Cyberaide onServe: Software as a Service on Production Grids

Tobias Kurze[1], Lizhe Wang[2], Gregor von Laszewski[2], Jie Tao[3]
Marcel Kunze[3], David Kramer[1], Wolfgang Karl[1]

1 Institut für Technische Informatik, Karlsruhe Institute of Technology, Germany
2 Pervasive Technology Institute, Indiana University, U.S.
3 Steinbuch Centre for Computing, Karlsruhe Institute of Technology, Germany

***Abstract*—The Software as a Service (SaaS) methodology is a key paradigm of Cloud computing. In this paper, we focus on an interesting topic – to implement a Cloud computing functionality, the SaaS model, on existing production Grid infrastructures. In general, production Grids employ a Job-Submission-Execution (JSE) model with rigid access interfaces. In this paper we develop the Cyberaide onServe, a lightweight middleware with a virtual appliance. The Cyberaide onServe implements the SaaS methodology on production Grids by translating the SaaS model to the JSE model. The Cyberaide onServe virtual appliance is deployed on demand, hosts applications as Web services, accepts Web service invocations, and finally the Cyberaide onServe executes them on production Grids. We have deployed the Cyberaide onServe on the TeraGrid infrastructure and test results show Cyberaide onServe can provide the SaaS functionality with good performance.**

Keywords – Virtual Appliance, Gird Computing, SaaS, Cloud Computing.

I. Introduction

Cloud computing [1], [2], [3] is quickly changing the technology landscapes and the way computing resources are used. Various advanced computing paradigms provided by Cloud computing, for example, Infrastructure as a Service, Software as a Service, Platform as a Service, can offer users nontrivial computing features, e.g., QoS guarantee and customized computing environment provision.

Grid computing, which emerged around 15 years ago, now has been widely adopted for large scale high performance computing applications. Many countries and organizations have developed some production Grid infrastructures, which are normally built across distributed computing centers and maintained with rigid access interfaces. As huge manpower and investment have been devoted into building production Grids, it is thus of great interests to bring Cloud computing paradigms and advances on production Grids to make a full usage of production Grid resources. Production Grids normally are well organized and managed in a good order. In this paper, we are interested in developing implementations that bring Cloud computing paradigms to production Grid without changing the interfaces and organizations of production Grids.

A production Grid basically uses a Job-Submission-Execution (JSE) model: a job description is generated by users, sent to a Grid system and is then finally executed. The Software as a Service (SaaS) model employed in Cloud computing is quite different: software is deployed and hosted as a service in Clouds, and clients use this software by remotely invoking services. The SaaS model distinguishes the features of Cloud computing from Grid computing: Cloud users can on demand get software execution environments from remote computing resources with a performance guarantee. Implementing SaaS on production Grids can bring various benefits to the users. For example, allowing Cloud users to use a large amount of existing Grid infrastructures. Grid users can access production Grid, on demand, by invoking their own software services, which can greatly level down the learning curve associated with Grid computing. This paper focuses on how to build Cloud functionalities on a production Grid, specifically, to enable the SaaS on production Grids by translating the SaaS model to the JSE model.

In this paper, we develop a light weight middleware, the Cyberaide onServe, to realize the SaaS model on a productional Grid. The Cyberaide onServe is developed based on the Cyberaide toolkit, which is a light weight middleware for accessing production Grids. The Cyberaide onServe is implemented as a virtual appliance which can be built on-demand. After it is deployed, the cyberaide onServe serves as an access layer for users to access production Grids. The Cyberaide onServe accepts users' software, deploys and hosts users' software as a service in the access layer. When users want to invoke the service deployed, the Cyberaide onServe translates users' requirements into the JSE model and executes the software on production Grids.

The contribution of this paper is two-fold:

- We develop a methodology to implement the SaaS model on production Grids – translating the SaaS model into the JSE model;
- We implement a light weight middleware, the Cyberaide onServe, which can be deployed and access on the fly for on-demand service provision on production Grids.

The rest of this paper is organized as follows: Section II introduces background and related work of Grid computing, virtual appliance, SaaS and Cloud computing. Section III discusses the Cyberaide toolkit, our lightweight middleware for Grid computing and Cloud computing. Section IV formally specifies the requirements to develop SaaS on production Grids. Section V overviews our solution for developing SaaS

on production Grids and the design of the Cyberaide onServe. Section VI details the implementation of Cyberaide onServe, which is developed based on the Cyberaide toolkit. Section VII discusses typical usage scenarios of the Cyberaide onServe. Section VIII then evaluates the Cyberaide onServe implementation. Finally this paper is concluded in Section IX.

II. Background and Related Work

A. Cloud computing and virtualization

Cloud computing has recently become a hot topic. A computing Cloud is a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing infrastructures on demand, which could be accessed in a simple and pervasive way [3]. Conceptually, users acquire computing platforms, or IT infrastructures, from computing Clouds and then run their applications inside. Therefore, computing Clouds render users with services to access hardware, software and data resources, thereafter an integrated computing platform as a service. The Cloud computing distinguishes itself from other computing paradigms, like Grid computing, Global computing, Internet Computing in the following aspects:

- Utility computing model: Users obtain and employ computing platforms in computing Clouds as easily as they access a traditional public utility (such as electricity, water, natural gas, or telephone network).
- On-demand service provisioning: Computing Clouds provide resources and services for users on demand. Users can customize and personalize their computing environments later on, for example, software installation, network configuration, as users usually own administrative privileges.
- QoS guaranteed offer: The computing environments provided by computing Clouds can guarantee QoS for users, e.g., hardware performance like CPU speed, I/O bandwidth and memory size. The computing Cloud renders QoS in general by processing Service Level Agreement (SLA) with users.
- Autonomous System: The computing Cloud is an autonomous system and it is managed transparently to users. Hardware, software and data inside clouds can be automatically reconfigured, orchestrated and consolidated to present a single platform image, finally rendered to users.

