CoRDAGe: towards transparent management of interactions between applications and ressources

Loïc Cudennec, Gabriel Antoniu, Luc Bougé

CORDAGE: Towards Transparent Management of Interactions Between Applications and Resources*

Loïc Cudennec
INRIA/University of Rennes 1, IRISA, France

Gabriel Antoniu
INRIA, IRISA, France

Luc Bougé
ENS Cachan/Brittany extension, IRISA, France

Abstract

Nowadays large-scale, grid-aware applications are intended to run for days or even weeks over hundreds or thousands of nodes. This requires new, and often painful operations for the user in charge of deployment and monitoring. We claim that the applications should themselves manage their run in an autonomic way, by requesting new resources on-demand.

In this paper, we introduce CORDAGE, a third-party tool, standing between applications and lower-level grid management tools. It provides generic and application-specific facilities to dynamically expand and retract the deployment of a grid-aware application according to its actual needs. A prototype has been implemented and a preliminary testing has been conducted on the GRID'5000 testbed.

1 Introduction

This work has been initiated within the context of large-scale, grid-aware applications, mostly scientific code-coupling applications, distributed execution frameworks and data-sharing services. These applications are designed to run on hundreds of nodes, for duration of the order of days or even weeks. In most cases, the needs in physical resources is not predictable before deploying. As most, if not all of the available grid testbeds are platforms shared among different users, it is obviously not possible to exclusively reserve all the physical resources for such a long time. Therefore, these applications have to be designed together with ad-hoc facilities to *expand* or *retract* their topology at run-time, that is, requiring or releasing computing nodes according to their actual needs, and/or to the requests of the other users.

This picture is reminiscent of so-called *Desktop computing* or *Internet Computing* for very large-scale, distributed applications, mostly based on a Master-Worker scheme. However, we address a more complex problem in this paper. The grid-aware applications should be able to continuously

*Contact author: Loïc Cudennec, IRISA, Campus de Beaulieu, F-35042 Rennes Cedex, France.Loic.Cudennec@irisa.fr.

This work has been supported by a grant of Sun Microsystems and a grant from the Regional Council of Brittany, France.

and *autonomically* interact with the grid management agents to handle *re-deployment* requests. Also, complex deployment requests have to be handled. For instance, in the case of coupled codes, multiple applications have to be deployed in a consistent way, and some additional connection actions (connecting pipes, installing consistent configuration files, etc.) have to be performed at *co-deployment* time.

Autonomic computing has become one of the most promising approach to ease the management of large-scale distributed systems. The idea of autonomic computing, first introduced in the IBM manifesto [1], consists in building systems which are self-managing to meet the administrator's goals. Such a feature is called *self-configurability*: installing, configuring and integrating large applications should be transparently achieved without the help of a human user. Some projects [2, 3, 4] offer dynamic deployment to execute tasks on a distributed execution overlay. In these projects, the resources are discovered and selected according to the needs of the application. However, this execution overlay has to be persistently deployed among the resources, which is not always possible on time-shared platforms. Dynamic reconfiguration of applications over computing grids has also been studied in Jade [5]. This system proposes an autonomous framework for the administration of clustered J2EE applications thanks to the use of a component model. Entities have to be encapsulated into Fractal components in order to be managed in a dynamic way. The Tune system [6] helps adapting applications to use the Jade framework, introducing a higher-level UML-like interface. Finally, systems like Dynaco [7] and Entropy [8] propose an adaptation framework based on a supervisor which monitors resources and applications, and takes decisions following a given policy that can trigger additional deployment or process migration.

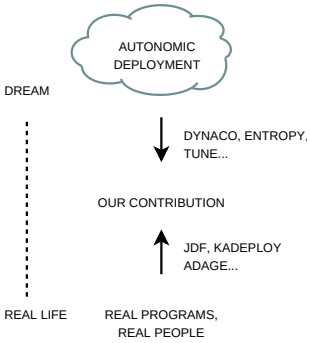

Figure 1: Standing between real-life deployment and autonomic deployment.

