

VIRTUAL APPLIANCES: A WAY TO PROVIDE AUTOMATIC SERVICE DEPLOYMENT *

G. Kecskemeti¹, P. Kacsuk¹, T. Delaitre², and G. Terstyanszky²

¹ *Laboratory of Parallel and Distributed Systems, MTA-SZTAKI*
{kecskemeti,kacsuk}@sztaki.hu

² *Centre of Parallel Computing, University of Westminster*
gemlca-discuss@cpc.wmin.ac.uk

Abstract Manual deployment of an application usually requires expertise both about the underlying system and the application. To support on demand service deployment or self healing services this paper describes an extension the Globus Workspace Service [12]. This extension includes creating virtual appliances for Grid services, service deployment from a repository, and influencing the service schedules by altering execution planning services, candidate set generators or information systems.

Keywords: grid, web services, deployment, virtual appliance, OGSA

1. INTRODUCTION

In this paper deployment is defined as a composite task. It starts with the *selection* of the site where further deployment steps take place. Then follows with the *installation* of the application's code, which is *configured* to match its new environment later on. After the initial configuration of the application the *activation* step enables its usage. The next deployment steps are closely related to system maintenance, and they are initiated later in the application's lifecycle. *Adaptation* is used when the application has to be reconfigured while it is running. Meanwhile other maintenance steps are always preceded by

*This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265)

deactivation, which disables the usage of the application temporarily. The two deactivation dependent steps are the *update* and the *decommission*. The update changes the application's code, which is usually followed by a configuration step before the application can be active again. Meanwhile the decommission removes all the unnecessary application code and removes the application permanently from the site.

Nowadays application deployment is done by system administrators, because researchers usually cannot deal with their applications (e.g. BLAST [1], CHARMm [14], GAMESS-UK [10]) complex deployment requirements. As a result in a system where the deployment tasks have to be done manually the researchers have to run their applications on those sites where they were installed by an administrator. Even if these researchers could have deployed these applications, site policies in production Grid environments usually do not allow users to deploy new applications. Therefore, they should ask the site administrators to install and configure the applications, they need.

If users have the source codes and they are able to compile these, they have binaries of the applications, which can be staged to the executor site eliminating the deployment. However, this solution has several drawbacks. First *the drawback for the user* is that the application has to be compiled for every site. Also the application usually cannot make network connections with the outside world - e.g. to a database. Finally, the submission has to prepare the executor environment for the application every time. The second drawback is the *increased network traffic* for the frequent full execution environment submissions. And further drawbacks appear on the *system administrator's side* where every new application is a possible danger to the system, because administrators have limited control over this code. Therefore, a single submission could affect other executions.

As Grid moves towards service-orientation trusted applications are exposed as services on the sites, and the users can only execute these services. As a result grid sites disable the "regular" execution of applications, on the other hand they provide service interfaces for their users. Users can access the deployed applications on the different sites through these interfaces. Installing a new application in a service-oriented Grid consists of two main steps. First deploying the application, installing its dependencies, configuring it, preparing network connections, etc. Then providing a service-oriented interface to access the application and deploy the interface on a Grid service container.

This service interface could be either a generic or specific one. The generic interface is provided by a Grid middleware (e.g. OGSA-BES implementations and JSDL repositories) or other generic application wrappers (e.g. AHE [5], GEMLCA [6], gFac [11]). The specific interface is specific to the application and written by either the application developers or by the user community.

Manual deployment of the application usually requires expertise both about the underlying system and the application. In several cases it cannot be done without the assistance from the system administrators. There is a need for a solution that can perform deployments without any specific expertise. This solution should also act between the limits set up by the system administrators. These kind of solutions are called automatic service deployment systems and they can both deploy the application and activate the service interface for further uses. There are two major approaches for automatic deployment: service-oriented and non-service-oriented solutions. The service-oriented solutions operate at service container level. Therefore they are not capable to handle the background application's management tasks. The non-service-oriented solutions manage the service container as a common software component on which the currently deployed software depends.

Automatic service deployment could be based on virtualisation. The new hardware architectures give more and more support for virtualisation. Therefore, the software also has to tackle this issue by automating deployment tasks of virtual appliances of Grid services. This paper defines virtual appliances (VA) as a single application and its service interfaces with all of their dependencies (including the supporting OS) in a virtual machine image. Grid researchers have already developed some solutions in this area, for example: the XenoServer platform [15], the Workspace Service (WS) [12] for Globus Toolkit 4 [7].