This paper aims to bring the advantages of aforementioned Cloud computing methodologies by implementing the SaaS mode for production Grids.

Many implementations of Cloud infrastructures use virtual machines and virtual appliances as basic building blocks, such as Eucalyptus [4], OpenNEbula [5], Xen Grid Engine [6], [7], and Nimbus [8]. Nowadays, popular virtual machines include XEN [9], VMware Server [10], KVM [11] and Microsoft Virtual PC [12]. To reduce the complexity of software development, a relatively new approach is to use virtual appliances. Some software systems are difficult to compile, to link, and to install and have been well tested only on a specific version of tools and platforms. A software publisher can bundle the necessary tools in an appliance and distribute it to users [13]. The Grid Appliance, an example of a virtual appliance, can facilitate the deployment of Grid middleware on distributed virtual machines [14]. Another interesting appliance is CERN VM [15] which is built using rBuilder [16] and provides a minimal Linux base to run LCG (LHC Computing Grid [17]) applications.

In this paper, we develop the Cyberaide virtual appliance for users, which can be built on demand to provide Web services and access to production Grids.

B. Production Grid computing

A production Grid [18], [19] can provide production level services for high end computation and data intensive scientific applications. Normally a production Grid is deployed on a large-scale distributed computing infrastructure across multiple administrative domains. From the viewpoint of Grid users, a production Grid can be accessed with rigid security interfaces and rules. Key features of a production Grid include:

- a production Grid is typically deployed on a large-scale distributed computing infrastructure across multiple administrative domains,
- a production Grid provides multiple production level Grid services, and
- a production Grid is normally accessed with strict secure interface, for example, with x.509 Certificates and Proxies.

Typical examples of production Grids include WLCG [17], [20], TeraGrid [21], [22], and EGEE [23].

In General, production Grids employ a Job-Submission-Execution (JSE) model:

- Grid resource administrators install and deploy users' application on compute resources,
- Grid users specify job submission with some job description language, then submit the jobs to Grid resources for execution.

Compared with the SaaS model, the JSE model has some drawbacks, for example, Grid users sometimes have difficulties in obtaining a customized computing environment such as operating systems and software libraries.

Our work of implementing the SaaS model on production Grids, which are featured by above discussions.

C. On-demand Service Provision

Software as a Service is a key feature of Cloud computing. Software or an application can be hosted as a service and provided to customers across the Internet. This model eliminates the need to install and run the application on the customer's local computers. The SaaS methodology therefore alleviates the customer's burden of software maintenance and reduces the expense of software purchases by on-demand pricing.

There have been significant effort of developing service on demand for Grid computing or distributed computing. SODA [24] is a Service-On-Demand Architecture that enables on-demand creation of application services by dynamically

deploying virtual machines in Grids. [25] investigates a framework for integrating the legacy business systems into Grid environment. A universal factory service [26] provides a dynamic Grid service deployment mechanism and a resource broker called door service. [27] presents a prototype Grid hosting system, in which a set of independent Grids share a network of cluster sites. [28] employs a business-oriented model for Grid computing incorporating dynamic negotiation of service level agreements. [29] proposes an on-demand service architectures for virtual machine based on-demand service provision on Grids. [30] automates the provision of HPC applications as Grid services for on-demand supercomputing and simplifies the construction of client-side applications. The Otho Toolkit [31] develops application-specific Grid service wrappers based on specifications of scientific legacy programs. [32] enhances on-demand QoS provision of Grid services by integrating new functionalities to enable the parties of a WS-Agreement to renegotiate and modify its terms during the service provision. [33] proposes a Grid-based architecture and an implementation to enable dynamic service overlays, which uses Grid factories for creation of execution-environment and service-processes components. An OGSI-compliant software information service is implemented as part of NASA's Information Power Grid project for reconciling information from periodic, on-demand, and user-specified sources [34]. HLAGrid [35] is a distributed simulation framework, which allows resources on the Grid to be utilized on demand by using Grid services. Above work is implemented in following methods:

1) QoS on-demand Grid provision via SLA negotiation [28], [32],
2) Applications specific deployment via language support such as [31], [30], [25], [33],
3) On-Demand service deployment in virtualized machines of Grid resources [24] [29].
4) Implementation via new Grid services [26] [35] [34].

Method 1) discusses on demand provision of QoS for Grid services, which is different from our research focus. Method 2) provides various programming language level support for on demand deployment of users' applications. This method is application specific, which cannot widely applied for different types of users' applications. Production Grids are accessed with strict interfaces, for example, job submission model, which currently do not allow virtual machines deployment of method 3). In method 4) Some Grid services w are developed in Grids new access interfaces, which cannot adopted by production Grids. Therefore previous research cannot solve the research issue of implementing the SaaS model on production Grids.

In this paper, we propose an innovative solution of on demand deployment of all kinds of applications in production Grids, which can comply strict interfaces of production Grids.

III. Cyberaide: A Light Weight Middleware for Production Grids

The solution for the Saas on production Grids presented in this paper is developed based on the Cyberaide toolkit, a light weight middleware for accessing Grids and Clouds [36], [37].