Some tools have been proposed to specifically assist the deployment of applications. The JXTA *Distributed Framework* (JDF) [9] helps in automating the deployment of large JXTA-based P2P networks using a symbolic description of the overlay. TakTuk [10] is designed to execute a given command on a bunch of nodes, retrieve the output data and ease the communications between processes using a logical interconnection network. Kadeploy [11] makes a step further in deployment support by replacing the node runtime environment with a cloned image that includes the entire operating system. Finally, ADAGE [12] goes generic, by proposing a deployment framework that converts specific descriptions of applications into an internal representation which serves as a basis to plan deployment. Explicit placement constraints regarding the mapping of processes can also be expressed. As far as we know, ADAGE is the only adaptable deployment framework to address a large variety of complex applications.

However, all these deployment systems are designed for one-shot deployments: once the application has been started, there is no support for additional deployment nor smart process removal. The goal of this paper is to introduce a preliminary attempt at addressing the problem of *joint co-deployment* and *dynamic re-deployment* of complex distributed applications on a grid. Section 2 describes a motivating scenario. Section 3 explains the various aspects of our CORDAGE tool, starting from the high-level description down to the low-level operations on the grid. Section 4 provides some details about our prototype and its performance.

2 A motivating example: co-deploying JUXMEM and GFARM

As a motivating case-study, we describe the deployment of JUXMEM [13]. We provide here a very brief description of this software, emphasizing its deployment requirements.

JUXMEM is a data-sharing service for the grid that enables transparent data sharing through the use of a unique global identifier. JUXMEM is inspired by both distributed shared-memory systems (DSM) regarding the transparent access to memory, and by peer-to-peer systems (P2P) regarding the support of dynamic reconfiguration. The data stored in the JUXMEM service are replicated on so-called *provider* peers, distributed within several JUXMEM groups. Within each group, a *manager* peer is in charge of locally connecting the peers and processing allocating requests by finding enough providers to replicate the data. Finally, some *client* peers are in charge of the interactions between the main user application and the data-sharing service.

The storage capacity provided by JUXMEM mainly depends on the number of providers involved in the deployment. As of today, this information has to be determined before the deployment of the service. To do so, the maximum peak of storage load has to be estimated from predictions about the client application behavior. Once a JUXMEM topology has been determined to satisfy the expected needs of the client (number of managers, providers, clients, etc.), the user has to reserve physical resources on the grid and to deploy all the JUXMEM entities. This is done using external tools provided elsewhere by the grid environment: reservation and scheduling tools, deployment tools, monitoring tools, etc. Observe that no later tuning of resource usage is possible: even if it happens that only a few providers are needed at run-time, the reserved resources are frozen until the end of the reservation period.

A much more attractive scenario would of course consist in initially deploying a minimal topology of JUXMEM only. This topology would be composed by only one group, in which only one manager peer is deployed and no provider. This initial topology would then self-expand and self-retract, in a transparent and autonomic way, depending on the actual needs of the client.

A first approach would be to patch ad-hoc pieces of code into JUXMEM to interact with the scheduling and deployment external tools, so as to reserve physical nodes and to deploy additional JUXMEM entities as needed. However this would not be a generic approach, whereas many other applications [2, 14, 4] might be interested by this kind of dynamic feature.

Furthermore, deploying several applications of different types at the same time is not a rare thing. Most of the scientific applications and prototypes are in fact made by coupling multiple sub-applications coming from different teams. As an example, we presented in a previous paper [15] how to build a distributed hierarchical memory for the grid, relying on the JUXMEM service for fast memory accesses, and the GFARM [14] global file system for long-term storage. The idea is to leverage the storage space by adding a secondary persistent storage. This joint architecture raises challenging deployment issues. Each JUXMEM provider also acts as a GFARM client, so that the deployment plans of JUXMEM and GFARM have to be set up together. Moreover, the deployment phase of GFARM has to create specific configuration files at the nodes where GFARM clients, that is, JUXMEM providers, are expected. In a previous work, it has been shown how to handle such elaborate constraints within the ADAGE deployment tool [12] in a static deployment context. The challenge here is to extend this capacity to a dynamic and unpredictable context as well.

