecture at a large scale (using thousands of servers and clients) and then to test some deployments of the TLSE architecture for future production use.

In the current availability of Grid'5000 platform, the deployment of DIET with TLSE server works in three phases. The first step consists in sending one OAR requests at each site, to reserve a maximum of available nodes. The second phase consists in receiving OAR information to know which nodes are given by reservation. The third phase generates an XML file with the dynamic information as well as names of nodes at each site. This files will be used by GoDIET to deploy DIET. Our main goal during this first experience is to corroborate a theoretical study of the deployment with the hardware capability of Grid'5000 platform (CPU performance, bandwidth, etc.) to design a hierarchy that achieves a good scalability and a good efficiency for DIET. From this XML file, GoDIET deploys agents (or schedulers), servers and services bound to DIET as Corba services (i.e. naming service) and a distributed log tool designed for the visualization tools (VizDIET).

6 Deployment system evaluation

In this section, we present the evaluation of the deployment and reboot system of Grid'5000. Evaluation of other parts of Grid'5000 will be presented in future articles. The deployment and reboot system is certainly the most important mechanism of Grid'5000, enabling a rapid turnaround of experiments on the platform. A typical deployment and reboot mechanism for cluster would require tens of minutes and even hours [**?**] for managing more than a thousand nodes. Our objective is to provide a reconfiguration time (boot to boot: B2B) lower than 10 minutes for the 5000 CPUs of the platform. This means: 1) deploying the software image on all the nodes of every site (a site may contain up to 500 nodes), 2) issuing the reboot order on all Grid'5000 nodes and 3) the reboot of all nodes from the deployed software image. As previously mentioned, Kadeploy uses more steps, booting a light kernel to prepare the user partition to boot from for the experiment.

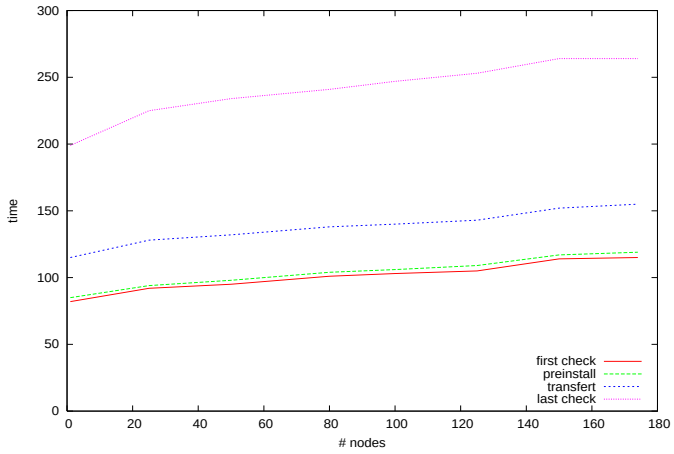

Figure 4. Time (in seconds) to deploy and boot a new OS on a cluster with Kadeploy, according to the number of nodes.

The B2B time depends not only on the performance of Kadeploy but also on the OS to be booted (OS have different configurations and run different set of services).

Figure 4 presents the BTB time according to the number of nodes, in a single site, for a simple kernel without service, on a cluster of 200 nodes.

The figure decomposes the B2T time in 4 steps: 1) the time to boot the preparation OS launching a light kernel (in red), 2) the time the prepare the disk partitions before the installation of the user environment (in dash green), 3) the time to transfer the user environment archive (in dash blue) and 4) the time to boot the user OS (in dash purple). First, the figure shows that the boot time depends on the number of nodes. This is because the boot time is different for all machine and we consider only the slowest one. In contrary, the disk preparation and environment transfer times increase negligibly with the number of nodes. The time to reboot the 2 OS largely dominates the environment transfer time. Altogether, the figure clearly shows a B2B time evolving linearly with the number of nodes following an affine function that could be evaluated as $B2B_time = 200ms + (0.33 \times X)$, X, being the number of nodes.

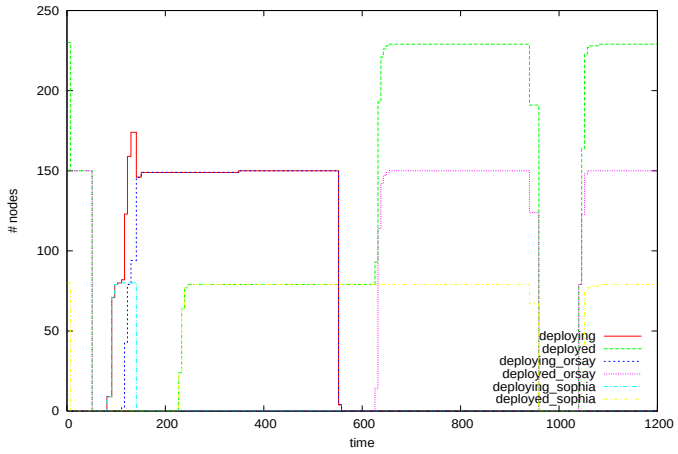

Figure 5. Time diagram for the deployment and reboot of a user environment on 2 sites of Grid'5000.

