EXPERIMENTATIONS WITH CORDAGE, A GENERIC SERVICE FOR CO-DEPLOYING AND RE-DEPLOYING APPLICATIONS ON GRIDS*

Loïc Cudennec[†‡]
CEA LIST, Saclay, France
Loic.Cudennec@cea.fr

Gabriel Antoniu
INRIA, IRISA, Rennes, France
Gabriel.Antoniu@irisa.fr

Luc Bougé
ENS Cachan/Brittany, IRISA/INRIA, Rennes, France
Luc.Bouge@bretagne.ens-cachan.fr

Abstract Computer grids are made of thousands of heterogeneous physical resources that belong to different administration domains. This makes the use of the grid very complex. In this paper, we focus on deploying distributed applications at a large scale. As the application requirements may often not be anticipated, dynamic re-deployment is needed; if various applications have to co-operate within a workflow, they should also be co-deployed in a consistent way. In a previous paper, we have described the CORDAGE deployment model and its architecture. It meets the three properties of *transparency*, *versatility*, and *neutrality*. We report in this paper on its application to a real co-deployment over the GRID'5000 experimental platform, using different configurations, including multiple clients, multiple applications and multiple grid sites.

Keywords: Grid Computing, Re-deployment, Co-deployment, Autonomic Computing.

*Contact author: Gabriel Antoniu, IRISA, Campus de Beaulieu, F-35042 Rennes Cedex, France. Contact: Gabriel.Antoniu@irisa.fr.
†This work has been conducted at the University of Rennes 1, IRISA/INRIA, Rennes, France
‡CEA LIST, Embedded Real Time Systems Laboratory, Point Courrier 94, Gif-sur-Yvette, F-91191 France

1. Introduction

Federating physical resources located in different universities, institutes and companies leads to the concept of grid computing. These infrastructures are particularly outfitted to support the heavy data-computing and data-management demand coming from distributed applications, for instance heavy simulation scientific codes. Unfortunately, both applications and infrastructures are complex to deal with, especially in the phase of initial deployment. In this paper we focus on the deployment of such applications over large-scale environments.

Two aspects have to be considered. First, such environments are often dynamic, in the sense that resources may appear and disappear while the applications are running. Such a dynamicity can be supported by task migration and checkpointing mechanisms embedded in the applications. These mechanisms lead to a dynamic reconfiguration of the application, which includes some form of deployment. In this paper we do not address this aspect, which has been studied in many contributions.

In contrast, we address another aspect, which has often been overlooked by (pure...) scientists, but which **practical** impact is of major importance. Actually, the behavior of such applications is often unpredictable at launching time: the needs in physical resources can vary during the execution time. For instance, a distributed data-storage system cannot predict its future requirements of storage nodes, as they depend on clients demands. Another example is a scientific application which consists of a succession of computational steps. Each of these steps often requires a varying set of computing nodes. Thus, the system must dynamically expand over additional storage nodes during the run. However, for the sake of usage fairness, it should as well retract from them when they are not needed. We call *re-deployment* such a process of expanding and retracting at runtime.

Another issue regarding deployment comes from the dynamic co-ordination of several applications. As an example, consider a workflow of two applications A and B, where B consumes the flow of results produced by A. One has to make sure that B is deployed and is ready to consume data before A starts producing. Furthermore, one has to make sure that both applications are located on resources that are close enough to efficiently transfer data. This results in a set of temporal and spatial constraints. This close co-ordination has to be guaranteed even in the case of dynamic re-deployment. We call *co-deployment* such a process. However, this aspect is not addressed in the experiments reported in this paper.

The purpose of the CORDAGE service is precisely to provide support to automatically handle the co-deployment and the re-deployment of applications on grids. Such a system should meet three requirements:

Transparency: The applications, and even more the user, should remain unaware of the low-level interactions with services and tools that manage resources.

Versatility: In addition to standard generic deployment actions, such as configuring, launching and monitoring, the service should also provide support for ad-hoc, application-specific actions.

Neutrality: The service should handle application without any restriction regarding their software model (component-based approach, message-passing approach, etc.), and it should be minimally intrusive into their source code.

