Management of an Academic HPC Cluster
The UL Experience

Sébastien Varrette* and Pascal Bouvry* and Hyacinthe Cartiaux* and Fotis Georgatos†
* Computer Science and Communications (CSC) Research Unit
† Luxembourg Centre for Systems Biomedicine (LCSB)
University of Luxembourg, 16, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg, Luxembourg

***Abstract*—The intensive growth of processing power, data storage and transmission capabilities has revolutionized many aspects of science. These resources are essential to achieve high-quality results in many application areas. In this context, the University of Luxembourg (UL) operates since 2007 an High Performance Computing (HPC) facility and the related storage by a very small team. The aspect of bridging computing and storage is a requirement of UL service – the reasons are both legal (certain data may not move) and performance related. Nowadays, people from the three faculties and/or the two Interdisciplinary centers within the UL, are users of this facility. More specifically, key research priorities such as Systems Bio-medicine (by LCSB) and Security, Reliability & Trust (by SnT) require access to such HPC facilities in order to function in an adequate environment. The management of HPC solutions is a complex enterprise and a constant area for discussion and improvement. The UL HPC facility and the derived deployed services is a complex computing system to manage by its scale: at the moment of writing, it consists of 150 servers, 368 nodes (3880 computing cores) and 1996 TB of shared storage which are all configured, monitored and operated by only three persons using advanced IT automation solutions based on Puppet [1], FAI [2] and Capistrano [3]. This paper covers all the aspects in relation to the management of such a complex infrastructure, whether technical or administrative. Most design choices or implemented approaches have been motivated by several years of experience in addressing research needs, mainly in the HPC area but also in complementary services (typically Web-based). In this context, we tried to answer in a flexible and convenient way many technological issues. This experience report may be of interest for other research centers and universities belonging either to the public or the private sector looking for good if not best practices in cluster architecture and management.**

Keywords*—*HPC, Puppet, Xen, Capistrano

I. Introduction

Many organizations have departments and workgroups that could benefit from High Performance Computing (HPC) resources to analyze, model, and visualize the growing volumes of data they need to conduct business. This also applies to academic institutes such as the University of Luxembourg (UL) where high-quality results and/or models could be achieved in many application areas thanks to the use of an HPC platform. Unfortunately, these groups often do not have sufficient IT support and may lack the specialized IT skills required to run their own HPC clusters. Our objective in this paper is not to contradict the fact that running such a complex platform

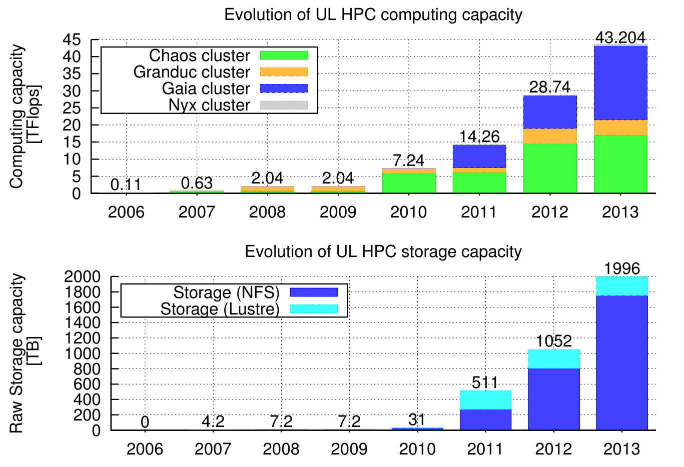

Fig. 1. Evolution of the computing and storage capacity within the UL HPC platform.

requires strong skills and serious investment in man-hours, but rather to share the tools, practices, and more generally our experience in the systems administration of an heterogeneous HPC facility, also in association to complementary services in relation to researchers' accessibility needs (typically Web-based). Indeed, since 2007, the UL has invested in the hardware acquisition and the running of a modest HPC facility at the disposal of the research units. As of now, the cumulative hardware investment has been upwards of 5.7 million euros and leaded to the evolution both in terms of computing power and storage capacity depicted in the Figures 1. The platform now reached a descent size, featuring 368 nodes (3880 computing CPU cores) and a raw shared storage capacity of 1996 TB. It's concrete usage (as measured in CPU-Hours *i.e.* the work done by a CPU in one hour of wall clock time) is exponentially growing, as illustrated in the Fig.2. Being operated by 3 persons, it forced us to refine the collaborative workflow of IT operations to make them easily distributed among the systems administration team. With the reinforcement and the automation of the IT management strategy, a complementary service has been offered to the research units and interdisciplinary centers present within the UL. Indeed, the basic deployment (*bootstrapping*) and configuration of servers is proposed, in such a way, that currently manage more than 150 servers which address various needs within the research units (typically web services, database hosting or code repositories) in addition

TABLE I. CHARACTERISTICS OF THE UL HPC COMPUTING NODES.

	Date	Vendor	Proc. Description	#N	#C	R_{peak}
chaos	2010	HP	Intel Xeon L5640@2.26GHz 2 × 6C,24GB	32	384	3.472 TFlops
	2011	Dell	Intel Xeon L5640@2.26GHz 2 × 6C,24GB	16	192	1.736 TFlops
	2012	Dell	Intel Xeon X7560@2,26GHz 4 × 6C, 1TB	1	32	0.289 TFlops
	2012	Dell	Intel Xeon E5-2660@2.2GHz 2 × 8C,32GB	16	256	4.506 TFlops
	2012	HP	Intel Xeon E5-2660@2.2GHz 2 × 8C,32GB	16	256	4.506 TFlops
			`chaos` TOTAL:	**81**	**1120**	**17.026 TFlops**
gaia	2011	Bull	Intel Xeon L5640@2.26GHz 2 × 6C,48GB	72	864	7.811 TFlops
	2012	Dell	Intel Xeon E5-4640@2.4GHz 4 × 8C, 1TB	1	32	0.307 TFlops
	2012	Bull	Intel Xeon E7-4850@2GHz 16 × 10C,1TB	1	160	1.280 TFLops
	2013	Dell	Intel Xeon E5-2660@2.2GHz 2 × 8C,64GB	5	80	1.408 TFlops
	2013	Bull	Intel Xeon X5670@2.93GHz 2 × 6C,48GB	40	480	5.626 TFlops
	2013	Bull	Intel Xeon X5675@3.07GHz 2 × 6C,48GB	32	384	4.746 TFlops
			`gaia` TOTAL:	**151**	**2000**	**21.178 TFlops**
g5k	2008	Dell	Intel Xeon L5335@2GHz 2 × 4C,16GB	22	176	1.408 TFlops
	2012	Dell	Intel Xeon E5-2630L@2GHz 2 × 6C,24GB	16	192	1.536 TFlops
			`granduc/petitprince` TOTAL:	**38**	**368**	**4.48 TFlops**
Testing cluster:						
nyx	2012	Dell	Intel Xeon E5-2420@1.9GHz 1 × 6C,32GB	2	12	0.091 TFlops
viridis	2013	Viridis	ARM A9 Cortex@1.1GHz 1 × 4C,4GB	96	384	0.422 TFlops
			`nyx/viridis` TOTAL:	**98**	**392**	**0.52 TFlops**

