Kadeploy3: Efficient and Scalable Operating System Provisioning for HPC Clusters

Emmanuel Jeanvoine, Luc Sarzyniec, Lucas Nussbaum

RESEARCH REPORT
N° 8002
June 2012
Project-Teams AlGorille and SED Inria Nancy – Grand Est

ISSN 0249-6399   ISRN INRIA/RR--8002--FR+ENG

Kadeploy3:
Efficient and Scalable Operating System Provisioning for HPC Clusters

Emmanuel Jeanvoine*, Luc Sarzyniec†*‡, Lucas Nussbaum†*‡

Project-Teams AlGorille and
SED Inria Nancy – Grand Est

Research Report n° 8002 — June 2012 — 18 pages

Abstract: Operating system provisioning is a common and critical task in cluster computing environments. The required low-level operations involved in provisioning can drastically decrease the performance of a given solution, and maintaining a reasonable provisioning time on clusters of 1000+ nodes is a significant challenge. We present Kadeploy3, a tool built to efficiently and reliably deploy a large number of cluster nodes. Since it is a keystone of the Grid'5000 experimental testbed, it has been designed not only to help system administrators install and manage clusters but also to provide testbed users with a flexible way to deploy their own operating systems on nodes for their own experimentation needs, on a very frequent basis. In this paper we detail the design principles of Kadeploy3 and its main features, and evaluate its capabilities in several contexts. We also share the lessons we have learned during the design and deployment of Kadeploy3 in the hope that this will help system administrators and developers of similar solutions.

Key-words: HPC, cluster provisionning

* Inria, Villers-lès-Nancy, F-54600, France
† Université de Lorraine, LORIA, UMR 7503, Vandoeuvre-lès-Nancy, F-54500, France
‡ CNRS, LORIA, UMR 7503, Vandoeuvre-lès-Nancy, F-54500, France

**RESEARCH CENTRE
NANCY – GRAND EST**

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Kadeploy3: Efficient and Scalable Operating System Provisioning for HPC Clusters

Résumé : Operating system provisioning is a common and critical task in cluster computing environments. The required low-level operations involved in provisioning can drastically decrease the performance of a given solution, and maintaining a reasonable provisioning time on clusters of 1000+ nodes is a significant challenge. We present Kadeploy3, a tool built to efficiently and reliably deploy a large number of cluster nodes. Since it is a keystone of the Grid'5000 experimental testbed, it has been designed not only to help system administrators install and manage clusters but also to provide testbed users with a flexible way to deploy their own operating systems on nodes for their own experimentation needs, on a very frequent basis. In this paper we detail the design principles of Kadeploy3 and its main features, and evaluate its capabilities in several contexts. We also share the lessons we have learned during the design and deployment of Kadeploy3 in the hope that this will help system administrators and developers of similar solutions.

Mots-clés : HPC, cluster provisionning

1 Introduction

Traditional users of computing resources do not often care about the way in which the operating system has been installed on the hardware as long as their applications can be compiled and executed.

However the task of installing an operating system can be very tedious on large scale clusters. Since it is not realistic to install the nodes independently, disk cloning or imaging [1, 9, 12, 14] is a common approach. In this case the administrator must keep updated only one node (sometime called golden node) that can be replicated to other nodes. Usually a small post-installation step has to be performed to customize certain node-specific parameters (e.g., identity details).

The work presented in this paper has emerged from quite specific requirements mandated by the Grid'5000 experimental testbed [6, 15]. The goal of Grid'5000 is to provide the users with a fully customizable testbed in order to perform advanced experiments in all areas of computer science related to parallel, large-scale or distributed computing and networking. Thus, the testbed required a tool to allow users to deploy their own operating system (whatever flavor). In many cases, this capability is essential since users might need to gain root on the nodes in order to install specific packages or to tune kernel parameters, or simply to install a non-standard OS. The computing resources of Grid'5000 are distributed over several sites (10 in 2012), mostly in France. Each site can support several hundred nodes. The major constraints of OS provisioning on the Grid'5000 testbed include:

- the operating system on any grid node can be modified by any user at any time, potentially several times a day. To avoid any conflicts between users, the tool must interact with the batch scheduler to specify which user has the right to deploy a specified set of nodes during a time slice.
- the reconfiguration time of an entire cluster (100+ nodes) must be small enough (on the order of ten minutes) and the reconfiguration process reliable enough to let users quickly set up their experiments.
- several clusters of a given site (with different hardware characteristics) and even several grid sites can have to be reconfigured in one shot, in the case of a grid experiment for instance.
- even though Linux is the most widely used operating system on Grid'5000, non-Linux/*nix based operating systems must also be supported.
- some users do not need to deploy a specific environment; a standard production environment can be used instead. Whenever possible, the standard environment should be maintained between experiments so as to avoid total reconfiguration and longer setup times. This requires several technical restrictions, e.g., the new operating system must be installed on a separate partition and the master boot record must not be modified.
- since users can reconfigure the nodes as they see fit, nodes may be in almost any state after experiment completion. Thus no assumption must be made about the state of a node before a reconfiguration. Grid'5000 mechanisms allow to check if nodes have been altered after experiment completion in order to redeploy the standard environment.
- most users are accustomed to scripting their experiments, so the deployment tool must not only offer a command line interface, but also an API.