There are a lot of scenarios where an advanced cyberinfrastructure is needed, but it might be difficult to use one. A possible solution to this dilemma is provided by Cyberaide. Several tools have been developed under the Cyberaide banner; well-known examples are Cyberaide toolkit and Cyberaide Shell. Consecutively, Cyberaide toolkit's architecture will be shortly introduced and explained.

Cyberaide enjoys the following essential features:

- *Ease of use*: makes the JavaScript based API and interfaces useful for Grid and Web developers.
- *Low installation footprint*: supports fast downloads as well as easy maintenance through a small manageable code base.
- *Security*: gains access to Grid resources in order to avoid compromising the system. This is especially important due to known limitations of JavaScript.
- *Basic Grid functionality*: provided for developers to create Grid-based client applications.
- *Advanced functionality*: offered as many developers do not want to replicate functionality provided by other Grid middleware and upperware.

IV. SaaS on production Grids: Research Definition

In this paper we focus on the SaaS model on production Grids: users can dynamically deploy their applications as Web services, which in turn execute users' applications in production Grids.

To develop the SaaS functionality on production Grids, a solution has to comply interface requirements:

- The solution cannot change production interfaces and functionalities, which typically are the job submission model, security model and data transfer services. The deployed software services can be accessed, published, monitored and manipulated like a normal Web service.
- When a job is submitted to a production Grid, it is required to access Grid infrastructures on the fly, like security interfaces, resource selection and provision.

In addition to the above functional requirements, some other concerns should be kept in mind in order to build a scalable and flexible Cloud system:

- Toolkits, middleware, and other solutions that are known to work well should be reused whenever possible.
- The solution's architecture should be modularized or structured in modules, so that independent parts can be used separately.
- Provided user-interface(s) should be easy to use and efficient. The most important functions should be provided through a single interface.
- Finally, a solution should be easy to install and to deploy, as well as it should be accessible and usable by more than one person, i.e., a group of people.

V. Cyberaide onServe: Overview of the Solution

This section overviews our solution for implementing the SaaS on production Grids. Our methodology of implementing

SaaS on production Grids is shown as follows (see also Figure 1):

1) Users dynamically start Cyberaide virtual appliance, which serves as an access layer for production Grids.
2) Clients then submit their software executables to Cyberaide virtual appliance.
3) Cyberaide virtual appliance then, on demand, deploys Web services for these executables.
4) Users thereafter invoke their Web services.
5) The invoked Web service forwards a user's invocation to the Cyberaide virtual appliance.
6) The Cyberaide virtual appliance translates the Web service invocation and submits jobs to production Grids for job execution.

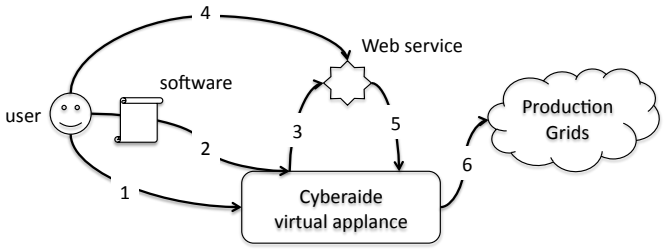

Fig. 1. Methodology of SaaS on production Grid: an overview

In our solution we provide a dynamic access layer, which can be deployed by users on demand to deploy Web services and access production Grids. By introducing an access layer, production Grids are remained unchanged. All intermediate functionalities are implemented in the access layer.

The solution exposes its functionalities through a Web-Interface which basically is the user-interface. The generated Web services may then be used through their Web service interfaces which are described by the associated WSDL description files. All the created Web services are published in an UDDI registry together with the descriptions, the WSDL files, and the service endpoint to make it easier to find a service.

As shown in Figure 2, the Cyberaide onServe is a multi-layer solution:

- Users at the user layer access the Cyberaide layer via multiple interfaces, such as Web portal or Web service interfaces.
- The Cyberaide onServe virtual appliance at the access layer:
  - A Cyberaide portal provides user access interfaces such as software upload and the Cyberaide service management.
  - A UDDI registry is the location where deployed services are published
  - A database stores the uploaded executables
  - A SOAP server runs the deployed Web services as well as some services related to the Cyberaide toolkit.
  - Other Control and Management components.

  The access layer can be deployed locally by a user, or deployed in a shared remote location and used by multiple users.
- The production Grid Layer comprises all Grid related services and tools (for example MyProxy, CoG Kit, etc.) as well as the Grid itself.

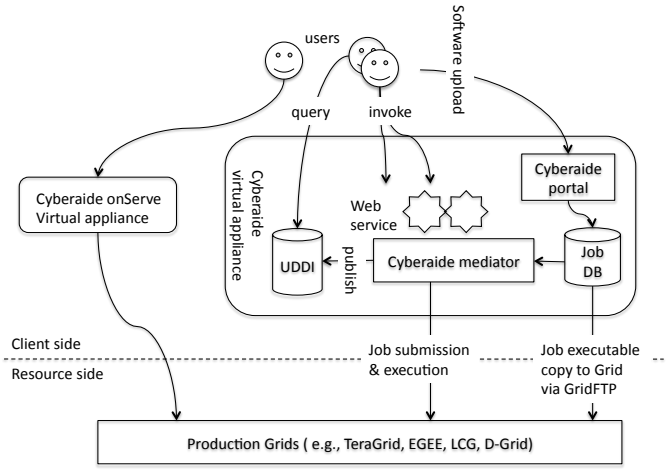

Fig. 2. Cyberaide onServe Architecture

VI. Cyberaide onServe: Implementation

While the previous sections give a rough overview of the extended architecture, this section wants to shed light on some details of the architecture, such as the communication between the different components for instance. Basically, it is a Java implementation that handles all the communication and control mechanisms between each of the components. The implementation is organized in the following packages:

- Database: The "DbManager" class implements the basic functions permitting the storage and retrieval of information. The SQL connection is handled through the "java.sql.Connection" interface. In the case of MySQL, just the MySQL-connector is needed which handles the communication with the MySQL server.
  All functionality concerning the database is organized in the "dataIO" package.
- UDDI: The communication with the UDDI registry is done using the "javax.xml.registry.Connection" interface. The UddiManager class uses this interface to provide the necessary functions to Cyberaide onServe. As jUDDI [38] is an implementation of the Universal Description, Discovery, and Integration specification for Web Services, it is not necessary to use any jUDDI specific code.
  The *"UDDI"* package contains the UddiManager class that provides the necessary functions to publish a Web service in the UDDI registry.
- The GridService "template-class" contains the code that actually initializes the execution of an associated executable on the Grid. It is part of the *"service"* package.
- The *"datastructures"* package contains representations for executables and for Web services.
- The *"tools"* package contains tools like a watchdog class, that is used to react correctly in some situations where a problem may occur. (For example when a process takes too long to complete.)

- Cyberaide Web service client: To create and submit the job to the Grid, Cyberaide agent methods are used. The Cyberaide agent is a Web service and exposes its functions as Web methods. To use these functions, a Web service client has been generated using *wsimport*. This tool automatically creates the required classes to call Web service methods by parsing the WSDL document of the concerned service. Finally, communication with the agent is performed through these generated classes. The Cyberaide Web service client is part of the *"client"* package.
- Build script: Another important detail is the build script generating Web services. There is an ANT build file [39] that provides two basic build options: build the tool and build the Web service. The first option generates the tool that is used directly through the Web interface. The second, and the more interesting option, builds a Web service for a specified file name. Basically, it uses a Web service template file and modifies its name and the initial value of an instance variable. Then it modifies the service description file and generates an aar-file that is finally copied into the Web service framework's service directory. The ant-build-file using the second option is called by the tool itself after an executable has been uploaded and the service is generated.
- Cyberaide portal: The chosen solution reuses the original Cyberaide portal and extends it with a new function. This new function is accessible through an additional button that has been added to the existing Cyberaide portal toolbar. By clicking the new button, the "Upload file and generate Web Service" dialog is displayed. (see Figure 3) The dialog permits the user to select a file that should be uploaded to the portal server and to additionally specify a description and information about possible parameters, such as name and type.
  By clicking the "Upload file and generate Web Service" button, the form's information is passed through a JSP file. The JSP file then loads the chosen file to the portal server and generates a parameter string. This parameter string is used to call the Cyberaide onServe function that generates and publishes a corresponding Web service.

VII. Cyberaide onServe: use scenarios

Our solution provides the following two basic functionalities:

- Accept executables and their descriptions through the Web interface and generate and publish a related Web service.
- When a Web service is used, a Grid job has to be generated and sent to the Grid together with the related executables and parameters.

This section discusses the use scenarios for above functionalities.

A. Uploading executable, generating and publishing Web service

In this scenario a user wants to upload an executable to generate a Web service that might then be executed on the Grid. The principal workflow is illustrated in Figure 4.