to HPC services. In this context, this article presents the strategy put in place to operate these resources either from a technical or an administrative (cost analysis etc.) point of view. Most design choices or implemented approaches have been motivated by several years of experience to address research needs.
This article is organized as follows: Section II presents an overview of the current facility operated within the University, together with the virtualization infrastructure based on the Xen hypervisor that has been put in production. The section III review the network architecture put in place to address in a secure and isolated way the services described in this article. At this level, we also detail our advanced SSH "tricks" used to facilitate the daily maintenance of our server. The Sysadmin Warrior (SaW) toolkit is also presented. It implements a distributed workflow based on Git and Capistrano to pilot and maintain the full platform operational manageable from the controlling terminals of the systems administrators. Section IV details the distributed Puppet infrastructure put in place to handle in an automated way the deployment and the configuration of all the critical servers operated by the UL HPC systems administration team. Section V reviews the other components put in place to handle the HPC facility, whether from an administrator (Fully Automatic Installation (FAI), OAR, Monitoring tools etc.) or a user point of view (Environment Modules, EasyBuild, HPCBIOS etc.). Finally, the section VI concludes the paper by reviewing the main lessons learned from this infrastructure.

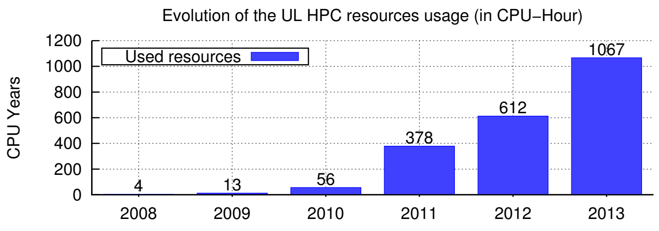

Fig. 2. Evolution of the used resources (total: 2482 CPU-years) of the UL HPC platform.

II. OVERVIEW OF THE MANAGED IT INFRASTRUCTURE

Founded in 2003, the UL is the first and only university of the Grand Duchy of Luxembourg, a small country located at the heart of Western Europe, close to the European institutions and Luxembourg's financial centre. Multilingual, international and research-oriented, the UL is composed of about 190 professors, assistant professors and lecturers, supported by 650 professional experts in various domains. Among them, the UL HPC management team, *i.e.* 3 system administrators experts headed by a full professor, handles two types of equipment and services within the University:

- The UL HPC facility. At the time of writing, it consists of 4 clusters spread on 2 geographic sites featuring a total of 368 computing nodes (for 3880 cores) and a cumulative shared storage capacity of 1996 TB.
- A set of servers serving dedicated needs within the research units (mainly web-based).

On total, it is more than 150 servers in addition to 368 computing nodes (detailed in table I) that are managed. Obviously, it would be time consuming to handle such a number of machines in a pure manual way and over the years, we put in place a management infrastructure based on cutting-edge IT automation and deployment tools that are detailed in this article. Note that for now 3 years, we make an intensive usage of the Xen virtualization framework [4] to improve the general usage of our physical servers. For instance, over the above mentioned 150 servers, 101 are in practice Virtual Machines (VMs) (*domU* in Xen terminology) Also, it greatly facilitates the setup of newly servers for specific services within the research units of the university that generally consumes not so much resources (whether at the level of the CPU or the network

TABLE II. CONFIGURED VLANS.

`prod`	production network containing all servers 10GbE interfaces and all nodes, servers and virtual machines primary ethernet interfaces
`mgmt`	management network containing all management card (BMC, DRAC, ILO, MGMT SAN, etc.)
`IPoIB`	non routed network for IP over IB serving fast filesystems (NFS,Lustre), with its low-latency high-throughput properties
`access`	entry point on the HPC isolated network.

bandwidth) such that it would be both time consuming and cost-ineffective to acquire a fresh new physical server (to be placed in a generally overcrowded server room) just to serve a web site. As we will see in the Section IV, we are able to bootstrap "*from scratch*" and setup such a server in about 6 minutes. Another nice feature offered by our virtualized environment is the configuration of the servers on top of LVM. Logical Volume Manager (LVM) is part of the Linux kernel and manages disk drives and similar mass-storage devices in a flexible way. In particular, it permits to dynamically resize partitions (Logical Volumes (LV) in LVM's terminology) online by concatenating extents which permits to easily extend VM disk image on demand. Also, LVM permits to create read-write snapshots of logical volumes, thus providing a flexible and reversible container able to test without harm and in real condition the effect of applying a Puppet configuration.

III. THE NETWORK INFRASTRUCTURE

A. The Network Backbone and the VLAN configurations

As mentioned in the previous section, the UL HPC system is scattered between two geographically distributed sites, separated by a distance of 40 km. The network backbone between the two campuses is operated by RESTENA [5] which offer to the UL a dedicated lambda on a DWDM infrastructure to establish a 10Gbps link between the two sites. Then inside each campus, a classical network environment features a 1GbE wired interconnection, in particular to the equipment we manage. The network equipment of the clusters is independent of the University network and implements a set of rules detailed in the sequel. Additional fast interconnect elements (based on Cisco 10GbE or Mellanox InfiniBand (IB) QDR 40Gbps switches) are present as we will see in the section V.
We have defined consistent network rules to be followed for a sane network infrastructure and an easier global administration of our HPC infrastructure. The aim was to have a network organization which is a) easily understandable b) designed for risk containment, as regards security aspects and, c) sufficiently safe and flexible for daily systems operations. In particular, the VLANs defined in table II are implemented on each campus via a Layer 3 extension. The Access Control Rules among the network segments & systems follow the onion approach: outgoing traffic is allowed from higher security level to lower. The number of entry points must remain as low as possible, in order to reduce the vulnerability surface. The main idea of our security plan is to minimize & harden the user entry points, and be less strict on the internal network, so that HPC traffic is unhampered. In accordance with this plan, the traffic is not filtered within each VLAN.