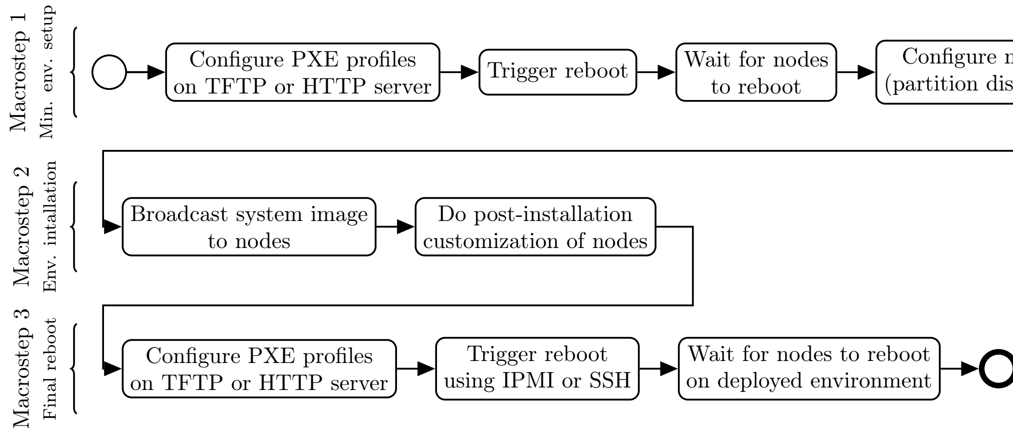

Figure 1: Kadeploy deployment process, composed of three macro-steps

Kadeploy3 has been designed to be used beyond the Grid'5000 testbed, so it is also possible to use it in a more common way (e.g., as an administration tool such as xCAT [14] or SystemImager [12]). Compared to its competitors, Kadeploy3 has better scalability properties, a property that is paramount since we would like to target petascale and future exascale clusters. The solution is relying on standard mechanisms (such as network boot), tunable (to be compliant with any technologies) and hardware independent. As a consequence Kadeploy3 is potentially usable with any kind of HPC cluster.

In this paper, we present an experiment in which we deploy 4,000 nodes in a single administrative cycle. Kadeploy3 is flexible enough to load or release dynamically certain features in order to fit a wide range of existing infrastructures and requirements.

The paper is organized as follows. Section 2 gives an overview of Kadeploy3 concerning the deployment process and its ecosystem. Section 3 describes the designs principles, in particular those related to the software architecture and to the scalability. Section 4 shows qualitative and quantitative evaluation elements that demonstrate the feasibility and the efficiency of our solution even in large scale environments. Section 5 details the lessons we have learned through the design, the development and the use of the tool. Section 6 compares Kadeploy3 to the state of the art. Finally, section 7 concludes the papers and presents the future directions we plan to investigate.

2 Overview

In this section we give an overview of Kadeploy3. First, we detail the steps that comprise the deployment process and how this process can be reliable. Then, we explain how Kadeploy3 interacts with the infrastructure ecosystem and in particular, the issue of node booting. Finally we present the suite of tools associated to Kadeploy3.

2.1 Deployment Process

Kadeploy3 belongs to the family of the *disk imaging and cloning* tools. Thus it takes as input an archive containing the operating system to deploy, called an *environment*, and copies it on

the target nodes. As a consequence, Kadeploy3 does not install an operating system following a classical installation procedure and the user has to provide an archive of the environment (as a *tarball*, for Linux environments).

As shown in figure 1, a typical deployment with Kadeploy3 is composed of three major steps, called *macro steps*.

1. *Minimal environment setup*: the nodes reboot into a *trusted* minimal environment that contains all the tools required for the deployment (partitioning tools, archive management, ...) and the required partitioning is performed.

2. *Environment installation*: the environment is broadcast to all the nodes and extracted on the disks. Some post-installations operations can also be performed.

3. *Reboot on the deployed environment.*

Each *macro step* can be executed via several different mechanisms to optimize the deployment process depending upon required parameters and the specific infrastructure. For instance, the *Reboot on the deployed environment* step can perform a traditional reboot or it might instead rely on a call to `kexec` [1] for a shorter reboot.

Reconfiguring a set of nodes involves several low-level operations that can lead to failures for various reasons, e.g., temporary loss of network connectivity, reboot longer than planned, etc. Kadeploy3 is designed to identify these failures as quickly as possible and improve deployment reliability by providing a *macro step* replay mechanism on the nodes of interest. To illustrate that, let's consider the last deployment *macro step* that aims at rebooting on the deployed environment. Kadeploy3 implements, among others, the following strategies:

1. directly load the kernel inside the deployed environment thanks to `kexec`;

2. perform a *hard* reboot using an out-of-band management hardware without checking the state of node.

Thus it is possible to describe strategies such as: try to launch the first strategy; then if some nodes fail, try to launch the second strategy two time if required.

2.2 Interaction with the Ecosystem

Kadeploy3 does not directly take control of the nodes since it would require some specific and uncommon hardware support. Instead, it uses common network boot capabilities based on the PXE protocol [11]. This works as follows:

1. at boot time, the network device of the node makes a DHCP query;

2. a DHCP server responds to the node with the address of the server containing the PXE profile (i.e., the information describing how the node is supposed to boot);

3. the node fetches the PXE profile from the server and performs the required operations. Depending upon the profile, some operations may require retrieval of more files, such as a kernel.

Using such a mechanism, combined to the capability to dynamically update the PXE profiles of the nodes and with the capability to reboot the nodes in a reliable way (thanks to BMC, RSA or PDU capabilities for instance), it is possible to take the control of the nodes and to specify what they are booting.

[1] `http://www.ibm.com/developerworks/linux/library/l-kexec/index.html`

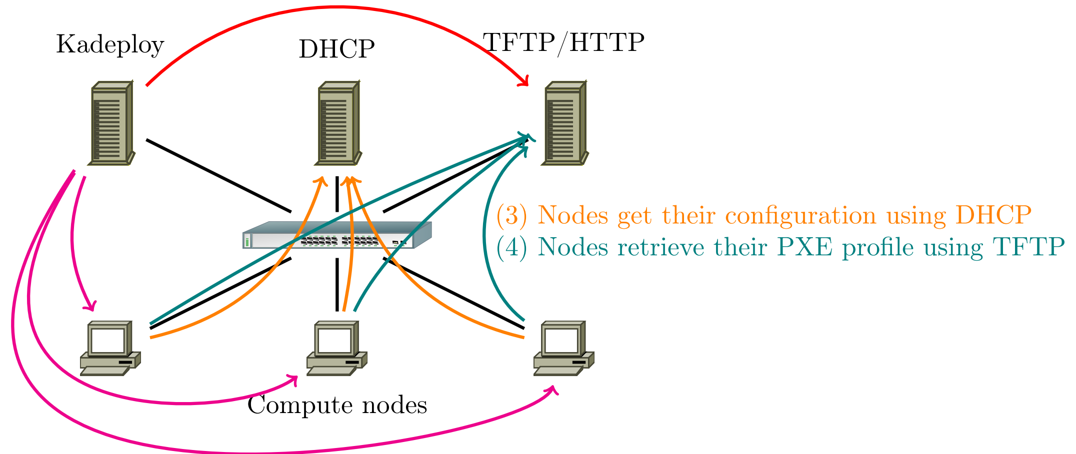

Figure 2: Interactions between Kadeploy and its ecosystem

Figure 2 shows the interaction between Kadeploy3, the infrastructure services and the cluster nodes.