A large number of systems have been proposed to investigate various approaches towards the deployment of complex applications over large-scale infrastructures. One of the most promising approach relies on the concept of *Autonomic Computing* [18] proposed by IBM. Autonomic systems are able to self-manage, as exemplified by systems such as Dynaco [4], Entropy [16], and Jade [20]. These systems can dynamically reconfigure themselves, including the ability to redeploy some additional parts during the execution. Thus, Autonomic Computing offers transparency regarding resources management, as well as versatile support for application-specific operations. Unfortunately, most of the systems are very intrusive, as they require the use of particular software models such as component-based models.

Another approach consists in submitting applications to a remote execution environment, through a client interface. This approach is illustrated by systems such as XtreemOS [10], Vigne [17], DIET [8], and Zorilla [11]. They are in charge of selecting and allocating physical resources, then deploying the application in a transparent way. This approach is also very close to job schedulers such as OAR [5] and GridEngine [14], in which the user submits a set of jobs to be executed over the grid. Remote execution environments and job schedulers offer a transparent resource management, whereas being neutral and non-intrusive for the application. Unfortunately, it is very difficult, or even impossible, to take into account very specific needs in term of deployment, like the co-deployment of multiple applications. These systems are mainly designed to address load balancing and task migration issues.

Finally, one can use low-level toolboxes like Globus [13], GridLabGAT [1], and Saga [15], as well as deployment systems like ADAGE [19], GoDIET [7], APST [9], and DeployWare [12]. They are widely used on grid infrastructures, as they do not require the installation of heavy middleware, nor dedicated operating systems. They specifically assist the deployment of applications, offering basic services to discover and reserve physical resources, transfer files and remotely start processes. However, as a counterpart, these systems are far from making resource management and application deployment transparent for the user.

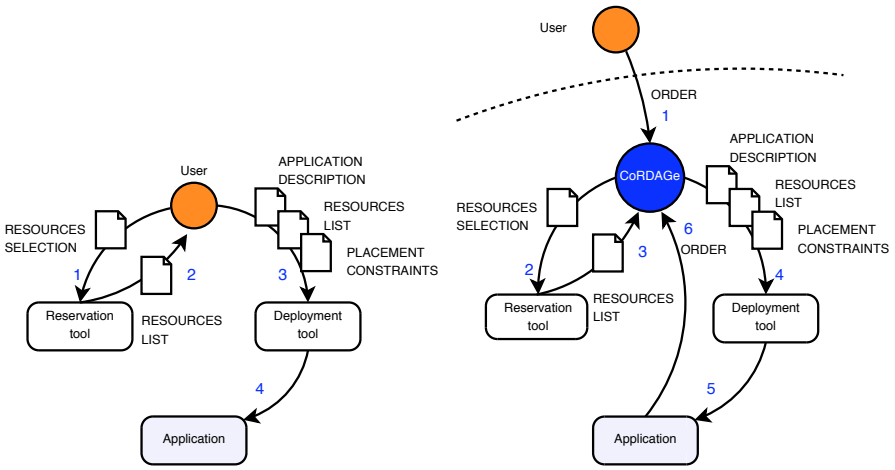

Figure 1. Deploying an application (a) by hand, and (b) using the CORDAGE tool.

The CORDAGE approach is to leverage the capabilities of a number of dynamic deployment tools, in order to meet the three properties of Transparency, Versatility and Neutrality. In our case, these are the OAR scheduler [5] and the ADAGE deployment tool [19], which enjoy Neutrality, but lack Versatility and Transparency.

The problem with low-level tools such as OAR and ADAGE is that the user is still responsible for scheduling the numerous and tedious elementary actions needed to perform and monitor a dynamic deployment, as shown on Figure 1(a). The user has still to 1) select and reserve some resources, then 2) retrieve the resource list, then 3) provide the deployment tool information like the description of the application, the resource list and some additional placement constraints, so that 4) the deployment tool can launch the application. In contrast, CORDAGE takes the user *out of the loop* and orchestrates these multiple actions on the behalf of the user according to a set of high-level orders, as displayed on Figure 1(b). As far as the user is concerned, everything is then transparent, versatile and neutral, as all the specific, ad-hoc details are handled within CORDAGE.

