A Model-Driven Approach for Virtual Machine Image Provisioning in Cloud Computing

Tam Le Nhan, Gerson Sunyé, Jean-Marc Jézéquel

A Model-Driven Approach for Virtual Machine Image Provisioning in Cloud Computing

Le Nhan Tam[1], Gerson Sunyé[1,3], and Jean-Marc Jézéquel[1,2]

[1] INRIA Rennes - Bretagne Atlantique, France
[2] University of Rennes 1
[3] University of Nantes
{tam.le_nhan,gerson.sunye,jean-marc.jezequel}@inria.fr

Abstract. The Cloud Computing Infastructure-as-a-Service (IaaS) layer provides a service for on demand virtual machine images deployment. This service provides a flexible platform for cloud users to develop, deploy, and test their applications. However, one major issue of application deployment is to ensure the compatibility of software components installed in a virtual machine image. This paper describes a model-driven approach to manage, create configurations, and deploy images for virtual machine image provisioning in Cloud Computing. This approach considers virtual image as product lines and uses feature models to represent their configurations. It uses model-based techniques to handle automatic deployment and reconfiguration, making the management of virtual images more flexible and easier than traditional approaches.

Keywords: Model-driven deployment, cloud computing, virtual image provisioning, feature models

1 Introduction

Cloud Computing [2, 13] has been a hot topic in both of research and industry community recently. It can be described as a new kind of computing in which dynamically scalable and virtualized resources are provided as services over the Internet. Cloud users can access cloud system and use the service through different devices and interfaces. They only have to pay what they use according to Service Level Agreement contracts established between Cloud providers and Cloud users [5]. One of the main features of Cloud computing is the virtualization in which all cloud resources become transparent to the user. They do not need any longer to control and maintain the underlying cloud infrastructure. The virtualization in Cloud Computing combines a number of virtual machine images (VMIs) on the top of physical machines. Each virtual image hosts a complete software stack: it includes operating system, middleware, database, and development applications. The deployment of a VMI typically involves booting the image, as well as installation and configuration of software packages. In the traditional approach, the creation of a VMI to fit user's requirements and deploying it in the Cloud environment are typically carried out by the technical

division of the Cloud service providers. They provide a platform as a service to the user according to SLA contracts signed between the service provider and the user. Usually, it is a pre-packaged platform with installed and configured software components. The standard VMI contains many software packages, which rarely get used and thus the image is typically larger than what would be necessary. This can lead to several difficulties, such as wastage of storage space, memory, operating costs, and waste of network bandwidth when cloning an image and deploying it on the cloud nodes [1].

In the traditional approach, when a cloud user requests a new development platform, the service provider administrators select an appropriate VMI for cloning and deploying on cloud nodes. If there is no match found, then a new one is created and configured to match the request. It can be generated by modifying from the closest-fit existing VMI or from scratch. Several concerns would need to be addressed by the cloud providers, such as: (i) How to create an optimal configuration? (ii) Which software packages and their dependencies should be installed? (iii) How to find the best-fit existing VMI and how to obtain a new VMI by modifying this one?

Cloud service providers want to automate this process because the complexity of interdependency between software packages, and the difficulty of maintenance [7] is time-consuming for the creation of standard VMIs. In other words, they want to give users more flexibility when choosing the appropriate VMI to satisfy their requirements, while ensuring benefits for providers in terms of time, operating costs, and resources. In this paper, we present an approach for managing VMI for Cloud Computing environments, providing a way to adapt to the needs of auto-scaling and self-configuring virtual machine images. In this approach, we consider VMIs as a product line and use feature models to represent VMI configurations and model-based techniques to handle automatic VMI deployment and reconfiguration. We claim that this approach makes the management (i.e., creation, configuration and adaption) of virtual image faster, more flexible and easier than the traditional approach. We validate this approach by an example showing that, given a base model representing all available artifacts, one can easily derive a configuration model (a specific use of a subset of artifacts) and generate all needed configuration scripts to generate its corresponding VMI. The paper is organized as follows. Section 2 describes our solution of managing virtual machine image configurations by using feature models and using the model-driven approach for virtual machine image deployment in Cloud Computing environment. Section 3 introduces an example about deploying a Java web application development platform. Section 4 shows the experiment evaluations. Section 5 discusses the related work, and is followed by the conclusion and future work in Section 6.