2.3 The Kadeploy3 software suite

Kadeploy3 is packaged with a set of complementary tools that are briefly described in this section.

Management of images
the *Kaenv* tool enables users and administrators to manage a catalog of deployment images, either shared between users or private to one user.

Rights management
Karights is used to define deployment permissions for users. It also provides the glue to integrate Kadeploy with a batch scheduler, making it possible to allow a given user to deploy a set of nodes for the duration of his job on the cluster.

Statistics collection
deployment statistics (durations, success/failures) are collected continuously, and available through the *Kastat* tool. This data can be leveraged by system administrators to identify nodes with hardware issues.

Frontends to low-level tools
Tools such as *Kareboot*, *Kaconsole* and *Kapower* act as frontends to lower-level tools (such as those based on IPMI [7] and integrate with the Kadeploy rights management system, to allow users to reboot, power-off/on, and access the remote serial console of nodes.

3 Design Principles

3.1 Software Architecture

We detail here several aspects of the Kadeploy3 architecture and the reasons behind our choices.

3.1.1 Client/Server Architecture

Kadeploy3 implements a client/server architecture. The majority of the deployment code is located on a server that requires write access to the PXE profile repository. All tools presented in section 2.3 can be used either using a lightweight client or using an RPC API. A typical installation would use one Kadeploy server for a set of clusters that are reachable on the same local network. In our terminology, a cluster is a set of nodes with identical characteristics (for performance concerns) located in the same area (typically the same room where the latency between the nodes is very low). When using one Kadeploy server to manage several clusters (what we call a site), the latency between the server and the nodes is supposed to be quite low also since several infrastructure services are shared (e.g., TFTP server for PXE boot, database, etc). In this case the clusters are supposed to be inside the same building. This design has several benefits:

- Support for concurrent deployments is one objective of Kadeploy3. Indeed, in the Grid'5000 context, several users can easily deploy nodes at the same time. Having a global view of all deployments allows certain optimizations such as simplified database access (required for rights management, environment management and some logging operations), support for an environment cache on the server, and support for mastering some concurrent operations that would hurt the infrastructure in case of massive execution (e.g., reboots, power operations, etc);
- It is possible, from a single client, to control deployments from multiple servers (typically located at different grid sites) with a single client. This feature is currently used on the Grid'5000 testbed to perform concurrent deployments on several grid sites in a single administrative cycle;
- Since the PXE repository requires root access, the server needs to run Kadeploy tasks as root, but the clients can run as a normal user. This limits potential security issues.

3.1.2 Deployment Process Management

As stated in section 2.1, several mechanisms may be available to carry out each *macro step*. Kadeploy3 implements a minimal workflow engine that manages the lifecycle of the deployment process.

First, it is possible to specify a predefined mechanism for each *macro step* in the configuration. Robustness is also supported by telling the system to perform several rounds of a given *macro step* in the case of failures in a subset of nodes. Furthermore, *macro step* mechanisms may be combined in different permutations across multiple attempts. This may increase reliability by, e.g., using an optimized but unreliable mechanism in a first round and then using safer but less optimal mechanisms in subsequent rounds to ensure eventual success. The last macro-step of rebooting nodes in a deployed environment illustrates this approach.

In order to accommodate special requirements, complex deployment processes can be altered or completely redefined by users on the command line. Furthermore, the deployment process can be enhanced at any time. This enables users to act like administrators and perform specific operations that are not necessary for a majority of users.

Finally, the *Environment installation macro step* has the capability to execute post-configuration operations once the environment is copied to node disks. This allows customization of the environment and is often required to assign some specific configuration parameters. Kadeploy3 defines two levels of post-configuration. The first is specified by administrators and is applied in every deployment and the second can be defined by users to tailor their own deployment. More specifically, the deployment process can be augmented at any time by allowing users to add hooks in the workflow engine.

3.1.3 Booting into the deployed environment

Once the environment is copied on the nodes and the post-installation operations are performed, the environment has to be launched. This is done by updating the PXE profile of the nodes followed by a reboot. Since the Grid'5000 context adds the constraint of keeping the master boot record of the nodes unmodified, it is not possible to leverage on a classical boot method.

Kadeploy3 offers two methods to launch an environment, both based on network boot.

The first method, called *pure PXE boot*, consists in fetching the kernel files (vmlinuz, initrd, and possibly an hypervisor) over the network in order to directly boot a node (the environment files are written on the disk). This methods only works with environments based on Linux. Because the deployed environment can vary, the kernel files must be extracted from the environment on the Kadeploy server before being placed in the PXE repository for a future fetch. To help the extraction, the environment creator must specify in the environment description, the paths to the necessary kernel files. Although a cache mechanism can be used to reduce the burden of repetitive extraction when the same environment is deployed, PXE boot remains heavy-handed. Furthermore, the kernel files (several MBs) have to be fetched, over TFTP or HTTP (in the best case), by all deployed nodes, which contributes to network congestion. These limitations make this method suitable only for uncommon situations, e.g., when the hardware is not compliant with the advanced Syslinux [13] features detailed in the rest of this section.

The second method, called *chainload boot*, uses the Syslinux `chain.c32` comboot to boot a partition directly on a node. In this case, the PXE profile contains a reference to `chain.c32` and the number of the partition from which to chainload. At boot time, after the PXE profile fetch, the node fetches `chain.c32` and boots directly off the specified partition. This method assumes that a bootloader is installed in the given partition. Here, we have two situations:

- the deployed environment is based on Linux. According to the paths specified in the environment description, Kadeploy3 generates a Grub2 configuration and installs it on the target partition;
- the deployed environment is not based on Linux. In this case, the environment is contained within a raw partition image that already embeds a bootloader. The image is written directly to the target partition on the node.