This paper is based on an improved version of the CORDAGE service already presented in a previous paper [2], as described in Section 2. Section 3 describes our case study, based on the JUXMEM data-sharing service [3], which is dynamically deployed over the GRID'5000 experimental grid testbed [6]. Then, Section 4 provides a detailed report on the experimental behavior of CORDAGE deploying and re-deploying JUXMEM under various near-real conditions. These experimentations show the impact of using multiple clients

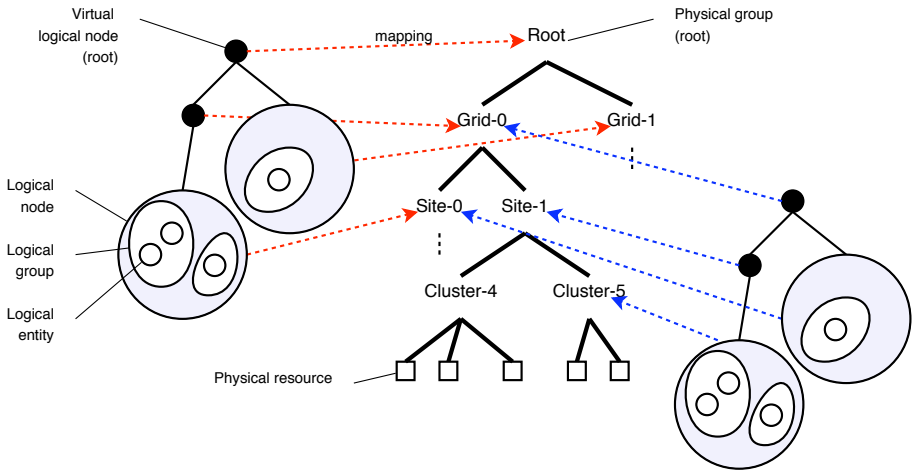

Figure 2. Two ways of mapping the same logical tree on the physical resource tree. Left is top-most, which ensures a greater distribution of entities over resources. Right is bottom-most, used to deploy entities on resources as close as possible.

and multiple applications, as well as the impact of spreading the service across multiple grid sites.

2. Towards dynamic deployment

CORDAGE introduces a model, in which applications and resources are represented by logical trees. This hierarchical approach allows to perform a consistent *pre-planning* of the deployment, by mapping the application components onto a set of physical resources with respect to the hierarchical constraints as shown on Figure 2.

In order to allow dynamic application *re-deployment*, the representation can then be updated during the execution. It makes it possible to dynamically expand or retract the configuration of the deployed application, respectively adding or deleting logical groups and nodes. Such a alteration is triggered by some *application-specific* requests sent by CORDAGE clients. These clients can be embedded in user applications, monitoring tools, or execution systems that are part of an autonomic framework.

The model can also handle the *co-deployment* of multiple applications. A logical tree is built for each application involved in the global deployment. These trees are then merged into a single tree, following specific rules. The resulting tree is thereafter mapped onto the physical tree and all applications are deployed in a co-ordinated manner.

```
#include cordage.hh
cdgclient* app1 = new cdgclient(conf_app1);
cdgclient* app2 = new cdgclient(conf_app2);

app1→add_sub_application(app2);
app1→deploy_application();
app2→perform_specific_action(ADD_PROVIDER, 2);

delete app1;
delete app2;
```

Figure 3. Minimal coding scheme to let an application interact with the CORDAGE server.

This model is *transparent*, as the mapping concept allows the user to express his high-level deployment requests without any reference to the low-level interactions with services and tools that manage resources.

As CORDAGE is seen as an external service by the application, it can be freely expanded with ad-hoc plugins to offer a set of additional specific actions as needed. This enables arbitrary *versatility* without any modification of the application.

The CORDAGE architecture is based on a client-server paradigm, which is as *neutral* as possible regarding the software model of the application. The only requirement is to insert a few lines of code at suitable places within the application, typically at places where the application makes decisions regarding expanding or retracting its configuration. To get a concrete feeling of using CORDAGE, the code lines displayed on Figure 3 are sufficient to let an application app1 request the CORDAGE server to add another application app2 as its sub-application, trigger the co-deployment of app1 and app2 in a co-ordinated way, and request the execution of an application-specific action ADD_PROVIDER to update the current configuration with 2 additional providers.

3. Deploying JUXMEM, a data-sharing service for the grid

The CORDAGE approach has been tested and evaluated in the context of the deployment of a distributed data-sharing service called JUXMEM [3] over the GRID'5000 experimental grid platform, using various configurations, including multiple clients, multiple applications and multiple grid sites.

The JUXMEM grid data-sharing service offers transparent and efficient access to data stored in the physical memory of grid processing nodes, later referred as *providers*. Providers self-organize as multiple *cluster groups*, which can be deployed on possibly different grid sites. The number of providers determines the overall storage capacity and the level of fault tolerance of the

service. JUXMEM is primarily intended to be used as a reliable storage for scientific grid applications.

