Project no. IST-033576

XtreemOS

Integrated Project
BUILDING AND PROMOTING A LINUX-BASED OPERATING SYSTEM TO SUPPORT VIRTUAL ORGANIZATIONS FOR NEXT GENERATION GRIDS

First version of XtreemOS testbed user manual D4.3.1

Due date of deliverable: November, 30, 2006
Actual submission date: December, 19, 2006

Start date of project: June 1^{st} 2006

Type: Deliverable
WP number: WP4.3

Responsible institution: INRIA
Editor & and editor's address: Yvon Jégou
IRISA/INRIA
Campus de Beaulieu
Rennes Cedex
FRANCE

Version 0.1 / Last edited by Yvon Jégou / Dec, 19, 2006

Project co-funded by the European Commission within the Sixth Framework Programme		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	06/10/27	Yvon Jégou	INRIA	Initial document
0.9	06/11/24	Yvon Jégou	INRIA	first review
1	06/11/27	Yvon Jégou	INRIA	cleanup
1.1	06/12/19	Yvon Jégou	INRIA	final comments

Abstract

Operating system developments must be validated on real hardware. The behavior of a grid operating system depends on so many parameters such as the number of nodes, their heterogeneity (memory, cpu, devices), the structure of the Grid (small or large clusters), the interconnection network (structure, latency, bottlenecks), the dynamicity of the grid, the stability of the grid (node failures), the efficiency of grid services, service bottlenecks etc. that their is no possible way to evaluate it through simulation. Grid operating systems such as XtreemOS must be validated on a realistic grid platform.

The grid testbed must provide a significant number of computation nodes for scalability evaluation. The testbed nodes must allow full reconfiguration of the software stack from the low level communication layers up to the grid services.

The Grid'5000 platform is distributed on 9 sites in France, provides a significant number of computation nodes (>1000 dual nodes end 2006, expected 2500 dual/quad nodes end 2007) and allows full reconfiguration of the node software from the operating system kernel to the grid middleware. Grid'5000 users can reserve nodes on Grid'5000 sites, define new operating system environments (kernel and distribution) and deploy these environments on the nodes.

Grid'5000 will be the initial testbed for the XtreemOS project.

This deliverable describes the Grid'5000 platform and the tools made available to users for the construction, registration, deployment and debugging of their operating system environments.

Contents

Chapter 1

Introduction

From the XtreemOS description of work document:

> *"A Grid testbed will be set up to experiment with XtreemOS software and test cases. The objective of WP4.3 is to make available a grid experimentation platform to XtreemOS partners."*

Large scale distributed systems like grids are too complex to be evaluated using theoretical models and simulators. Such systems must be experimented on real size, real life experimental platforms. In order to prove the effectiveness of the results, the experimentations must be reproducible. The purpose of WP4.3 is to setup an experimentation platform for XtreemOS. Because XtreemOS is an operating system, this platform must allow all experimentations on the whole software stack. In order evaluate the scalability of XtreemOS, this platform must offer thousands of computation nodes.

> *"All partners involved in the development of an XtreemOS component or a software related to an XtreemOS use case will use for initial tests resources already available in their institution or experimental grid platforms they have access to."*

The WP4.3 testbed is not a development platform. Each XtreemOS partner will develop and run initial tests of his software on his own platform.

> *"During the first half of the project, the French Grid'5000 infrastructure will be used by all partners as the XtreemOS grid testbed."*

Grid'5000 [2] will be the first testbed used in XtreemOS. It is an appropriate environment for experimenting XtreemOS. The main purpose of the Grid'5000 platform is to serve as an experimental testbed for research in Grid Computing. It allows experiments in all the software layers between the network protocols up to the applications including the OS layer. Further, it is already operational and already gives access to one thousand of nodes.

The extension of the initial XtreemOS testbed to DAS-3 (a grid for research in the Netherlands) and to the China National Grid (CNGrid) will be studied during the next months. The issue of mobile device testbeds will be addressed in future documents, once the requirements and specifications for XOS-MD are defined.

Chapter 2 of this document describes the Grid'5000 harware platform and chapter 3 the user tools. Chapter 4 is a small user tutorial. Appendices A and B contain the reference manuals of Grid'5000 `OAR` and `Kadeploy` user softwares.

Chapter 2

Grid'5000 platform

2.1 Motivations

The design of Grid'5000 [8] 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.

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.) and problem solving environments. Research in these layers concerns scalability (up to thousands of CPUs), performance, fault tolerance, QoS and security.

For researchers involved in protocols, OS and Grid middleware research, the software settings for their experiments often require specific OSes. Some researchers need Linux, while others are interested in Solaris10 or Windows. For networking researches, FreeBSD is preferred because network emulators like Dummynet and Modelnet run only on FreeBSD. Some research on virtualization, process checkpoint and migration need the installation of specific OS versions or OS patches that may not be compatible between different versions of the OS. Even for experiments over the OS layers, researchers have some preferences: for example some prefer Linux kernel 2.4 or 2.6 because of scheduler differences. Researcher needs are quite different in Grid middleware: some require Globus (in different versions: 3.2, 4, DataGrid version) while others need Unicore, Desktop Grid or P2P middleware. Some other researchers need to make experiments without any Grid middleware and test applications and mechanisms in a multi-site, multi-cluster environment before evaluating the middleware overhead. According to this inquiry on researchers' needs, Grid'5000 provides 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 his software image on it, reboots all the machines on this partition, runs the experiment, collects results and frees the machines. This reconfiguration capability allows all researchers to run their experiments in the software environment exactly corresponding to their needs.

As discussed earlier, the Grid'5000 architecture provides an isolated domain where communication between sites is not restricted and communication with the outside world is not possible. Mechanisms based on state-of-the-art technology like public key infrastructures and X509 certificates, produced by

the Grid community to secure all resources accessed, are not suitable for the Grid'5000. The GSI high level security approach imposes a heavy overhead and impacts the performances, biasing the results of studies not directly related to security. Therefore, 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.

Because researchers are able to boot and run their specific software stack on Grid'5000 sites and machines, it is not possible to 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 constraints increase 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, a two level security design is implemented with the following rules: a) Grid'5000 sites are not directly connected to the Internet — no communication packet from the Internet will be able to directly enter Grid'5000 and no communication packet from Grid'5000 will be able to directly exit to the Internet — and b) there is no limitation in the communication between Grid'5000 sites. The first rule ensures that Grid'5000 will resist hacker attacks and will not be used as basis of attacks (i. e. massive DoS or other more restricted attacks).

These 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. First, a strong authentication and authorization check is done to enter the lab. A second authentication and authorization check is done when logging from the lab to the Grid'5000 nodes.

Performance evaluation in grid computing is a complex issue. Speedup is hard to evaluate in heterogeneous hardware. In addition, hardware diversity increases the complexity of the deployment, reboot and control subsystem. Moreover, multiplying the hardware configurations directly increases day-to-day management and maintenance costs. Considering these 3 parameters, it has been decided that 2/3 of the total machines should be homogeneous. However, as grid are heterogeneous by definition, a reasonable amount of the machines (one third) are not homogeneous in Grid'5000.

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 conduct their experiments in a reproducible way 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). Grid'5000 software set provides a reservation tool (`OAR`, [7, 4]), a deployment tool (`Kadeploy`, [3]) and a several monitoring tools.

Local observation of processor memory, disk and network is difficult at the hardware level. Since the users can use their own 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.

2.2 Architecture

The Grid'5000 architecture implements the principles described in the previous section. Figure 2.1 presents an overview of Grid'5000. All sites are connected by high speed access links (10 Gbps by the end of 2005) to the RENATER national research and education network (NREN).

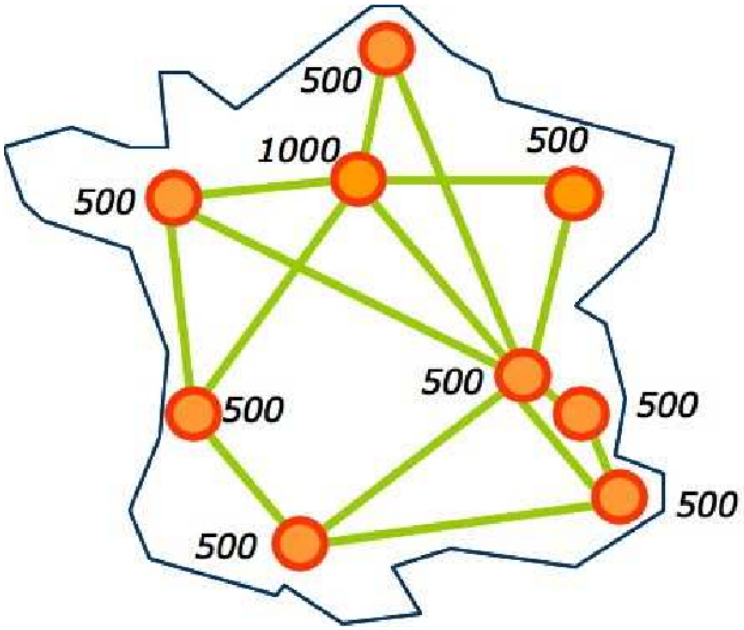


Figure 2.1: Overview of Grid'5000.

Numbers in the figure give the number of CPUs expected in 2007 for every site. 2/3 of the nodes are dual CPU 1U racks equiped with 64 bits x86 processors (AMD Opteron or Intel EM64T).

Each site hosts one or more clusters. Each cluster is equiped with one special node, the cluster head. A cluster head supports the management tools for the cluster (reservation tools, deployment tools, monitoring tools). The cluster head has the same configuration (hardware and software) as the cluster nodes. This node can be used as a development node: deployment of experiments, compilation, ... All nodes are interconnected through 1 gigabit ethernet. Other network equipments can be present on some clusters, for instance Myrinet 10g for 33 machines and Infiniband for 66 machines in Rennes.

Grid'5000 users can access all cluster heads without restrictions. Before using a cluster node, a user must reserve this node (with `OAR`, section 3.3 or GridPrems, section 3.5) for the duration of his experiment.

2.2.1 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 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 his 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 (also the pathname of his home-dir is the same on all sites). 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 outside of Grid'5000 are restricted to secure tools to prevent identity spoofing. and Public key authentication is used to prevent brute-force attacks.

2.2.2 Hardware

2.2.3 Programming environment

Maintaining a uniform programming environment accross all Grid'5000 sites would be a heavy task. Moreover, selecting a common programming environment for all users is impossible. Grid'5000 users can define and deploy their own system image. This possibility relaxes the dependency of user experimentations on site configurations. Grid'5000 sites manage their clusters independently. Each site provides a basic Linux installation on his clusters. This basic installation provides a standard programming environment: Fortran and C/C++ compilers, geb, java... Various Linux distributions are present on Grid'5000: Ubuntu at Rennes, Fedora-core at Toulouse, Debian at Grenoble...

2.2.4 Accessibility policy

As mentionned earlier, communication between Grid'5000 nodes and the outside world is very restricted. The accessibility policy considers Grid'5000 sites, hosting domains, satellite sites, internet servers and external user sites.

Grid'5000 sites. The Grid'5000 sites communicate through a private network. Except for some limited protocols, ethernet broadcast for instance, there is no limitation in inter site communications.

Hosting domains. Grid'5000 sites are hosted in public research laboratories in France. Each site has limited connectivity with his hosting domain:

- `ssh` from any Grid'5000 node to the domain portal,
- `ssh` from any domain node to local Grid'5000 node (can be restricted to the Grid'5000 portal),
- local Grid'5000 servers (not accessible to users) can use some internet services from the hosting domain: `dns`, `ntp`, `http`, `https`, `smtp`...
- web servers on the local Grid'5000 site can be accessed from the hosting domain.