3.1.4 Reliability of Reboot and Power Operations

Since reboot and power operations are essential to control of cluster nodes, and ultimately the entire deployment process itself, they must behave correctly and reliably. Several methods can be used to reboot a nodes:

1. *soft reboot*: direct execution of the `reboot` command;
2. *hard reboot*: via the reboot capability of out-of-band management hardware with protocols such as IPMI. Various kinds of reboot can be executed: reset, power cycle, etc;

3. *very hard reboot*: use the power management capability of the power distribution unit (PDU).

Obviously it is better to use a method that leverages available hardware parts. Performing a `reboot` is the best solution with regards to speed and cleanliness. But it may not be an option if the target node is unreachable via in-band methods such as ssh (e.g., the node is already down, the OS has crashed, an unfriendly operating system is installed, etc). In this scenario, we would use IPMI-like features if available. Also, it might be better for speed to perform a reset rather than a power cycle since it bypasses the power on self test, but sometimes this is not sufficient. Finally, if onboard management hardware is unreachable, we may be required to use the capabilities of a remotely manageable PDU.

Kadeploy3 provides administrators with a way to specify several levels of commands in order to perform escalation if required. This allows them to perform highly reliable deployments if the clusters have the appropriate hardware. Advanced scripts that deal with multiple reboot methods can also be used to improve flexibility and reliability. Unfortunately, depending on the methods chosen, reboot escalation comes at a cost. Thus a balance must be struck between desired reliability and the time to deployment. Typically escalation operates by first performing a *soft reboot*. In case of failure on some nodes, a *hard reboot* is performed and a *very hard reboot* if necessary.

Implementing a generic command escalation feature can be a bit difficult. Actually, some commands might involve several nodes, e.g. those attached to the same PDU. In this case, performing a PDU power-off can shutdown several nodes. Kadeploy3 allows one to specify groups of nodes in relation to a given command level. These groups can be used for instance to describe the nodes plugged into the same PDU. When a deployment involves some nodes that require such an operation, it is only performed on the nodes that do not belong to any group or on the nodes whose entire group is involved in the deployment.

3.1.5 Configuration

A Kadeploy server is supposed to manage several clusters in the same LAN. Because different clusters have different hardware specifications (e.g. NICs, hard-drives, out-of-band management hardware, etc), they have to be managed independently from the deployment point of view.

Configuring Kadeploy3 on the server side consists in filling four kinds of YAML files:

1. a common configuration file defines general parameters of a Kadeploy server (e.g., the access to the database for rights and environments management, the windows sizes for the reboot commands, etc).

2. a global file defines the different clusters managed by a server as well as the nodes (host-names and IPs) included in each cluster.

3. as much specific configuration files as the number of defined clusters. Each cluster configuration file defines the specific configuration elements peculiar to a cluster (e.g., the minimal environment and its parameters, the reboot/power commands, the reboot timeouts, or the definition of the deployment workflow composed of the different *macro steps*).

4. for specific requirements, it is possible to define specific reboot or power commands for some nodes. This specific configuration overloads the commands defined into the cluster configuration files and avoid the creation of unnecessary Kadeploy clusters for some exceptions.

On the client side, it is only necessary to specify the location of the server. If multiple servers are defined into the client configuration file, Kadeploy3 also offers a facility to perform deployment on multiple servers in a single administrative cycle.

3.2 Key Features for Scalability

3.2.1 Parallel commands

The deployment workflow contains several operations that reduce to executing a command on a large set of nodes. The exit status of the command returns to the deployment process the subset of nodes for which the command was executed correctly.

Thanks to SSH, one can execute commands remotely and retrieve their outputs (`stdout`, `stderr`, exit status, etc). Launching SSH commands on a large number of nodes in sequence does not scale at all. Furthermore, launching all commands simultaneously can impose an extreme load on the client and saturate all of its file descriptors.

Several tools have been built to overcome these limitations. For instance, Pdsh [10] and ClusterShell [2] are designed to execute SSH commands on many nodes in parallel. Both tools use windowed execution to limit the number of concurrent SSH commands and both also allow retrieval of command outputs on each node.

We choose to leverage TakTuk [16] as our mechanism for parallel command execution and reporting. TakTuk is based upon a model of hierarchical connection. This allows it to distribute the execution load on all the nodes in a tree and to perform commands with low latency. Using such a hierarchical mechanism would normally require the tool to be installed on all nodes. Fortunately, TakTuk includes a convenient auto-propagation feature that assures the tool's existence on all necessary nodes. The tool also uses an adaptive work-stealing algorithm to improve performance, even on heterogeneous infrastructures.

3.2.2 File broadcast

The broadcast of the system image to all nodes is a critical part of the deployment. In cluster environments where the most important network for applications is using Infiniband or Myrinet, the Ethernet network is often composed of a hierarchy of switches (e.g.; one switch per rack) that is hard to leverage for a high-performance broadcast. File distribution to a large number of nodes via any sequential push or pull method is not scalable. Kadeploy3 provides system administrators with three scalable file distribution approaches during the *Environment installation macro step* to minimize deployment time.

With tree-based broadcast, a file is sent from the server to a subset of nodes, which in turn send the file to other subsets until all the nodes have received. The size of the subsets, called tree arity, can be specified in the configuration. A large arity can reduce the latency to reach all nodes but transfer times might increase because global bandwidth is equal to the bandwidth of a network link divided by the tree arity. The opposite effect occurs when the arity is small. In general, this broadcast method does not maximize bandwidth and should be used primarily for the distribution of small files. This method is also inefficient when used in hierarchical networks. We implement tree-based broadcast using TakTuk.