B. SSH Configuration

Our servers are connected to different networks and VLANs, typically behind the University firewall over which we have little or no direct control. Also, we obviously need to operate all our remote servers from abroad. For the very few cases (14) where the server is configured with a public IP, this is not an issue. For all the others, we configured a special entry point from the outside called the "bastion". This is a special host reserved to the system administrators and separated from the user-accessible part of the infrastructure. This bastion benefits from specific network ACLs and allows us to reach nearly all points of our network. If not, we use a specific feature offered by SSH called `ProxyCommand` in combination with the `netcat` utility to setup a transparent multi-hop connections framework which allow us to reach any of our managed hosts in a single command. Here is an example for configuring an entry for `internal-server.ext_ul`, where %h is automatically replaced by the hostname, and %p by the port:

```
Host internal-server.ext_ul
  Hostname 10.1.2.3
  Port 8022
  ProxyCommand ssh bastion "nc -q 0 %h %p"
```

Coupled with the bastion host, it greatly ease our daily system administration duties while offering several key advantages compared to the "regular" SSH tunnelling approach: (1) there is no need to explicitly initiate the intermediate SSH tunnels that compose the connection path toward the target host; (2) it is sufficient to configure our set of SSH public keys on each server (which we have to do in all cases, anyhow), automatized by Puppet (see §IV); (3) Using the `ControlMaster` feature, we can configure SSH to reuse an existing connection (typically to the bastion) by sharing the same tcp connection. This means subsequent connections are much faster to open. The SaW toolkit presented in the next section not only configure in an automated way these configurations, it also benefit from this setup in the tasks that run remote commands and we will see with the experiment presented in the Figure 3.

C. SAW: the Sysadmin Warrior toolkit

To facilitate the definition and the efficient deployment of commands on the servers we manage, the Sysadmin Warrior (SaW) toolkit has been developed. To permit a collaborative work on the server definition and status, the configuration piloted by SaW is hosted on a Git repository. When dealing with the problem of running and storing in an automated way information associated to IT equipment, we have to face the following questions:
(1) how to organize the files and data such that they can be easily accessed in a human-readable way, typically over a command-line terminal; (2) how to automatically generate configuration files (SSH, Puppet manifests etc.) that match the system configuration of each managed host; (3) how to effectively group and run remote commands (system upgrade, FAI or Puppet deployment etc.) on all or, a subset of the configured hosts?
In this context, several design choices have been made in the implementation of SaW. For instance, instead of relying on a complex relational database system such as MySQL

TABLE III. MAIN SAW TASKS CURRENTLY IMPLEMENTED.

Task name	Description
User management	
`accounts:user:new`	Create a new HPC user (ldap entry,nfs dirs and welcome mail)
`accounts:project:new`	Create a new HPC project (ldap entry, nfs dirs)
Site/server management	
`site:[host:]add`	Add a new site/host
`site:[host:]mod`	Modify an existing site/host
Cluster management	
`cluster:{add,mod}`	Add / modify a cluster
`cluster:{server,node,network,storage}:add`	Add a new server/computing node/interconnect/storage equipment
`cluster:{server,node,network,storage}:mod`	Modify (add invoice, alter configuration etc.) of one of the above element
`cluster:fai:deploy:{devel,prod,testing}`	Run FAI on the computing node in the `{devel,prod,testing}` environment
Puppet infrastructure management	
`puppet:bootstrap`	Bootstrap a new node to later on use puppet
`puppet:env:update:{devel,prod,testing}`	Update the `{devel,prod,testing}` environment on the Puppet servers (either via `git pull` or `rsync --delete` operations)
`puppet:deploy:{devel,prod,testing}`	Run puppet in the `{devel,prod,testing}` environment
`puppet:on_snapshot:{devel,prod,testing}`	As above, yet on a LVM snapshot of the VM to permit system rollback
`puppet:vault:{open,close,deploy}`	Open, close or deploy the content of the EncFS vault
Useful system administrator commands	
`conf:list:by:{cluster,site,dom0,ip}`	List the managed hosts, grouped by cluster/site/Xen Dom0/IP
`conf:ssh`	Generate the SSH configuration file `~/.ssh/config`
`cmd:{uptime,reboot...}`	Run the command `uptime`/`reboot` etc.
`sys:upgrade`	Upgrade the target system(s) (by `yum upgrade` or `apt-get upgrade`)

or PostgreSQL that would add an unnecessary dependency on an external software while rendering difficult the interaction with the data stored in a command-line, SaW stores the information in independent YAML [6] files. YAML is a human friendly data serialization standard evenly suitable for all programming languages. Developed in Ruby, SaW is based on Capistrano [3], a utility and framework for executing commands in parallel on multiple remote machines, via SSH. It uses a simple Domain-Specific Language (DSL) (borrowed in part from Rake, a simple ruby build program with capabilities similar to GNU `make`) that allows to define tasks, which may be applied to machines in certain roles. It also supports tunneling connections via some gateway machine to allow operations to be performed behind VPN's and firewalls. Capistrano was originally designed to simplify and automate deployment of web applications to distributed environments, and originally came bundled with a set of tasks designed for deploying Rails applications. Up to now, 87 tasks have been implemented. A subset of them which permits to pilot the most useful tasks are detailed in the Table III.