Users authenticated on the information systems of these laboratories can log in their local Grid'5000 using `ssh`.

Satellite sites. It is possible to open connections with sites outside Grid'5000 for experimentation purposes. A satellite site must have a security policy equivalent to Grid'5000 policy. Non priviledged ports can be opened in both directions. Communication through priviledged ports is limited to `ssh`, `http` and `https`. Satellite sites must be aware that they can be subject to attacks from malicious Grid'5000 users. Declaring a production network as a satellite site is a bad idea.

Node types	Apple Xserve G5	Dell PowerEdge 1600SC	Dell PowerEdge 1750	HP Integrity RX2600	HP ProLiant DL145G2	IBM eServer 325	IBM eServer 326	IBM eServer 326m	Sun Fire V20z	Sun Fire X4100	Site total
Bordeaux						48					**48**
Grenoble		32		103							**135**
Lille							53	20			**73**
Lyon						56			70		**126**
Nancy					47						**47**
Orsay								342			**342**
Rennes	32		64		99				64		**259**
Sophia						105				56	**161**
Toulouse									58		**58**
Nodes total	32	32	64	103	146	209	53	362	192	56	**1249**

Table 2.1: Distribution of nodes on Grid'5000 sites

Processor types	AMD Opteron 246	AMD Opteron 248	AMD Opteron 250	AMD Opteron 252	AMD Opteron 275	Intel Itanium 2	Intel Xeon IA32	PowerPC	Site total
Bordeaux		96					64		96
Grenoble						206			270
Lille		106		40					146
Lyon	112		140						252
Nancy	94								94
Orsay	432		252						684
Rennes	198	128					128	64	518
Sophia	210				224				434
Toulouse		116							116
Nodes total	1046	446	392	40	224	206	192	64	2610

Table 2.2: Processor types on Grid'5000 sites

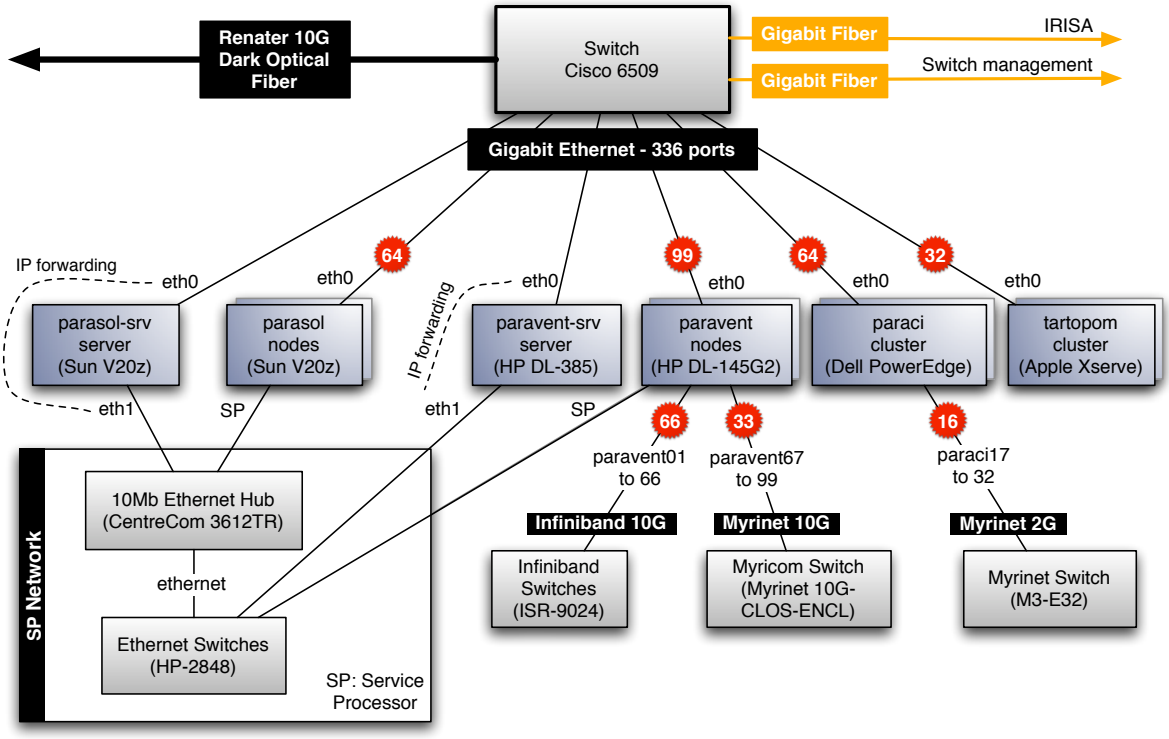


Figure 2.2: Rennes platform

Internet servers. Internet connectivity is helpful during software development and installation. For instance, direct access to Linux distribution mirroring sites facilitates system configuration. Grid'5000 site administrators can provide indirect (temporary) access to such sites through proxies.

External user sites. Grid'5000 can provide access to external users such as XtreemOS members. It is possible to open limited connectivity (`ssh+http+https`) between a Grid'5000 cluster head and an Internet IP address (or small network).

2.3 Rennes Grid'5000 site

All XtreemOS members from INRIA work in the PARIS research group in Rennes. The PARIS research group is also in charge of the Grid'5000 platform in Rennes. Rennes' platform will be the access point to the Grid'5000 testbed for XtreemOS partners.

Rennes Grid'5000 platform provides 4 clusters:

paraci cluster:	model:	Dell PowerEdge 1750
	Nb nodes:	64
	CPU:	Intel Xeon IA32 2.4 Ghz
	memory:	2 Gb
	network:	$2 * 1$ Gb ethernet
	storage:	37 Gb SCSI

parasol cluster:	model:	Sun Fire V20z
	Nb nodes:	64
	CPU:	AMD Opteron 248 2.2 Ghz
	memory:	2 Gb
	network:	$2 * 1$ Gb ethernet
	storage:	73 Gb SCSI
paravent cluster:	model:	HP Proliant DL145G2
	Nb nodes:	99
	CPU:	AMD Opteron 246 2.0 Ghz
	memory:	2 Gb
	network:	$2 * 1$ Gb ethernet
		33 nodes with Myrinet 10G
		66 nodes with Infiniband
	storage:	80 Gb SATA
tartopom cluster:	model:	Apple Xserve G5
	Nb nodes:	32
	CPU:	PowerPC 2.0 Ghz
	memory:	1 Gb
	network:	$2 * 1$ Gb ethernet
	storage:	80 Gb IDE

The parasol and paravent clusters are equiped with remote management capacity: users can deploy their own operating system distribution on these two clusters.

The Cisco 6509 switch of the platform is in charge of separating IP traffic:

- traffic to other Grid'5000 sites is routed to the 10G fiber,
- all other allowed traffic is routed to the laboratory (IRISA) router which supports Internet connectivity,
- the rest of the paquets are dropped.

In order to detect network misconfiguration, the source of incoming traffic is also checked: only traffic from Grid'5000 sites is accepted from the 10G fiber and paquets from Grid'5000 sites received from the Internet (laboratory) fiber are dropped.

2.4 Grid'5000 interconnection

RENATER is the French National Telecommunication Network for Technology, Education and Research. More information can be found on RENATER's web site [5]. RENATER offers about 30 POPs (Points Of Presence) in France, at least one POP for each region, on which metropolitan and regional networks are connected. More than 600 sites (universities, research centers, ..) are interconnected through RENATER.

The network, in its current phase (RENATER-4), was completely deployed in November 2005. The "standard" architecture is based on 2,5Gbit/s leased lines and provides IP transit connectivity, interconnection with GEANT-2 [1], overseas territories and the SFINX (Global Internet exchange) (figure 2.3).

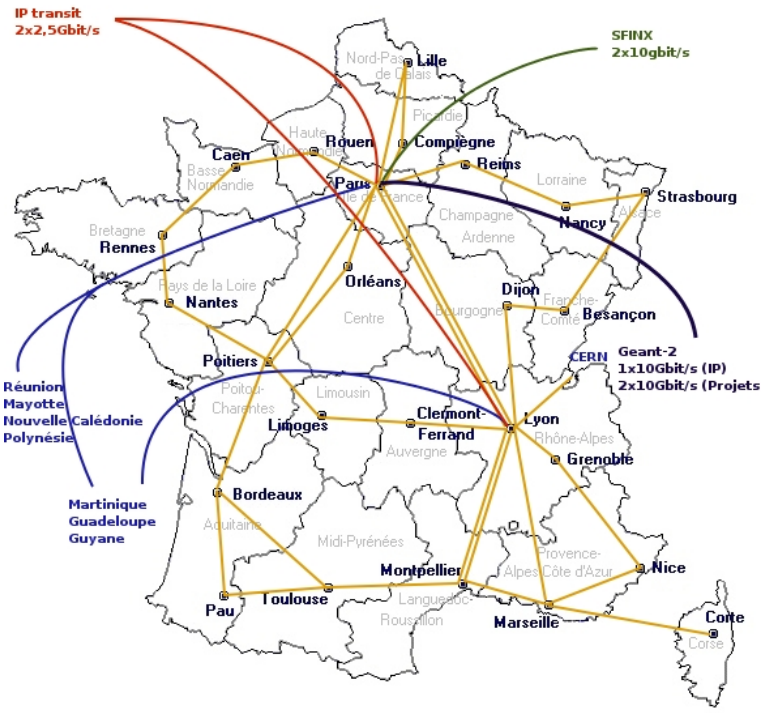


Figure 2.3: Renater network

The initial design of Grid'5000 sites interconnection has been adressed within the RENATER backbone using a Ethernet Over MPLS (EoMPLS) solution. It is a full mesh topology based on MPLS tunnels (LSPs) established between the RENATER POPS on which are connected the Grid'5000 sites. In practice sites are interconnected through 1Gbit/s VLANs.

RENATER-4 also introduced a dark fibre infrastructure allowing to allocate dedicated 10Gbit/s "lambdas" for specific research projects. It also provides interconnection with GEANT-2, with increased capacity compared to GEANT-1 and dedicated interconnection for projects.

This new infrastructure is still under deployment and Grid'5000 sites will be connected to it, one by one, as soon as it becomes available on the local POPs. Six Grid'5000 sites are currently (October 2006) connected to their local POP with the dark fibre infrastructure, three of them at 10Gbit/s – Rennes, Nancy, Sophia – and three others at 1Gbit/s. With the dark fibre infrastructures, Grid'5000 sites will eventually see each other inside the same VLAN (10Gbit/s).

Figure 2.4 shows the current state of the architecture (along with the maximun number of lambdas that can be allocated on each link).

Inside the RENATER POPs, Grid'5000 sites are directly connected to the switches (see figure 2.5), bypassing the routers which are used for "standard" IP traffic (and Grid'5000 sites that are still using the EoMPLS initial solution).