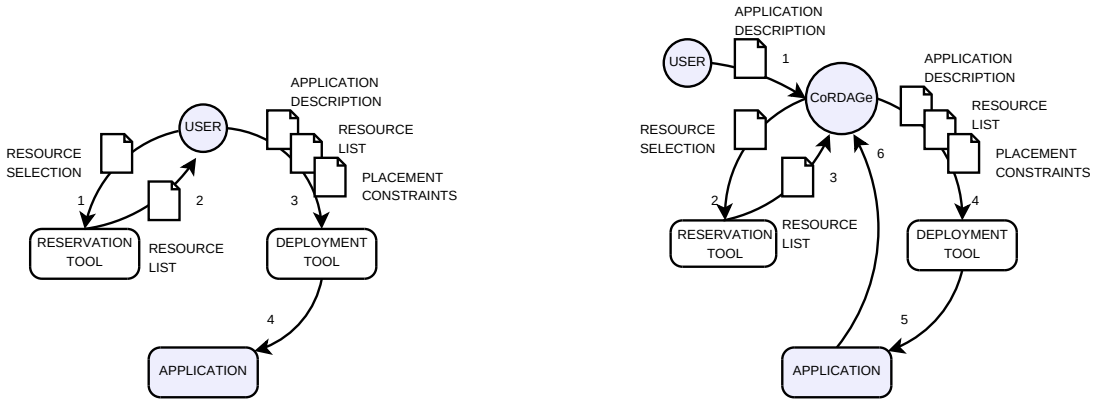

Figure 2: Deploying an application (a) by hand, and (b) using the CORDAGE tool.

These sophisticated applications are designed to be executed on a grid architecture, that is, a set of distributed resources interconnected by a network and belonging to different administration domains. This is particularly true for the GRID'5000 [16] platform, a French experimental testbed that gathers up to 5,000 processors distributed over 9 sites nation-wide. The resource nodes from this highly reconfigurable testbed can be reserved thanks to a distributed batch scheduler named OAR [17]. This scheduler allows grid users to share resources by requesting reservations starting as soon as possible or in the near future. Information about the wall-time of the reservation and the requested properties of the reserved resources are specified through to a basic command-line interface. Each grid site hosts a database in charge of managing the local reservation requests. There is no advanced support for multi-site reservations: they have to be handled using shell scripts. Additional supervision tools generate live Gantt diagrams for reservations in order to help the user select a list of sites with enough free resources.

As of today, experimentations on GRID'5000 involving complex applications are deployed *by hand*. Figure 2(a) shows the complex pattern of interactions between the user and the grid management tools to achieve a very simple deployment.

We claim that a third-party tool is needed to *autonomically* manage all these intricate reservation and deployment operations. We introduce our CORDAGE proposal: a *co-deployment* and *redeployment* tool based on ADAGE. (*Cordage* also means *rigging* in French.) CORDAGE should be generic enough to handle at the same time the management of several applications of different types. Figure 2(b) displays the interactions standing between the CORDAGE tool, the user, and original grid management tools. The main difference with Figure 2(a) is that the user is not in charge of requesting resources, neither deploying them. All these steps are now performed by this tool in a transparent way on behalf of the user. The only information needed from the user is the initial application description. This information could even be left empty in order to deploy a minimal default configuration. Once deployed, the application can freely interact with CORDAGE in order to request expansion or retraction, without any user intervention.

Action	Meaning
BASE_REGISTER	Register a new application with a given type and a given id
BASE_SET_APPDESCPATH	Set the application description file path
BASE_DEPLOY	Reserve resources and deploy the application using the current application description
BASE_DISCARD	Discard a given entity, request entity termination, delete associated reservation
BASE_TERMINATE	Terminate deployment tool and unregister application
JUXMEM_ADD_GROUP	Add a JUXMEM group (1 manager and a given number of providers)
JUXMEM_ADD_PROVIDER	Add a given number of providers in a given JUXMEM group

Table 1: Some CORDAGE generic and specific actions.

3 Contribution

In this paper we propose a grid tool, called CORDAGE, that transparently manages all the interactions between user applications and other, lower-level grid tools. CORDAGE is adapted to any kind of distributed applications that can interact with CORDAGE through a remote procedure call (RPC) interface. RPC to CORDAGE trigger the execution of generic actions offered by the CORDAGE kernel base class (CorKBase). These actions are available for all registered applications. Registering a new application creates an instance of the CorKBase class that features a logical representation of the application, a representation of the physical resources and other state variables that can be accessed by all actions. In order to adapt CORDAGE to a new type of applications, a set of specific actions have to be defined. These actions are part of a new class, specific to the application, that extends the CorKBase class. Therefore, specific actions can also access the logical representation and other state variables. Some of the actions are mandatory, like building the CORDAGE logical representation from the application description file. This action is called each time a new description file is sent. Other actions can be very specific, like deploying a provider in a given JUXMEM group. Some examples are listed in Table 1.