There are many features offered by SaW (EncFS vault management, automatic doc extraction etc.) that we do not describe here in detail due to page restrictions. Yet the main objectives of interest fulfilled by SaW for the scope of this article are (1) the automatic generation of the SSH configuration (in `~/.ssh/config`) to permit direct and transparent connections to any of the configured hosts, whether inside or outside the UL network (see §III-B):

```
$> ssh name        # inside the UL network
                   # OR publicly reachable host
$> ssh name.ext_ul              # outside the UL
```

(2) permits the parallel run over the Capistrano framework of a set of predefined commands, typically to pilot an Operating System (OS) upgrade, a Puppet or a FAI deployment:

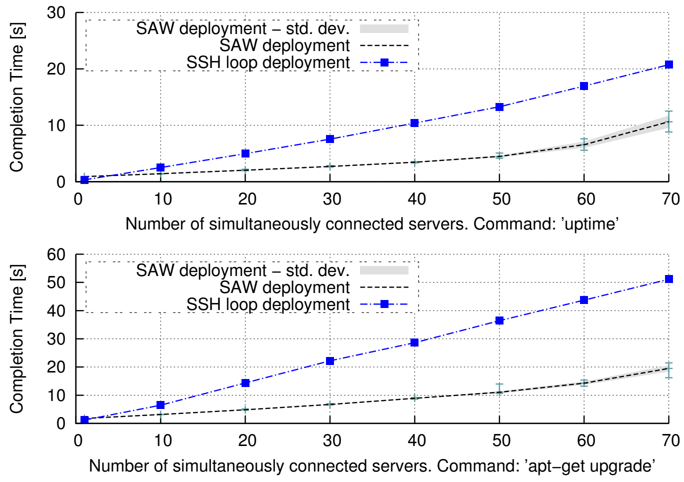

Fig. 3. Performances of SaW in the parallel execution of commands on an increasing number of managed servers.

```
$> saw sys:upgrade   # upgrade ALL configured hosts
     # idem yet limited to gaia cluster servers
$> saw sys:upgrade CLUSTER=gaia
     # similar , yet restricted to CSC hosts
$> saw sys:upgrade SITE=CSC
     # run on a single host
$> saw sys:upgrade HOST=dns.chaos
```

One advantage of the above approach is that it normalizes the access to the configured hosts and the operations performed in a given context among the system administrator. Also, while automating daily tasks, SaW also permits to conduct them in an efficient way. To demonstrate it, we have performed the following experiment: for two typical commands – a simple `uptime` (which displays the length of time the system has been up and running) and an OS upgrade via `apt-get upgrade` on Debian host (yet in `--no-op` mode to facilitate the reproducibility of the experiment). These commands have been performed by SaW on an increasing number of hosts and the completion time has been compared to the one required by a SSH loop command repeated sequentially. Each test has been reproduced 100 times and the results are depicted in the figure 3. We can see that SaW permits to reduce the completion time on 70 hosts by 48.7% for a simple command (`uptime`) and up to 61.9% for a more complex one (the system upgrade, involving around 22 package updates).

IV. THE PUPPET INFRASTRUCTURE

The configuration of the UL HPC platform is piloted by Puppet [1]. Puppet is a mature IT automation software based on a master/slave approach that helps system administrators to manage their infrastructure throughout its life cycle, from provisioning and configuration to patch management and compliance. Using Puppet, it is easy to automate repetitive tasks, quickly deploy critical applications, and pro-actively manage change in a scalable way: at the moment of writing, the UL Puppet infrastructure manages around 150 servers configuration, while the one of the Grid'5000 grid handles more than 400 servers.
In practice, Puppet uses a declarative, model-based approach to IT automation. In particular, the desired state of the infrastructure's configuration is defined using Puppet's declarative configuration language. At this level, one can combine the interfaces provided by Puppet *modules* which are self-contained bundles of code and data that can eventually be shared with other users by publishing the module on Opensource collaborative platforms such as GitHub. Up to now, we have implemented more than 55 original modules. After a Puppet configuration is deployed, a Puppet agent is run on each node to automatically enforce the desired node state, correcting any configuration drift. The Puppet agent of a node sends *Facts*, or data about its state, to the Puppet Master which compiles a *Catalog*, or detailed data about how the node should be configured, and sends this back to the Puppet agent. After making any changes to return to the desired state, the Puppet agent sends a complete Report back to the Puppet Master. Detailing the implemented modules is clearly out of the scope of this paper. We simply briefly review now the hierarchical approach put in place to define the Puppet *manifests* of the configured nodes and servers. These manifests contain classes instances and chunks of code that configure a specific aspect or feature of the machine. In practice, all the servers we manage are configured by inheriting from at least two other generic node definitions.

```
node 'basenode'  {
   # This  install  the  basic  packages/ services
   class { 'generic': }
   class { 'exim4':   # mail  notifications
       configtype  => ' satellite ',
       smarthost   => 'xxx.uxx'
   }
   class { 'ssh :: server':
       port                    => '8022',
       permitrootlogin         => 'no',
       passwordauthentication  => 'no',
   }
   # Configure  the  local  system  administrator
   $sysadmin_login = 'localadmin'
   class { 'sysadmin': ensure => 'present' }
   include  'user_admin1'
   include  'user_admin2'
   include  'user_admin3'
}
```

Then for each cluster, we define another generic definition derived from this `basenode` definition:

```
node 'generic_gaia'  inherits  'basenode' {
   include  network
   # Define  default  network  setup
   network :: interface  { 'eth0':
        comment  => "Prod network (10.xx.0.0/16)−VLAN xxx",
        netmask   => '255.255.0.0',
        network   => '10.xxx.0.0',
        broadcast  => '10.xxx.255.255',
        gateway    => '10.xxx.0.1',
        dns_nameservers => '10.xx.xx.xx,
        dns_search        => 'gaia−cluster.uxx'
   }
   # configure  resolv .conf
   bind :: resolver  { 'gaia−cluster.uxx':
       nameserver => [ '10.xx.xx.xx', '10.yy.yy.yy' ],
   }
   class { 'ntp':  server_list  => [ '10.xx.xx.xx' ] }
   class { 'puppet :: agent':
       server      => 'puppet−csc.uxx',
       ca_server  => 'puppetca.uxx'
   }
}
```

Finally, a new server manifest is written on the following basis (a DHCP server in the below example):

```
node 'dhcp.gaia−cluster.uxx'  inherits  'generic_gaianode'  {
   # Specialization  of  the  IP  for  eth0
   Network:: Interface ['eth0'] {
       address    => '10.xx.xx.ZZ',
   }
   # The DHCP server may serve IP for some BMC cards
   network :: interface  { 'eth1':
       comment  => "Mgmt network (10.yy.0.0/16)−VLAN yyy",
       address    => '10.yy.yy.ZZ',
       netmask    => '255.255.0.0',
       network    => '10.yyy.0.0',
       broadcast  => '10.yyy.255.255',
       # gateway    => '10.yyy.0.1'
   }
   include  'dhcp :: server'
}
```

Many other elements have been put in place at the level of our Puppet infrastructure - eg. hierarchical Certificate Authority (CA), distributed workflow based on Git, Puppet environments etc. We have depicted the complete layout of this architecture in figure 4. To illustrate its efficiency, the following experiment has been performed. The objective was to setup a working server in four typical scenarios met in our environment. Each of them corresponded to a desired running configuration on the deployed server summarized in the Table IV.