Chain-based broadcast facilitates the transfer of files with high bandwidth. In practice, this broadcast method is a special case of tree-based broadcast with an arity of 1. A classical chain-based broadcast suffers from the establishment time of the chain in large scale clusters. Indeed, since each node must connect to the next node in the chain (usually via SSH), a sequential initialization would drastically increase the entire broadcast period. Thus we perform the initialization of the chain with a tree-based parallel command. This kind of broadcast is near-optimal in a

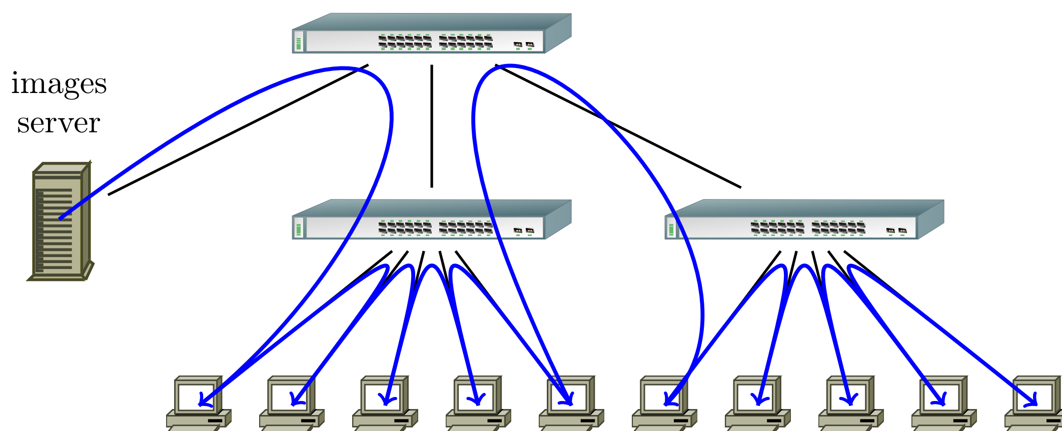

Figure 3: Topology-aware chained broadcast. Data is pipelined between all nodes. When correctly ordered, this ensures that inter-switches links are only used once in both directions.

hierarchical network if the chain is well ordered since, as shown in Figure 3, all the full-duplex network links can be saturated in both directions, and the performance bottleneck becomes the backplane bandwidth of the network switches. For this method, we implement chain initialization using TakTuk and perform transfers using other custom mechanisms.

BitTorrent-based broadcast allows us to send files at large scale without making any assumptions about the quality of the network. Furthermore, BitTorrent is able to efficiently handle churn, an important property in large scale systems like petascale and future exascale clusters. Currently, our experiments show that there are two scenarios in which the performance of this broadcast method is inferior to the other methods. The first pathological case is one in which we are broadcasting on a small-scale cluster with a high-speed network, and the second is one in which we are broadcasting small files. In both cases, BitTorrent exhibits high latency and the overhead of the protocol dominates the time to broadcast. Also, as shown in Figure 4, the large number of established connections between nodes induced by the protocol can lead to bottlenecks (in red) depending on the network topology.

3.2.3 Windowed operations

Our experience in the development of a cluster deployment tool shows that several low-level operations must be performed carefully. For instance, it can be disastrous to reboot a huge number of nodes at the same time in the same technical room since it can generate electric hazard. It can also flood the network with DHCP requests and accidentally lead to multiple boot failures.

Kadeploy3 addresses this problem by supporting the definition of windows for multiple operations like reboots, power operations or nodes check (this leads to scan some ports).

Let's take the reboot example to illustrate that. According to the infrastructure capabilities (e.g., network, service nodes, etc) the administrator of a cluster estimates that no more than 100 nodes should be rebooted at once, and a grace period of 10 seconds should be observed after having rebooted 100 nodes to reboot more nodes. Imagine that you have 400 nodes to reboot. In this configuration, Kadeploy3 reboots 100 nodes, waits 10 seconds, reboots 100 nodes, and so

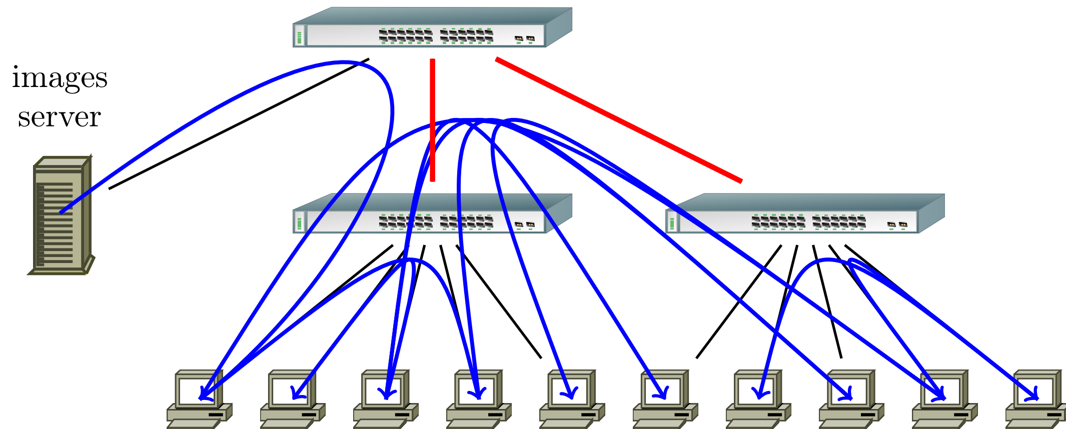

Figure 4: Bittorrent broadcast. The random selection of peers leads to saturation of the inter-switches links.

on until all the nodes have been rebooted.

To be effective in a context where concurrent deployments can occur, windows are applied globally on a Kadeploy3 server.

4 Evaluation

In this section, we describe and evaluate Kadeploy3 along several axes: its source code, its current usage, some benchmarks on physical infrastructures, and scalability benchmarks on virtualized infrastructures.

4.1 Software

Kadeploy3 is developed in Ruby and has about 10,000 SLOC. Source code is available under a free license, as well as Debian and Red Hat packages. As other tool developers have discovered (e.g. Puppet, Chef, etc), Ruby is well-suited to the objectives of this type of project thanks to its high expressiveness and its meta-programming features. Furthermore, the core functionality of Kadeploy3 has no special performance requirements except for parallel command execution and file broadcast, which are easily delegated to other tools. Thus we favor the simplicity of development and a high maintenance potential.