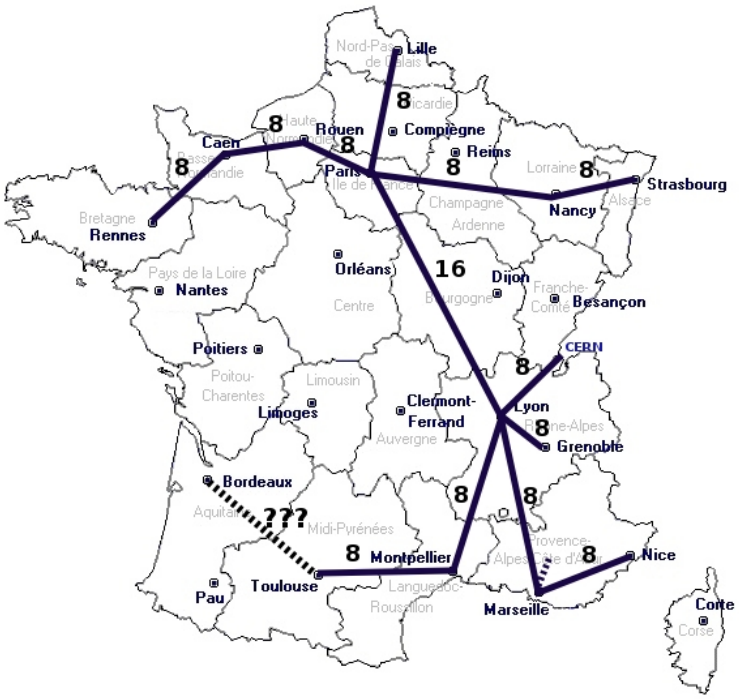


Figure 2.4: Grid'5000 network

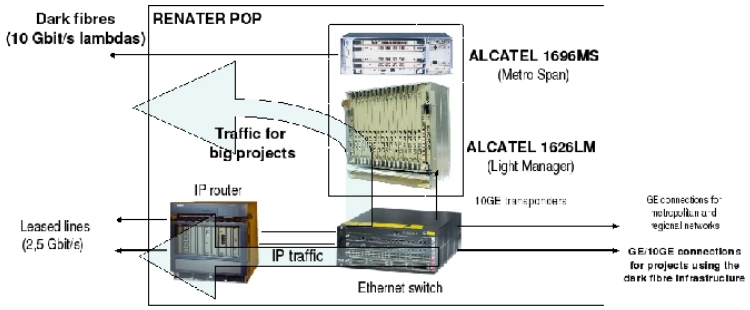


Figure 2.5: Traffic separation in RENATER POPs

Chapter 3

User tools

3.1 Grid'5000 user portal

The official Grid'5000 web site (see figure 3.1) is accessible from Internet at <https://www.grid5000.fr/>. This site gives access to the real time status of the platform (see figure 3.2), the monitoring tools, the job reservations on each site, the reservation history and the software manuals.

3.2 Grid'5000 monitoring tools

Various views of the platform state can be obtained from the Grid'5000 web site: global view (figure 3.2) or ganglia view (figure 3.3). User-deployed nodes are not monitored by ganglia in general, unless a ganglia configuration has been provided in the deployed environment: only nodes running the basic Linux environment are monitored.

3.3 OAR

OAR is the resource manager (or batch scheduler) used to manage Grid'5000 platform's resources. OAR allows cluster users to submit or reserve nodes either in an interactive or a batch mode. Administrators can tune OAR to force an exploitation policy. See figures 3.4 and 3.5 for two screenshots of OAR visualization tools.

This batch system is built on a database (MySql), a script language (Perl) and an optional scalable administrative tool (component of Taktuk framework, [6]). It is composed of several modules which interact only with the database, and are executed as independent programs. So formally, there is no API, the system is completely defined by the database schema. This approach eases the development of specific modules. Indeed, each module (such as schedulers) may be developed in any language having a database access library.

The main features of OAR are:

- Batch and Interactive jobs
- Admission rules

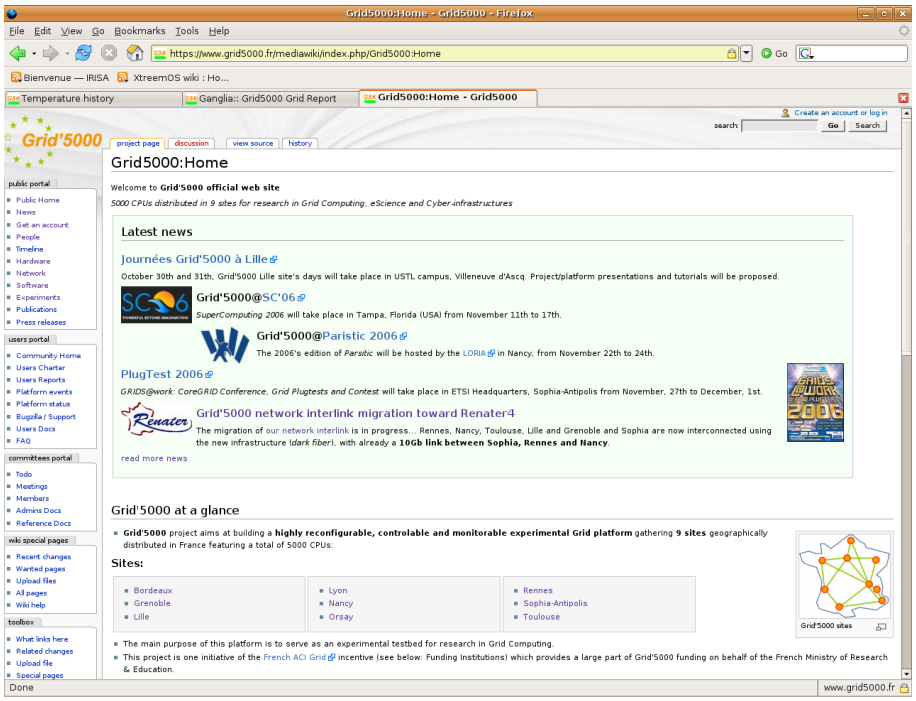


Figure 3.1: Grid'5000 user portal

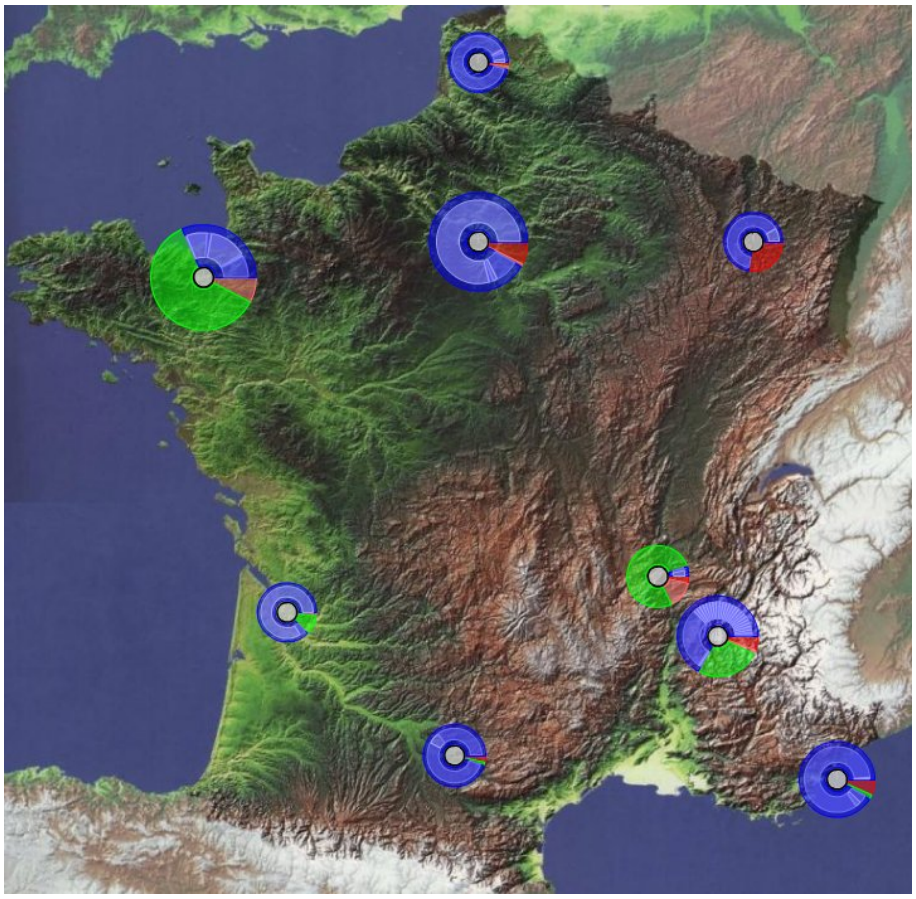


Figure 3.2: Grid'5000 status

Figure 3.3: Ganglia platform monitoring

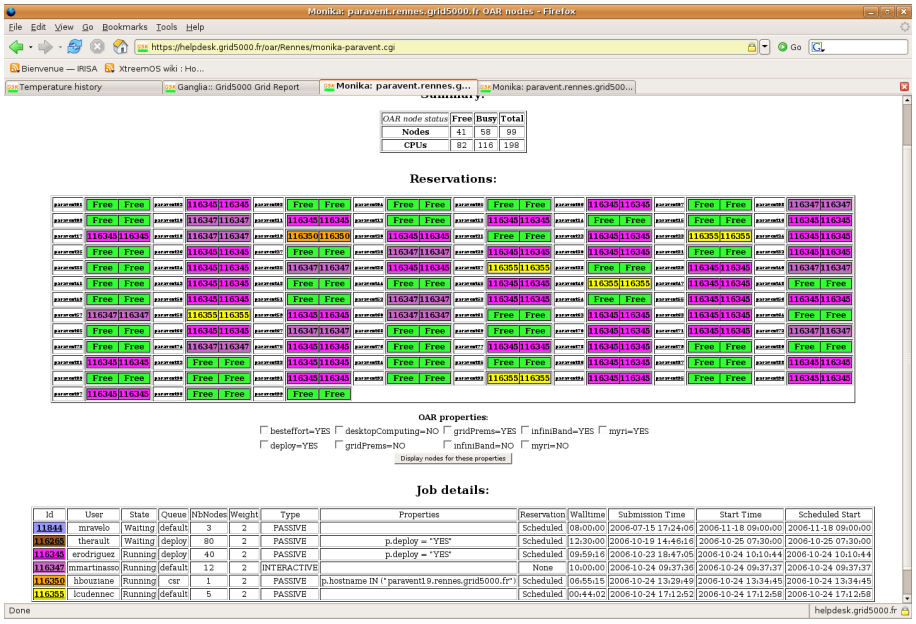


Figure 3.4: OAR queue

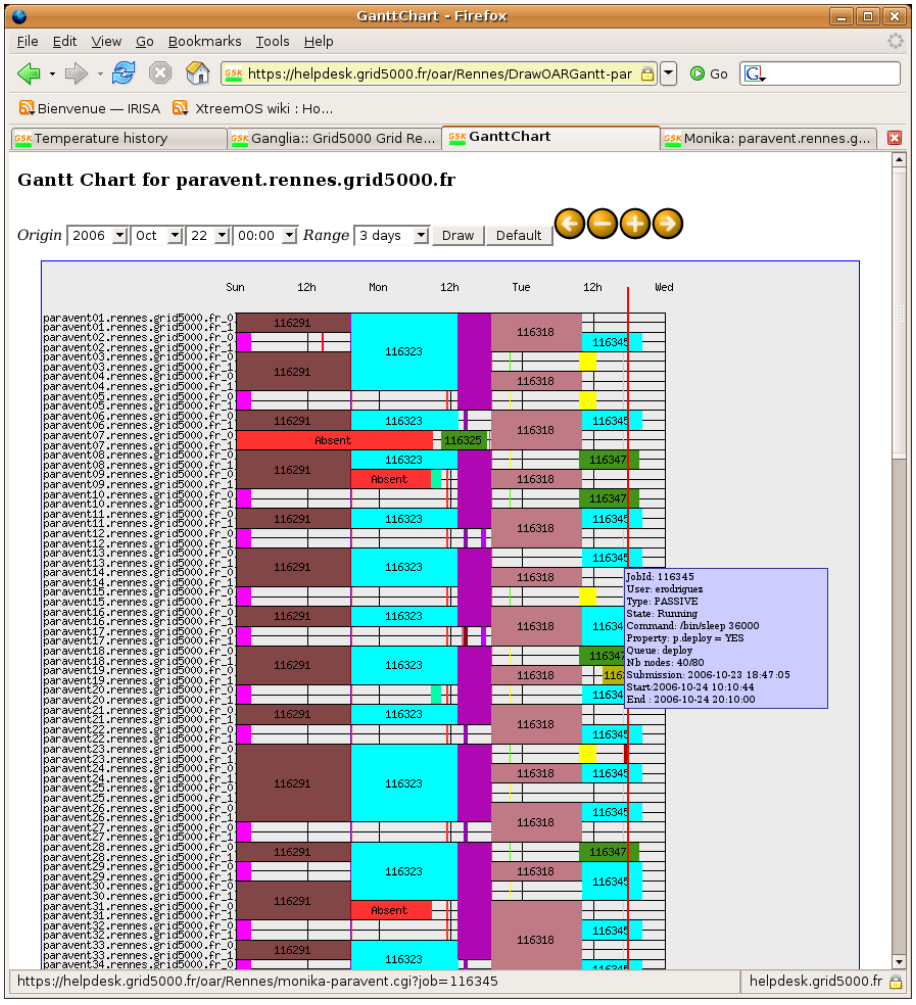


Figure 3.5: Gantt chart reservation history

- Walltime
- Matching of resources (job/node properties)
- Hold and resume jobs
- Multi-scheduler support (simple fifo and fifo with matching)
- Multi-queues with priority
- Best-effort queues (for exploiting idle resources)
- Check compute nodes before launching
- Epilogue/Prologue scripts
- Activity visualization tools (Monika)
- No Daemon on compute nodes
- rsh and ssh as remote execution protocols (managed by Taktuk)
- Dynamic insertion/deletion of compute node
- Logging
- Backfiling
- First-Fit Scheduler with matching resource
- Advance Reservation
- Environnement of demand support (Ka-tools integration)
- Grid integration with Cigri system
- Simple Desktop Computing Mode

The main `OAR` commmands are:

`oarsub` submit a job to the scheduler,

`oarstat` get job state,

`oarnodes` get informations about cluster nodes,

`oardel` delete a job.

`OAR` manages interactive and batch jobs. The standard output of a batch job is returned in file `OAR.`*BaseName*`.`*JobId*`.stdout` in the user homedir. Errors are in file `OAR.`*BaseName*`.`*JobId*`.sterr`.

3.3.1 OAR examples

Allocation of 1 node in interactive mode

```
yjegou@parasol-dev:~$ oarsub -l nodes=1 -I
Host:Port = parasol-dev:33022
 IdJob = 29739
Interactive mode : waiting
yjegou@parasol08:~$
```

On success of an interactive reservation request, the user is automatically logged on the first allocated node (`parasol08` in this example). The reservation ends when the user exits this initial shell.

Running `/bin/date` in batch mode, on a dynamically allocated node

```
yjegou@parasol-dev:~$ oarsub -l nodes=1 /bin/date
IdJob = 29741
yjegou@parasol-dev:~$
```

Result of job execution in `OAR.`*BaseName*`.`*JobId*`.stdout`:

```
yjegou@parasol-dev:~$ cat OAR.date.29741.stdout
Fri Nov 24 11:29:10 CET 2006
yjegou@parasol-dev:~$
```

Job submission in the future and interactive connection to the job once it is running

```
yjegou@parasol-dev:~$ oarsub -I -l "nodes=2,walltime=1:00:00" \
                             -r "2006-11-24 15:17:00"         \
                             /home/rennes/yjegou/sleep.sh
Host:Port = parasol-dev:33147
 IdJob = 29767
Reservation mode : waiting validation
Reservation valid --> OK
yjegou@parasol-dev:~$
...
```

Once his reservation has started, the user can request for an interactive shell on the first node of the reservation:

```
parasol-dev:~$ oarsub -I -c 29767
Connect to OAR reservation 29767 via the node parasol29.rennes.grid5000.fr
yjegou@parasol29:~$
```

Getting job status

```
parasol-dev:~$ oarstat -j 29767
Job Id: 29767
     queueName = default
     reservation = Scheduled
```

```
      submissionTime = 2006-11-24 15:15:31
      accounted = NO
      bpid =
      state = Running
      user = yjegou
      weight = 2
      startTime = 2006-11-24 15:17:54
      nbNodes = 2
      command = /bin/sleep 3600
      checkpoint = 0
      jobType = PASSIVE
      message =
      properties =
      maxTime = 00:59:06
      idFile =
      autoCheckpointed = NO
      stopTime = 0000-00-00 00:00:00
      launchingDirectory = /home/rennes/yjegou
      infoType = parasol-dev:33147

parasol-dev:~$
```

Deleting a job

```
parasol-dev:~$ oardel 29767
Deleting the job = 29767 ...REGISTERED.
The job(s) [ 29767 ] will be deleted in a near futur.
parasol-dev:~$
```

Submitting a moldable job. It is possible to use several "`-l`" `oarsub` option (one for each moldable description). By default the `OAR` scheduler will launch the moldable job which will end first. The scheduler can decide to start a job later because they will have more free resources and the job walltime will be smaller.

Best effort jobs. A best effort job will be launched only if there is enough resources and will be deleted if another job wants its resources.

Example:

```
oarsub -t besteffort /path/to/prog
```

3.4 Kadeploy

Kadeploy `Kadeploy` is the deployment system which is used by Grid'5000. It is a fast and scalable deployment system for cluster and grid computing. It provides a set of tools for cloning, configuring (post installation) and managing a set of nodes. Currently it successfully deploys linux, *BSD, Windows and Solaris on x86 and 64 bits computers.

The deployment system is composed of several tools or commands whose aim is the execution of all the needed actions to carry out deployments and other administrative tasks.

The environment to be deployed can either already exist and be registered in the database, or already exist but not be registered, or neither exist nor be registered. The first case is the simplest since nothing has to be done prior to the deployment itself. In the other cases, `kacreateenv` (`kaarchive`/`karecordenv`) are used to create and register (`create`/`register`) an environment in the database. The deployment itself i.e. of a given environment on target partitions of a set of cluster nodes is done using the `kadeploy` tool. A complete deployment is composed of the following steps:

- reboot on the deployment kernel via `pxe` protocol
- pre-installation actions (bench, partitionning and/or file system building if needed etc.)
- environment copy
- post-installation script sending and execution
- reboot on the freshly installed environment

If the deployment fails on some nodes, these are rebooted on a default environment. `Kadeploy` allows the customization of each node by 2 means:

- preinstallation script, executed before sending the system image
- postinstallation script, executed after having sent the system image

Originally, these two scripts are written in `ash`, which is a lightweight `bash`, but the way these scripts are designed could allow to add any script language.

Preinstallation script. This script is common to all environments. Its goal is to prepare the system to the hardware specification and the target hard disk drive for the deployment. It can load a specific IDE controller driver, improve deployment performance or make every kind of checks needed. This script is defined in the configuration file as `pre_install_script` and the associated archive as `pre_install_archive`. The preinstallation archive is a gzipped tar archive, containing the `pre_install_script` in its root directory. The directory structure allows to custom the tasks to the user needs.

Postinstallation script. This script is associated to the environment to deploy. Its goal is to adapt the raw system image to a bootable system. It is composed of a gunzipped tar archive that contains all the site files and a script `traitement.ash` in the archive's root directory. This archive is sent to the nodes, decompressed in a ramdisk and then the post_install_script is executed on every node. The script name is defined in the configuration file as `post_install_script`.

System modifications.

`/etc/fstab`	- a site-dependant `/etc/fstab` base file should be copied in the postinstall archive.
`/tmp`	- should have its rights modified.

Administrative aspects. Many other basic files can be handled by putting them in the archive and replacing the existing ones. Only the most important ones are listed here:

`/root/.ssh/authorized_keys`
- should contain the administrator's public key and the public key of the user, to allow him to get a root shell on every node. In order to be stored on the node images, the authorized_keys file has to be built and put in the postinstallation archive's root directory.

`/etc/hosts /etc/hosts.allow /etc/hosts.deny`
- should be set to fit the cluster's configuration, and ensure network connection within the cluster's network.

Various modifications can be applied by the postinstallation script, from authentification server to tailored modification depending on the node's IP. It's a good idea to modify `rc` scripts to prevent the first boot hard disk drive verification, because it is just a waste of time, and avoid all the manual intervention that could occur on system boot: for example, by default, many distributions ask the root password before checking a filesystem at boot time.

Two remote management tools are available to diagnose and, if possible, take control of cluster nodes that are in an unstable or undefined state after a deployment failure.

The `kaconsole` tool enables to open a console on a remote node. It needs special hardware equipment and special command configuration.

The `kareboot` tool enables to do various reboot types on cluster nodes. The possibilities are numerous: the reboot on a given (already installed) environment or on a given partition or even on the deployment kernel for instance. It also enables to hard reboot nodes appropriately equiped.

`kaaddnode` and `kadelnode` enable to add and remove nodes from the deployment system if the cluster composition changes.

3.5 GridPrems

GridPrems is a resource reservation middleware used on the clusters in Rennes. GridPrems is a web application implemented using PHP and MySQL.

When making a reservation through GridPrems, one can specifie:

- Which nodes to use. Nodes can be chosen by the user, or automatically selected.
- When to reserve. Reservations may be repeated daily or weekly.
- Whether the reserved machines can be shared. Exclusive reservations are typically used for benchmarking, whereas shareable reservations are used for testing and debugging programs.

GridPrems is not a batch scheduler. Therefore, it does not automatically run jobs on the reserved nodes. It is the user's duty to connect to the reserved nodes and perform his tasks.

Figure 3.6: GridPrems

3.6 Getting an account

To get a user account on the Grid'5000 platform, XtreemOS members should send a mail to <mailto:xtreemos-projectleader@irisa.fr> with the following informations:

- First name, last name,
- ssh public key in a file attachement,
- home fixed IP (single IP, or address range),
- entrypoints on Grid'5000 (all nodes / only clusters heads),
- requested protocols (ssh, http, https, non priviledged),
- in the case where connectivity to all nodes, or non priviledged-protocols have been requested: the security policy on the home site must be described.

Chapter 4

Grid'5000 deployment tutorial

This short tutorial describes, step by step, how a Grid'5000 user connects to a Grid'5000 site, locates a standard deployable operating system image, tunes this image to his needs, registers this new environment and deploys it on Grid'5000 nodes.

4.1 Connection to Grid'5000

The user must first connect to the frontend node of the cluster he is granted access to:

```
outside: ssh yjegou@parasol-dev.rennes.grid5000.fr
yjegou@parasol-dev:~$
```

If needed, the user may then internally jump onto the frontend of another cluster following his deployment needs:

```
yjegou@parasol-dev:~$ssh oar.another_site.grid5000.fr
```

4.2 Finding and deploying existing images

From the cluster frontend, the user can locate registered environments, find deployable partitions, make reservations on deployable nodes and deploy his own operating system image using kdeploy.

