

# Automating Provisioning of a Complete Software Stack in Grid Environment

Bikash Agarwalla\*

Georgia Institute of Technology  
801 Atlantic Drive NW  
Atlanta, GA 30332 USA  
bikash@cc.gatech.edu

Vanish Talwar, Sujoy Basu, Raj Kumar

Hewlett-Packard Laboratories  
1501 Page Mill Road  
Palo Alto, CA 94304 USA  
{vanish.talwar,sujoy.basu,raj.kumar}@hp.com

## Abstract

*With scaling of data centers, clusters, and grids, it is going to be increasingly difficult to download, configure, install, update, and manage the software stack on the constituent nodes. Automation of this process is needed for the system to scale beyond a few hundred nodes. We present system architecture for a middleware service that performs this task in an automated manner. We propose that various components of our architecture can be implemented as grid services with clearly defined interfaces. Furthermore, we demonstrate the usefulness of the architecture through sequence diagrams and a prototype implementation. Our prototype implementation automates the whole process of provisioning a complete software stack by integrating various existing solutions, such as kickstart and smartfrog, along with building intelligence into the middleware to handle such tasks. Preliminary experimental results show that the installation time increased slowly at an acceptable rate of 30% as we moved from 1 machine to 6 machines.*

## 1 Introduction

Grid Computing has its roots in solving compute intensive problems. Traditional use of grid has been in solving compute intensive medical research problems, weather forecasting, and scientific visualization applications. But the application domain for grid is continually expanding with recent efforts focusing on the use of grid in supporting data centers and interactive application environment. In a data center environment, grid computing can be used to provide various features such as security, virtualization, naming, and discovery.

With scaling of data centers, clusters, and grids, we predict that it is going to be increasingly difficult to install, configure, update, and manage the software stack installed on the machines. For a small data center, most of the configuration management tasks can be performed manually, but when the size of the data center increases to a few hundreds or thousands of nodes, manually performing the configuration tasks become practically impossible. Additional complications arise due to the heterogeneity in the software and hardware configuration. If all the machines in the grid have same hardware and software configuration, a simple script may be enough in configuring and maintaining them, but such a homogeneous configuration is not possible due to various reasons. Differences in the hardware configuration

---

\*Work done during internship at HP Labs. Current affiliation: College of Computing, Georgia Tech, Atlanta, GA.

may come due to newly bought machines or upgraded machines in the data center. The software configuration may become different because of different customer needs. All of the different configurations are necessary if a data center is to be used efficiently with minimal financial impact. Because of these heterogeneity in the configurations, a higher level representation of configuration attributes, and validation is needed. In addition, the system should also have monitoring and fault recovery services and security features to protect the resources from malicious use.

Thus, we argue that automation of the whole process of installation and deployment of complete software stack is needed for the system to scale beyond a few hundred machines. Many design choices exist, however, we describe a general system architecture that can take care of the issues mentioned above. In our prototype implementation, we focus on automating the whole process by integrating various existing solutions along with building intelligence into the middleware to handle such tasks. We have identified following set of tasks as crucial in automating provisioning of complete software stack in a grid environment.