4.2 Qualitative Elements

Kadeploy3 has been used intensively on the Grid'5000 testbed since the end of 2009. In that time, approximately 620 different users have performed 117,000 deployments. On average each deployment has involved 10.3 nodes. The largest deployment involved 496 nodes (in multi-site mode). To our knowledge, the deployed operating systems are mostly based on Linux (all flavors) with a sprinkling of FreeBSD. Although the Grid'5000 use case does not exercise all the goals targeted by Kadeploy3 (e.g., scalability), it shows the tool's adequacy with regard to most characteristics, such as reliability.

	min	big
Average time for the 1st macro step	229	255
Average time for the 2nd macro step	56	143
Average time for the 3rd macro step	240	230
Average file broadcast time	31	126
Average deployment time	564	675

Figure 5: Deployment times (seconds)

Kadeploy3 is also used in more straightforward contexts, for administration tasks on production clusters in several french research institutes at least. In this case, several features are not used but scalability and reliability contributes to ease some administration tasks and to minimize the downtime.

4.3 Validation in Real Infrastructure

4.3.1 Deployment of a Regular Cluster

In this section, we provide the reader with an idea of the time required to deploy regular clusters. We perform a set of deployments on 130 nodes of the `graphene` cluster from the Nancy Grid'5000 site. We have launched 5 deployments with a 137 MB environment (called *min deployments*) and 5 deployments with a 1429 MB environment (called *big deployments*).

In those deployments, all the nodes were deployed correctly. Actually some temporary failures occurs (between 0 to 4 nodes each time), but the robustness mechanisms of Kadeploy3 where used to ensure the full deployment success. In particular, we add a possible retry on the first *macro step*.

Figure 5 details some measures made during the deployments. We can see that deploying 130 nodes in one shot takes between 9 to 12 minutes, depending on the environment size. The deployment time is mainly spent in two steps: the wait of rebooted nodes (in the 1st and in the 3rd *macro step* and the broadcast file (in the 2nd *macro step*). The 1st and the 3rd *macro steps* are of the same kind for *min* and *big* deployments, the difference is due to the small sampling. Indeed rebooting 130 nodes can take a variable time according to several uncontrolled parameters in the infrastructure.

To ensure reliability of the deployments we chose to use a non-aggressive configuration where large timeouts have been set in order to give slow nodes a chance to appear after a reboot and also to enable retries. Faster deployment times would have been reached with a more aggressive configuration, at the price of a few node loss. To be rigorous, we have to precise that the results shown in figure 5, excepted for the average deployment time, do not take into account the time spent for nodes that had needed a second try. Actually, the time spent in the retry is partially covered by the main deployment because rebooting just a few nodes is faster than rebooting a large set and other operations like file broadcast are also faster on a small node set. That is why adding the average time spent in the 3 *macro steps* give a smaller value than the average deployment time.

4.4 Scalability Validation in Virtualized Infrastructure

Validating scalability on large physical infrastructures can become complex because it requires privileged rights on many components (e.g., access to management cards, modification of PXE

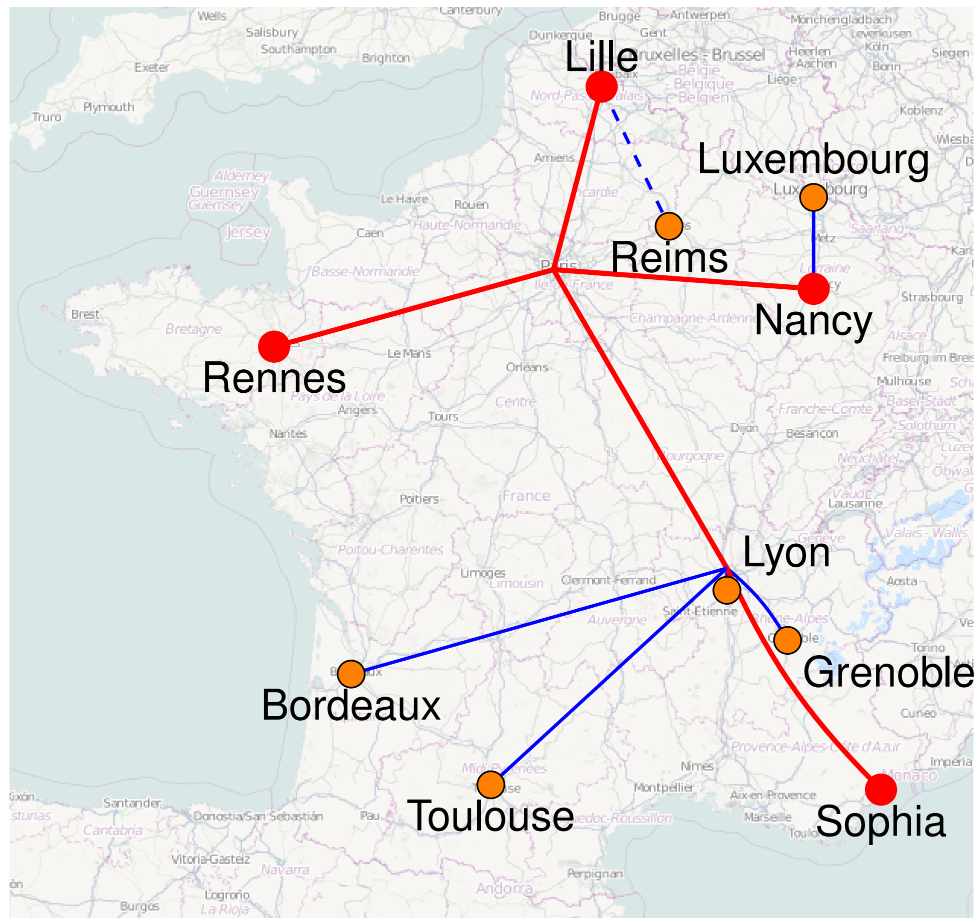

Figure 6: Grid'5000 network topology

profiles, etc). Thus we chose to build our own large-scale virtual testbed on Grid'5000, leveraging important features such as link-layer isolation, and Kadeploy3 of course.