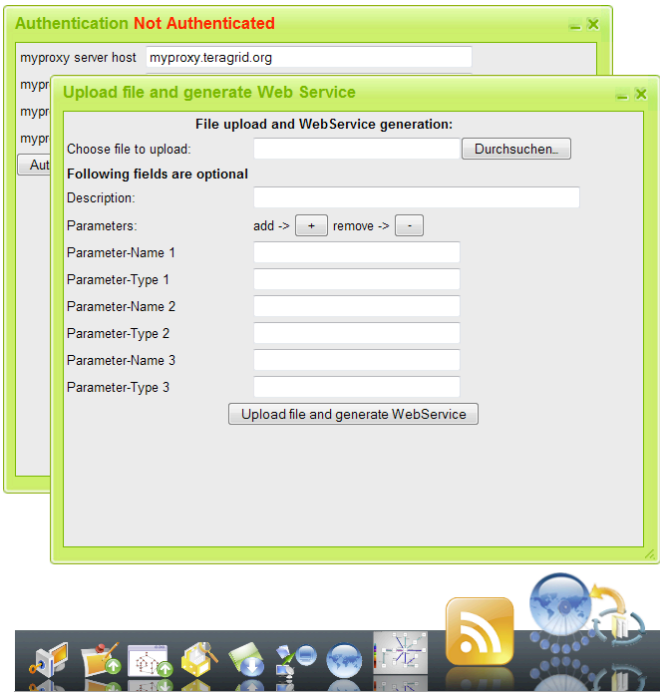

Fig. 3. Cyberaide onServe portal

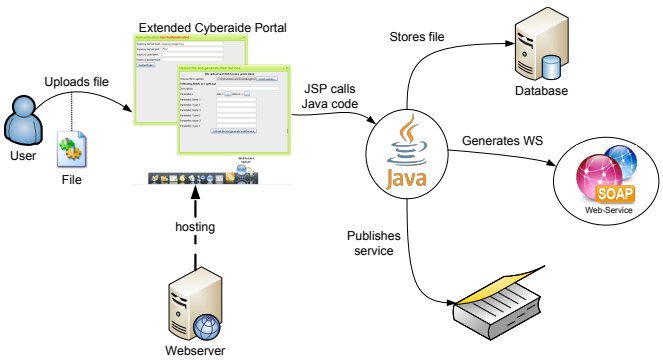

Fig. 4. Upload executable and Web service generation

- **Upload:** The user starts his/her browser and points it to the extended Cyberaide portal Web site. Then, the user selects the "Upload and generate Web service" option. In the appearing dialog, a file has to be chosen and a description may be provided, as well as details concerning parameters and their types that might be passed through to the executable. By confirming the dialog, the file is loaded to the server and a small JSP script creates a parameter list that is then used to start the Java program that conducts further treatment.
- **Further treatment:** This Java program then performs the following steps:
  - *Storage:* It first stores the uploaded file in the MySQL database, together with the its description and the details about the parameters if provided by the user.

- *Service build:* It then calls the ant-build script to generate a Web service that is "linked" with the uploaded file, in the sense, that when the execute function of the Web service is called, the associated file will be executed on the Grid.
- *Publishing:* Finally the generated Web service is published in the jUDDI registry and is then ready to be used.

B. Using generated Web service

The second scenario describes what happens if a user wants to invoke a Web service that has been generated before by Cyberaide onServe. The principal workflow of this scenario is illustrated in Figure 5.

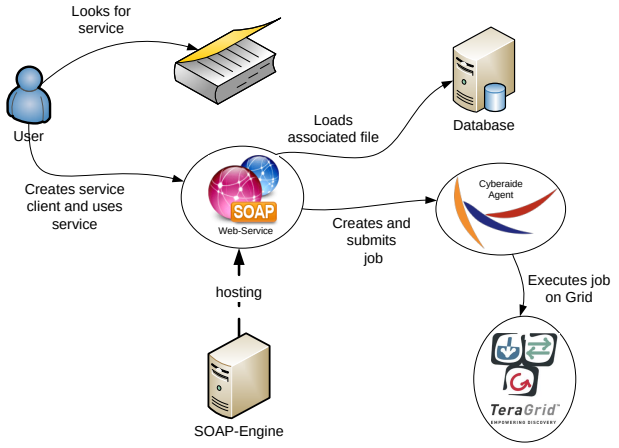

Fig. 5. Use of Web service and execution on Grid

- **Service Discovery and Web service client creation:** First of all, the user examines the jUDDI registry to find the appropriate service. Once the service has been discovered, a Web service client may be created by using the corresponding WSDL document. This WSDL file location is also specified in the UDDI entry for the discovered service.
- **Client execution requested:** When the client is created and the Web service executed, to specify the appropriate parameters, the following steps are performed:
  - *File retrieval:* The first thing that happens is the lookup of the associated file in the database. It is loaded from the database and then stored in a temporary location.
  - *Authentication:* Before any use of the Grid is possible, an authentication is required and performed by the Cyberaide agent.
  - *Upload:* After the successful authentication, the file is uploaded to the Grid by using the functions provided by the Cyberaide agent.
  - *Job description generation:* Subsequently, a job description is generated by using the specified parameters and the name of the executable.
  - *Job submission:* Finally, the job is submitted to the Grid again using functions of the Cyberaide agent and the generated job description.

VIII. Performance Evaluation and Discussion

A. Test organization

We take a performance study on the TeraGrid [21], [22]. The TeraGrid is a production Grid infrastructure which contains 11 supercomputing centers across U.S. We measure the system performance metrics when a Cyberaide virtual appliance works in the TeraGrid test bed. The following subsections discuss the performance of various components of the Cyberaide onServe.

B. Web service client

When requesting the execution of a Web service, respective of the executable represented by the Web service, the associated executable will be loaded from the database and stored temporarily on the disk. Obviously, this step takes more time if the performance of the database and/or the hard disk is poor. Figure 6 shows the CPU utilization, the I/O of the hard disk and the data sent over the network. To generate the graph, a very small file (some bytes) has been used and loaded to a Grid node. It is notable that the hard disk utilization is very low as well as the amount of data sent to the Grid. A relatively large part of the measured traffic is generated by the security credential request and the associated answer. The CPU usage achieves a high value while loading and decompressing the file from the database and a second time when the job is being created and submitted to the Grid.

Another very important factor influencing the performance is the available network bandwidth. The slower the network, the longer the upload of the executable will take. An upload strategy that avoids frequent uploads of the same file may finally result in a better overall performance in this case.

By replacing the small file used in the test before with a much larger file (∼5MB), the bandwidth limitation becomes visible. Figure 7 shows the network and hard disk I/O graphs. The first blue peak indicates the moment the file is written temporarily to the hard disk. Clearly, the hard disk is not the limiting factor in this test, but the network bandwidth is. It takes about 60 seconds to upload the file to the Grid node. The transfer rate is almost constant all the time at about 80 to 90 KB/s. The upload time for larger files, or multiple simultaneous uploads, will probably take even longer.

The current implementation of the solution does contain some workarounds as some features provided by the Cyberaide toolkit didn't work as expected. One result of these workarounds is, that the actual status of the job can't be retrieved and that the local client has to request the output tentatively. Finally this may result in a service customer that requests the application's output more often than necessary which may reduce the network performance even more. This effect is visible in Figure 6 and Figure 7. In a relative constant interval the output of the running job is written to the hard disk, resulting in periodic hard disk write access peaks.

