Scalability Testing of the Kadeploy Cluster Deployment System using Virtual Machines on Grid'5000

Luc Sarzyniec, Sébastien Badia, Emmanuel Jeanvoine, Lucas Nussbaum

Scalability Testing of the Kadeploy Cluster Deployment System using Virtual Machines on Grid'5000

Luc Sarzyniec, Sébastien Badia, Emmanuel Jeanvoine, Lucas Nussbaum
LORIA — INRIA Nancy – Grand-Est — Université de Lorraine
firstname.lastname@inria.fr

1 Introduction

All users of computing resources such as clusters or clouds know that scalability is a key feature of their applications. What users might not be aware of is that the infrastructure software used to run such platforms share the very same needs in terms of scalability. Typical examples are the applications used to copy a full software environment (operating system, libraries, user tools, etc.) to each node, which are used by system administrators during maintenance to install new nodes or update the software image on current nodes. More efficient deployment solutions mean shorter downtimes due to maintenance: on clusters composed of thousands of nodes, the process of re-installing all nodes can take hours or even days if the deployment solution does not exhibit the required scalability.

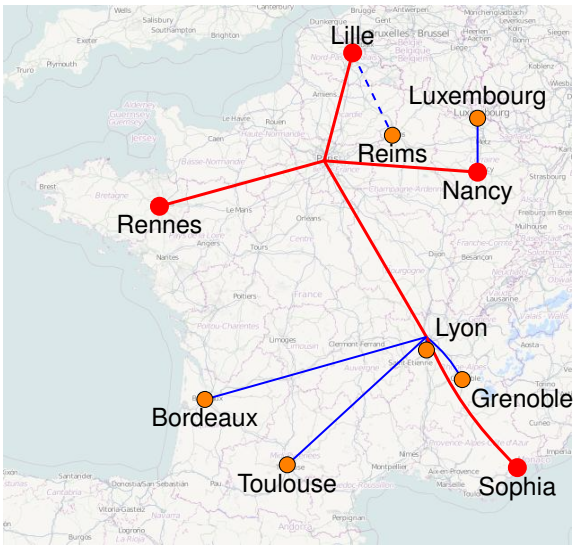
Kadeploy [1, 2] is a cluster deployment solution designed with scalability in mind. In this paper, we describe a set of experiments conducted on the Grid'5000 testbed to explore the scalability of Kadeploy, using up to 3999 virtual machines to simulate nodes being deployed. Grid'5000 is a testbed for research on distributed systems (Cloud, Grid, HPC, P2P systems), composed of 1700 nodes, which provides users with a unique set of advanced features and services, including the deployment of user-provided software environments on the nodes (using Kadeploy), and network isolation of experiments using dynamic VLAN reconfiguration (with KaVLAN). It is therefore a very suitable testbed for performing such experiments.

The rest of this paper is organized as follows. First, section 2 provides a quick overview of Kadeploy internals. Section 3 describes the steps involved when running this experiment on Grid'5000. Section 4 presents scalability results that were obtained during those experiments, and section 6 concludes the paper.

2 Overview of Kadeploy internals

Kadeploy is a cluster deployment solution which has been developed at INRIA since 2002. The current version, version 3, was developed in 2008 to provide a robust and scalable deployment solution for Grid'5000. A typical Kadeploy deployment is composed of three steps:

1. The nodes being deployed are rebooted to a specific *deployment environment* by Kadeploy. The reboot itself is usually triggered using a remote administration card, e.g., using the IPMI protocol. In order to avoid issues such as voltage spikes, nodes are rebooted by small delayed groups (*reboot window*). Nodes then boot using network boot (PXE); Kadeploy writes PXE profiles to indicate how the node should boot. The retrieval of the *deployment environment* during the PXE boot is a critical step. In the worst case, it is performed by the bios using TFTP, which is rather unreliable and slow. If the NIC supports it (using an iPXE firmware, for example), it can be performed via HTTP.



(a)

Site	Cluster	Nodes	VM / node	VMs
lille	chicon	6	3	18
lille	chimint	18	8	144
lille	chinqchint	31	7	231
lille	chirloute	5	7	35
nancy	graphene	132	4	528
nancy	griffon	91	8	728
nancy	talc	113	8	904
rennes	paradent	54	8	432
rennes	paramount	19	4	76
rennes	parapide	24	8	192
rennes	parapluie	5	18	90
sophia	helios	54	3	192
sophia	sol	39	3	117
sophia	suno	44	8	352
Total		635	/	3999

(b)

Figure 1: Grid'5000 resources used during the experiment. Sites in red on (a) were used for the experiment, with the repartition of virtual machines given in (b).