TABLE IV. SERVER CONFIGURATION IN THE PUPPET EXPERIMENT.

Name	Description
`base`	Minimal Generic yet secure configuration
`www`	Web server (`base` + Apache, MySQL...)
`hpcfront`	HPC frontend (`base` + {nfs,oar,ldap}-client ...)
`nfs`	NFS server (`base` + nfs-server and exports, multipath...)

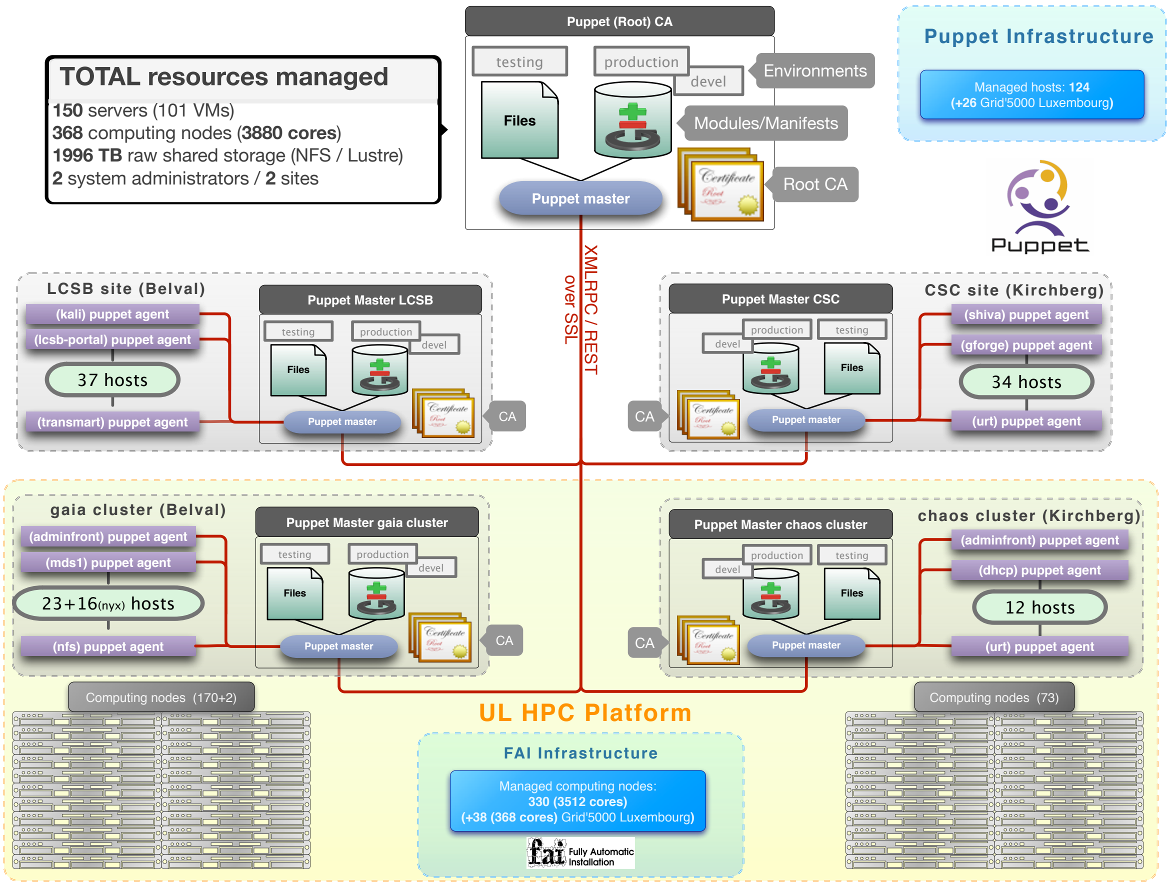

Fig. 4. Overview of the Puppet infrastructure put in place at the UL.

These configurations have been successively deployed on a newly created VM (a Xen DomU) configured with a minimal OS (a Debian `netinstall`), and where the specialized configuration inherent to our environment (NTP server, enhanced SSH configuration, local administrator account etc...) or the situation (Apache, NFS server etc.) has to be performed. This typically reflects the situation of a newly arrived server where the default system has been installed. In practice, we have applied our customized `xen::domU` Puppet module to generate such a minimal system (in around 130s of time) and created a LVM snapshot on top of it to easily revert any changes applied by the successive Puppet runs. Then each the above mentioned configurations has been applied by our puppet infrastructure three times and we stored the completion time of each run. This test has been repeated 100 times for each configuration and the results are proposed in the Figure 5. It follows that the first runs permit to completely bootstrap a running system with the fully configured services. Most of the time is spent on the many packages that have to be installed – on average between 161s and 364s, meanwhile it still remains quite fast to get a fully configured and running server. The remaining runs (2 and 3) simply reapply the selected configuration thus mainly ensuring that the packages and configuration files are present and coherent, or that the services (apache, nfs etc.) are indeed running. On average, these steps cost on average between 20s and 31s and correspond to the typical time spent on the puppet runs that are applied on a regular basis on all the servers we manage within our infrastructure.

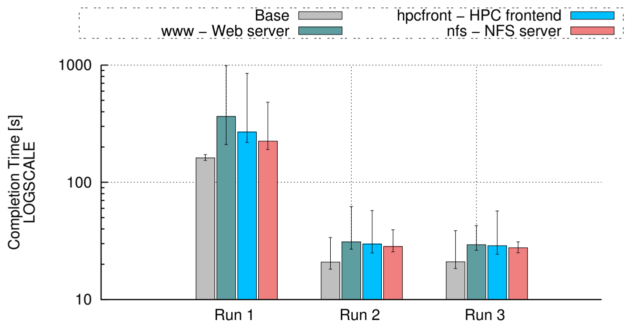

Fig. 5. Performances of our Puppet infrastructure in the repetitive deployment of 4 typical configurations.