The paper is organised on the following way. Section 2 discusses related works, Section 3 introduces some of the most relevant issues which could be solved by the deployment architecture proposed in Section 4, then Section 5 describes an advanced virtual appliance creation service, and Section 6 provides solutions to alter an execution schedule to include deployments, finally Section 7 concludes this work.

2. RELATED WORKS

WS [12] (workspace service) as a globus incubator project supports wide range of scenarios. These include virtual workspaces, virtual clusters and service deployment ranging from installing a large service stack like ATLAS, to deploying a single WSRF service. For service deployment purposes the Virtual Machine (VM) image of the service should be available. The WS is designed to support several virtual machines - XEN [3], VMWare, VServer - to accomplish its task.

The XenoServer open platform [15] is an open distributed architecture based on the XEN virtualization technique. It is aiming for global public computing. The platform provides services for server lookup, registry, distributed storage and a widely available virtualization server.

VMPlants [13] project proposes an automated virtual machine configuration and creation service heavily dependent on software dependency graphs. This project stays within cluster boundaries.

These solutions are focusing on the deployment process itself and do not leverage their benefits on higher levels like automation. As an opposite the solution presented in this paper is focusing on possible extensions on the current deployment systems. The proposed architecture integrates the openness of the Xenoserver platform, the widespread virtual machine support of the Workspace Service, and the DAG based deployment solution presented by the VMPlants. This paper also introduces higher-level services supporting the service lifecycle on grid architectural level.

3. ISSUES OF A VIRTUALIZATION BASED DEPLOYMENT SYSTEM

On demand deployment and self healing services are among the most important improvements automatic service deployment could add to the deployment process. With *on-demand* deployment the applications are transferred, installed and configured on the target site prior their execution. The deployment could be initiated by a broker or an advance reservation system. As these systems aggregate the invocation requests, they have an overview on the level of demand. After the initiation step, the application's code is transferred either from a repository or from the location the requester specifies. Repositories can be the source of trust for the system administrators who usually do not trust arbitrary code. Therefore on demand deployment uses either centralised or distributed repositories. Centralised repositories store all the required and dependent software packages in the same location with their configuration. In distributed repositories, packages are spread around the Grid. Distributed repositories could store pre-packaged software components and they have references to their dependencies. Alternatively software components could be distributed around the Grid using replication solutions.

The newly deployed software components might interfere with the already deployed ones. For example the software installed might need outgoing network connections, therefore on every local hardware resource (e.g. disks, network, processing power) the service should have strict limitations according to the service level agreements. If malfunction occurs in a Grid service and the source of malfunction can be identified the service can reconfigure or reinstall the affected components. Service faults can be recognized through advance fault prediction or active service calls. Advance fault prediction uses component monitors to determine their misbehaviour. Active service calls can return regular output messages or fault messages. If the service response is a fault message the service should identify the cause of the fault which could be

internal or external. In case of internal causes the component causing the fault should be repaired. If the service responds with a regular response message the response has to be validated against the semantics of the service and violations should be reported back to the service initiating the *self-healing* process.

The service deployment with virtualisation can support both the on-demand deployment and the self-healing services in a secure way even at the site level. Using virtualisation techniques, a software repository for on-demand deployment should hold the virtual appliances of the services. Requirements against the hosting environment should also be stored together with the virtual appliances because the virtual appliances are virtualisation technique (e.g. Xen, VMWare, VirtualPC) dependent. Virtual machines could also provide restrictive domains for the deployed software in them. Therefore the limitations - e.g. outgoing network bandwidth, IP address - declared in the service level agreements of the site can be enforced via the virtualisation software. In order to support self-healing, virtual appliances should be distributed in several packages - a base package, and delta packages. The base package is a minimal and robust package on which the delta packages are built. It should contain the necessary components to configure, install further components of the application before their execution, and it should be capable to reinstall these components when malfunction arises. The delta packages should represent those software components which the self healing service is able to repair. The delta packages should be distributed with their individual configuration details in order to support on-demand reconfiguration.

4. AUTOMATIC SERVICE DEPLOYMENT

This paper proposes the extension of the deployment system of the Virtual Workspace Service (WS) with the following two new services, which solve the issues identified by Section 3.