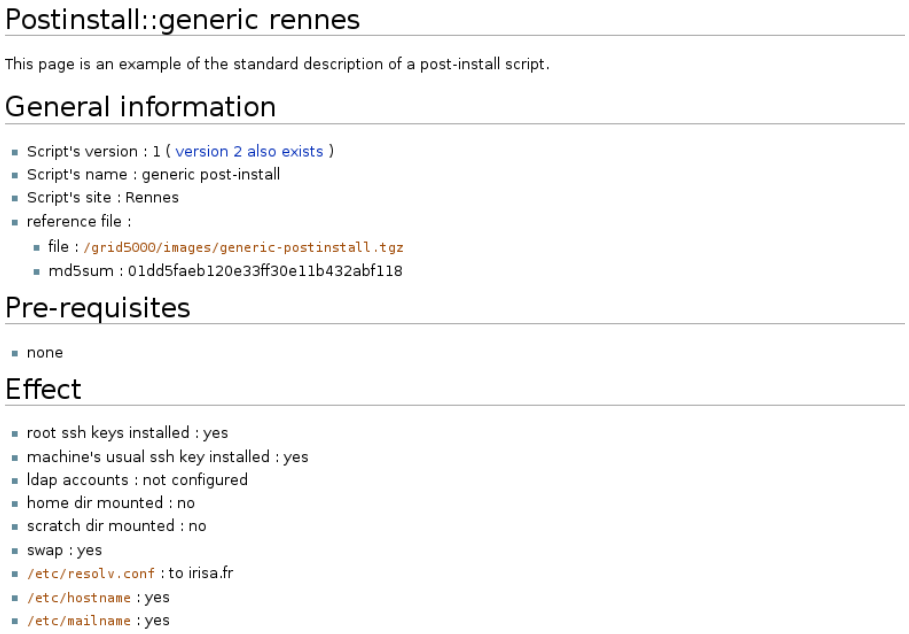
Image localization

Each site maintains an environment library in the `/grid5000/images` directory of the `OAR` node. Image descriptions can be located from the Grid'5000 wiki at

```
Sidebar > User Docs > Kadeploy > Kadeploy environments directory
```

The following environments are currently available:

Debian4all: Debian environment that should comply with all basic needs, and all sites. This environment is to be taken as the base to build more complex Debian environments.



Postinstall::generic rennes

This page is an example of the standard description of a post-install script.

General information

- Script's version : 1 (version 2 also exists)
- Script's name : generic post-install
- Script's site : Rennes
- reference file :
  - file : /grid5000/images/generic-postinstall.tgz
  - md5sum : 01dd5faeb120e33ff30e11b432abf118

Pre-requisites

- none

Effect

- root ssh keys installed : yes
- machine's usual ssh key installed : yes
- ldap accounts : not configured
- home dir mounted : no
- scratch dir mounted : no
- swap : yes
- /etc/resolv.conf : to irisa.fr
- /etc/hostname : yes
- /etc/mailname : yes

Figure 4.1: Rennes postinstall wiki page

Debian4RMS: Debian environment with several RMS aka batch scheduler systems installed:

- `OAR` + Cigri
- Torque + MAUI
- SGE/Sun One Grid Engine
- ...

Debian4kexec: Debian environment with kexec tool for testing purposes.

Globus4: The globus 4 Toolkit has been deployed on the debian4all image. It is shipped with scripts to automate configuration.

Ubuntu4all: Ubuntu environment that should comply with all basic needs, and all sites. This environment is to be taken as the base to build more complex Ubuntu environments. Available for x86_64 and also for i386

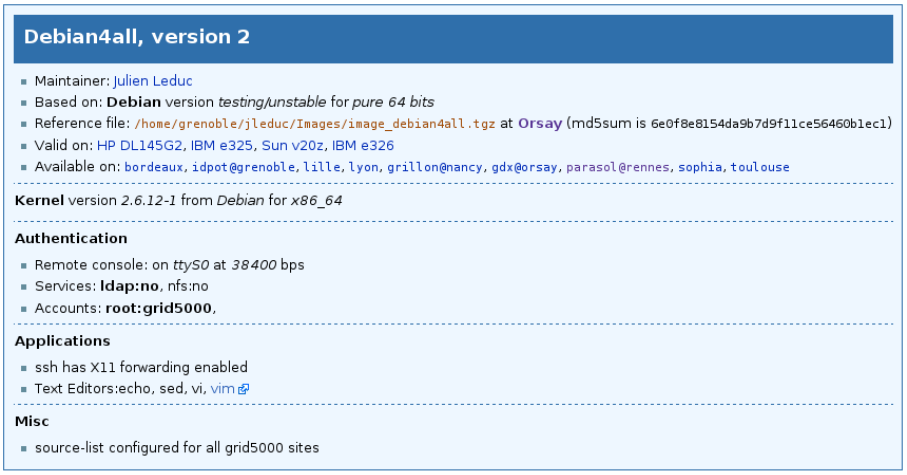
Fedora4all: Fedora environment that should comply with all basic needs, and all sites. This environment is to be taken as the base to build more complex Fedora environments.

Rocks4all: Rocks environment that should comply with all basic needs, and all sites. This environment is to be taken as the base to build more complex Rocks environments.

Each environment is described on a wiki page: reference file, valid platforms and sites... Figure 4.2 shows the wiki page of `Debian4all`.

To deploy an environment, the user must know its name as registered in the `Kadeploy` database. It is the first information on the environment description page. This tutorial uses the debian4all environment.

Being able to reproduce the experiments that are done is a desirable feature. It is possible to check the exact version of the environment present on the target cluster using `kaenvironments` on the cluster frontent:



Debian4all, version 2

- Maintainer: Julien Leduc
- Based on: **Debian** version *testing/unstable* for *pure 64 bits*
- Reference file: /home/grenoble/jleduc/Images/image_debian4all.tgz at **Orsay** (md5sum is 6e0f8e8154da9b7d9f11ce56460b1ec1)
- Valid on: HP DL145G2, IBM e325, Sun v20z, IBM e326
- Available on: bordeaux, idpot@grenoble, lille, lyon, grillon@nancy, gdx@orsay, parasol@rennes, sophia, toulouse

Kernel version *2.6.12-1* from *Debian* for *x86_64*

Authentication

- Remote console: on *ttyS0* at *38400* bps
- Services: **ldap:no**, nfs:no
- Accounts: **root:grid5000**,

Applications

- ssh has X11 forwarding enabled
- Text Editors:echo, sed, vi, vim

Misc

- source-list configured for all grid5000 sites

Figure 4.2: Debian4all wiki page

```
yjegou@parasol-dev:~$ kaenvironments --environment debian4all
#debian4all v2

name = debian4all
id = 108
version = 2
description = Image Debian 3.1 minimale generique
author = julien.leduc@lri.fr
filebase = file:///grid5000/images/debian4all.x86_64-2.tgz
filesite = file:///grid5000/images/generic-postinstall-v3.tgz
size = 1000
initrdpath = /initrd.img
kernelpath = /vmlinuz
kernelparam =
fdisktype = 131
filesystem = ext2
siteid = 1
optsupport = 0
user = kadeploy
yjegou@parasol-dev:~$
```

The user can also check that the filebase file is the expected one by checking its name and its checksum:

```
yjegou@parasol-dev:~$ md5sum /grid5000/images/debian4all.x86_64-2.tgz
6e0f8e8154da9b7d9f11ce56460b1ec1 /grid5000/images/debian4all.x86_64-2.tgz
yjegou@parasol-dev:~$
```

A post-install script adapts an environment to the site it is deployed on. In the same way as for environments, a user should be able to find a description of the post-install script for each site. For instance, see figure 4.1 for the postinstall description on the Grid'5000 Rennes site.

4.3 Making a reservation on a deployable node

For this example of the tutorial, reservation made are interactive (`-I`), on the deploy queue (`-q deploy`), on only one machine (`-l nodes=1`). The reservation is limited to 4 hours with `-l walltime=4`.

```
yjegou@parasol-dev:~$ oarsub -I -q deploy -l nodes=1 -l walltime=4
Host:Port = parasol-dev:32788
 IdJob = 29702
Interactive mode : waiting

Message from oar@parasol-dev on (none) at 14:48 ...

Your Kadeploy reservation is beginning (oar job id: 29702).
You can deploy your image on nodes: parasol32.rennes.grid5000.fr
Kadeploy partitions are sda3
The escape sequence for kaconsole is ^.
yjegou@parasol-dev:~$
```

For a reservation on the deploy queue, a new shell is opened on the frontend node and not on the first machine of the reservation as for standard reservations. When this shell exits, the reservation ends. The shell is populated with `OAR_*` environment variables:

```
yjegou@parasol-dev:~$ printenv|grep OAR
OAR_JOBID=29702
OARDIR=/opt/oar
OAR_NODENUM=1
OAR_USER=yjegou
OAR_WORKDIR=/home/rennes/yjegou
OARUSER=oar
OAR_NB_NODES=1
SUDO_COMMAND=/bin/sh -c /opt/oar//oarexecuser.sh /tmp/OAR_29702\
 1 29702  yjegou /bin/bash /home/rennes/yjegou I
OAR_FILE_NODES=/tmp/OAR_29702
OAR_NODEFILE=/tmp/OAR_29702
OAR_NODECOUNT=1
OAR_O_WORKDIR=/home/rennes/yjegou
yjegou@parasol-dev:~$
```

As usual, if the reservation is successfull, the reserved machine names are stored in file `$OAR_FILE_NODES`:

```
yjegou@parasol-dev:~$ cat $OAR_FILE_NODES
parasol32.rennes.grid5000.fr
parasol32.rennes.grid5000.fr
yjegou@parasol-dev:~$
```

4.4 Deploying an environment

First, the user needs to select a deployable partition. To this end, each site describes the partition scheme it uses in its "message of the day", that is displayed upon succesful login.

```
yjegou@parasol-dev:~$ cat /etc/motd
...
-- This cluster is "parasol": 64 dual Opteron nodes, 2.2Ghz --
  Node names: parasol{01..64}.rennes.grid5000.fr
  Operating System: Ubuntu 5.04 32/64bit
  Deploy Partitions: sda3 <============== NEW 21/09/2006
  Local scratch dir: /local (sda5)
  NFS scratch dir: /site/data0
  Disk quotas active on /home (check your status with 'quota')
yjegou@parasol-dev:~$
```