Finally, the provided solution is quite good in a scenario using a lot of relatively small files. The network limitation doesn't play a huge role in this case and K-GRAM permits to submit a large number of jobs quite efficiently. Large files naturally need a lot more of time to be loaded to the Grid and

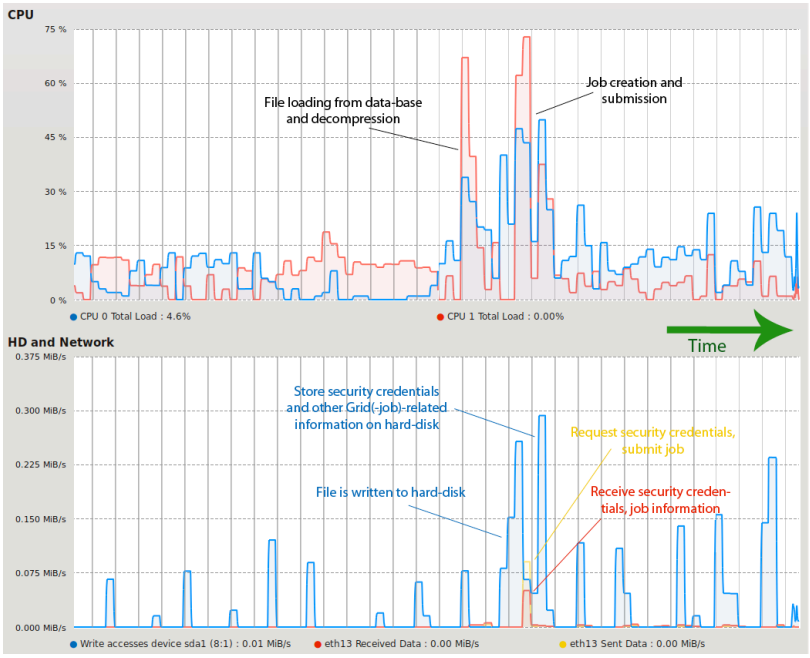

Fig. 6. Web service execution: CPU utilization, network and hard disk I/O (3 seconds interval)

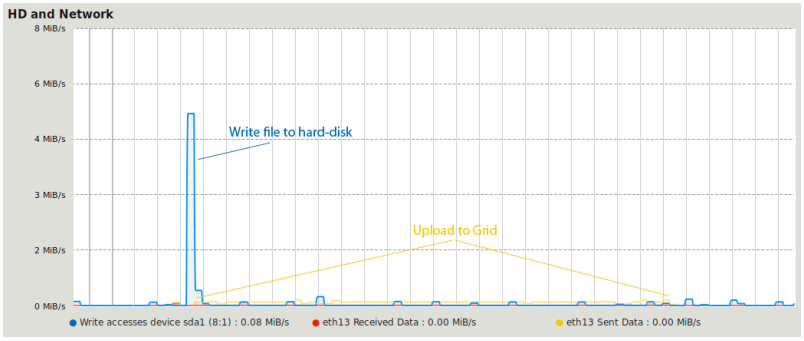

Fig. 7. Web service execution, larger file: network and hard disk I/O (3 seconds interval)

will even be reloaded when executed a 2nd time. This behavior may not be optimal depending on the usage scenario.

On the other hand, there may be files that need a lot of time to finish on the Grid. Depending on the available Grid resources, its workload and the complexity of the executable itself, the execution time may vary tremendously. The additional overhead added by Cyberaide onServe should be quite small compared to the runtime of a typical executable a Grid-Web service is generated for.

C. Portal and Web service generation

The Web Portal performance is basically influenced by two factors, which are the same for the Web service client except for the Grid-service's performance which logically doesn't play a role here.

When a file is uploaded through the Web portal, it is stored in a temporary location. Figure 8 shows a high peak of the network input graph, indicating the reception of the file. The used network operates at 1000Mbit/s, explaining the peak's height. The CPU utilization is very high due to the reception and storage of the file and also because of tomcat handling the request and loading the java-classes. Also, the Web service is being created. In the case of multiple simultaneous uploads, this could pose a problem.

Another implementation related problem is visible in the hard disk graph. Two peaks indicating write hard disk activity show, that the file is written two times. The problem is, that the file is first stored temporarily and then in the database. This solution is not optimal and may be improved. This second storage operation also implies a high CPU utilization.

D. Further Discussion

1) Scalability: Finally, this Section will end with some thoughts concerning the scalability of the presented solution and some of the solution's flaws. It is quite obvious that the solution's scalability is limited either by the system's hard disk I/O-performance or its network connection's performance. The solution doesn't need a lot of CPU time nor a lot of memory,