2. After the node has been booted in that *deployment environment*, Kadeploy controls the node remotely using Taktuk [3] (a scalable and hierarchical SSH wrapper), does all the required configuration (disk partitioning, etc.) and broadcasts the system image to the nodes before doing the final configuration (per-node customizations). In order to achieve efficient broadcast, Kadeploy organizes the nodes into a chain, which provides near-optimal performance even on hierarchical networks (since Kadeploy uses the cluster's Ethernet network, and not for example Infiniband, hierarchical networks providing low performance are common).
3. Once all nodes have been configured, Kadeploy reboots the nodes again, this time to the newly deployed system image.

In the following section, we detail how Grid'5000 was used to build a development and test environment for Kadeploy.

3 Platform setup and experimental process

Building a development and test environment for Kadeploy requires the deployment of a set of intrusive network services, including a DHCP server (for PXE), and a TFTP and/or an HTTP server (to send the deployment environment). This yields several challenges, that were overcome using several Grid'5000 features.

First, those services must run as the *root* user. Grid'5000 enables users to deploy their own system images and get *root* access on the nodes using the testbed-provided Kadeploy installation.

Second, installing a second DHCP on an existing network might cause disruptions. To avoid any problem, we used the KaVLAN service provided on Grid'5000, that implements network isolation using dynamic reconfiguration of Ethernet switches. Another benefit of KaVLAN is that it provides VLANs spanning several sites at the Ethernet level (OSI level 2): this made it possible to use nodes from different Grid'5000 sites in the same deployment.

Grid'5000 provides a large number of nodes but only 783 of them had the required capabilities for our experiments and they are rarely all available at the same time. However, our goal was to experiment at a

much larger scale. We therefore used virtualization to provide from 3 up to 18 virtual machines on each physical node, depending on the physical node capabilities (Figure 1). The virtualization technology used was KVM. Due to our rather specific constraints, we did not reuse an existing Cloud solution but instead wrote a set of scripts to handle the KVM virtual machines creation, startup, shutdown and reboot.

Our experimental process is fully automated, and can be decomposed into several steps:

1. We reserve the required physical nodes, deploy our own system image using the Kadeploy installation provided by Grid'5000, and isolate the nodes into a specific VLAN using KaVLAN.
   This step took 20 minutes to reserve and deploy 668 nodes.
2. We instantiate virtual machines and prepare each physical node:
   - Nodes that will be dedicated to host services (DHCP, ...) are connected to the testbed network;
   - Each remaining physical node is configured to host virtual machines: dynamic detection of hosting capability and configuration of the network bridge used to connect the virtual machines;
   - We instantiate virtual machines on each physical node, creating virtual hard disk images (stored in RAM to optimize disk I/O performance) and assigning IP addresses.

   This step took 5 minutes to prepare 668 nodes (33 services and 635 hosts) and 3999 virtual machines.
3. We configure all nodes:
   - For the whole testbed, one Kadeploy server is configured, as well as one DNS server and one DHCP server. Every virtual machine is registered with each of these services;
   - A varying number of nodes on each site (depending on the number of local virtual machines) is dedicated to being HTTP servers serving the deployment environment to virtual machines during the network boot. The DNS server points virtual machines to a local HTTP server in round-robin assignment.

   Configuring our testbed composed of 4 Grid'5000 sites took 15 minutes.

At this point, it is possible to run Kadeploy deployments. The next section presents some timing results.

4 Results

Several experiments were done during our tests, but we only present here the results of the execution using the maximum number of nodes. This experiment was run with 668 physical nodes: 33 nodes were used to host services (DHCP, HTTP, ...) and 635 nodes were used to host virtual machines.

As mentioned before, the number of virtual machines hosted by each physical node varies. We used a script to determine how many virtual machines a node could host taking into account various constraints: at most one VM per core; 914 MB of RAM per VM (564 MB for the RAM-stored hard disk, and 350 MB for the VM itself); at most 18 VMs per node to avoid network congestion.

Since our VMs are located on distant sites, several hundreds of kilometers away, and since some services such as DHCP and DNS are sensitive to network overload, we decided to use Kadeploy's reboot window mechanism. This mechanism allow us to reboot our nodes by small groups and wait a specified amount of time between the reboot of each group. In this run we used a window size of 150 nodes and a delay of 20 seconds between the groups.

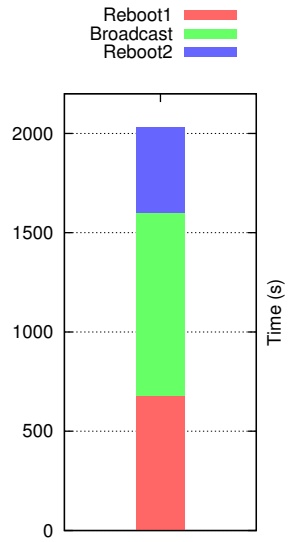


Figure 2: Timing results

Figure 2 shows the result of the experiment. The evaluation of our system is based on 3 criteria:

- The time of the first reboot (*Reboot1*): shutting down and relaunching each VM, then waiting for every VM to be up.