Figure 5 presents the time diagram of a deployment and reboot phase involving 2 Grid'5000 sites for a total of 230 nodes (150 nodes at Orsay and 80 nodes at Sophia). The vertical axis corresponds to the number of nodes in deployment and reboot state. On the leftmost time, all the nodes are running an OS. At $t = 15s$, a deployment sequence is issued. At $t = 60s$, all nodes are rebooting the deployment kernel. At $t = 150s$ all nodes have rebooted and are preparing the user partition. The clusters start the second reboot at $t = 150s$ for Sophia and $t = 550s$ for Orsay. Sophia nodes are rebooted with the user OS at $t = 230s$. All nodes are rebooted with the user OS (including Orsay) at $t = 650s$. At $t = 950$ a reboot order is issued making all nodes rebooting on the deployment kernel. This figure demonstrates that the current B2B time at the Grid level is only exceeding the 10 minutes mark by 10%. The deployment and reboot system is in Alpha version. It is not yet tuned and that there are many optimization opportunities.

7 Conclusion

Grid'5000 belongs to a novel category of research tools for Grid research: a highly reconfigurable, controllable and monitorable real life platform. We have presented the motivations, design and architecture of this platform. The main difference between Grid'5000 and previous real life experimental platforms is its degree of reconfiguration, allowing researchers to deploy and install the exact software environment they need for every experiment. This capability raises a security difficulty, solved in Grid'5000 by establishing a virtual domain spanning over several sites connected by the Internet, strongly controlling the communications at the domain boundaries and relaxing restrictions for intra-domain communications. The platform provides the users many network probes capturing the network traffic during experiments. We have describes some configuration examples, illustrating the variety of experiments that can be done with Grid'5000. We also presented the performance of the reconfiguration system allowing a "boot to boot" time lower than 10 minutes on the full platform.

Ongoing work concerns several points: 1) ease the software image construction for the users, 2) automatic validation of software images, 3)

8 Acknowledgments

We would like to thank the French Ministry of research and the ACI Grid and ACI Data Mass incentives, especially Thierry Priol (Director of the ACI GRID) and Brigitte Plateau (Head of the Scientific Committee of the ACI Grid). We also thank Dany Vandromme, Director of RENATER for the involvement of RENATER in this project.

References

[1] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):00–00, July 2003.

[2] Atsuko Takefusa, Satoshi Matsuoka, Kento Aida, Hidemoto Nakada, and Umpei Nagashima. Overview of a performance evaluation system for global computing scheduling algorithms. In *HPDC '99: Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing*, page 11, Washington, DC, USA, 1999. IEEE Computer Society.

[3] H. Casanova. Simgrid: a toolkit for the simulation of application scheduling. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01),Brisbane, Australia*, pages 430–437, may 2001.

[4] Henri Casanova, Arnaud Legrand, and Loris Marchal. Scheduling distributed applications: the simgrid simulation framework. In *Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, may 2003.

[5] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14(13-15), 2002.

[6] C. Dumitrescu and I. Foster. Gangsim: A simulator for grid scheduling studies. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05), Cardiff, UK*, may 2005.

[7] William H. Bell, David G. Camerona, Luigi Capozza, A. Paul Millar, Kurt Stockinger, and Floriano Zini. Optorsim - a grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing Applications,*, 17(4), 2003.

[8] Kavitha Ranganathan and Ian Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *HPDC '02: Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC'02)*, page 352, Washington, DC, USA, 2002. IEEE Computer Society.

[9] Sugree Phatanapherom, Putchong Uthayopas, and Voratas Kachitvichyanukul. Fast simulation model for grid scheduling using hypersim. In *Winter Simulation Conference (WSC '03)*, New Orleans, Louisiana, USA, 2003.

[10] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The microgrid: a scientific tool for modeling computational grids. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 53. IEEE Computer Society, 2000.

[11] Huaxia Xia, Holly Dail, Henri Casanova, and Andrew Chien. The microgrid: Using emulation to predict application performance in diverse grid network environments. In *Proceedings of the Workshop on Challenges of Large Applications in Distributed Environments (CLADE'04)*. IEEE Press, 2004. Published in conjunction with the Thirteenth IEEE International Symposium on High-Performance Distributed Computing, Honolulu, Hawaii, June 2004.

[12] Xin Liu, Huaxia Xia, and Andrew Chien. Validating and scaling the microgrid: A scientific instrument for grid dynamics. *The Journal of Grid Computing*, Volume 2(2):141 – 161, 2004.

[13] Y. Georgiou, O. Richard, P. Neyron, G. Huard, and C. Martin. A batch scheduler with high level components. In *Proceedings of CCGRID'2005*. IEEE Computer Society, 2005.

[14] W. Hoarau and S. Tixeuil. Fail: A language driven tool for fault injection in distributed systems. In *Technical report 1399, Laboratoire de Recherche en Informatique, Paris South University, France*, Feb, 2005.

[15] P. Augerat, C. Martin, and B. Stein. Scalable monitoring and configuration tools for grids and clusters. In *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*. IEEE Computer Society, 2002.

[16] E. Caron and F. Desprez. DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid. *International Journal of High Performance Computing Applications*, 2005. To appear.

[17] K. Seymour, C. Lee, F. Desprez, H. Nakada, and Y. Tanaka. The end-user and middleware apis for gridrpc. In *Workshop on Grid Application Programming Interfaces, In conjunction with GGF12*, Brussels, Belgium, September 2004.

[18] An overview of the grid-tlse project. http://www.irit.enseeiht.fr/tlse.refs.html, 2002.

[19] Michel Dayd, L. Giraud, Montse Hernandez, Jean-Yves L'Excellent, Marc Pantel, and Chiara Puglisi. An overview of the grid-tlse project. In *Proceedings of 6th International Meeting VECPAR 04*, Valencia, Spain, June 2004.