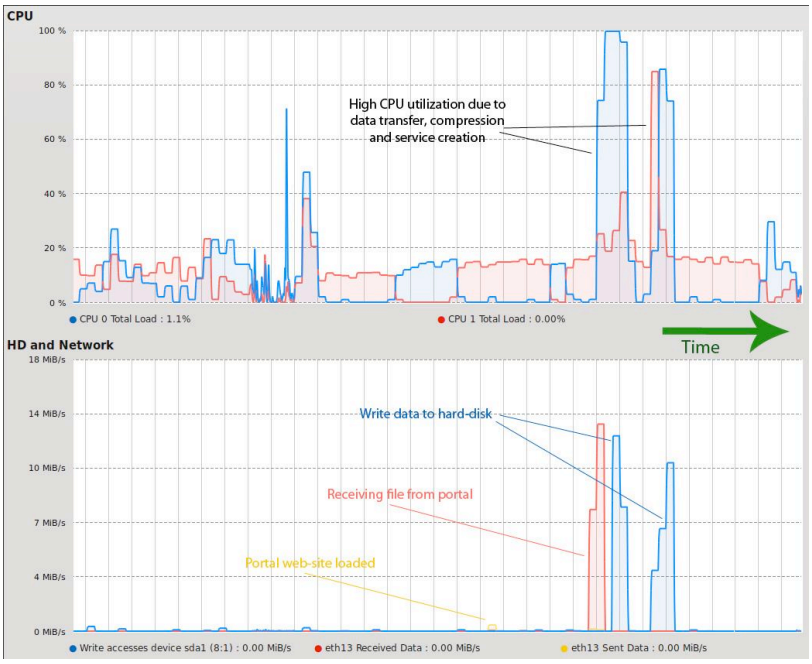

Fig. 8. Upload file and generate Web service: CPU utilization, network and hard disk I/O (3 seconds interval)

even with multiple simultaneously requests - neither of them should hence be the bottleneck.

2) Network connection: A system that only possesses a slow network connection will naturally treat requests much slower than a system with a more powerful network connection. In the case of multiple simultaneous requests, the system's performance might suffer significantly.

This goes for both basic use cases of the presented solution (upload and Web service generation scenario and Web service usage scenario) and is related to the fact that in any case large files (some MBs) might have to be transferred between the involved machines. In a stress-test-scenario, when multiple up- and downloads from and to the system have to be performed, a poor network connection might become a bottleneck slowing down the treatment of the requests.

3) Hard disk I/O: In the "upload and Web service generation scenario", the bottleneck of a "standard" system that is equipped with a "normal" hard disk and a good network connection might become its I/O performance. This is also due to a non-optimal solution when storing the files into the database.

When a file is loaded to the server, it is first stored into a temporary location and then loaded from this location into the database. Hence there are at least two write "operations" and one read "operation" necessary just to store one file in the database. This is not optimal and may lead to performance drops.

When using a Web service the situation is a bit different, as two reads and just one write "operation" are necessary, and also "mandatory", this scenario is not a solution's flaw.

4) Usability: This is a point quite difficult to evaluate as usability is somewhat of a "matter of taste". As mentioned before, the upload and Web service creation functionality is provided through a single interface – a Web-portal (see Figure 3). This Web portal should be easy to use for almost anybody. It is a tool that also should be considered to be very efficient by a majority of users.

To use the generated services, a user should examine the UDDI registry provided by the solution. The user has to do so by using external tools as the presented solution doesn't come with a tool to examine UDDI registries.

To actually use the generated and provided services, a Web service client must be generated. The most easiest solution is to parse the WSDL document with an appropriate tool, such as "wsimport", which then generates all needed classes permitting to use the Web service in a comfortable way. An even more comfortable solution may provide the necessary files as a download, but the proposed wsimport solution also seems to be quite user-friendly as the purpose is to provide dynamically created Web services.

IX. Conclusion

Software as a Service (SaaS), a key functionality of Cloud computing, brings a number of benefits for users, like on demand deployment of software environment, location independent access and efficient software management. This paper focuses on the research to build the SaaS model on production Grids. Production Grids normally use the Job-Submission-Execution (JSE) model with strict access interfaces. The Cyberaide onServe is developed to implement the SaaS functionality on production Grids by translating the SaaS model to the JSE model. The Cyberaide onServe is implemented in a virtual appliance, thus it can be deployed on demand by users to build the SaaS model on production Grids. Test results and performance evaluation on the TeraGrid infrastructure show that the Cyberaide onServe implements the SaaS model on production Grids with a good performance.

ACKNOWLEDGMENT

Work conducted by Gregor von Laszewski and Lizhe Wang is supported (in part) by NSF CMMI 0540076 and NSF SDCI NMI 0721656. The authors want to thank Mr. Jai Dayal for his comments on the paper.

REFERENCES

[1] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC-08, IEEE CS Press, Los Alamitos, CA, USA)*, 2008, pp. 5–13.

[2] L. Wang, J. Tao, M. Kunze, A. Castellanos, D. Kramer, and W. Karl, "Scientific cloud computing: Early definition and experience," in *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications*, September 2008, pp. 825–830.

[3] L. Wang, G. von Laszewski, M. Kunze, and J. Tao, "Cloud Computing: a Perspective Study," *New Generation Computing*, vol. 28, March 2010. [Online]. Available: http://cyberaide.googlecode.com/svn/trunk/papers/08-lizhe-ngc/08-ngc.pdf

[4] "Eucalyptus," [Online], http://www.eucalyptus.com/.

[5] "Opennebula," [Online], http://www.opennebula.org/doku.php?id=start.

[6] "Xge - xen grid engine," [Online], http://mage.uni-marburg.de/trac/xge.

[7] N. Fallenbeck, H. J. Picht, M. Smith, and B. Freisleben, "Xen and the art of cluster scheduling," in *Virtualization Technology in Distributed Computing, 2006. VTDC 2006. First International Workshop on*, 2006, p. 4. [Online]. Available: http://dx.doi.org/10.1109/VTDC.2006.18

[8] "Nimbus Project." [Online]. Available: http://workspace.globus.org/clouds/nimbus.html/

[9] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization." in *SOSP*, M. L. Scott and L. L. Peterson, Eds. ACM, 2003, pp. 164–177. [Online]. Available: http://dblp.uni-trier.de/db/conf/sosp/sosp2003.html#BarhamDFHHHN03

[10] "Vmware server," [Online], http://www.vmware.com/products/server/.