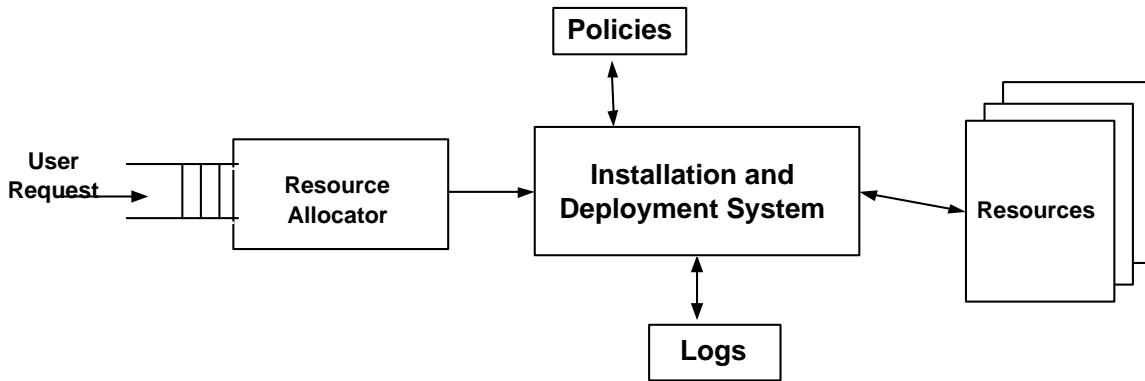
- **Software Install:** Software installation step includes installation of OS as well as the application packages. We consider a data center providing resources for hosting e-commerce applications. In this case, the various tiers of a n-tier e-commerce application should be installed in addition to the OS. The system architecture should take care of dependencies between various tiers. In addition, the system should also automate this process with support for remote administrations in order for it to be scalable.
- **Version Upgrade:** This process involves applying routine upgrades to already installed applications on various machines. The upgrades should be done seamlessly with minimal system downtime.
- **Software Configuration:** The software installed on the machines may need to be configured properly before they can be deployed. A web server typically is configured to run on a specific port and access data from specific directory. Similar configuration steps are needed for application servers and database servers. Some of the configuration aspects may be related to security while others may be related to runtime parameters. Our system architecture performs these software configurations automatically based on higher level policies specified typically by a grid administrator.
- **Deployment:** This step involves running the applications on the specific nodes of the data center. The system should take care of dependencies between various applications and deploy them in order.
- **Life-cycle Management:** The life-cycle management component monitors the deployed applications and take corrective actions based on the system load, or application failures. The various decisions related to life-cycle management are taken based on policies. Life-cycle management task is typically performed in conjunction with the resource allocator.
- **Termination:** The deployed applications may need to be terminated based on the user request or by the system as part of life-cycle management.
- **Software Uninstall:** Software may need to be uninstalled driven by the user request or by the need for efficient utilization of resources in the grid.

Separate tools exist which address some of the problems mentioned above, but none of the existing solution is capable of automating the whole process in an intelligent way. We have taken into account all of the above issues in coming up with a solution for automatic provisioning of complete software stack (operating system as well as an e-commerce application) in a grid environment. Our contributions in this paper are as follows:

1. we present a system architecture that can be used to automatically install and deploy the complete software stack in a grid environment
2. we propose that the various components of our architecture can be implemented as grid service
3. we demonstrate the usefulness of the architecture through sequence diagrams and a prototype implementation.

The rest of the paper is organized as follows: In section 2 we describe our system architecture. In section 3 we discuss sequence diagrams and our system implementation in detail which provide primary validation for the usefulness of the architecture. We put our work in the context of other related works in section 4 and finally, present our conclusions in section 5.

## 2 Architecture



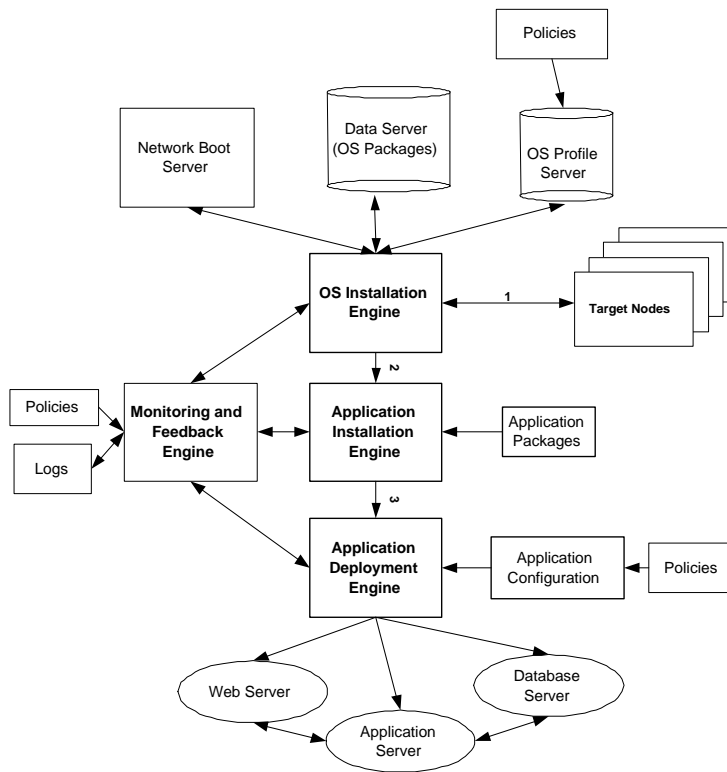
**Figure 1. System Architecture**

Our system architecture is represented in figure 1. The user submits request for deploying an e-commerce application through the grid portal. The resource allocator takes the requests from the input queue and assigns resources based on scheduling algorithms. Details of our resource allocator system can be found from [13].