The deploy partition should be selected from the deploy partitions list, in general `hda3` or `sda3`, according to the most recent universal partition scheme that has been adopted on Grid'5000.

```
yjegou@parasol-dev: kadeploy -e debian4all \
                             -m parasol32.rennes.grid5000.fr \
                             -p sda3
Checking variable definition...
Checking database access rights...
WARNING : there is no environment named debian4all with user yjegou
specified env debian4all does not exist for user yjegou
searching for env debian4all with user \texttt{kadeploy}
Checking user deployment rights...
invalidating deployments older than 1000
Warning 0 deployment have been corrected automatically
Checking command definition...
child
Forking child 10771
Checking variable definition...
OK
kernel duke-vmlinuz.x86_64, initrd  from 84 to 84
nmapCmd: /usr/bin/nmap
WARNING : there is no environment named debian4all with user yjegou
Waiting for all the nodes to reboot during 200 seconds
<BootInit 0>
there on check:  131.254.202.132
<BootInit 1>
All nodes are ready!
First Check: 128
 /usr/local/bin/DKsentinelle -lroot -crsh -c timeout=10000 -v             \
                             -m131.254.202.132 -- " umount /mnt/dest "
<PreInstall>
Retrieving preinstall
Command Mcat: /opt/kadeploy//bin/mcat_rsh.pl -p 14052                    \
              -sc "cat /opt/kadeploy/scripts/pre_install.tgz"            \
              -dc "cat > /pre_pipe"tranfert OK
Use of uninitialized value in string eq at /opt/kadeploy//bin/kadeploy   \
line 376.
Command Mcat: /opt/kadeploy//bin/mcat_rsh.pl -p 14053                    \
              -sc "/opt/kadeploy//sbin/kasetup -printpreinstallconf"     \
              -dc "cat > /mnt/rambin//preinstalldisk.conf"tranfert OK    \
Executing preinstall... /usr/local/bin/DKsentinelle -lroot -crsh         \
              -c timeout=10000 -v  -m131.254.202.132
              -- " /mnt/rambin//init.ash fdisk "
```

```
Done
Preinstall: 0
<Transfert>
filebase: //grid5000/images/debian4all.x86_64-2.tgz
Formatting destination partition /dev/sda3 on the nodes...                  \
              /usr/local/bin/DKsentinelle -lroot -crsh -c timeout=10000 -v \
              -m131.254.202.132 -- " mkfs -t ext2 /dev/sda3 "
 /usr/local/bin/DKsentinelle -lroot -crsh -c timeout=10000 -v              \
              -m131.254.202.132 -- " mount /dev/sda3 /mnt/dest "
Done
<tar Transfert>
Sending Computing environment to the nodes...
Command Mcat: /opt/kadeploy//bin/mcat_rsh.pl -p 14054                       \
              -sc "cat //grid5000/images/debian4all.x86_64-2.tgz"           \
              -dc "cat > /dest_pipe"tranfert OK
Done
Transfert: 15
<PostInstall>
Executing postinstall...Command Mcat: /opt/kadeploy//bin/mcat_rsh.pl        \
            -p 14055 -sc "cat //grid5000/images/generic-postinstall-v3.tgz"\
            -dc "cat > /post_pipe"tranfert OK
 /usr/local/bin/DKsentinelle -lroot -crsh -c timeout=10000 -v               \
            -m131.254.202.132 -- " /rambin/traitement.ash /rambin "
Done
Postinstall: 10
Umounting fs...  /usr/local/bin/DKsentinelle -lroot -crsh -c timeout=10000 \
            -v  -m131.254.202.132 -- " umount /dev/sda3  "
Done
No kernel parameter, taking default ones defined in the configuration file
generating pxe
[VERBOSE] fork process for the node 131.254.202.132
[VERBOSE] Execute command : rsh -l root 131.254.202.132 reboot_detach
[VERBOSE] job 11652 forked
[VERBOSE] Child process 11652 ended : exit_value = 0, signal_num = 0,       \
          dumped_core = 0
131.254.202.132 : GOOD (0,0,0)
rebooting the nodes...
Transfert: 21
nmapCmd: /usr/bin/nmap
You want this node: parasol32.rennes.grid5000.fr
Waiting for all the nodes to reboot during 300 seconds
<BootEnv 0>
there on check:  131.254.202.132
<BootEnv 1>
Deployment finished!
<Completed>
Last Reboot: 121

Deploy  State
------  -----
14742   terminated

Node    State           Error Description (if any)
----    -----           --------------------------
```

```
parasol32.rennes.grid5000.fr    deployed



Sumary:
        first reboot and check: 128
        preinstall:0
        transfert: 21
        last reboot and check: 121
yjegou@parasol-dev:~$
```

Node discovery can be automated:

```
yjegou@parasol-dev: kadeploy -e debian4all \
                             -m `head -1 $OAR_FILE_NODES` \
                             -p sda3
```

Once `kadeploy` has run successfully, the allocated node is deployed under the requested ("debian4all") environment. On most environments, there is at least one configured account for login through `ssh` (`kadeploy` checks that `sshd` is running before declaring success of a deployment). This account parameters (`user/password`) can be found in the environment description page.

```
yjegou@parasol-dev: ssh root@parasol32
Password:
Last login: Thu Nov 23 15:22:24 2006 from parasol-dev.irisa.fr
Linux (none) 2.6.12-1-amd64-k8-smp #1 SMP Wed Sep 28 02:57:49 CEST 2005 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Nov 23 15:22:51 2006 from parasol-dev.irisa.fr on pts/0
(none):~# uname -a
Linux (none) 2.6.12-1-amd64-k8-smp #1 SMP Wed Sep 28 02:57:49 CEST 2005 \
x86_64 GNU/Linux
(none):~# cat /etc/issue
Debian GNU/Linux testing/unstable \n \l
(none):~#
```

4.5 Tuning the environment to build another one: customizing authentification parameters

Once a standard environment has been successfully deployed, it it possible to tune/modify it and to record the result in a new environment. It is possible to log in the deployed environment through `ssh` and the default account (see exemple above) or through a machine console.

In case of connection difficulty, there is a troubleshooting on wiki page

```
Sidebar > User Docs > Kadeploy > Kadeploy documentation > \
          Deployment troubleshooting guide
```

When playing with unstabble environments, it is possible to loose services such as `sshd` or `rshd` and then become unable to get a shell on the deployed machine. An alternative solution to the connection through `ssh` is to open a machine console using `kaconsole` command (for the OS, it is seen as physical access, thanks to remote console capabilities) provided nobody else has already opened a write access to the remote console and that the post-install script has remote console properly configured:

```
yjegou@parasol-dev:~$ kaconsole -m parasol22.rennes.grid5000.fr
Checking variable definition...
Checking command definition...
This is ttyS0 on parasol22. Type ^. to terminate the session
-
root
Password:
Last login: Thu Mar  2 12:27:00 2006 from devgdx002.orsay.grid5000.fr on pts/0
Linux (none) 2.6.12-1-amd64-k8-smp #1 SMP Wed Sep 28 02:57:49 CEST 2005 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
(none):~#
```

`kaconsole` shows the console screen as if the user was locally connected onto the machine. Remote console should be kept working all along the boot process, which is useful to debug an environment (see also expert-level practices). Typing the proper escape sequence gets out of this remote console without having to kill the shell or terminal window.

4.5.1 Customization

For automated deployment of experiments, the easiest is to enable password-less connections to the machines. Therefore, this example shows how to create create an environment where the user `ssh` key is stored in the `authorized_keys` file of the root account. Note: post-install scripts sometimes change this file. In this case, it is possible to use `authorized_keys2`. Therefore, the user can:

1. generate a passphrase-less `ssh` key on the front-end;
2. connect as `root` to the machine where the environment is deployed;
3. edit `/root/.ssh/authorized_keys2`, and add to it the new public `ssh` key;
    - because `oar` submission does not transfer `ssh` authentication through agent-forwarding, a passphrase-less `ssh` key is necessary;
4. check that connection is now possible from his account on the frontend node, with no interaction.

Using the root account for all experiments is possible, but in general, it is better to create a user accounts for all users of this new environment. However, if the number of users is greater than 2 or 3, it is faster to configure an `LDAP` client by tuning the post-install script or using a fat version of this script (beyond the scope of this example). Two ways of doing things.

The simplest is to create a dedicated account (e.g. the generic user `g5k`) and move in all experiment data at the beginning and back at the end of an experiment, using `scp` or `rsync`.

A more elaborate approach is to locally recreate the user Grid'5000 account with the same `uid/gid` on the deployed environment. This second approach could simplify file rights management if temporary data must be stored on shared volumes: The user Grid'5000 account parameters can be obtained from the frontend using `id`.

```
yjegou@parasol-dev:~$ id
uid=19000(yjegou) gid=19000(rennes) groups=19000(rennes),19001(staff35)
yjegou@parasol-dev:~$
```

Then, as root, this account can be created on the deployed machine:

```
(none):~# addgroup --gid 19000 rennes
Adding group 'rennes' (19000)...
Done.
(none):~# adduser --uid 19000 --ingroup rennes yjegou
Adding user 'yjegou'...
Adding new user 'yjegou' (19000) with group 'rennes'.
Creating home directory '/home/yjegou'.
Copying files from '/etc/skel'
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
...
(none):~#
```

Finally, as `root`, the user can place his `ssh` key and switch to his new account.

```
(none):~# cp /root/.ssh/authorized_keys2 /home/yjegou/.ssh
(none):~# chown yjegou:rennes /home/yjegou/.ssh/authorized_keys2
(none):~# su - yjegou
yjegou@(none):~$
```

4.5.2 Creating a new environment from a customized environment

The user can now save this customized environment, to be able to use this account again each time he deploys it. The first step to create an environment is to create an archive of the customized node. Because of the various implementations of the `/dev` filesystem tree, this can be a more or less complex operation. The state-of-the-art command for archive creation can be found on the wiki:

```
Sidebar > User Docs > Kadeploy > Kadeploy documentation > \
          Main archive creation
```

```
yjegou@parasol-dev:~$ ssh root@parasol22.rennes.grid5000.fr \
                          "tar --posix --numeric-owner      \
                               --one-file-system -zcf - /"  \
                          > archive.tgz
Password:
tar: Removing leading '/' from member names
tar: Removing leading '/' from hard link targets
tar: /dev/log: socket ignored
yjegou@parasol-dev:~$
```

If /dev is mounted using "udev" (it's probably the case if a /dev/.udevdb directory exists), the directions to follow can be found on the wiki at:

```
Sidebar > User Docs > Kadeploy > Environments requirements > \
          udev and devfs
```

Warning. Kadeploy insists archives be named with a .tgz extension and not .tar.gz. Moreover, the system user deploy should have "read access" to the archive file (this hence also means "execute access" on each subdirectories from the user homedir to that file) in order to deploy an environment based on this archive.

4.5.3 Recording this environment for future deployments

This is done using the karecordenv command. It allows to create a pointer in the Kadeploy database to the archive and associate to it a name, that can be recalled by the kadeploy command, as well as a post-install archive to finely adjust some system files and make the system bootable on each different deployed node. An environment thus consists of this association between an image archive, a name and a post-install archive, along with some other minor parameters.

To find all the parameters of this command, one method is to read the output of karecordenv -h.

```
Usage : karecordenv
        -n  | --name             environment name
        -v  | --version          version              (default is 1)
        -d  | --description      description
        -a  | --author           author's email
        -fb | --filebase         environment image path
        -ft | --filesite         post-installation file path
        -s  | --size             size (Mo)
        -i  | --initrdpath       initrd path
        -k  | --kernelpath       kernel path
        -p  | --param            kernel parameters
        -fd | --fdisktype        fdisk type           (default is 83)
        -fs | --filesystem       file system          (default is ext2)
        -o  | --optsupport       optimisation support (default is 0)
        -fe | --file_environment
        -h                       get this help message
```

Here, the -i and -k parameters are pointers to system files within the user environment needed to build the correct boot loader (Kadeploy uses "grub"). The files in the following example correspond to the default kernel installed in the debian4all environment, which are still valid in the customized environment (unless the user has compiled another kernel...).

```
yjegou@parasol-dev:~$ karecordenv \
                       -n "debian4all-jegou"                    \
                       -v 1                                     \
                       -d "customized debian4all by Yvon Jegou"   \
                       -a Yvon.Jegou@irisa.fr \
                       -fb file:///home/rennes/yjegou/archive.tgz \
                       -ft file://usr/local/kadeploy/Scripts/postinstall/post\
```

```
install4all_gen.tgz                                                    \
                       -i /boot/initrd.img-2.6.12-1-amd64-k8-smp  \
                       -k /boot/vmlinuz-2.6.12-1-amd64-k8-smp     \
                       -fd 83 -fs ext3
Checking variable definition...
Environment registration successfully completed.
yjegou@parasol-dev:~$
```

When customizing an already existing environment, it could be easier to restart from the original record, since `karecordenv` also accepts a file as a single parameter, following the output format of `kaenvironments -e`. For instance:

```
yjegou@parasol-dev:~$ kaenvironments -e debian4all > myImage.env
Checking variable definition...
user selected : yjegou
yjegou@parasol-dev:~$
```

The user may then edit the content of the `myImage.env` file:

1. update the name, description, author and filebase lines
2. delete the id, size, version and user lines
3. save
4. and then record the environment with:

   ```
   frontend: karecordenv -fe myImage.env
   ```

4.6 Adding software to an environment

Grid'5000 machines cannot access the outside public internet in general. But for practical reasons, some holes are poked to be able, for instance, to update an environment with the latest packages from the distributor or to add usefull software (in `.deb` or `.rpm` forms). One of these holes is the `apt-cacher` available on most the sites at `apt.site.grid5000.fr` (Nancy, Orsay or Rennes for example). Therefore, the `debian4all` environment, has the following lines in the `/etc/apt/sources.list` file:

```
deb http://apt.orsay.grid5000.fr/apt-cacher/ftp.de.debian.org/debian/\
         sid main
```

These lines enable the `apt-get` commands to work as expected. It is therefore possible to update the environment (to add missing libraries, or remove unnecessary packages so that sizes down the image and speeds up the deployment process, etc.) using:

- `apt-get update`
- `apt-get upgrade`
- `apt-get install` list of desired packages and libraries
- `apt-get -purge remove` list of unwanted packages

- `apt-get clean`

See `apt-get` man pages for more infos.