Automated Virtual Appliance Creation Service (AVS). This service supports deployment by creating virtual appliances of Grid services. The virtual appliances should be stored in an appliance repository, for example in the Application Contents Service [9] (ACS). The ACS provides a simple interface for managing Application Archives (AA) [9], which holds both the application packages and their configuration. The WS should access the contents of the service by using the endpoint references provided by the ACS for each AA. The virtual appliances (or service images) should be minimized because the repository may hold large number of virtual appliances, and because smaller sized appliances could be delivered faster to the target site. As a result, a virtual appliance shrinker should optimize the appliance's archive, even if it

site but not available in any repositories. Then in step (2) AVS generates a basic virtual appliance from the XEN [3] Domain M. If its size can be decreased, the service shrinks (or optimizes) the virtual appliance. Which is followed by step (3) where the AVS stores the base appliance and the delta packages in an Application Archive (AA) instance, and adds the configuration and dependency information to support later deployment requests. Finally in step (4) the AVS acknowledges the virtual appliance (or image) creation by sending its ACS End Point Reference (EPR) to the client.

Phase II - Service Deployment from a repository. This phase might not directly follow the virtual appliance creation phase, but it can be repeated as many times as required and it is always preceded by the single execution of phase I. Step (5) is the first step of this phase, where the client asks the virtual workspace service to deploy the VM image using the ACS EPR returned by the previous phase. In this step the client is an entity having an ACS EPR, e.g. a regular user initiating a deployment, a Grid broker trying to load balance between sites. Then step (6) follows with the WSFactory request to the ACS in order to provide the application archive (AA) instance and its configuration details (including the configuration of the Virtual Workspace). As a result the VM image is transferred to the Grid Site B in step (7). Then in step (8) the WS creates a virtual workspace according to the configuration using the VM image embedded in the AA instance. The endpoint of the created workspace is returned in step (9). Finally in step (10) the client forwards any further service requests to the re-deployed Grid service on Site B. If necessary it manages the VM instance through the previously received EPR (VM termination and configuration changes).

The Automated Virtual Appliance Creation Service and the Scheduler Assistant Service are described in Section 4 and Section 5, respectively.

5. AUTOMATIC VIRTUAL APPLIANCE CREATION SERVICE (AVS)

The Automatic Virtual Appliance Creation Service creates and stores virtual appliances in an ACS repository [9] to make the appliances available for WS. The service is built on the ACS-WS interface, which enables deployment in Grids. The interface should enable WS to deploy virtual appliances retrieved as AA instances from an ACS repository. The AA instances store the virtual appliances and their states. The state of the virtual appliance is composed of WS resource properties and VM configuration parameters. This service has the following functionality:

Creating virtual appliances. The AVS service implements AVS-WS interface. The first implementation of this service uses the XEN Virtual Machine [3]. The service provides four operations. First, it generates an AA instance. Secondly, the service instructs the XEN VM to make a disk and memory image of the domain and store these images in the AA instance. Thirdly, it collects a VM setup, such as XEN domain specific setup and if exists, the network shaping setup for the Domain 0 VM and convert these parameters into WS resource properties. Finally, it uploads the state of the VM image to the AA.

Optimizing virtual appliances. To minimize the size of virtual appliances, they should be shrunk. Efficient image shrinking can be achieved by active fault injection, which is a flexible dependency detection algorithm. After the image shrinking process each WSRF (Web Services Resource Framework) service should be validated against the developer provided tests. The subsystem dependencies detected during the shrinking process should be stored for future use, for example in a CDL [4] (Configuration Description Language) document or in a model driven deployment descriptor.

Repackaging the appliance. The service builds deployment DAGs [13] to define deployments. These DAGs are built on top of configuration and installation nodes. Each DAG starts from a base virtual appliance - which is replicated over the Grid sites. The base virtual appliance is a non functional virtual appliance capable of holding further packages required for the service. The optimal selection of the base appliance is crucial, because the system has to make compromises between transfer time (deliver a complete VA from a remote location) and on site deployment time (construct the required VA from more accessible pieces on site).

Identifying base appliances. The AVS regularly analyzes the AAs in the repository and checks the similarities of their deployment DAGs [13]. The

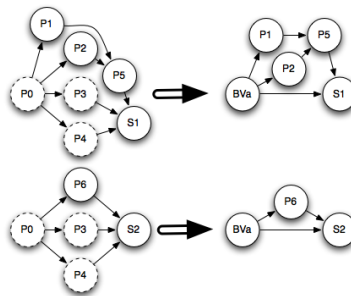


Figure 2. Service Deployment DAGs

similarities mean common roots in the DAGs. Figure 2 presents multi package (Px) deployment strategies for 2 services (Sx). The dashed packages are the similar deployment tasks in the two deployments. If the similar items exceed a system-level threshold, then the common nodes can be stored in a base virtual appliance (BVa). This base appliance is going to be spread over the Grid by replication. Finally, the Application Archives used for building the base virtual appliance has to be revised. The archives should build on this newly created base appliance. The deployment of an archive built on a base appliance is composed of two phases: first the base appliance is deployed, and then this base appliance is started in a self-healing state. During the self-healing process the BVa installs and configures the missing packages for healthy operation.