Before CORDAGE, the deployment of JUXMEM was essentially based on the OAR reservation tool and the ADAGE deployment tool, as introduced in Section 1. In this former approach, the user has first to manually issue a request to the OAR scheduler to be granted a set of grid nodes. This request returns the user with a reservation identifier. Then, the user manually invokes the ADAGE deployment tool with this OAR reservation identifier. The user also has to manually provide ADAGE with a static description of the JUXMEM service, including the declaration of the various cluster groups, the configuration of the providers to be deployed (listening port numbers, memory space allocated to the service, maximal service time, etc.), and a set of spatial and temporal deployment constraints. All these information are provided by the user as XML files. Based on this information, ADAGE generates a deployment plan: it transfers the proper configuration files towards the storage nodes and there starts the processes. All this latter phase is done in a generic and transparent way.

This former approach has several drawbacks. First, the user has still to manually handle a large and complex set of information: the user has to interact with a number of sophisticated tools, consuming the data produced by one to feed another, translating the data format on the fly by hand. Moreover, if the initial reservation request to OAR fails, then the user is left alone to react and to provide OAR with another request. All these steps have to be made in the case of the deployment of any application over a grid.

Second, the case of JUXMEM is even more complex, as it is obviously very difficult to predict how large data will be stored into the service during its lifetime. It may well happen that the initial configuration falls short of storage space, and that deploying additional providers is needed at some point to serve the storage requests. Conversely, some storage providers may be left unused at some point. A fair share of the grid resources commends then to release this nodes to the other grid users. Therefore, the service configuration has to dynamically expand and retract all over its lifetime, based on the external requests. In this former approach, it is up to the user to take care about all the elementary actions needed for such a reconfiguration.

The goal of the CORDAGE approach is to make the deployment of JUXMEM transparent for the user. CORDAGE automates all these elementary actions, regarding both the initial deployment and successive expansions and retractions. For instance, CORDAGE is able to dynamically redeploy providers, taking into account specific requests from the JUXMEM service itself. A key aspect is to enforce that all the providers of each individual JUXMEM cluster group are deployed on the same grid site for performance

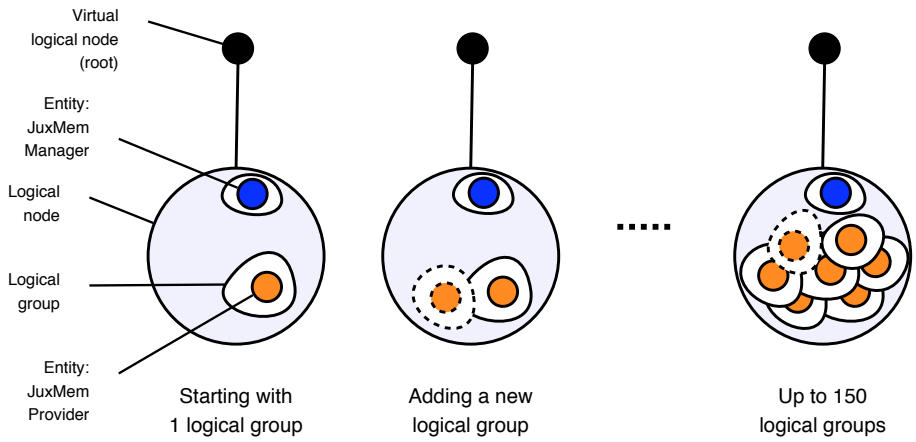

Figure 4. Expanding the JUXMEM topology by sequentially adding new providers into the same cluster group.

purpose. In contrast, the various cluster groups should be deployed over different grid site for fault tolerance purpose.

The CORDAGE hierarchical model described in Section 2 has precisely been introduced for this purpose. The corresponding logical tree contains one virtual root node, each child of which corresponds to a *cluster group* to be deployed on a particular grid site. As shown on Figure 4, the entities added into the same second-level logical node will be deployed within the same grid site. In contrast, as shown on Figure 7, the entities belonging to different logical nodes will be deployed over different grid sites.

Specific actions have been added to the CORDAGE server to enable the JUXMEM service to reconfigure itself: expanding by creating new storage providers into existing cluster groups, or even creating new cluster groups; retracting by deleting providers or groups. This is done by creating or deleting logical nodes, groups and entities within the logical tree.