4.7 Scripting a deployment

It is often necessary to script most of the work to avoid too many interactive steps. The main points to note are the following:

- a non-interactive reservation on queue `deploy` will see the given script running on the node from which deployments are run (usually the `OAR` or frontend node);
- `kadeploy` accepts the `-f` parameter to read the list of machines to deploy on from a file;
- in non-interactive mode, scripts started by `OAR` write their output in file whose names are
  - `OAR.scriptName.jobId.sdterr` and
  - `OAR.scriptName.jobId.sdtout`.

The following shell script can be adapted for automatic deployment.

```
#!/bin/sh
# written by David Margery for the 2006 Grid'5000 spring school
# feel free to copy and adapt

echo -n "Script running on : "
hostname

#$OAR_FILE_NODES file may contain some nodes more than once
#We will create a second file with each node only once

OAR_FILE_NODES_UNIQUE=${OAR_FILE_NODES}-unique
cat $OAR_FILE_NODES | sort -u > $OAR_FILE_NODES_UNIQUE

#deploy our test environnement
kadeploy -e ubuntu4all-`whoami` -f $OAR_FILE_NODES_UNIQUE -p sda2

#do stuff with this deployed environment
for i in `cat $OAR_FILE_NODES | sort -u`
do
 echo attempt to get information from $i
 ssh root@$i -o StrictHostKeyChecking=no cat /etc/hostname
 ssh root@$i -o StrictHostKeyChecking=no uname -a
done
```

The latest version of this script is available in the subversion repository of the Grid'5000 on InriaGforge. To get a copy of this script, as well as other shared material:

```
outside: svn checkout svn://scm.gforge.inria.fr/svn/grid5000/tools\
                      Grid5000_tools
```

1. a user version of this script can be saved in user's home directory (`/home/mySite/myLogin/-myPath/auto_deploy.sh`);
2. the environment must be updated in the script;
3. this script can be submitted to OAR:

```
frontend: oarsub -l nodes=2 -q deploy \
                /home/mySite/myLogin/myPath/auto_deploy.sh
```

To follow progress of the job:

```
frontend:  oarstat -j jobId
```

or

```
frontend:  tail -f OAR.auto_deploy.sh.jobId.stdout
```

To see the associated error log:

```
frontend:  less OAR.auto_deploy.sh.jobId.stderr
```

Appendix A

OAR reference manual

This manual section has been extracted from the official OAR web site at [4]. More information can be found on this site.

All OAR user commands are installed on cluster head nodes.

A.1 oarstat

This command prints jobs in execution mode on the terminal.

Options

-f	- prints each job in full details
-j job_id	- prints the specified job_id information (even if it is finished)
-g "d1,d2"	- prints history of jobs and state of resources between two dates.
-D	- formats outputs in Perl Dumper
-X	- formats outputs in XML
-Y	- formats outputs in YAML

Examples

```
# oarstat
# oarstat -j 42 -f
```

A.2 oarnodes

This command prints information about cluster resources (state, which jobs on which resources, resource properties, ...).

Options

```
-a              - shows all resources with their properties
-r              - shows only properties of a resource
-s              - shows only resource states
-l              - shows only resource list
-D              - formats outputs in Perl Dumper
-X              - formats outputs in XML
-Y              - formats outputs in YAML
```

Examples

```
# oarnodes
# oarnodes -s
```

A.3 oarsub

The user can submit a job with the `oarsub` command. So, what is a job in our context?

A job is defined by needed resources and a script/program to run. So, the user must specify how many resources and what kind of them are needed by his application. Thus, `OAR` system will give him or not what he wants and will control the execution. When a job is launched, `OAR` executes user program only on the first reservation node. So this program can access some environment variables to know its environment:

```
$OAR_NODEFILE       - contains the name of a file which lists all reserved nodes for this job
$OAR_JOBID          - contains the OAR job identificator
$OAR_RESOURCE_PROPERTIES_FILE
                    - contains the name of a file which lists all resources and their properties
$OAR_NB_NODES       - contains the number of reserved nodes
```

Options:

```
-q "queuename"      - specify the queue for this job
-I                  - turn on INTERACTIVE mode (OAR gives the user a shell instead of
                      executing a script)
-l "resource description"
                    - defines resource list requested for this job; the different param-
                      eters are resource properties registered in OAR database; see ex-
                      amples below. (walltime: Request maximun time. Format is
                      [hour:mn:sec|hour:mn|hour]; after this elapsed time, the job
                      will be killed)
-p "properties"     - adds constraints for the job (format is a WHERE clause from the SQL
                      syntax)
-r "2007-05-11 23:32:03"
                    - asks for a reservation job to begin at the date in argument
-C job_id           - connects to a reservation in Running state
-k "duration"       - asks OAR to send the checkpoint signal to the first processus of the job
                      "number_of_seconds" before the walltime
--signal "signal name"
                    - specify the signal to use when checkpointing
-t "type name"      - specify a specific type (deploy, besteffort, cosystem,
                      checkpoint)
```

`-d "directory path"`
: - specify the directory where to launch the command (default is current directory)

`-n "job name"`
: - specify an arbitrary name for the job

`-a job_id`
: - anterior job that must be terminated to start this new one

`--notify "method"`
: - specify a notification method (mail or command); ex:
`--notify "mail:name@domain.com"`
`--notify "exec:/path/to/script args"`

`--stdout "file name"`
: - specify the name of the standard output file

`--stderr "file name"`
: - specify the name of the error output file

`--resubmit job_id`
: - resubmit the given job to a new one

`--force_cpuset_name "cpuset name"`
: - Specify the cpuset name instead of using the default `job_id` for the cpuset (WARNING: if several jobs have the same cpuset name then processes of a job could be killed when another finished on the same computer)

Examples

```
# oarsub -l /nodes=4 test.sh
```

This example requests the reservation of 4 nodes in the default queue with the default walltime and then the execution of the `test.sh` shell script on the first node of the allocated node list.

```
# oarsub -q default \
         -l walltime=50:30:00,/nodes=10/cpu=3,walltime=2:15:00 \
         -p "switch = 'sw1'" /home/users/toto/prog
```

The "`/home/users/toto/prog`" script will be run on the first of 10 nodes with 3 cpus (so a total of 30 cpus) in the default queue with a walltime of 2:15:00. Moreover "`-p`" option restricts resources only on the switch '`sw1`'.

```
# oarsub -r "2004-04-27 11:00:00" -l /nodes=12/cpu=2
```

A reservation will begin at "`2004-04-27 11:00:00`" on 12 dual-nodes one.

```
#  oarsub -C 42
```

Connects to the job 42 on the first node and set all `OAR` environment variables.

```
# oarsub -I
```

Gives a shell on a resource.

A.4 `oardel`

This command is used to delete or checkpoint job(s). They are designed by their identifier.

Option

`-c job_id` - send checkpoint signal to the job (signal was definedwith "`--signal`" option in oarsub)

Examples

```
# oardel 14 42
```

Delete jobs 14 and 42.

```
# oardel -c 42
```

Send checkpoint signal to the job 42.

A.5 `oarhold`

This command is used to remove a job from the scheduling queue if it is in the "Waiting" state.

Moreover if its state is "Running" `oarhold` can suspend the execution and enable other jobs to use its resources. In that way, a SIGINT signal is sent to every processes.

A.6 `oarresume`

This command resumes jobs in the states Hold or Suspended.

Appendix B

Kadeploy reference manual

This manual section has been extracted from the official Kadeploy web site at [3]. More informations can be found on this site.

All Kadeploy user commands are installed on cluster head nodes.

B.1 kaaddnode

```
kaaddnode info_file
```

kaaddnode registers nodes and their disk and partition features for the deployment system. It can be used at the very beginning to describe a whole cluster (after having completed the installation for instance) or later when new nodes are added to a cluster already managed by the deployment system.

Parameter

info_file - info_file is the name of the file that describes all the information the system needs.

It contains two main parts and has to be formatted as described below:

```
# first part: node set description

node_name1    mac_address1    ip_address1
node_name2    mac_address2    ip_address2
node_name3    mac_address3    ip_address3
...           ...             ...
```

N.B.: each node is described on a separate line; fields on each line are separated by tabs or spaces.

```
# second part: descriptive information about
# the type of disk and the partitioning

device_name    device_size
partition_number1    partition_size1    deployed_env1
partition_number2    partition_size2    deployed_env2
partition_number3    partition_size3    deployed_env3
...                  ...                ...
```

N.B.: each partition is described on a separate line; fields on each line are separated by tabs or spaces. All size are given in Mbytes. The specified `deployed_env` should already exist in the deployment system otherwise default value `undefined` will be switched to the declared one during the execution.

First and second part are separated by a – strange and arbitrary – line starting with the following characters: `\#`. Empty lines and commentaries (starting with `#`) are ignored. Before doing any modification in the deployment system database, `kaaddnode` tries to check if the expected description syntax is respected in both parts.

Example:

```
kaaddnode description.txt
```

with description.txt as below:

```
# Nodes
node1    00:02:73:49:9C:8D    192.168.10.1
node2    00:02:73:49:9C:8E    192.168.10.2
...      ...                  ...

\#

# Disk & Parts
hda  80000
1    5000    swap
2    20000   debian
...  ...     ...
```

B.2 kadelnode

```
kadelnode -m|--machine hostname
```

`kadelnode` unregisters nodes and their disk and partition features from the deployment system.

Options

`-m, --machine hostname` - specifies a host. This option can be used more than once to specify several hosts

Example

```
kadelnode -m node1
```

Remove all information relative to `node1` from the deployment system database.

B.3 kadeploy

```
kadeploy -e|--environment environment_name \
         -m|--machine hostname             \
         -p|--partition partition
```

kadeploy deploys the specified environment on the requested nodes. When successfully over, the nodes are available under the new and freshly installed environment. It needs critic variables to be defined in the /etc/kadeploy/deploy.conf configuration file; see configuration file section for further details.

Options

-e, --environment environment_name
- the name of the environment to deploy

-m, --machine hostname
- specifies a host. This option can be used more than once to specify several hosts. By default, it tries to deploy on all the nodes.

-p, --partition partition
- the target partition. This option can't be used more than once and the partition is the same for all the nodes.