3.1 Step 1: Describing and configuring applications

In the CORDAGE model, an application is defined by a set of *types of entities* (ToE). The type of an entity is defined as the program to be executed on a physical resource, without taking the initial parameters into account. As an example, the JUXMEM data-sharing service is represented by a set of three ToE that correspond to the *manager*, the *provider* and the *client*.

Configuring applications consists in instantiating types of entities that compose an application into *entities*. An entity is a unit-element managed by CORDAGE that needs to be deployed on a single physical resource. During this step, two main information are specified prior to deployment. 1) Dimensioning the application is performed by attaching a cardinality to each type of entities. This cardinality can be equal to zero for some entities. If we consider the JUXMEM scenario mentioned above, we can configure the application following Listing 1: one single manager m, two providers p_1 and p_2, and one single client c. 2) Semantic information is then attached to each entity. This information can be initial parameters to use at startup or a description on how entities have to

Listing 1: Application description sample (JDL): overlay definition of a JUXMEM service.

```
<overlay>
 <superpeers>
  <superpeer id="my_manager" cardinality="1"/>
 </superpeers>
 <peers>
  <peer id="my_provider" cardinality="2" superpeer="my_manager"/>
  <peer id="my_writer" cardinality="1" superpeer="my_manager"/>
 </peers>
</overlay>
```

interact. Regarding the JUXMEM scenario, we can specify that p_i and c have to be connected to manager m. The set of entities generated at this step composes the *configured application*. This step is still at the charge of the final user who wants to describe the initial configuration of his application. However, we can imagine that a minimal description is provided for each kind of application, to be later modified by CORDAGE.

3.2 Step 2: Building the logical representations, groups and trees

This step helps CORDAGE manage the entities, by building a logical representation of the configured application. This representation is generic against the type of application. We propose a first representation based on *logical groups* that we generalize into a *logical tree* representation.

This representation decomposes the set of entities defining a configured application into a set of logical groups (see Figure 3(a)). Logical groups will be later mapped to a set of physical resources. Each logical group gathers entities that belong to the same *class*. These classes are defined thanks to an *affinity relation* α which is specific to the application. Its evaluation depends on the type of entities and their attached semantic information. As an example, if we consider an application based on the JUXMEM service, a smart α relation should express that all entities belonging to the same JUXMEM group have to be placed in the same logical group. This strategy will reveal helpful to take advantage of resource proximity, mainly in terms of network latency.

A specific action, *generate_logical_representation*, is called each time the application description has been modified in order to build the logical groups. This action has to be written for each supported application: it is in fact the implementation of the α relation. Listing 2 displays the algorithm used to build logical groups from a JUXMEM application description (as shown in Listing 1).

Listing 2: Building logical groups.

```
foreach (superpeer manager) {
  gid = new logical_group(); // we create a new logical group
  add_entity(gid, manager);  // we add the manager to this group
  foreach (peer p connected to manager) {
    add_entity(gid, p);      // we add p to the manager group
  }
 }
```

Logical groups may fit the needs of applications built around a simple organization model. How-

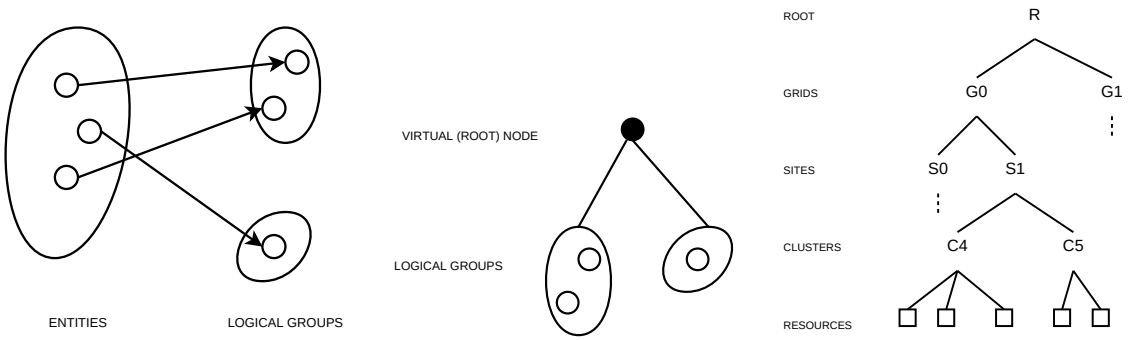

Figure 3: (a) Representing the application using logical groups. (b) Representing the application using a logical tree. (c) Representing physical resources using a physical tree.