V. The UL Clusters environment

Each of the computing clusters are organized in a similar fashion illustrated on figure 6, thus features (1) an `access` server used as an SSH interface for the user to the cluster that grants the access to the cluster internals; (2) a user `frontend` (eventually merged with the access node), used to reserve nodes on the cluster etc. (3) an `adminfront`, a Xen `Dom0` (*i.e.* the initial domain started by the Xen hypervisor on boot that runs the Xen management toolstack, and has special privileges like being able to access the hardware directly). This node is used to host the different services (one per Xen *domU*) required to manage the cluster. Among them, we can

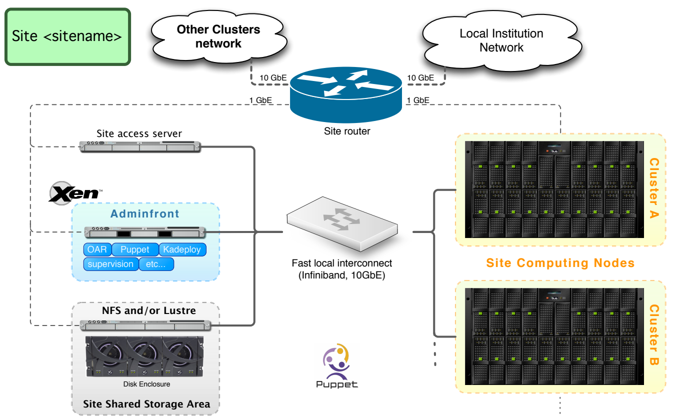

Fig. 6. Organization of the UL clusters.

cite the FAI server, used to automatically install and deploy the computing nodes (see §V-A); the Puppet server, used to manage various configuration aspects on the site (see §IV), the OpenLDAP server, which manages user authentication or the OAR server that acts as a resource manager (and batch scheduler) for the computing nodes of the cluster; (4) an NFS server (eventually completed with a Lustre infrastructure), used for data sharing (homedirs etc.) among the computing nodes and the user frontend (see §V-B); (5) the computing nodes and (6) the fast interconnect equipment (based on 10 GbE or IB QDR technologies).

A. Computing node deployment strategy

FAI [2] is an operating system installation tool, which permits to automate the process of installation and customizaton of a server or a set of computing nodes. The FAI service requires a DHCP server, a Debian Mirror, and delivers a deployment system using NFS together with a TFTP server. Once the FAI server is set up, the biggest part of the work is to write an installation plan in order to specify the final configuration of the node: disk partitioning, extra packages, scripts and hooks, configuration files, which in our case consists of 329 files. The deployment in itself is illustrated in the figure 7: The deployment system is booted via the network card (PXE), and applies our installation plan. In terms of performance, our FAI infrastructure is currently able to deploy 70 nodes of each cluster in around 30 minutes.

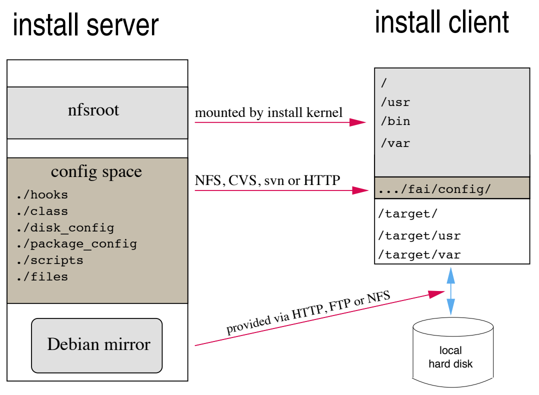

Fig. 7. Overview of an FAI deployment. (Source:[2])

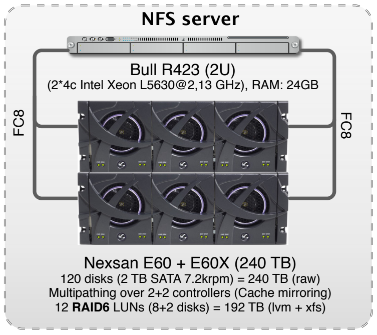
NFS server
Bull R423 (2U)
(2*4c Intel Xeon L5630@2,13 GHz), RAM: 24GB
FC8
FC8
Nexsan E60 + E60X (240 TB)
120 disks (2 TB SATA 7.2krpm) = 240 TB (raw)
Multipathing over 2+2 controllers (Cache mirroring)
12 RAID6 LUNs (8+2 disks) = 192 TB (lvm + xfs)

Fig. 8. Configuration used for each NFS server.

B. Shared Storage infrastructure

The shared storage facilities offered on the clusters are of two types: (1) a traditional shared storage relying on the Network File System (NFS) protocol; (2) a highly performant Lustre [7] based storage. It has been chosen for its stability and for simplicity, along with scalability properties that are of interest in an HPC environment.
The choice for NFS was motivated by the well-known stability it offers, thus being designated to host user home directories. We decided to assign to each cluster on each site its dedicated and independent NFS storage. It appeared indeed extremely hazardous to authorize a File System (FS) synchronization (even asynchronous) between sites geographically separated by around 40 km. In practice, we have build our NFS service as depicted in figure 8. The basic brick of our storage infrastructure are high density enclosures (provided by Nexsan or NetApp) which features 60 disks (each with 2TB of raw capacity). We organized them in 6 RAID6 Logical Unit Numbers (LUNs) over 8+2 disks, thus featuring an effective storage capacity of 96TB per enclosure. Redundant Fibre Channel (FC) links are used to connect the NFS server to the enclosure. Also, it was crucial for us to feature high-performant redundant controller with a cache-mirroring to guarantee both sufficient I/O performance and fault-resilience to offer the best protection to data loss. On top of them, LVM was configured and an `xfs` File System (FS) was setup with quota operations enabled.

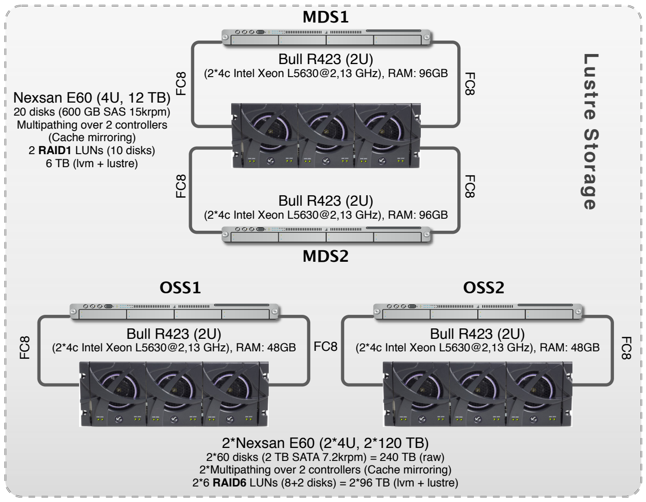
MDS1
Bull R423 (2U)
(2*4c Intel Xeon L5630@2,13 GHz), RAM: 96GB
FC8
FC8
Nexsan E60 (4U, 12 TB)
20 disks (600 GB SAS 15krpm)
Multipathing over 2 controllers
(Cache mirroring)
2 RAID1 LUNs (10 disks)
6 TB (lvm + lustre)
FC8
FC8
Bull R423 (2U)
(2*4c Intel Xeon L5630@2,13 GHz), RAM: 96GB
MDS2
Lustre Storage
OSS1
Bull R423 (2U)
(2*4c Intel Xeon L5630@2,13 GHz), RAM: 48GB
FC8
FC8
OSS2
Bull R423 (2U)
(2*4c Intel Xeon L5630@2,13 GHz), RAM: 48GB
FC8
2*Nexsan E60 (2*4U, 2*120 TB)
2*60 disks (2 TB SATA 7.2krpm) = 240 TB (raw)
2*Multipathing over 2 controllers (Cache mirroring)
2*6 RAID6 LUNs (8+2 disks) = 2*96 TB (lvm + lustre)