In this paper, our focus is on the installation and deployment system. Based on the resource assignments, the installation system performs OS installation on the target nodes. The system takes decisions based on policies specified by the grid administrator. Based on policy, the system will decide the packages that should be installed on the target nodes. Once the installation is complete, the e-commerce application will be deployed using the deployment system.

Figure 2 shows the architecture of the installation and deployment system in detail. We present a component based architecture with each component responsible for a specific part of the installation and deployment process. There are four main components of the system, OS installation engine, application installation engine, application deployment engine and the monitoring engine. Each of these components can be implemented as a grid service. In the remainder of this section, we describe these various components and their interfaces in detail.

The OS installation engine performs the automated OS installation. It contacts the network boot server for bootstrapping the installation process. The network boot server will typically be a bootp server used for performing network installation. The data servers will contain the installation packages and the kernel images for initiating the installation and can potentially be distributed for load balancing purpose. The OS profiles will determine the particular softwares that will be installed on a machine. These profiles are generated automatically based on policies. For example, all machines in the product



**Figure 2. Installation and Deployment System Architecture**

group of an enterprise may require a specific machine configuration which is different from the machines in the research and development group. These policies can be used to customize the grid environment. In order to integrate with the application installation engine, the OS installation engine performs the following steps.

- OS installation engine sends a completion message to the application installation engine so that application installation can be initiated.
- OS installation engine installs a specific application at the end of OS installation. This specific application is responsible for installing and deploying the e-commerce application. In our implementation, we used SmartFrog[8] as the specific application.
- OS installation engine should send progress information to the monitoring engine periodically so that any failure can be identified and corrective action can be taken.

Once OS installation engine has completed its task, the application installation engine takes over the target nodes and is responsible for installing the applications. In this paper we focus on e-commerce application installation in a grid environment though our solution is generalizable to other applications as well. The application installation engine will consult the appropriate application packages (webserver, app server, and database server in our case) and install it on the allocated machines. The application installation engine integrates with the application deployment engine through the exchange of following messages:

- Application installation engine sends a completion message to the application deployment engine so that the application deployment can be initiated.
- Application installation engine should send progress information to the monitoring engine periodically so that any failure can be identified and corrective action can be taken. Failure in this case

can be related to application packages being unavailable or improper configuration of the target node. The monitoring engine can instruct the application installation engine to install all the dependent packages before proceeding with the application installation. The monitoring engine can also decide to choose another target node for the application installation in case of node failure. The monitoring engine will contact the resource allocation system for getting access to a new target node.

Once the application is installed, the application deployment engine is responsible for starting and stopping the application. The deployment engine is also responsible for managing the life-cycle of the running applications. This requires continuous monitoring of the running applications and adapting the running application based on system load. System wide policy will dictate the actions that need to be taken in case the system load exceeds a threshold. An example of policy that a typical e-commerce application may specify is to have certain number of instances of the various servers running at any time depending on the load. Such policies are enforced by the application deployment engine as part of life-cycle management. The various configuration parameters related to application deployment and life-cycle management are read from the application configuration module. These application configurations are generated at run time based on policies. In our implementation, the application deployment system is used to deploy the webserver, application server, and the database server. The application deployment system is integrated with the other components through the exchange of following messages:

- Application deployment engine sends progress information periodically to the monitoring engine.
- The life-cycle management component of the application deployment engine should notify the monitoring engine in case of any failure. Failure can be due to changes in the configuration of the target node which prohibit deployment of the application. The monitoring engine can try to correct the node configuration or may decide to deploy the application on a different target nodes depending on policy. Deploying the application on a different node will require notifying all the dependent applications so that connectivity is maintained between the different components of the e-commerce application.

The monitoring engine is responsible for monitoring the progress of the different system components. It monitors the progress of the OS-installation engine, the application installation engine and the application deployment engine. The system progress is written to the log files for analysis. In case of failures of any particular component, the monitoring engine provides feedback based on various policies as mentioned above.