ever, for more complex applications, who need to express hierarchical relations between entities, a flat representation may not suffice. This hierarchy may for instance express some proximity relation (e.g. in terms of physical or semantic distance among the entities). To generalize our solution to a logical tree structure, we introduce the notion of *virtual node* which will stand at the root level of the tree. All logical groups are then attached as children of this virtual node (see Figure 3(b)). Starting from this idea, we define the logical tree representation as a tree whose nodes and leaves are either a logical group, or a virtual node.

3.3 Representing physical resources

The next step of the CORDAGE model uses this logical representation of the configured application in order to make a pre-selection of the physical resources that will be involved in the deployment. As for the application, CORDAGE proposes a generic representation of the physical resources. A *physical resource* is defined in the CORDAGE model as a physical device that refers to a computing node, with the ability to support the execution of at least one logical entity. The goal of this representation is to help CORDAGE perform a sub-selection of resources likely to host a logical group. Physical resources are organized using *physical groups* in a hierarchical way, which define a *physical tree*. Each physical resource belongs to at least one physical group.

A physical tree is built thanks to a leaf-set of physical resources and to a node-set of physical groups. Figure 3(c) shows some resources, represented by squares, that are organized within a physical tree. Because most of the client applications are designed to be deployed in a multi-grid context, and also for the need of simplicity, we organize these resources with the criteria to make the physical tree mirror the hierarchical grid infrastructure. This representation is mainly based on network properties in terms of latency.

3.4 Step 3: Mapping the logical tree onto the physical tree

Once both logical and physical representations have been built, the next step of the CORDAGE model consists in mapping the logical tree to a sub-tree of the physical tree. Each logical group has to be mapped to a physical group, with the respect of the tree hierarchy. We assume the

physical tree to be deeper and larger than the logical tree so that it is possible to find different physical groups for each logical groups. This step generates preliminary placement constraints that associate a set of logical entities to a set of physical resources. At the end of this step all resource reservations have been made. Then, the final mapping of entities to resources will be done by the deployment tool in a further step.

Figure 4 displays two different mappings of the same logical tree over one single physical tree. The first mapping, on the left, is a *top-most* mapping in which we try to select a physical sub-tree as close to the root as we can. This approach is motivated by the need to map entities from different logical groups on distant physical resources. On the other side, we represent a *bottom-most* mapping that selects a sub-tree that is close to the leaves. This approach is motivated by the need to map entities of the same logical group to resources that are as close as possible. In this example, the top-most mapping distributes the two logical groups within two different grids ($G0$ and $G1$) whereas the bottom-most mapping only selects resources from grid $G0$.

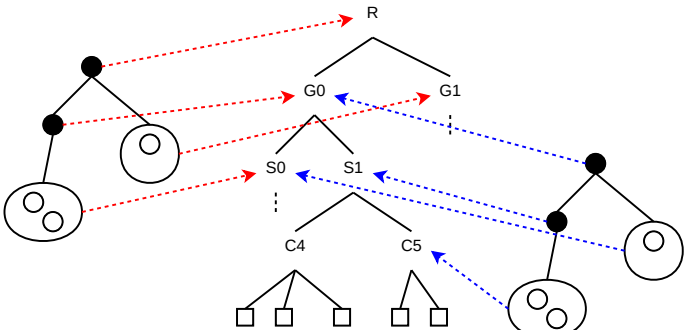

Figure 4: Two ways of mapping the same logical tree on the physical resource tree. Left is top-most, right is bottom-most.

At this step, CORDAGE iteratively searches for valid mappings of the logical tree to a sub-tree of the physical tree: for each possible solution, it requests the reservation of the corresponding physical resources. This iterative process stops if all reservations succeed. However, since the testbed is time-shared among multiple users, multiple iterations may be necessary before succeeding in finding available resources that satisfy the hierarchical mapping. If no single solution is found at a given level of the physical resource tree, the algorithm can go up a level and look for higher-level sub-trees (which provides access to a larger number of physical resources).