Figure 6 shows the network topology of the Grid'5000 platform. Our testbed comprised four network-isolated sites (red nodes and associated links). We had access to 635 nodes upon which we launched a variable number of KVM virtual machines, depending on the node's capabilities. In total, 3,999 virtual machines were launched and participated in a single virtual network. Building the testbed was tricky for several reasons like the high latency between two sites that affects the infrastructure services. Those reasons are more deeply explained in section 5.3. Thanks to the versatility of the Grid'5000 platform, we were able to construct the testbed without any privileged user rights and without suspending access to the platform by other legitimate users.

Once the testbed was launched, we were able to perform deployments within a single cluster of 3,999 nodes. During the largest run, a 430 MB environment was installed on 3,838 virtual machines in less than an hour. 161 virtual nodes were lost due to network or KVM issues. A significant amount of time was also wasted because of the high latency between sites (10-20 ms), which affected some infrastructure services like DHCP and the PXE protocol.

Here are few statistics of the time spent in the time-greediest steps: the first reboot took 11

minutes, the broadcast of the image archive took 15 minutes, the second reboot took 7 minutes. We should also highlight the fact that this experiment was designed a non-friendly way for Kadeploy. Indeed, Kadeploy is designed as a cluster oriented tool and in this experiment, we described our resources as resources of the same cluster while in reality the physical machines were distributed over four sites separated by hundreds of kilometers. This was unfavorable to the parallel commands execution mechanism that works effectively with small latency differences between the nodes involved in a deployment.

This experiment demonstrates the effectiveness of the Kadeploy3 design in terms of scalability and reliability. In particular, hierarchical parallel commands and chain-based broadcast are well-suited to deployments at this scale. With regard to reliability, roughly 96% of the nodes were deployed successfully. Deployment to other nodes failed due to infrastructure issues but a large number of those (e.g., the ones not affected by KVM problems) could have been recovered using a fallback approach, at the cost of a longer deployment time.

We are confident that Kadeploy3 will scale to larger infrastructures and will be a good choice for node reconfiguration on a variety of current and future clusters.

5 Lessons Learned

In this section we highlight three technical points that gave us a bit of trouble in the design and testing of Kadeploy3. The goal is to allow other administrators to save time with the lessons we have learned.

5.1 Dealing with Unreliable Reboot Mechanisms

Since our node control mechanism relies heavily on remote reboot, it is essential to ensure that it is performed correctly. Unfortunately, we learned that some of these mechanisms are not reliable. Much of the problem can be attributed to varying and sometimes incorrect vendor implementations of IPMI and other protocols. This leads to situations in which the management card is not able to reboot a node that is in an unexpected state or otherwise unreachable. Furthermore, IPMI is sometimes implemented on the same Ethernet port that the node uses for its own traffic. Traffic competition can lead to unresponsive management cards.

In section 3.1.4, we presented the escalation command mechanism to increase the reliability of the reboot and power control operations. This mechanism depends upon the return status of the executed commands to decide if escalation is required on some nodes. If the return status of the executed commands is not correctly reported, it can lead to unreliable reboots. We found that instead of using raw IPMI commands (or their equivalent for other protocols) we could improve reliability by wrapping the commands in ad-hoc scripts that test several preconditions and postconditions.

5.2 First Steps of the Boot

Still related to the node control mechanism, we address the first steps of the boot. As stated in section 2.2, just after the DHCP step, nodes must retrieve their PXE profile and possibly some associated files. To achieve this, a typical DHCP configuration would instruct the nodes to retrieve `pxelinux.0` (from the Syslinux project [13]). Then the nodes are able to continue retrieving other necessary files. All transfers are performed over TFTP, which is not reliable and not scalable, and which can lead to boot issues.

We found that one way in which to increase reliability and speed is to use `gpxelinux.0` (also from the Syslinux project) instead of `pxelinux.0`. Even though the retrieval of `gpxelinux.0`

still has to be done over TFTP, other retrievals can be made over higher level protocols such as HTTP. This greatly improves scalability and reliability since, e.g., HTTP servers can deal with high numbers of static requests, as opposed to the horrible performance exhibited by TFTP servers under high load.

We also found that PXE implementations in many BIOSes are often poor. When possible, we highly recommend that network card firmware be updated to use something such as iPXE [8]. This is an open-source boot firmware that enhances a classical PXE implementation with features such as booting directly from an HTTP server, from iSCSI, or from an Infiniband network. This approach completely bypasses retrievals via TFTP and highly increases the speed and reliability of the boot stage.

5.3 Scalability Testing

Building scalable tools always brings the question of the scalability evaluation. Tools like Kadeploy3 require a complex infrastructure to be tested and cannot use methods like simulation. As said in section 4.4, we have built a tool that allows the deployment of a large-scale infrastructure thanks to virtualization and to advanced features of Grid'5000. We would like to underline some issues we encountered.

The first problem when dealing with a large number of nodes was related to the poor reliability and scalability properties of the TFTP protocol. From dozen of simultaneous requests, TFTP server was quickly overloaded, resulting in a denial-of-service (DoS) that leads to multiple boot failures. To solve this problem, we have configured virtual machines with an iPXE compliant NIC in order to be able to use HTTP protocol instead of TFTP for PXE profiles and kernel files fetching. Using HTTP allows to boot hundreds of nodes at the same time. Furthermore, thanks to some HTTP server tunings, we were able to serve twice as much requests, with an increased bandwidth.

Relying on a more scalable protocol was useful to improve the performance but it was not enough to get rid of DoS issues. A service accessed from thousands of nodes at the same time can quickly become overloaded, especially if communication sockets are not quickly closed (e.g., when broadcasting large files). Thus we had to replicate the infrastructure services. To enable PXE profiles and kernel files distribution from multiple HTTP servers, we tuned the DNS server to split the requests over several HTTP servers. Because the testbed is distributed over several Grid'5000 sites, the iPXE HTTP implementation also suffers from the high latency between the grid sites. Thus we had to replicate the HTTP servers on every site, and thanks to another DNS tuning, the nodes of a given site were only served by local HTTP servers.