The above system architecture can be used to deploy any OS (linux, windows, solaris) and any e-commerce application in an automated manner. We have identified the interfaces between the various engines so that the OS installation engine, application installation engine and the application deployment engine can be implemented as grid services. In the next section we briefly describe our system implementation.

### **3 Implementation**

In our implementation, we chose to install redhat linux OS and a 3-tier application in an automated manner. The 3-tier application consists of a web server, an application server, and a database server. We have used kickstart [11] with preboot execution environment (PXE) [9] as the underlying technology for automating OS installation in a grid environment. The 3-tier application is installed and deployed using SmartFrog [8]. Though we demonstrate the usefulness of the system through these specific examples, our architecture is general enough so that it can be used to perform similar tasks for other applications and OS as well. The details of each of the system components are described below.

### 3.1 OS Installation

In a large data center or grid environment, it is not possible to do OS installation using floppy drive or CD-ROM as is traditionally done. Even when OS is installed through the network, the bootstrap method traditionally used makes use of a floppy with the minimal linux kernel. This set up is not scalable as the number of machines increases beyond a few hundreds. Our system uses a combination of PXE and Kickstart to completely automate the process of OS installation without the need for any floppy or cd-rom. The system includes a DHCP Server, TFTP Server, and NFS server. The target nodes use the above servers for getting the different pieces required for OS installation. The servers may all be located on one machine or they may be distributed. There may be multiple servers for load sharing in our system.

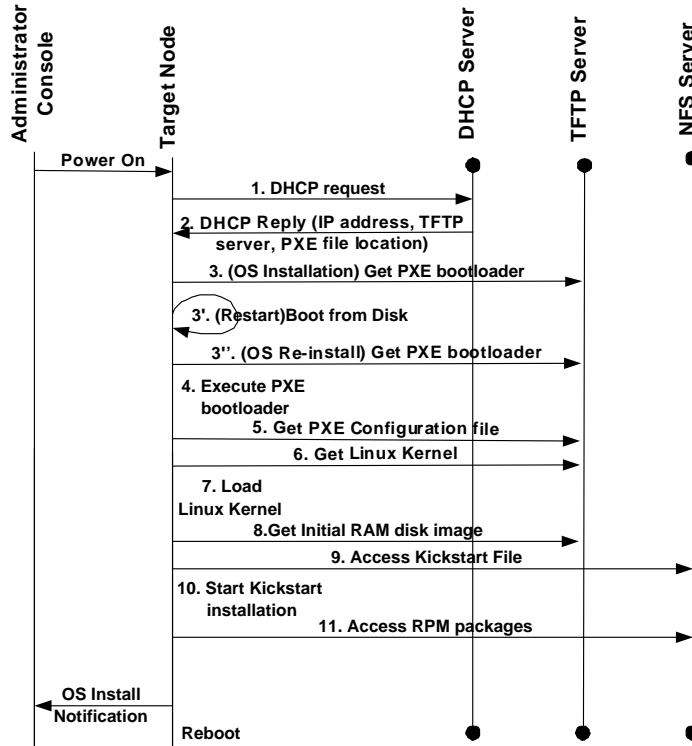


Figure 3. Sequence Diagram for OS installation

The figure 3 describes the sequence diagram for automating the operating system installation. The system administrator will power on the target node to initiate the OS installation. The DHCP server is used to assign an IP address to the target nodes. The TFTP server is used to provide various files to the target nodes. For OS installation or re-installation, the target node first gets the network boot program(PXE bootloder) from the TFTP server and starts booting from it. In contrast, when the machine is restarted after a normal shutdown, the target node starts booting from the disk. The TFTP server is configured at the end of OS installation to enable this switching. These different use cases are shown in the sequence diagram in step 3.

In the case of OS installation and re-installation, the PXE bootloder gets the linux kernel and the image files from the TFTP server and starts loading the kernel. The kernel is passed, as argument, the location of the kickstart file on the network (NFS server). The NFS server is used to provide the kickstart file as well as the RPM packages that need to be installed to the target nodes. The kickstart configuration file contains the choices that are to be made during the OS installation. It also contains the location of the packages that are to be installed.

In our particular implementation the NFS server, DHCP server, and TFTP server were running on a single machine with two network card, one for connecting to the internal network and another for

connecting to the internet. The target nodes were on the internal network. The installation progress can be visualized through this external network interface on an administrator console.