2 Model-Driven Approach

In this section, we present a model-based approach for image provisioning. This approach uses an image with a minimal configuration, containing the operating

system, some monitoring tools, and an *execution model*. The goal of the execution model is to install and configure software packages, after booting the deployed images.

2.1 Feature Modeling for VMI Configuration Management

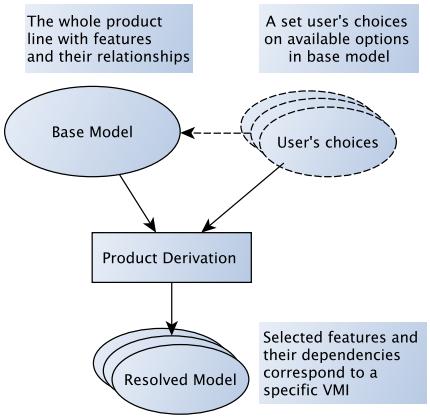

Fig. 1. Feature Modeling Approach

Our approach uses feature modeling [11] to manage the configuration of virtual-machine images. In terms of configuration derivation, a feature model describes:

- The software packages that are needed to compose a Virtual Machine Image, represented as configuration options.
- The rules dictating the requirements, such as dependent packages and the libraries required by each software component.
- The constraining rules, which specifies how the choice of a given component restricts the choice of other components, in the same Virtual Machine Image.

Feature models have a tree structure, with features forming the nodes of the tree and groups of features representing feature variability. There are four types of feature groups: *Mandatory*; *Optional*; *Alternative*; and *Or*. The model follows some rules when specifying which features should be included in a variant. If a variant contains a feature, then:

- All its *mandatory* child features must also be contained;
- Any number of *optional* child features can be included;
- Exactly one feature must be selected from an *alternative* group;
- At least one feature must be selected from an *or* group.

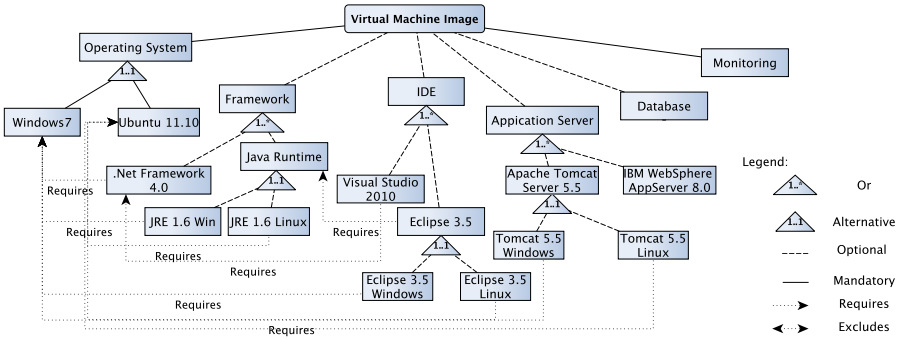

Fig. 2. Feature Diagram Represents a Base Model

Feature models support two cross-tree constraints: *Requires*; and *Excludes*. Given two features, f_a and f_b: if f_a requires f_b, then the selection of f_a implies the selection of f_b; if f_a excludes f_b, then the selection of f_a prevents the selection of f_b.

Our approach deals with two models: base and resolved. The base model represents the whole product line, with all its features, their relationships, and constraints; The resolved model is obtained after the product derivation process, it contains selected features and their dependencies.

Base Model The base model represents configuration options which would be used for composing a VMI. The elements of the base model are features of the configuration options of a VMI, they represent software packages and their dependencies. These elements become elements of resolved models, according to the resolutions of the corresponding selection models.