The goal of the experiments below is to present and analyze the behavior of CORDAGE when used for managing the dynamic deployment of the JUXMEM service.

4. Experimenting with CORDAGE

Various configurations of the CORDAGE service are studied. In each of them, a single CORDAGE server is set up to serve the requests of possibly multiple clients regarding possibly multiple JUXMEM applications.

The experimentations are conducted within the GRID'5000 platform, an experimental and distributed testbed that gathers up to 5000 nodes distributed

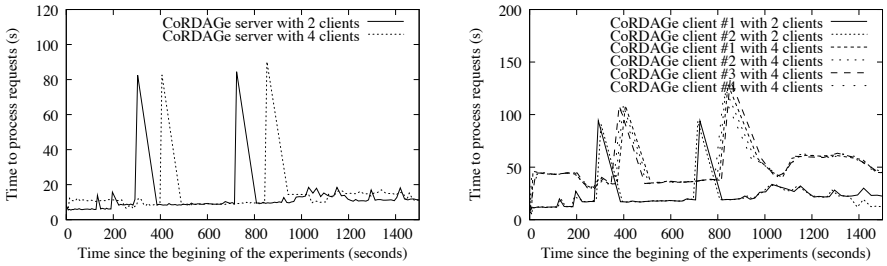

Figure 5. Multiple clients: time to process requests (a) on the server and (b) on the clients.

in 9 sites France-wide. The protocol starts with the deployment of one CORDAGE server, followed by the deployment of a minimal JUXMEM topology. This topology consists of one node, in charge of organizing the set of JUXMEM providers. Then, synthetic clients perform re-deployment requests to the CORDAGE server, as the JUXMEM service would do in a real behavior.

4.1 Using multiple clients

In a first scenario, we consider that the CORDAGE server is in charge of managing one single JUXMEM application. Several clients perform concurrent requests to add new providers to the service. In order to keep the application representation consistent, the CORDAGE server does not allow parallel operations on the same application. Therefore, the requests of the various clients are separately processed in a FIFO-ordered queue. Figure 4 displays the evolution of the logical representation, as the requests for adding a new provider are processed. Each request creates a new logical group in the logical node corresponding to the single JUXMEM cluster group. This new group contains the single new provider entity to be deployed.

We run two experiments: the first one uses two concurrent clients and the second one uses four concurrent clients. The clients repeatedly add new providers into the unique instance of JUXMEM, without any intermediate synchronization. The clients requests are accumulated into a FIFO queue at the CORDAGE server level.

Figure 5(a) displays the time to process each successive request on the CORDAGE server, whichever the sender is, during the whole experiment time. The experiment with 2 clients leads to processing 150 requests in 28 minutes. The average time to process a request is 11 seconds. The experiment with 4 clients leads to processing 143 requests in 29 minutes. The average time to process a request is 12 seconds. Processing a request on the server breaks down into 3 phases:

Update the logical representation. This includes all synchronization mechanisms, the mapping of the logical tree onto the physical tree as well as some ancillary operations. In both configurations, this step takes less than 0.20 seconds per request, that is, less than 2 % of the overall processing time.

Interact with the OAR reservation tool. This step takes an average time of around 7 seconds per request in the 2-client configuration and 9 seconds in the 4-client one. This step takes nearly 70 % of the overall processing time.

Interact with the ADAGE deployment tool. This updates the configuration of the JUXMEM service by transferring the configuration files and starting the processes. This step takes an average time of 3 seconds per request in both configurations. This step takes around 30 % of the overall processing time.

One major observation coming from Figure 5(a) is that the processing time for each request on the server is not significantly impacted by the number of clients acting on the same application. This can be explained by the serialization of the requests by the CORDAGE server: each request is fully processed by the server without any time-sharing mechanism.

In contrast, the overhead coming from the number of clients can be witnessed on Figure 5(b). This figure displays times measured in the same experiment as above. Yet, the request time is measured on the client side instead of the CORDAGE server side, including the time spent by the requests in the server queue. The more clients are acting on the same application, the longer they have to wait for their requests to be processed. The clients wait an average time of 8 seconds in a single-client configuration, 21 seconds in a 2-client configuration and 46 seconds in a 4-client configuration.