[11] "Kvm - kernel based virtual machine," [Online], http://www.linux-kvm.org/page/Main_Page.

[12] "Microsoft virtual pc," [Online], http://www.microsoft.com/windows/virtual-pc/default.aspx.

[13] C. P. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum, "Virtual appliances for deploying and maintaining software." in *LISA*. USENIX, 2003, pp. 181–194. [Online]. Available: http://dblp.uni-trier.de/db/conf/lisa/lisa2003.html#SapuntzakisBCZCLR03

[14] D. I. Wolinsky and R. J. Figueiredo, "Simplifying resource sharing in voluntary grid computing with the grid appliance," in *IPDPS*. IEEE, 2008, pp. 1–8. [Online]. Available: http://dblp.uni-trier.de/db/conf/ipps/ipdps2008.html#WolinskyF08

[15] "Cernvm," [Online], http://cernvm.cern.ch/cernvm/.

[16] "rbuilder," [Online], http://www.rpath.com/rbuilder/.

[17] "Lcg," [Online], http://lcg.web.cern.ch/LCG/.

[18] I. Foster and J. Gieraltowski, "The grid2003 production grid: Principles and practice," in *HPDC '04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 236–245.

[19] D. Lingrand, J. Montagnat, and T. Glatard, "Modeling user submission strategies on production grids," in *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*. New York, NY, USA: ACM, 2009, pp. 121–130.

[20] T. Antoni, D. Bosio, and M. Dimou, "Wlcg-specific special features in ggus. wlcg worldwide lhc computing grid," CERN, Geneva, Tech. Rep. CERN-IT-Note-2009-018, May 2009.

[21] P. Beckman, "Building the TeraGrid," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 363, no. 1833, pp. 1715–1728, 2005.

[22] C. Catlett, "The Philosophy of TeraGrid: Building an Open, Extensible, Distributed TeraScale Facility," in *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*, 2002, pp. 8–8.

[23] R. Berlich, M. Hardt, M. Kunze, M. Atkinson, and D. Fergusson, "Egee: building a pan-european grid training organisation," in *ACSW Frontiers '06: Proceedings of the 2006 Australasian workshops on Grid computing and e-research*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 105–111.

[24] X. Jiang and D. Xu, "Soda: A service-on-demand architecture for application service hosting utility platforms," in *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2003, p. 174.

[25] Z. Luo, "Research on on-demand grid services access," *Neural, Parallel Sci. Comput.*, vol. 12, no. 3, pp. 407–418, 2004.

[26] E.-K. Byun, J.-W. Jang, W. Jung, and J.-S. Kim, "A dynamic grid services deployment mechanism for on-demand resource provisioning," in *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 863–870.

[27] L. Ramakrishnan, D. Irwin, L. Grit, A. Yumerefendi, A. Iamnitchi, and J. Chase, "Toward a doctrine of containment: grid hosting with adaptive resource control," in *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 2006, p. 101.

[28] G. Engelbrecht and S. Benkner, "A service-oriented grid environment with on-demand qos support," in *SERVICES '09: Proceedings of the 2009 Congress on Services - I*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 147–150.

[29] Z. Huang, C. He, L. Gu, and J. Wu, "On-demand service in grid: Architecture, design and implementation," in *ICPADS '05: Proceedings of the 11th International Conference on Parallel and Distributed Systems - Workshops*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 674–678.

[30] S. Benkner, I. Brandic, G. Engelbrecht, and R. Schmidt, "Vge – a service-oriented grid environment for on-demand supercomputing," in *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 11–18.

[31] J. Hofer and T. Fahringer, "Presenting scientific legacy programs as grid services via program synthesis," in *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2006, p. 34.

[32] G. Di Modica, O. Tomarchio, and L. Vita, "Dynamic slas management in service oriented environments," *J. Syst. Softw.*, vol. 82, no. 5, pp. 759–771, 2009.

[33] O. Ardaiz and L. Navarro, "Grid-based dynamic service overlays," *Future Gener. Comput. Syst.*, vol. 24, no. 8, pp. 813–823, 2008.

[34] P. Z. Kolano, "A unified framework for periodic, on-demand, and user-specified software information," in *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 273–280.

[35] Y. Xie, Y. M. Teo, W. Cai, and S. J. Turner, "Servicing provisioning for hla-based distributed simulation on the grid," in *PADS '05: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 282–291.

[36] G. von Laszewski, A. Younge, X. He, K. Mahinthakumar, and L. Wang, "Experiment and Workflow Management Using Cyberaide Shell," in *4th International Workshop on Workflow Systems in e-Science (WSES 09) in conjunction with 9th IEEE International Symposium on Cluster Computing and the Grid*. IEEE, 2009. [Online]. Available: http://cyberaide.googlecode.com/svn/trunk/papers/09-gridshell-ccgrid/vonLaszewski-ccgrid09-final.pdf

[37] "The cyberaide project." [Online]. Available: http://multicore.amd.com/

[38] "Aparche juddi," [Online], http://ws.apache.org/juddi/.

[39] "Apache ant," [Online], http://ant.apache.org/.