The result of this step is a mapping of each logical group onto a physical group, which is consistent with the hierarchy. Each such mapping generates a placement constraint. This set of constraints and the corresponding set of reservations are given as an input to the deployment tool provided by the grid, which can then trigger the deployment of the configured application. The result of this deployment phase is a *deployed application*.

3.5 Expansion and retraction

In this paper we consider the dynamic nature of the application expressed as the need to make its topology expand or retract. We do not address, for example, the problem of migrating deployed entities to other physical resources. The idea is here to let the application request: 1) the deployment of new entities; and 2) the removal of some entities currently deployed. We define two corresponding basic operations, *expand* and *retract*, designed to be applied on the logical representation of the application, as shown in Figure 5.

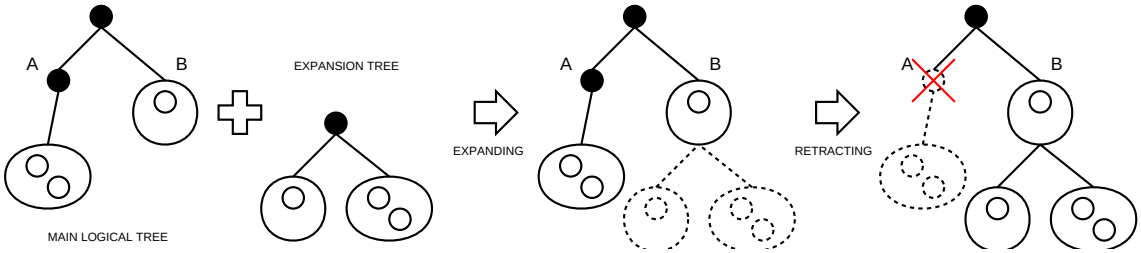

Figure 5: Expanding, then retracting an application.

Expanding a previously deployed application consists in adding new entities to the set of currently deployed entities. This operation is performed following the CORDAGE model introduced in the previous sections. New entities have to be organized into a logical tree, along the same lines as the main logical tree. This new tree is called the *expansion tree* and has to be attached as a sub-tree of a particular node or leaf of the main logical tree. This particular node or leaf is called the *expansion node* (node B tagged on Figure 5) and replaces the virtual node standing at the root level of the expansion tree. The application may provide additional information to determine the expansion node. The mapping step is then processed on the expansion tree, keeping in mind that its expansion node is already mapped to a physical group. Once the mapping done, CORDAGE can request the deployment tool to deploy the additional entities.

Retracting a previously deployed application consists in removing entities from the set of currently deployed entities. The corresponding CORDAGE operation is performed on the logical tree by removing a particular sub-tree. This sub-tree, called the *retraction tree*, is a complete sub-tree that includes the leaves. This choice has been made to avoid the reconfiguration of the logical tree that would imply some migrations of entities. The root of the retraction tree is called the *retraction node* (node A tagged on Figure 5) and is given by the application while requesting the retraction. Removing the retraction tree from the main logical tree implies the removal of entities from their physical resources and the deletion of associated node reservations. CORDAGE invokes the regular grid management tools with the appropriate parameters.

3.6 Co-deployment

The model proposed in this paper let CORDAGE describe the deployment and the evolution of an application during its runtime. The need to deploy and manage applications made of several sub-applications with all their cross-application constraints, can also be taken into account. This particular kind of deployment is called *co-deployment*. CORDAGE handles two types of additional deployment constraints. The first type of constraint is temporal constraints, indicating that an entity from sub-application X has to be deployed and started before launching another entity from sub-application Y. The second type of constraint is placement constraints, indicating that an entity, or a group of entities belonging to sub-application X has to be placed on the same physical resource than an other entity from sub-application Y, as illustrated in Figure 6.

In the CoRDAGe model, co-deployment boils down to a set of *cross-application placement constraints*. These constraints are used while processing the mapping of logical trees on the physical tree. This ensures for example that logical groups C and D will be mapped to the same physical group ($S1$ in this example). Choosing the same physical group for C and D also implies that all their parents, taken two by two in each upper level, will also share the same physical group. In this example, A and B are mapped to the same grid $G0$.