- The time spent in the user environment broadcast step (*Broadcast*): broadcasting the environment disk image file (430 MB) to all the VMs.
- The time of the second reboot (*Reboot2*): rebooting VMs using the Linux kernel *kexec* mechanism, enabling the reload of a new kernel without performing a hard reboot of a machine.

In this run we successfully deployed 3838 virtual machines, while 161 machines were lost due to network or KVM issues. The entire deployment process took 57 minutes.

5 Limits to scalability and future work

Two main factors limiting scalability have been identified: the mechanism used to reboot the nodes, and the image broadcast.

During the PXE reboot, each node gets its reboot parameters (IP, DNS, TFTP or HTTP server to retrieve its boot image) using DHCP, then uses TFTP or HTTP to retrieve its boot image. In the case of thousands of nodes rebooting simultaneously, network packet loss is not uncommon. Despite retry mechanisms, we still witnessed cases where nodes would not be able to get a DHCP reply, or where the image retrieval would time out. Those problems are mitigated in Kadeploy by rebooting nodes in smaller groups, but tuning the number of groups and the interval between the reboot of groups is difficult. The use of HTTP for system image retrieval, which is possible if the nodes use iPXE, improves things significantly. However, there is still margin for improvement, as iPXE implements a rather limited TCP stack (e.g. it does not support scaling the TCP window size to large values).

The broadcast of the system image to be installed on nodes is a key step of the deployment process, and uses a large share of the deployment time. Kadeploy's chain-based broadcast provides optimal performance if nodes are properly ordered in the chain, but it is sensitive to node failures (which, during our tests, have been mainly caused by ARP or TCP failures due to the network load). In the future, we will explore ways to improve the robustness of network protocols such as ARP and TCP – the Linux kernel has a large number of knobs to tune the number of retries, or the timeout delays. We are also considering adding fault tolerance features to the chain-based broadcast tool to be able to exclude a failing node during the broadcast.

Both issues are rooted in the quality of the network used to deploy nodes. On clusters equipped with a high performance network (Infiniband, Myrinet), the administration network is usually composed of a set of cheap ethernet switches. We encountered switches with a MAC table size of 8000 entries, which puts a rather low limit on the number of virtual machines on the network. It is worth noting that large-scale Cloud providers are known to use routing between server racks in order to limit the level-2 network size. In Kadeploy, we are planning to explore how some of the steps of the deployment, e.g. the image broadcast, could use an Infiniband network.

6 Conclusion

In this paper, we presented experiments conducted on the Grid'5000 testbed to explore the scalability of the Kadeploy cluster deployment solution using up to 3999 KVM virtual machines hosted on up to 668 physical machines. During our largest run, we installed a 430 MB environment on 3838 virtual machines in less than 60 minutes.

First, those experiments proved that Kadeploy performs well at large scale, but also uncovered some areas of future improvements, such as the broadcasting of the environment, using iPXE for reboots (that would allow the download of the *deployment image* via HTTP), and reboots using *kexec*.

Second, this work demonstrates the capabilities of Grid'5000 as an experimental testbed. We used up to 668 physical machines, deployed our own environment on them and had administrative privileges on our nodes. Inside the set of nodes we reserved, we were able to deploy a complete infrastructure including intrusive network services such as DHCP in less than 20 minutes. Furthermore, everything performed

during this work was done as a normal Grid'5000 user with no special privileges, and all nodes used were automatically returned to normal Grid'5000 usage immediately after the end of the job.

The versatility exhibited by Grid'5000, with user-provided system images and network isolation on a large number of resources available for reservation, opens the path to other deployments of complex infrastructures. For example, other cluster deployment solutions could be deployed and compared with Kadeploy. And Grid'5000 could be used to test Grid and Cloud middleware solutions in a realistic setting during their development (our group already has experience deploying a gLite grid and an OpenStack Cloud inside Grid'5000).

Acknowledgements

This work was performed in the context of the INRIA ADT KADEPLOY project.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN-G5K development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr). We would like to thank the Grid'5000 technical team, and specifically Nicolas Niclausse and Tina Rakotoarivelo, for their help with those experiments.

Bibliography

[1] Yiannis Georgiou, Julien Leduc, Brice Videau, Johann Peyrard, and Olivier Richard. A tool for environment deployment in clusters and light grids. In *SMTPS'06*, 2006.

[2] Kadeploy3 website. `http://kadeploy3.gforge.inria.fr/`.

[3] Benoit Claudel, Guillaume Huard, and Olivier Richard. TakTuk, adaptive deployment of remote executions. In *HPDC'09*, 2009.