The new target node needs to be configured for booting over the network before the installation can proceed. This step may require changing the BIOS configuration to enable booting over the network. Once the target node is configured for booting over the network, it is powered up by the system administrator through an administration console. After that, all the installation steps are carried out without any human input. Once OS installation is complete, a notification message is sent so that application installation and deployment step can be carried out. As part of the post-installation process, we performed the installation and configuration of smartfrog [8] which is the special software we used for installation and deployment of application.

### 3.2 Application Installation and Deployment

This section describes our implementation for performing a 3-tier application installation and deployment. The automation process goes through the sequence of downloads, installation, configuration, deployment, life-cycle management, and termination for the 3-tier application. In our set up, we chose to follow these steps for the application server (JBoss) [10], the database server (postgres) [2], and a servlet application (guest book) built by us. The *guest book* application allows a guest to enter his name and some messages through a browser. The messages are stored in the database and the system responds with all the messages that are stored as well as the number of times the user has logged in after querying the database. This servlet application uses *JDBC* [1] API's to connect to a database. In our particular testbed, we chose not to install and configure the web server component, though the system can easily support it as well.

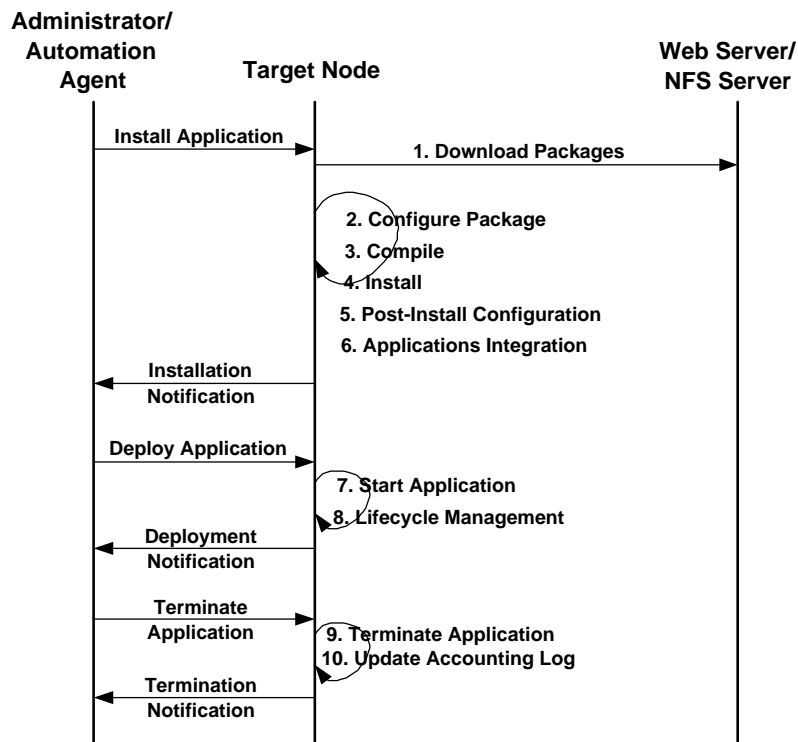


Figure 4. Sequence Diagram for Application installation

Figure 4 shows the sequence diagram for installation and deployment of a generic application. The installation and deployment process is initiated by an administrator or an automation agent. As a result, the target node downloads the application packages from a web server or a NFS server. After the application package is downloaded, it is configured, compiled, and installed at the target node. After

that, some post-install configuration may be needed. For example, when installing a database server, special directories for holding the database may need to be created. Also, additional configurations may be needed for integrating various applications. In our system, the applications server, and the database servers are configured to work together as part of this integration step. Once the software is installed, it is deployed by the administrator or the automation agent. As part of the deployment process, the life-cycle management component of the application is started. Finally, the application is terminated on a future request and the accounting logs are updated. Please note that some of these steps may not be needed depending on the applications targeted. However, an automated system should be able to take care of each of these steps.

We use SmartFrog [8] as the underlying technology for automating the installation and deployment of application. SmartFrog is a framework for describing, deploying, igniting, and managing distributed applications. SmartFrog focuses on the higher level of the configuration task like application service configuration and ignition. The SmartFrog framework consists of a configuration description language, component model, and the deployment infrastructure. Its life-cycle management infrastructure is able to deal with a software system as a single entity even though it may be running over multiple, distributed nodes.