6. SCHEDULER ASSISTANT SERVICE (SAS)

This service is built on an ACS repository [9] which is usually prepared by the AVS. It's task to define a schedule for executing service requests taking into consideration sites where service deployments are feasible. If the already deployed services are busy (or in a degraded state), then it checks whether any other Grid sites can deliver the service with a new deployment.

OGSA-EPS [8] has two main connections with the outside world: the Candidate Set Generators, and the Information Services (IS). In order to influence the schedules the EPS makes, the assistant service could be installed on any of the following components or on their combination:

Candidate Set Generators. The scheduler assistant generates extra candidate sites for execution. These sites are the ones where the requested services have not been deployed.

Execution Planning Services. The scheduler assistant queries the CSG to retrieve the list of sites, which can handle the service request and which can deploy the service in a given time period. If no site can deliver the requested service, then the EPS makes a decision upon the results of the second query and adds two separate entries to the schedule - the deployment task, and the service call.

Information Services. The scheduler assistant generates virtual entries in the information services. Since both the CSG and the EPS heavily rely on the IS the assistant can include information which might alter their decision. This information states service presence on sites where the service is not even deployed. The QoS information stored in the virtual entries are degraded to represent the latency the deployment would cause before serving the first request. This solution has serious problems compared with the previous two

ones. The IS has to be filled with the full service site matrix, which could increase query times and load on the IS nodes. Non-realistic information is introduced in the information systems this might affect some systems.

Installation of the assistant service or services next to an OGSA-EPS enabled site depends on the grid policies. The assistant will be able to cooperate with an existing EPS as a CSG or an IS source, or it can offer a new, enhanced EPS on deployment enabled Grids.

The *CSG assistant* is a distributed, ontology-based adaptive classifier, to define a set of resources on which a job can be executed. The CSG can build its classification rules using the specific attributes of the local IS. Each CSG may have a feedback about the performance of the schedule made upon its candidates in order to further improve its classification rules using a semi-supervised learning. The CSGs build a P2P network and the EPS's candidate nomination request might spread over the neighboring CSGs for refinement - the request is sent when the quality of the current candidates is below a certain limit. When a new Grid with a CSG is deployed it inherits the rules of the neighboring classifiers at the startup. A SAS extended CSG has three interfaces to interoperate with other CSGs and the EPS. The P2P network formed by CSGs has two interfaces. The first P2P interface manages the ontology of different ISs by sharing the classifier rules and the common ontology patterns distributed as an OWL schema. While the second P2P interface supports distributing the decision-making process, and the resulting candidate set can be sent directly to the EPS without traversing through peers. The third interface lays between the EPS and the CSGs to support the supervised learning technique of the CSGs - the EPS reports the success rate of the received candidate set to the originator.

The *EPS assistant* has different implementations depending on how the other parts of the SAS are deployed. If both the CSG assistant and the EPS assistant are deployed then the EPS can make smarter decisions. After receiving the candidate site-set the EPS estimates the deployment and usage costs of the given service per candidate. To estimate the deployment costs the EPS queries the WS with an ACS endpoint reference, which identifies a particular virtual appliance. The usage costs also include, for example the cost of the inter-service communications - e.g. if a service is deployed closer to its dependencies then the costs decrease. Therefore, the EPS estimates the change in the communication cost between the affected endpoints (e.g. it checks for latencies and available bandwidth). The SAS has a plug-in based architecture in order to support different agents discovering different aspects of the deployment. If the EPS assistant is deployed without the CSG assistant then the EPS could generate deployment jobs on overloaded situations, these deployment jobs are simple service calls to the Virtual Workspace Service with the proper ACS EPR.

The *IS assistant* provides information sources on sites which can accept service calls after deployment. The SAS calculates the necessary metrics specified

by the GLUE schema [2] - like EstimatedResponseTime, WorstResponseTime - for each site in the Grid according to the local service availability, and publishes them in the ServiceData entry.