Fig. 9. Configuration used for the UL Lustre storage.

To cope with the growing storage needs of the UL, especially since the advent of a Bio center with its growing volumes of data, we had to select a complementary storage solution based on a parallel and distributed FS that would provide a scalable solution, whether in terms of capacity or access efficiency. We selected Lustre [7], a highly performant FS scaling to tens of thousands of nodes and petabytes of storage with groundbreaking I/O and metadata throughput. The configuration we put in place is depicted in the Figure 9. It features two redundant Meta-Data Server (MDS) in master/slave mode, and currently two Object Storage Serverss (OSSs) each of them attached to a disk enclosure (an Object Storage Target (OST) in Lustre terminology). The advantage of Lustre is that for each pair of OSS/OST added, the global capacity of the Lustre storage infrastructure is increased by the capacity of the OST, and its throughput is added to the global throughput of the system.

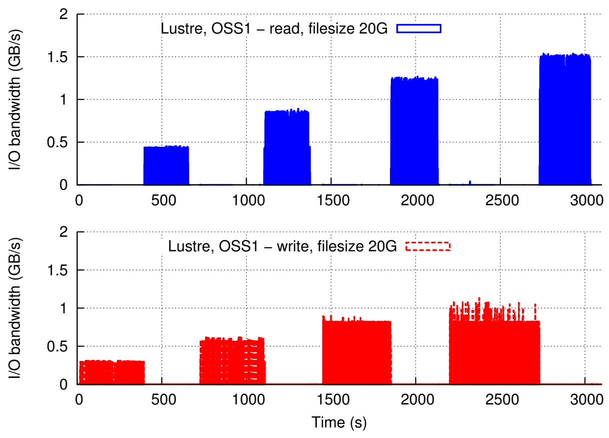

Fig. 10. UL HPC Lustre OSS IOZone performances (R/W).

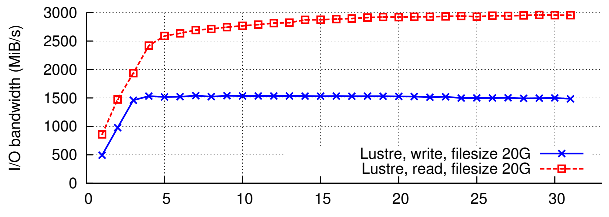

Fig. 11. UL HPC Lustre I/O performances using IOR.

Of interest for the user, to evaluate the performance of the Lustre infrastructure, we selected IOR [8] in addition to IOZone [9] as benchmarks. All results are displayed in the figures 10 and 11. They demonstrate sustainable performances close to the controllers theoretical limits (1.5GB/s in read, 750MB/s in write) on each OSS (only the results of the first one are displayed due to space constraints). Also, the IOR benchmark results of the Fig.12 reveal a concrete performance of 3GB/s (in read) and 1.5GB/s (in write) for the shared storage option, which is coherent with the current sizing of the infrastructure (which features two OSS and thus two controllers).
We also evaluate in the Fig.12 the performance of the different storage options available on the platform, whether local (RAM *i.e.* `/dev/shm`, Hard Disk or SSD *i.e.* `/tmp`) or shared (NFS or Lustre) since it appears important to advertise to the users of the system the characteristics of each possible approaches. Indeed privileging local solutions permits to limit the impact of distributed workflow on the shared storage and not all users were aware of the possibility (and the performance) featured by the `/dev/shm` device for instance. Of course, local approaches are restricted by the inherent size of the corresponding device and suffer from the additional synchronization work users have to take care of, which they are generally reluctant to investigate. Nevertheless, our experience demonstrate that continuously advertise the benefits of local storage to reduce the load of the shared solutions drastically improved the productivity of users and the statility of the platform.

C. User Job Management

The job scheduler used within the UL HPC platform is *OAR* [10], a versatile resource and task manager (also called a batch scheduler) which provides a simple and flexible exploitation of a cluster. OAR manages resources of clusters as a traditional batch scheduler (as PBS [11], Torque [12], LSF [13] or SGE [14]). In other words, it does not really execute users job on the resources but manages them (reservation, access granting) in order to allow the user to connect these resources and exploit them. Its design is based on high level tools, more precisely a relational database engine (MySQL or PostgreSQL), a scripting language (Perl), a confinement system mechanism based on a feature proposed by recent (2.6) Linux kernels called `cpuset`. OAR also use a scalable administrative tool, component of the `Taktuk` [15] framework. OAR is flexible enough to be suitable for production clusters and research experiments and presented for us several advantages: it is Open-source and no specific daemon is run on nodes. Also, there is no dependence on specific computing libraries like MPI such that all sort of parallel user applications