The reader will have noticed the presence of two peak points on both figures, in both configurations. These points correspond to the resource saturation of the grid site currently used to deploy new providers: no more nodes are available for reservation by the OAR tool. CORDAGE incurs an almost 80-second overhead. The server checks OAR about the availability of the requested nodes every second. After 45 unsuccessful checks, the server decides the current grid site cannot be used any longer. It selects another one from a list of available sites, and then requests OAR for new resources in this latter site. These peaks occur more or less at the same moment in both configurations because the experiments were run on the same day, without any major changes in the overall grid reservation status: the sites reached saturation after adding the same number of providers in both experiments. Yet, there is a noticeable shift between the peaks in the 2-client configuration and the 4-client configuration. This is due to the fact that the average time for processing requests in the 4-client configuration is slightly larger than that in the 2-client configuration at

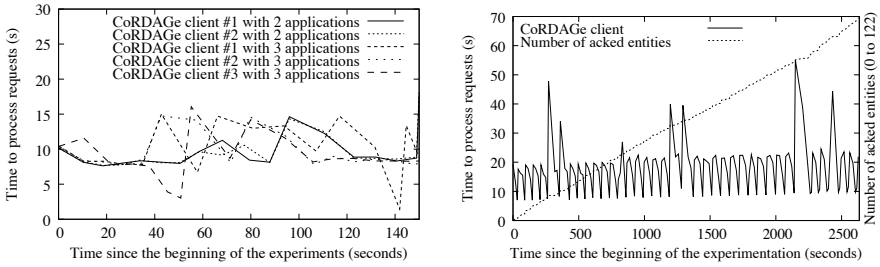

Figure 6. On the left, multiple sites: time to process requests on the clients and number of ready. On the right, multiple applications: time to process requests on the clients.

the beginning of the experiments. Therefore, all subsequent request occurrence times are shifted away by some fixed amount.

4.2 Impact of using multiple applications

The CoRDAGe server is also able to manage several concurrent applications at the same time. This makes for example sense in the context of co-deployment. In this section, we study the impact of managing several applications with only one server on the request processing time. We use the previous configuration involving a minimal topology of JuxMem that is expanded by sequentially adding new providers. We run two successive experiments, one with 2 independent instances of the JuxMem service, and then 3 instances of it, all the instances being managed by the same CoRDAGe server. For each instance of the JuxMem service, one specific synthetic client repeatedly issues requests to the CoRDAGe server to add new providers. At the CoRDAGe level, all the JuxMem instances share the same instance of the OAR reservation tool, whereas a dedicated instance of the ADAGE deployment tool is used for each of them. This arrangement actually suits the Grid'5000 environment, as most of the other grid platforms: all users must interact with a unique, centralized job scheduler, but they are free to use their own deployment tool.

Figure 6(a) displays the time to process requests measured on the clients, using the 2- and the 3-application configurations. The average time is around 11 seconds. This time is stable and comparable to results with one client and one application: managing additional applications does not increase the client waiting time for a request to be processed. Actually, the client requests do not have to wait for any lock before being processed by the CoRDAGe server. Then, the concurrent *client requests* at the level of the CoRDAGe server result in concurrent *reservation requests* at the level of the OAR scheduler, which are effectively handled in parallel, with a stable request time. Also, they result in

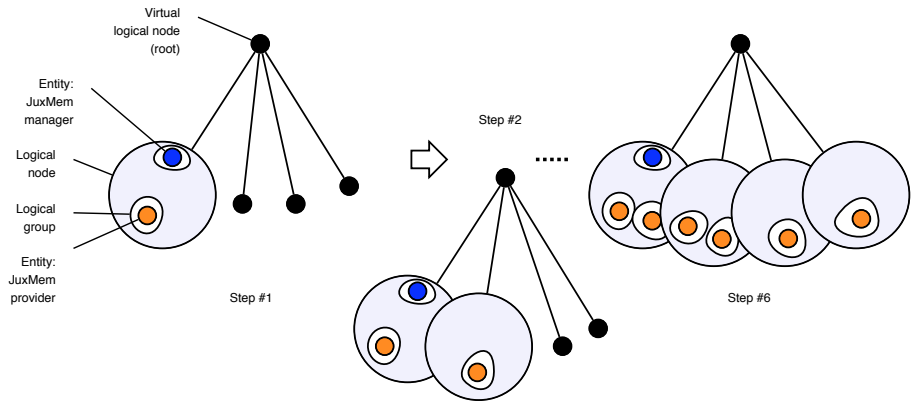

Figure 7. Expanding the JUXMEM topology by sequentially adding providers into four cluster groups located on different grid sites.

concurrent *deployment requests* at the level of the ADAGE deployment tools: all these requests are effectively handled in parallel as one instance of ADAGE is used per application instance.