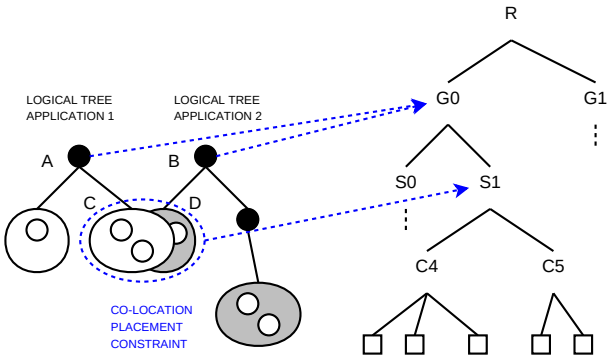

Figure 6: Mapping a co-deployment.

4 Implementation and preliminary evaluation

We have designed and implemented a prototype to validate the model described in Section 3. This prototype features the management of applications using both logical groups and tree representations. It is also possible to perform co-deployment of several sub-applications using a logical tree merging procedure, which is not explained in this paper.

The CoRDAGe prototype consists in a server in charge of processing all the deployment requests coming from an application. Figure 7 displays an overview of the prototype. The main generic action offered by CoRDAGe is the *deploy* action. It performs the deployment of a set of sub-applications according to their internal descriptions. A important practical aspect not discussed above is the *objective end-time* to be provided to the reservation and scheduling tool. In CoRDAGe, it is set by the user on the initial deployment, and all the subsequent deployments inherits of this end-time.

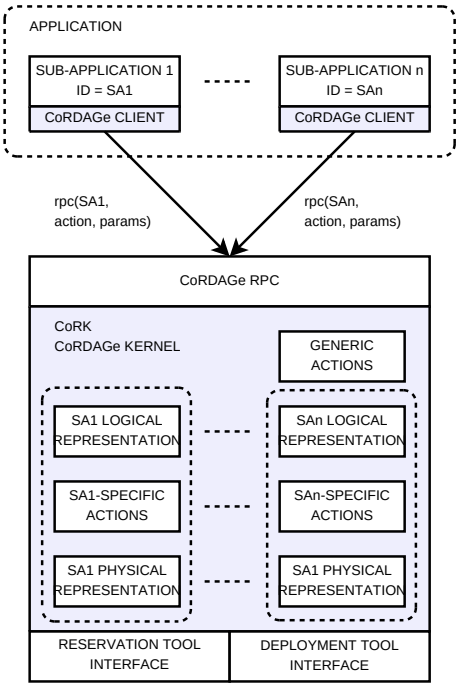

Figure 7: A modular design.

The prototype has been written in C, C++ and Perl languages and uses the XML-RPC protocol specification. CoRDAGe currently relies on the ADAGE deployment tool and the OAR grid reservation tool available on the GRID'5000 testbed. In order to validate the prototype we consider the following basic scenario. A synthetic application is used to simulate the behavior of a JuxMem manager in charge of allocating data in the data-sharing service. This application first requests CoRDAGe to locally deploy a minimal topology of JuxMem made of 1 manager, 1 provider and 2 clients that perform repeated read and write operations on a data. This deployment is ordered with a end-time of 10

minutes later. Once deployed, it requests the expansion of the JUXMEM service by adding two new groups on two distant sites, each featuring a manager and a provider. Finally, the synthetic application repeatedly requests the deployment of new providers within each of these JUXMEM groups in turn. This is done until the initial end-time has been reached.

Our preliminary evaluations have been performed over GRID'5000 using 3 remote distant sites: Rennes, Lille and Sophia, the CORDAGE server being located in Rennes. Adding a new provider took an average time of 10 seconds, depending on the remote site. The time needed to achieve an expansion is actually mainly due to the reservation and deployment tools, especially when dealing with remote sites because of the higher latency.

5 Conclusion

Deploying grid-aware applications on modern large-scale testbeds tends to become more and more difficult. Configuring the application, selecting physical resources, interacting with grid management tools are tedious tasks. We claim that the applications should themselves manage their run in an autonomic way, by requesting new resources on-demand.

In this paper we have described a grid management tool called CORDAGE that helps managing dynamic deployment and co-deployment tasks in a transparent way. It is made possible thanks to the high-level model used to represent both the application and the physical resources. This model is generic enough to be used by various distributed applications. CORDAGE supports generic actions in charge of mapping the application on physical resources, as well as specific actions to adapt its behavior to the type of application.

A first prototype has been implemented to validate this approach. This paper reports on preliminary experiments with initial deployment and topology expansions. We are currently working on topology retraction, co-deployment of sub-applications and integration of CORDAGE in other grid-aware environments like GFARM and DIET. An interesting direction for future work would be to make different instances of CORDAGE communicate and collaborate together.