Configuration files

/etc/kadeploy/deploy.conf

Contains variables that should be defined prior to the command execution. These variables are grouped into several sections. A very short description is available for each variable and sometimes example values are given in commentary. The file is read once at the beginning of the execution and prior to any other action. The syntax is the following:

```
# section_name #
# ------------ #

# short description
var_name1 = var_value1

# short description
var_name2 = var_value2  # example_value

...
```

Also, an example of this file can be found at in the tools/cookbook/ folder.

/etc/kadeploy/deploy_cmd.conf

Contains entries describing reboot command needed for deployment. Please see kareboot man page for further details.

kadeploy needs read and write access to several folder and files to acheive a deployment. These rights should be granted to the deploy user. Here are the files and folder list:

- $KADEPLOYDIR/boot/ contains utilities (binaries etc.) for grub/pxe/reboot stories
- the tftp folder where grub and pxe files will be written and that should be defined in the /etc/kadeploy/deploy.conf configuration file
- and /tmp/ for a few temporary files

Example:

```
kadeploy -e debian -m node7 -p hda3
```

- deploys the debian on partition hda3 of node 7

```
kadeploy -e debian -m node1 -m node2 -p hdb6
```

- deploys the debian on partition hdb6 of nodes 1 and 2.

B.4 kareboot

```
kareboot [-n|--noreboot] [-s|--soft] [-h|--hard]          \
         [-d|--deploy] -m|--machine hostname              \
         [-e|--environment name] [-p|--partition partition]
```

`kareboot` can execute software or hardware reboots on given nodes. It can execute five types of query: no reboot, simple reboot, deploy reboot, environment reboot and partition reboot. For all of them, the user can specify one or several hosts.

Here is a brief description of the effects of each reboot type:

reboot: setups `pxe`, `grub` and `tftp` stuff for the given nodes without rebooting;

simple reboot: executes a 'normal' (classic) reboot of the given nodes;

deploy reboot: reboots the given nodes on the deployment kernel;

environment reboot: looks for the specified environment on the given nodes and reboots on it. The environment should thus already be installed;

partition reboot reboots the given nodes on the requested partition if an appropriate environment is found.

Consult the example section below for more explicit usage descriptions.

`kareboot` needs the appropriate command (`softboot`, `hardboot`, `deployboot`) to be defined in the configuration file in order to be able to execute the requested reboot on the given node; see configuration file section for further details.

Options:

`-n, --noreboot`	- setups `pxe`, `grub` and `tftp` stuff
`-s, --soft`	- requests a software reboot (default reboot type)
`-h, --hard`	- requests a hardware reboot
`-d, --deploy`	- specify to reboot on the deployment kernel
`-m, --machine hostname`	- specifies a host. This option can be used more than once to specify several hosts.
`-e, --environment`	- gives an (already installed) environment to boot. Only one environment can be specified.
`-p, --partition`	- gives a partition to boot on. Only one partition can be specified.

Configuration file

`/etc/kadeploy/deploy_cmd.conf`

For each node, it is possible to define the reboot commands for the soft, hard and deploy reboot types in the `/etc/kadeploy/deploy_cmd.conf` configuration file. Here is the description of the expected syntax: `host_name reboot_type command` where `reboot_type` can be either `softboot`, `hardboot` or `deployboot`.

Example for `idpot1` of `idpot` cluster:

```
idpot1 softboot ssh root@idpot1 -o ConnectTimeout=2 /sbin/reboot
idpot1 hardboot ssh localuser@ldap-idpot  \
                ~/commande_modules/logiciel/dio_reset 0x81 0
idpot1 deployboot ../boot/setup_pxe.pl \
                  192.168.10.1:label_deploy_tg3
```

In this example,

- soft reboot is actually a simple "reboot" command done via a `ssh` connection as root,
- hard reboot is a hard signal sent via a `ssh` connection on a precise server as a special user,
- deploy boot is a call to the (deployment system) `setup_pxe.pl` perl script with appropriate arguments.

Examples:

No reboot

```
kareboot -n -m node1 -e debian
                            - setups pxe, grub and tftp stuff for rebooting node1 on
                              debian environment (if already installed)
kareboot -n -m node7 -p hda5
                            - setups pxe, grub and tftp stuff for rebooting node7 on
                              partition hda5 (if an environment is found)
```

Simple reboot

```
kareboot -s -m node1        - executes a software reboot of node1
kareboot -h -m node7 -m node6
                            - executes a hardware reboot of nodes 7 and 6
kareboot -m node7           - executes a software reboot of node7
```

Deploy reboot

```
kareboot -d -m node2        - (soft) reboots on deployment kernel
kareboot -d -h -m node2
                            - (hard) reboots on deployment kernel
```

Environment reboot

```
kareboot -e debian -m node3
                            - looks for environment debian on node3 and (soft) reboots
                              on it if it exists
```

`kareboot -h -e debian -m node3`
- looks for environment debian on node3 and (hard) reboots on it if it exists

Partition reboot

`kareboot -s -p hda6 -m node4`
- checks the environment installed on partition hda6 of node4 and (soft) reboots on it if appropriate

B.5 kaconsole

```
kaconsole -m|--machine hostname
```

`kaconsole` opens a console on a remote node. This tool is designed to help the user that would try to find out the reason of a deployment failure or to follow a boot sequence for instance. It needs the appropriate command to be defined in the configuration file in order to be able to open a remote console on the given node; see configuration file section for further details.

Options

`-m, --machine hostname` - specifies a host. This option cannot be used more than once.

Configuration file

`/etc/kadeploy/deploy_cmd.conf`

For each node, it is possible to define the command to open a remote console on the specified node of the cluster. Here is the description of the expected syntax: `host_name console command`

Example for `idpot1` of `idpot` cluster:

```
idpot1 console ssh -t localuser@ldap-idpot kermit \
                   -l /dev/ttyUSB0 -b 38400 -c
```

In this example, `kaconsole -m idpot1` means to open a serial connection on idpot1 node via kermit software from ldap-idpot

Example

`kaconsole -m node` - opens a console on node (if appropriately equiped)

B.6 karecordenv

```
karecordenv
   -n |  --name        registration name
   -v |  --version     version            # default is 1
   -d |  --description description
   -a |  --author      author email
   -fb|  --filebase    environment image path
```

```
   -ft|  --filesite    post-installation file path
   -s |  --size        size (Mb)
   -i |  --initrdpath  initrdpath        # default is none
   -k |  --kernelpath  kernel path
   -p |  --param       kernel param
   -fd|  --fdisktype   fdisk type         # default is 83
   -fs|  --filesystem  file system        # default is ext2
```

`karecordenv` registers an environment in order to be able to use it with the deployment system.

Options

`-n, --name registration_name`
: - the registration name for the environment

`-v, --version version`
: - the version number of the environment (if needed); default is 1

`-d, --description "description"`
: - a brief description of the environment

`-a, --author author_email`
: - the author email address

`-fb, --filebase environment_image_path`
: - the complete path to the environment image

`-ft, --filesite post-installation_file_path`
: - the complete path to the post-installation file

`-s, --size size (Mb)`
: - the size of the environment; in order to perform partition size check before a deployment

`-i, --initrdpath`
: - the complete initrd path in the environment (including intrd name)

`-k, --kernelpath kernel_path`
: - the complete kernel path in the environment (including kernel name)

`-p, --param`
: - arguments passed to kernel at boot (parameter is set automatically), if left empty, default kernel parameters can be configured

`-fd, --fdisktype fdisk_type`
: - the fdisk type; default is 82

`-fs, --filesystem file_system`
: - the file system type; default is ext2

`Name`, `kernel_path`, `environment_image_path` and `post-installation_file_path` must be defined

Example

```
karecordenv -n debian -v 2 -d "debian ready for installation"
   -a katools@imag.fr
   -fb file://home/nis/jleduc/ImagesDistrib/image_Debian_current.tgz
   -ft file://home/nis/jleduc/Boulot/postinstall/traitement.tgz
   -size 650 -k /boot/vmlinuz -i /initrd
   -p "console=ttyS0,38400n8 ETH_DRV=tg3" -fd 83 -fs ext2
```

registers a debian image whose features are explicitly given in the parameter list.

B.7 kaarchive

```
kaarchive [-X | --exclude-from file]            \
          [-z | --gzip]                         \
          [--output-directory output_directory] \
          [-i | --image image_name]             \
          --root-directory root_directory
```

`kaarchive` creates a environment image from a running one.

Options

`-X, --exclude-from file`
: - excludes files named into file from the generated environment

`-z, --gzip`
: - filters the archive through gzip

`--output-directory output_directory`
: - sets the output directory for the generated environment; default is current directory

`-i, --image image_name`
: - sets the output image name; default is `output_image`

`--root-directory root_directory`
: - sets the root directory for the environment creation

B.8 kacreateenv

```
kacreateenv                             \
   -e | --environment environment_name  \
   -v | --version version               \
   -X | --exclude-from file             \
   -z | --gzip                          \
   --output-directory output_directory  \
   --root-directory root_directory      \
   --author email_address               \
   --description "description"          \
   --file-base file                     \
   --file-site file                     \
   --size size                          \
   --kernel-path kernel_path            \
   --fdisk-type fdisk_type_number       \
   --file-system file_system_type
```

`kacreateenv` creates and registers a new environment from a running one.

Options

`-e, --environment environment_name`
: - the registration and output image name for the environment

```
  -v, --version version      - the version number of the environment (if needed); default
                               is 1
  -d, --description "description"
                             - a brief description of the environment
  -a, --author author_email - the author email address
  -fb, --filebase environment_image_path
                             - the complete path to the environment image
  -ft, --filesite post-installation_file_path
                             - the complete path to the post-installation file
  -s, --size size (Mb)       - the size of the environment; in order to perform partition
                               size check before a deployment
  -k, --kernelpath kernel_path
                             - the complete kernel path in the environment
  -fd, --fdisktype fdisk_type_number
                             - the fdisk type; default is 83
  -fs, --filesystem file_system_type
                             - the file system type; default is ext2
  --root-directory root_directory
                             - sets the root directory for the environment creation
  -X, --exclude-from file    - excludes files named into file from the generated environ-
                               ment
  -z, --gzip                 - filters the archive through gzip
  --output-directory output_directory
                             - sets the output directory for the generated environment; de-
                               fault is current directory
```

Example

```
kacreateenv -e toto_env                                        \
            --description "toto's special image"              \
            --author toto@fai.fr                               \
            -fb file://home/images_folder/toto_env.tgz         \
            -ft file://home/images_folder/toto_spec_file.tgz   \
            --size 650 -k /boot/vmlinuz                        \
            --root-directory /home/toto_img/                   \
            -X /path/to/file_to_exclude.txt -z
```

creates and registers the environment according to the parameters.

Bibliography

[1] Geant-2 web site. <http://www.geant2.net/>.

[2] Grid'5000 web site. <http://www.grid5000.fr/>.

[3] Kadeploy web site. <http://www-id.imag.fr/Logiciels/kadeploy/index.html>.

[4] Oar web site. <http://oar.imag.fr/>.

[5] Renater web site. <http://www.renater.fr/>.

[6] Taktuk web site. <http://taktuk.gforge.inria.fr/>.

[7] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cyrille Martin, Grégory Mounié, Pierre Neyron, and Olivier Richard. A batch scheduler with high level components. In *Cluster computing and Grid 2005 (CCGrid05)*, 2005.

[8] Franck Cappello, Eddy Caron, Michel Dayde, Frederic Desprez, Emmanuel Jeannot, Yvon Jégou, Stephane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, and Olivier Richard. Grid'5000: a large scale, reconfigurable, controlable and monitorable Grid platform. In *Grid'2005 Workshop*, Seattle, USA, November 13-14 2005. IEEE/ACM.