4.3 Using multiple grid sites

The previous experiments were conducted within one grid site at a time. In this section we evaluate the CORDAGE prototype using 4 different sites at the same time. We start with a minimal JUXMEM topology made of one single JUXMEM cluster group, which is then expanded with 3 additional sibling cluster groups. Then, a synthetic client repeatedly adds new providers, one in each cluster group in turn. According to this scenario, the logical representation of the application evolves as displayed on Figure 7.

Figure 6(b) displays the time needed to process the successive provider launching requests. Observe that this request processing time, as measured on the client side, only includes launching the system process which runs the provider applicative code. It *does not include* the time needed to fully set up the provider and let it serve the application requests within the JUXMEM service. Once a provider has joined the JUXMEM service, we let it send an acknowledgment ready message to the CORDAGE server, so that Figure 6(b) can display the number of ready providers.

In this experiment, we use 4 grid sites. On each site, an instance of the OAR reservation tool is made available to locally manage the resources. These instances are running on various types of machines, and they manage various set of resources, so that the interaction times vary from site to site. This explains the periodic patterns displayed by Figure 6(b): the smaller values correspond

to requests made in a particularly reactive site. Again, peaks correspond to sites reaching saturation, as CORDAGE selects a fresh grid site if no reservation has been granted after a number of unsuccessful requests to OAR. In this experiments, the timeout was set to around 45 seconds, so as to include several site shifting within the overall time.

5. Conclusion

Today, the management of complex distributed applications over large-scale infrastructures requires the use of dedicated external systems. These systems help in facing, in a transparent way, the dynamicity introduced by both applications and infrastructures. In a previous paper [2] we have introduced CORDAGE, a co-deployment and re-deployment tool that satisfies the three properties of transparency, versatility and non-intrusiveness.

In this paper we have described the evaluation of a prototype based on this model, used to deploy the JUXMEM data-sharing service over the GRID'5000 platform. The CORDAGE server relies on a number of grid tools: a reservation tool (OAR) and a deployment tool (ADAGE). CORDAGE takes the user *out of the loop* and orchestrates these multiple interactions on behalf of the user according to a set of high-level orders. As far as the user is concerned, everything is then transparent, versatile and neutral, as all the specific, ad-hoc details are handled within CORDAGE.

We have shown that the CORDAGE server can effectively interact with these tools, with a minimal overhead (less than 2 %). It can handle several applications at the same time, either dependent as in a co-deployment, or independent, taking advantage of the parallelization of the requests.

An interesting perspective of this work is the ability to let multiple CORDAGE servers communicate. They could for instance exchange information about resources and applications in order to optimize deployments, and to provide load balancing and fault tolerance features. More generally, CORDAGE could be integrated as one of the executive tools of a generic autonomic framework like DYNACO [4].

Acknowledgments

This work has been supported by Sun Microsystems, the Regional Council of Brittany, and the LEGO (ANR-05-CIGC-11) and RESPIRE (ARA MDMSACS) Projects of the French National Agency for Research (ANR).

The experiments reported in this paper were carried out using the GRID'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see http://www.grid5000.fr/).

References

[1] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. van Nieuwpoort, A. Reinefeld, F. Schintke, T. Schott, E. Seidel, and B. Ullmer. The Grid Application Toolkit: toward generic and easy application programming interfaces for the Grids. *Proceedings of the IEEE*, 93(3):534–550, March 2005.

[2] Gabriel Antoniu, Luc Bougé, and Loïc Cudennec. CoRDAGe: towards transparent management of interactions between applications and ressources. In *International Workshop on Scalable Tools for High-End Computing (STHEC 2008), held in conjunction with the International Conference on Supercomputing (ICS 2008)*, pages 13–24, Kos, Greece, June 2008.

[3] Gabriel Antoniu, Luc Bougé, and Mathieu Jan. JuxMem: an adaptive supportive platform for data-sharing on the grid. *Scalable Computing: Practice and Experience (SCPE)*, 6(3):45–55, November 2005.

[4] Jérémy Buisson, Françoise André, and Jean-Louis Pazat. Dynamic adaptation for Grid computing. In *Advances in Grid Computing - European Grid Conference (EGC 2005)*, volume 3470 of *Lecture Notes in Computer Science*, pages 538–547, Amsterdam, The Netherlands, June 2005. Springer.