The smartfrog components are described using smartfrog description language. The description includes details that corresponds to the particular components to be deployed on a particular nodes and the configuration parameters needed by the components. The smartfrog daemon running on each node is responsible for deploying and terminating the various components. The daemon loads the configuration description and the component implementation code from the repositories and deploys the component at appropriate nodes. The smartfrog daemon running on various nodes discover and communicate using peer to peer protocols.

Smartfrog is a general and useful framework capable of maintaining the configuration of large scale, diverse, and dynamic environment. Appropriately built components will reduce the time needed to automate the configuration tasks considerably. We have used smartfrog's built-in components wherever possible and have written additional components as necessary. In order to ensure that the newly installed target node can run smartfrog components, we install smartfrog package as part of the OS installation. We also configure the target node so that the smart frog daemon starts up at boot time. All of this is done through kickstart configuration. As a result, when the OS installation is complete, the target node comes up with smart frog installed, and the smartfrog daemon running and configured to deploy applications. In the absence of smartfrog, custom written scripts can be used to aid in installation and deployment of applications.

In our set up, the application packages are downloaded from a web server. We extended the smartfrog in-built components to download as well as install jboss and postgres applications automatically. Some application specific configuration may be needed once the application is installed. In the case of postgres database, a directory for holding the data need to be created and initialized before anyone can use the database. Also, setting up accounts for creating and accessing database, creation of database itself may need to be performed. Each of these post-install configuration task was performed by writing a smartfrog component with properly defined attribute values. We performed special configuration needed to integrate jboss with postgres through appropriate smartfrog components. In order to automate the whole process of deploying the 3-tier application, jboss, postgres as well as the servlet application needs to be deployed in a particular order. Smartfrog component model takes care of the specific order in which these various applications are deployed.

In summary, we used smartfrog components to automate the whole process of application deployment. Using smartfrog for deploying any applications requires a description of the component in the smart frog language, an implementation of the component and its life-cycle management functionalities through a java program or wrapper. Various tasks such as setting up database and environment variable for all the users were performed through appropriate components.

### 3.3 Discussion

We were able to completely automate the process of OS installation and 3-tier application installation on small number of machines (<5) in the data center. In our preliminary experiments, we wanted to find out the benefits of automation. We performed postgres installation multiple times, varying the number of machines. These experiments were conducted on 6 machines, 3 of which have Intel XEON 2.2GHz processor, 1GB memory and 4 processors per node. The remaining 3 machines are Intel Pentium III 1.2GHz dual processors with 384MB memory. We calculated the total amount of time it took to install the software on all the machines through our system. As the result in figure 5 shows, the installation time increased slowly at an acceptable rate of 30% as we moved from 1 machine to 6 machines. The finish time for 6 machine configuration varied from 232 seconds to 358 seconds for the faster and slower machines respectively.

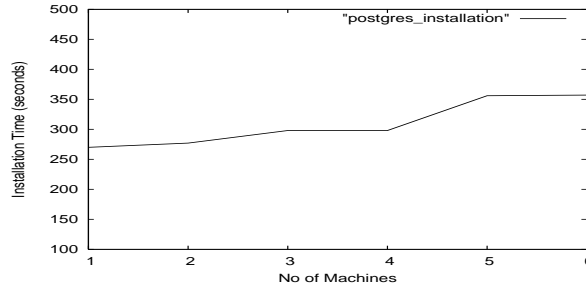


Figure 5. Postgres Installation

Further quantitative analysis of scalability of the system was not possible at the time of this writing. However, since the system takes decisions based on policies, and since resource allocation system is coupled with the installation and deployment system, hence, if system load exceeds beyond acceptable limit (as defined by policy), new servers can be installed and deployed. Future experimental studies will be conducted to give guideline for defining policies related to performance.