The SAS-enabled Grid services build on the fact that the ACS EPR is available for the services. Deployments use this EPR to initiate the installation process on the selected site. Therefore, the most crucial point in the system is the place of the repository reference. The SAS can collect the reference from various locations. First it can collect from the standard service repository, the *UDDI*, which can hold metadata for each service, if this metadata is a valid ACS EPR the SAS uses it for deployments. Secondly the SAS can also collect the EPR from the *GLUE-Schema*. The schema specifies Service entries per site, and every service entity has a ServiceData map, which holds key value pairs for extended use of the schema's information. One of the ServiceData objects in the per service description should be the ACS EPR. Finally the most commonly available service description in the web services world is the *WSDL*. It can be customized and further extended as described in the W3C standards. The *WSDL* can describe which virtual appliance was used for instantiating the service by including an ACS EPR element after the service element the *WSDL*. As a consequence both the *GLUE* schema and the *UDDI* is supported without the extensions on them, because they both hold references to the *WSDL* the service conforms with.

7. CONCLUSION AND FUTURE WORK

This paper defined an infrastructure on which an automated service deployment solution can build on. It has covered the lifecycle of service and application deployment from the packaging to the execution. In the future the interface of the self healing virtual appliance has to be further developed to support local and system wide malfunction alerts, and self monitoring systems. Also the SAS should be extended towards seamless integration with other execution management subsystems in OGSA like CDDLM [4].

REFERENCES

- [1] SF. Altschul, W. Gish, W. Miller, EW. Myers, and DJ. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, May 1990.
- [2] S. Andreozzi, S. Burke, L. Field, S. Fisher, B. Konya, M. Mambelli, J. M. Schopf, M. Viljoen, and A. Wilson. *GLUE Schema Specification version 1.2*, 2005.
- [3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt, , and A. Warfield. Xen and the art of virtualization. In *ACM*

Symposium on Operating Systems Principles (SOSP), 2003.

- [4] D. Bell, T. Kojo, P. Goldsack, S. Loughran, D. Milojicic, S. Schaefer, J. Tatemura, , and P. Toft. *Configuration Description, Deployment, and Lifecycle Management (CDDL) Foundation Document*, 2005.
- [5] P. V. Coveney, M. J. Harvey, , and L. Pedesseau. Development and deployment of an application hosting environment for grid based computational science. In *Proceedings of UK e-Science All Hands Meeting 2005*, 2005.
- [6] T. Delaittre, T. Kiss, A. Goyeneche, G. Terstyanszky, S. Winter, and P. Kacsuk. Gmlca: Running legacy code applications as grid services. *Journal of Grid Computing*, 3(1-2):75–90, June 2005. ISSN: 1570-7873.
- [7] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, 2005.
- [8] I. Fosterd, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, , and J. Von Reich. *The Open Grid Services Architecture, Version 1.5*, 2006.
- [9] K. Fukui. *Application Contents Service Specification 1.0*, 2006.
- [10] M.F. Guest, I. J. Bush, H.J.J. van Dam, P. Sherwood, J.M.H. Thomas, J.H. van Lenthe, R.W.A Havenith, and J. Kendrick. The gamess-uk electronic structure package: algorithms, developments and applications. *Molecular Physics*, 103(6-8):719–747, March 2005.
- [11] G. Kandaswamy, D. Gannon, L. Fang, Y. Huang, S. Shirasuna, and S. Marru. Building web services for scientific applications. *IBM Journal of Research and Development*, 50(2/3), March/May 2006.
- [12] K. Keahey, I. Foster, T. Freeman, X. Zhang, , and D. Galron. Virtual workspaces in the grid. In *ANL/MCS-P1231-0205*, 2005.
- [13] I. Krsul, A. Ganguly, J. Zhang, J. Fortes, , and R. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, Pittsburgh, PA, 2004.
- [14] A.D. MacKerel Jr., C.L. Brooks III, L. Nilsson, B. Roux, Y. Won, and M. Karplus. *CHARMM: The Energy Function and Its Parameterization with an Overview of the Program*, volume 1 of *The Encyclopedia of Computational Chemistry*, pages 271–277. John Wiley & Sons: Chichester, 1998.
- [15] D. Reed, I. Pratt, P. Menage, S. Early, , and N. Stratford. Xenoservers: Accountable execution of untrusted programs. In *7th Workshop on Hot Topics in Operating Systems*, Rio Rico, AZ, 1999. IEEE Computer Society Press.