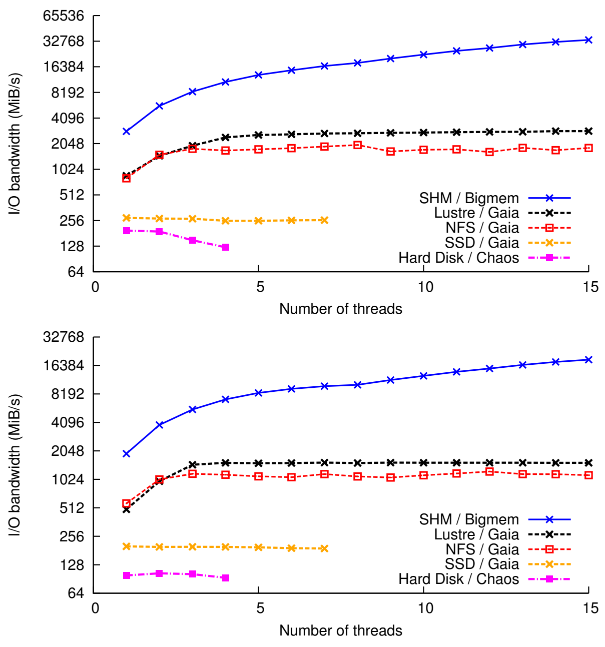

Fig. 12. UL HPC Storage Options I/O performances against the IOR benchmarking suite (Read on top, Write on bottom).

are supported. Finally, OAR support job checkpointing and resubmission and features best effort jobs: if another job wants the same resources then it is deleted automatically. We use it to propose to user a way to relax our default policy as regards the total number of resources a user can reserve in his job. Until now, we bounded this number to 10% of the considered cluster for daily jobs yet using best-effort jobs, users have no limitation. The price for them is simply that jobs submitted that way may be kill at any moment if the resource is to be used for a default job. The full detail of the scheduling policy would require far more (lengthy) developments. In a few lines, we put in place many rules to handle in a fair way the presence of stakeholders (which have a privileged access to dedicated paid resources), research project management and a (short) list of resources assigned to new users to limit their impact on the jobs run by more experimented users. These resources are changed on a weekly basis in a round-robin way. Also, one interesting feature of OAR fall in the area of accounting possibilities since it easiest the measure of CPU-Hour metrics per users and/or projects. For instance the Fig 2 was generated from such data, aggregated on a yearly basis. Finally, OAR is particularly robust and resilient: more than 3.8 millions jobs have been scheduled by OAR between 2008 and early 2014 on the `chaos` and `gaia` clusters.

D. Software Tools and Development Environment

The HPC Software Development environment includes a set of common tools which are needed for building and/or using scientific computing software on our clusters. In practice, we rely on three frameworks: (1) the `modules` environment [16] is a de facto standard in HPC systems, since it is a mechanism that allows a structured management of the environment variables for multiple software versions, specifically well adopted for usage under UNIX platforms. (2) The `EasyBuild` [17] toolkit, an open source project with the explicit aim to automate the building of software and modules for HPC environments. The use of such a tool is imperative in our context, in the sense that given sufficient time the complexity of HPC software stacks grows beyond what an individual human mind can handle with ease and, the related software building procedures can quickly become convoluted and error prone. EasyBuild, at the present moment, automatically builds more than 377 distinct applications and can deliver over 600 modules, serving varying needs of HPC communities. (3) The `HPCBIOS` [18], a project effort initiated within the UL in order to automate and standardize the delivery of scientific software for both the needs of local and other collaborating HPC user communities.

VI. Conclusion & Perspectives

Administrating research infrastructures requires a multitude of skills and very focused expertise in High Performance Systems management. The experience at the UL highlights that the selection of tools is crucial, in order to allow scaling the infrastructure without suffering from constant need to deliver support man-hours, which is the most common bottleneck in such environment. Deliberate choices in both hardware (*i.e.* privileging a common processor architecture and the same network vendor solution) and software components, that permit automation and collective remote control and monitoring are the key to allow a systems population to grow in a super-linear fashion, in relation to the size of a system administrators team. The specific set of technologies that are in use now at UL, have proven their integration capabilities and can be recommended as best practices for an HPC infrastructure of this scale, even with a small team of system administrators:

1) FAI, Kadeploy, Puppet, Xen - for fabric management
2) SaW - for configuration management; featuring git, Capistrano, YAML
3) ssh, Ganglia - remote control & monitoring featuring RRDtool
4) OAR, NFS, Lustre - for HPC computational & storage cluster facilities
5) environment modules, EasyBuild, HPCBIOS - for advanced complex HPC software stacks.

Certainly, the quest to advance the systems management and its quality aspects does not end here; an advice that can be provided to fellow sysadmins is to shop carefully their solutions from the available open source projects, or not, and take calculated risks as regards support effort. This mentality is what has allowed a small team of 3 persons to deliver complex services, that now hundreds of very expert and specialized scientists can benefit from.

References

[1] P. Labs, “Puppet, an automated administrative engine,” [online] http://projects.puppetlabs.com/projects/puppet.

[2] T. Lange, “Fully automated installation,” [online] http://fai-project.org/features/.

[3] J. Buck, “Capistrano,” [online] https://github.com/capistrano/capistrano/wiki.

[4] P. Barham and al., “Xen and the art of virtualization,” in *Proc. of the 19th ACM symposium on Operating systems principles*, ser. SOSP ’03. ACM, 2003, pp. 164–177.

[5] “Restena,” [online] http://www.restena.lu/.

[6] C. Evans, “Yaml: Yaml ain’t markup language.”

[7] P. Schwan, “Lustre: Building a file system for 1,000-node clusters,” in *Proc. of the Linux Symposium*, 2003, p. 9.

[8] “IOR HPC benchmark,” [online] http://sourceforge.net/projects/ior-sio/.

[9] “Iozone filesystem benchmark,” [online] http://www.iozone.org/.

[10] N. Capit and al., “A batch scheduler with high level components,” in *Cluster computing and Grid 2005 (CCGrid05)*, 2005.

[11] Altair, “PBS Professional, HPC workload management and job scheduling solution,” [online] http://www.pbsworks.com/.

[12] I. Adaptive Computing, “Torque resource manager,” [online] http://www.adaptivecomputing.com/.

[13] IBM, “Ibm platform lsf, a workload management platform for demanding, distributed hpc environments.” [online] http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/lsf/.

[14] Oracle, “Sun grid engine, an open source batch-queuing system for hpc clusters,” [online] http://www.oracle.com/technetwork/oem/grid-engine-166852.html.

[15] INRIA, “taktuk,” [online] http://taktuk.gforge.inria.fr/.

[16] J. L. Furlani and P. W. Osel, “Abstract yourself with modules,” in *Proc. of the 10th USENIX conf. on System administration*, ser. LISA ’96, 1996, pp. 193–204.

[17] K. Hoste, J. Timmerman, A. Georges, and S. Deweirdt, “Easybuild: Building software with ease,” in *SC*, 2012.

[18] F. Georgatos, “Hpcbios,” https://hpcbios.readthedocs.org.