## 4 Related Work

We are not aware of any system which performs the automatic provisioning of the complete software stack in a grid environment. However, separate solutions exist which perform the OS installation, application installation and deployment. NPACI Rocks [6], Kickstart [11], Jumpstart [12] are some of the tools for automating the OS installation. SmartFrog [8], LCFG [4], and cfEngine [7] are some of the tools used for configuration management of a node. cfEngine can be used to perform specific configuration tasks such as file operations(linking, setting permissions), editing text files, bringing up network interfaces, and monitoring important files for changes. However, cfEngine doesn't provide support for OS installation or e-commerce application deployment. SmartFrog provides a general framework for deployment and life-cycle management of applications but it doesn't provide support for OS installation. Moreover, our implementation uses specialized SmartFrog component for deploying e-commerce applications. LCFG provides support for automating the OS installation but as such, it is not suitable for provisioning e-commerce applications in grid environment.

GridWeaver [5] is closest to our work. They use SmartFrog in combination with LCFG for automating the OS and application installation. In contrast, we present a general architecture for automating software provisioning in grid environment and demonstrate the usefulness of the architecture through sequence diagrams and a prototype implementation. Also, our focus is on provisioning e-commerce applications in the grid environment. The virtual data toolkit(VDT) [3] is an integration of grid middleware that can be easily installed and configured. We believe that VDT is complementary to our work, since we can

automate the complete process of installing VDT through our architecture. Also, our system takes care of OS installation in addition to application installation.

In summary, none of these existing tools is suited alone for the provisioning of the complete software stack in a grid. We have integrated SmartFrog and kickstart with intelligence in the middleware, which leads to a more scalable solution. We have also identified the various components which can be implemented as grid services.

## 5 Conclusion

As the grid scales to thousands or millions of nodes, it will be crucial to automate the process of software installation and deployment on those nodes. We take the initial step in that direction by demonstrating the automation of installation, configuration, and deployment of software stack in a data center environment. We presented our system architecture and identified various components of the architecture which can be implemented as grid services. The usefulness of the architecture was demonstrated through a specific implementation. Our implementation was used to install linux OS and 3-tier application as well as deploy the 3-tier application in a data center. It used kickstart and PXE for performing OS installation and smartfrog in order to install as well as deploy applications on the target nodes. All of these steps were performed automatically without any human intervention.

Separating various system components, defining clear interfaces between the various components through grid services, policy based decision making, having multiple servers for load balancing, and automation of the whole process leads to a scalable solution to the problem of provisioning a complete software stack in grid setting. However, security, remote monitoring, policies, and algorithms for load sharing of the various server components are some of the issues that can be looked into in the future.

## References

- [1] JDBC 3.0 Specification. Available online from <http://java.sun.com>.
- [2] PostgreSQL Database. <http://sourceforge.net/projects/postgresql>.
- [3] The Virtual Data Toolkit. <http://www.cs.wisc.edu/vdt/index.html>.
- [4] P. Anderson. Towards a hi-level machine configuration system. In *Proceedings of the 8th Large Installations System Administration (LISA) Conference*, pages 19–26, October 1994.
- [5] P. Anderson, P. Goldsack, and J. Paterson. SmartFrog meets LCFG - autonomous reconfiguration with central policy control. In *Proceedings of the 2002 Large Installations Systems Administration (LISA) Conference*, Berkeley, CA, 2003. Usenix.
- [6] G. Bruno, P. Papadopoulos, and M. Katz. Tools and techniques for easily deploying manageable linux clusters. In *3rd IEEE International Conference on Cluster Computing (CLUSTER)*, volume 14, pages 258–270, October 2001.
- [7] M. Burgess. Cfengine: a site configuration engine. *USENIX Computing Systems*, 1995.
- [8] P. Goldsack. Smartfrog: Configuration, ignition and management of distributed applications, 2003. <http://www-uk.hpl.hp.com/smartfrog>.
- [9] Intel Corporation. Preboot Execution Environment (PXE) Specification, September 1999. <http://www.intel.com/labs/manage/wfm/wfmspecs.htm>.
- [10] JBoss Group. JBoss. <http://www.jboss.org>.
- [11] Red Hat Incorporated. Red hat linux 7.3: The official red hat linux customization guide, 2002. Available online from <http://www.redhat.com>.
- [12] Sun Microsystems Inc. Solaris 8 advanced installation guide, 2002. Available online from <http://docs.sun.com>.
- [13] V. Talwar, B. Agarwalla, S. Basu, R. Kumar, and K. Nahrstdt. Architecture for resource allocation services supporting interactive remote desktop sessions in utility grids. In *2nd International Workshop on Middleware for Grid Computing (MGC'04)*, Toronto, Canada, October 2004.