6 Acknowledgments

This work has been supported by Sun Microsystems, the Regional Council of Brittany and the French National Agency for Research project LEGO (ANR-05-CIGC-11). Experiments presented 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/). We would like to thank Voichita Almasan for her early contribution on this research topic.

References

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

[2] Caron, E., Desprez, F.: DIET: A scalable toolbox to build network enabled servers on the grid. Intl. J. High-Performance Computing Applications **20**(3) (2006) 335–352

[3] Jeanvoine, E., Rilling, L., Morin, C., Leprince, D.: Using overlay networks to build operating system services for large scale grids. Scalable Computing: Practice and Experience **8**(3) (2007) 229–239 Special issue on Practical Aspects of Large-Scale Distributed Computing.

[4] Drost, N., van Nieuwpoort, R.V., Bal, H.E.: Simple locality-aware co-allocation in peer-to-peer supercomputing. In: Proc. 6th Intl. Workshop on Global and Peer-to-Peer Computing (GP2P 2006), Singapore (May 2006)

[5] De Palma, N., Sicard, S., Bouchenak, S., Hagimont, D., Boyer, F.: Autonomic administration of clustered J2EE applications. In: Proc. Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 2005). (2005) 1248–1254

[6] Broto, L., Hagimont, D., Stolf, P., Depalma, N., Temate, S.: Autonomic management policy specification in Tune. In: Proc. Ann. ACM Symp. on Applied Computing (SAC 2008), Fortaleza, Ceará, Brazil, ACM (March 2008) (electronic medium)

[7] Aldinucci, M., André, F., Buisson, J., Campa, S., Coppola, M., Danelutto, M., Zoccolo, C.: An Abstract Schema Modelling Adaptivity Management. In: Integrated Research in GRID Computing. Springer (2007) 89–102

[8] Hermenier, F., Lorca, X., Cambazard, H., Menaud, J., Jussien, N.: Reconfiguration automatique du placement dans les grilles de calculs dirigée par des objectifs. In: Actes 6e Conférence Francophone sur les Systèmes d'Exploitation (CFSE 6), Fribourg, CH (February 2008)

[9] Antoniu, G., Bougé, L., Jan, M., Monnet, S.: Large-scale deployment in P2P experiments using the JXTA distributed framework. In: Euro-Par 2004: Parallel Processing. Number 3149 in Lect. Notes in Comp. Science, Pisa, Italy, Springer (August 2004) 1038–1047

[10] Martin, C., Richard, O., Huard, G.: Déploiement adaptatif d'applications parallèles. Technique et Science Informatiques (TSI) **24**(5) (2005) 547–565

[11] Kadeploy. Available at http://www-id.imag.fr/Logiciels/kadeploy/

[12] Lacour, S., Pérez, C., Priol, T.: Generic application description model: Toward automatic deployment of applications on computational grids. In: Proc. 6th IEEE/ACM Intl. Workshop on Grid Computing (Grid 2005), Seattle, WA, USA, Springer (November 2005)

[13] Antoniu, G., Bougé, L., Jan, M.: JuxMem: An adaptive supportive platform for data sharing on the grid. Scalable Computing: Practice and Experience **6**(3) (November 2005) 45–55

[14] Tatebe, O., Morita, Y., Matsuoka, S., Soda, N., Sekiguchi, S.: Grid datafarm architecture for petascale data intensive computing. In: Proc. 2nd IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (Cluster 2002), Washington DC, USA, IEEE Computer Society (2002) 102

[15] Cudennec, L.: Un service hiérarchique distribué de partage de données pour grille. In: Actes Rencontres francophones du Parallélisme (RenPar 18), Fribourg, CH (February 2008)

[16] Cappello, F., Caron, E., Dayde, M., Desprez, F., Jeannot, E., Jegou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Richard, O.: Grid'5000: A large scale, reconfigurable, controlable and monitorable grid platform. In: Proc. 6th IEEE/ACM Intl. Workshop on Grid Computing (Grid '05), Seattle, Washington, USA (November 2005) 99–106

[17] Capit, N., Costa, G.D., Georgiou, Y., Huard, G., Martin, C., Mounié, G., Neyron, P., Richard, O.: A batch scheduler with high-level components. In: Proc. 5th IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGrid '05), Cardiff, UK (May 2005) 776–783