We also had to deal with ARP tables overflow. Since some services (e.g., DHCP) were accessed by every nodes, the ARP tables of those servers were quickly saturated. This issue has been solved by increasing the ARP tables on the servers thanks to on-the-fly Linux kernel tuning facilities.

6 Related Works

This section present some tools that can be compared to Kadeploy3. We only present open-source solutions since we are not able to freely study the commercial ones.

xCAT [14] aims at doing operating system provisioning on physical or virtual machines. It supports scripted install, stateless deployment and cloning deployment. Concerning physical machines, it targets RPM based Linux distributions (RHEL, CentOS, Fedora), IBM AIX and some Windows versions. Even if it can be used more or less easily on any hardware, xCAT is strongly designed to run on IBM one. In this case, it offers a set of features to quickly set

up and control the management node services (DNS, DHCP, TFTP, HTTP). According to the developers, a single xCAT instance can scale up to 128 nodes clusters. Beyond this, several xCAT instances can be deployed hierarchically to scale far away. To perform broadcast operations, xCAT relies on hierarchical HTTP transfers or iSCSI. Comparing to Kadeploy3, it offers extra services like scripted install and stateless deployment. However it is less versatile concerning the operating systems and the supported hardware. It also does not allow to perform large scale deployments in a single shot since it does not leverage on scalable tools for parallel commands and file broadcast like Kadeploy3 does. Finally, xCAT only targets administration tasks and does not allow user deployments.

Clonezilla [1] is a cloning tool that supports almost any operating system since it is compliant with a lot of file systems. For unsupported file system, Clonezilla can do sector-to-sector copy using `dd`, like Kadeploy3. Clonezilla is proposed in two versions: Clonezilla Live and Clonezilla Server Edition. Since Clonezilla Live is dedicated to single machine backup, we focus on Clonezilla Server Edition. It relies on DRBL [4] that provides a diskless environment for cluster nodes. Clonezilla has no dependency to any specific hardware and can use either unicast or multicast transfer for environment broadcast. However, it is not designed for large scale clusters but instead infrastructures like classrooms.

SystemImager [12] aims at replicating a golden node on a cluster. It can either perform bare metal installation of cluster nodes or propagate updates from the golden node to the cluster nodes using `rsync`. In the case of bare metal installation, the environment broadcast can use `rsync`, multicast, and BitTorrent for a better scalability. Only file broadcast takes scalability concerns into account, thus only a few hundred of nodes can be deployed in one shot. Compared to Kadeploy3, SystemImager is less versatile since it targets only the administration purpose and only supports Linux based operating systems. SystemImager is widely used and is a keystone for higher level tools like Oscar [9].

Other tools like Cobbler [3] or FAI [5] aims at installing a Linux operating system on a cluster following the regular system installation process. These tools are not suitable for large scale clusters and only focus on Linux.

7 Conclusion and Future Works

This paper presents Kadeploy3, an efficient, reliable and scalable tool that aims at performing operating system provisioning on HPC clusters.

Kadeploy3 has been designed in the context of the Grid'5000 experimental testbed where the goal was to provide grid users with a way to fully reconfigure the grid nodes as they see fit during their experiments.

Kadeploy3 is also well suited for classical administration tasks. Indeed, the reliability and the scalability properties of Kadeploy3 allow to safely perform operating system provisioning on large scale clusters without inducing long downtimes like it is the case with some unreliable and poorly scalable tools of the state of the art.

Because it is used since 2009 on the Grid'5000 context by several hundreds of users and because it is also used for administrative tasks on production clusters, Kadeploy3 has proven to be useable, efficient, and reliable for day-to-day use.

We have also demonstrated the scalability of Kadeploy3 by presenting experiments where we have performed successfully a deployment on a petascale cluster comprising 2000 nodes, and where we also have performed deployments into a virtualized testbed comprising about 4000 nodes.

Future works will be dedicated to the improvement of the file broadcast techniques, in partic-

ular when the network topology is complex, and to the improvement of the hierarchical parallel command tool in the situation where the tree must not be randomly created, for instance when latency between the nodes is heterogeneous.

Acknowledgements

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the Inria ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see `https://www.grid5000.fr`). Special acknowledgements to Olivier Richard for his contribution to the previous versions of Kadeploy and to several ideas presented in this paper, and to Marc Chiarini for his help at improving the paper.

Availability

Kadeploy3 is freely available under the Open Source CeCILL v2 license on

`http://kadeploy3.gforge.inria.fr`

References

[1] Clonezilla. `http://clonezilla.org`.

[2] ClusterShell. `http://cea-hpc.github.com/clustershell`.

[3] Cobbler. `https://fedorahosted.org/cobbler`.

[4] Diskless Remote Boot in Linux. `http://drbl.sourceforge.net`.

[5] Fully Automatic Installation (FAI). `http://fai-project.org`.

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

[7] Intelligent Platform Management Interface (IPMI) Specification. `http://www.intel.com/design/servers/ipmi/spec.htm`.

[8] iPXE - Open source boot firmware. `http://ipxe.org`.

[9] Oscar: Open Source Cluster Application Resources. `http://svn.oscar.openclustergroup.org/trac/oscar/wiki`.

[10] Parallel Distributed Shell. `http://sourceforge.net/projects/pdsh`.

[11] Preboot Execution Environment (PXE) Specification. `http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf`.

[12] SystemImager. `http://systemimager.org`.

[13] The Syslinux Project. `http://www.syslinux.org`.

[14] xCAT: Extreme Cloud Administration Toolkit. `http://xcat.sourceforge.net`.

[15] Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.-G., and Irena, T. Grid'5000: a large scale and highly reconfigurable experimental Grid testbed. *International Journal of High Performance Computing Applications 20*, 4 (nov 2006), 481–494.

[16] Claudel, B., Huard, G., and Richard, O. TakTuk, adaptive deployment of remote executions. In *International Symposium on High Performance Distributed Computing (HPDC)* (2009), pp. 91–100.

RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399