[5] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cyrille Martin, Grégory Mounié, Pierre Neyron, and Olivier Richard. A batch scheduler with high-level components. In *Proc. 5th IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGrid '05)*, pages 776–783, Cardiff, UK, May 2005.

[6] Franck Cappello, Eddy Caron, Michel Dayde, Frédéric Desprez, Emmanuel Jeannot, Yvon Jegou, Stephane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, and Olivier Richard. Grid'5000: a large scale, reconfigurable, controllable and monitorable grid platform. In *Proc. 6th IEEE/ACM Intl. Workshop on Grid Computing (Grid '05)*, pages 99–106, Seattle, WA, USA, November 2005.

[7] Eddy Caron, Pushpinder Kaur Chouhan, and Holly Dail. GoDIET: a deployment tool for distributed middleware on Grid'5000. In *Experimental Grid Testbeds for the Assessment of Large-Scale Distributed Applications and Tools (EXPGRID workshop). Held in conjunction with HPDC 15.*, pages 1–8, Paris, France, June 2006.

[8] Eddy Caron and Frédéric Desprez. DIET: a scalable toolbox to build network enabled servers on the Grid. *Intl. J. High-Performance Computing Applications*, 20(3):335–352, 2006.

[9] Henri Casanova, Graziano Obertelli, Francine Berman, and Richard Wolski. The AppLeS parameter sweep template: user-level middleware for the Grids. *Sci. Program.*, 8(3):111–126, 2000.

[10] Toni Cortes, Carsten Franke, Yvon Jégou, Thilo Kielmann, Domenico Laforenza, Brian Matthews, Christine Morin, Luis Pablo Prieto, and Alexander Reinefeld. XtreemOS: a vision for a grid operating system. White Paper Available at http://www.xtreemos.eu/publications/research-papers/xtreemos-cacm.pdf, May 2008.

[11] Niels Drost, Rob van Nieuwpoort, and Henri E. Bal. Simple locality-aware co-allocation in peer-to-peer supercomputing. In *Proceedings of the 6th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '06)*, May 2006.

[12] Areski Flissi, Jérémy Dubus, Nicolas Dolet, and Philippe Merle. Deploying on the Grid with DeployWare. In *Proceedings of the 8th International Symposium on Cluster Computing and the Grid (CCGrid '08)*, pages 177–184, Lyon, France, may 2008. IEEE.

[13] I. Foster and C. Kesselman. Globus: a metacomputing infrastructure toolkit. *Intl. J. of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.

[14] W. Gentzsch. Sun Grid Engine: towards creating a compute power Grid. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGrid '01)*, page 35, Brisbane, Australia, May 2001. IEEE Computer Society. Sun Microsystems.

[15] Tom Goodale, Shantenu Jha, Hartmut Kaiser, Thilo Kielmann, Pascal Kleijer, Gregor von Laszewski, Craig Lee, Andre Merzky, Hrabri Rajic, and John Shalf. SAGA: a simple API for Grid applications - high-level application programming on the Grid. *Computational Methods in Science and Technology*, SC05:8(2), November 2005. special issue "Grid Applications: New Challenges for Computational Methods".

[16] Fabien Hermenier, Nicolas Loriant, and Jean-Marc Menaud. Power management in Grid computing with Xen. In *Frontiers of High Performance Computing and Networking - ISPA 2006 International Workshops*, volume 4331 of *Lecture Notes in Computer Science*, pages 407–416, Sorrento, Italy, December 2006. Springer.

[17] Emmanuel Jeanvoine, Christine Morin, and Daniel Leprince. Vigne: executing easily and efficiently a wide range of distributed applications in Grids. In *Proceedings of the International Euro-Par Conference on Parallel Processing (Euro-Par 2007)*, volume 4641 of *Lecture Notes in Computer Science*, pages 394–403, Rennes, France, August 2007. Springer.

[18] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[19] Sébastien Lacour, Christian Pérez, and Thierry Priol. Generic application description model: toward automatic deployment of applications on computational Grids. In *6th IEEE/ACM International Workshop on Grid Computing (Grid 2005)*, Seattle, WA, USA, November 2005. Springer.

[20] Noel De Palma, Sylvain Sicard, Sara Bouchenak, Daniel Hagimont, and Fabienne Boyer. Autonomic administration of clustered J2EE applications. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2005)*, volume 3, pages 1248–1254, Las Vegas, NV, USA, June 2005. CSREA Press.