Figure 2 depicts a part of based model that represents VMI configuration features. In this model, features and their relationships represent software packages:

- *Operating System* is a mandatory child feature of *Virtual Machine Image*, which must be selected when *Virtual Machine Image* is selected.
- *Operating System* includes alternative child features: *Windows* and *Linux*
- When the *Operating System* feature is selected, then one of *Windows 7* or *Ubuntu 11.10* must be selected.
- If the feature *Ubuntu 11.10* is selected, then all features that require *Windows 7* cannot be selected, for instance: *Visual Studio 2010*, *JRE 1.6 Windows*, etc.

Base models are built by IT experts of cloud providers, who have knowledge about systems and software packages used to compose Virtual Machine Images. The correctness of the base model relies on the correctness of the feature model

that represents the base models. Many approaches and tools were proposed to automate analysis of feature models [16, 4, 14]. They offer to validate, check satisfiability, detect the "dead" features and analyze feature models. In our implementation, we use constraints to ensure that the created feature model is valid, and that configurations, which are derived from this feature model, are also consistent with the base model. For example:

- Parent and child features cannot have a mutually exclusive relationship.
- Sibling features cannot be mutually exclusive.
- For two features f_1 and f_2, if f_1 requires f_2, then f_2 cannot require f_1.

Product Derivation Product Derivation is a process that is responsible for the creation of the final configuration. It supports to derive the VMI configurations from the base model [19]. To create a specific configuration of a VMI, the designer selects some features from the base model and uses a mechanism to produce a suitable configuration. The selection of each feature is checked and validated by the Product Derivation process to ensure the selection is valid. When a feature is selected, the Product Derivation process checks its relationships. Features connected to the selected feature by a mutually exclusive relationship become unavailable on the base model for next selections. All of the features that are required by the selected feature are also selected.

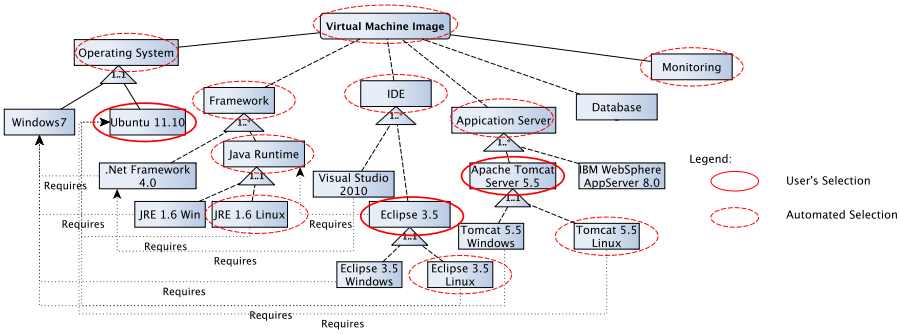

Fig. 3. A Resolved Model Derived by the Product Derivation Process.

Resolved Models A resolved model stores user's feature choices of the base model and their dependencies. It is derived from the Product Derivation process based on user 's selection on the base model. A resolved model corresponds to a specific configuration of a Virtual Machine Image. Figure 3 is an example of a resolved model that is derived from the base model presented in Figure 2 with the following user 's selections: operating system is Ubuntu 11.10, integrated

development environment is Eclipse 3.5, and Apache Tomcat 5.5 for application server. According to the base model, Eclipse 5.5 requires Java Runtime. Both features have two alternative children: Windows and Linux. However, since the selected operating system is Ubuntu 11.10, only the Windows version is available. Addition, since Monitoring is a mandatory feature of Virtual Machine Image it must be selected.

Comment: *BEGIN-EDIT* By using feature models, cloud providers have flexibility to create Base models representing resources for VMI provisioning. Features represent software packages or hardware options, such as RAM or virtualization technology (e.g., KVM or Xen). These feature could also be used to store other informations: time, cost, memory usage, etc., to support finding optimal configurations. The first time for creating the base models might take time and need experts on software packages and their dependencies. However, once the Base model is created, it helps cloud users during the selection and the creation of VMI configurations, reducing time, complexity, and errors during the manipulation. **Comment**: *END-EDIT*

2.2 Model-Based Deployment Architecture

Comment: *BEGIN-EDIT* Unlike the traditional approach, where software packages are installed and configured when the VMI template is created, the model-driven deployment approach installs and configures software packages at runtime when a VMI template is booted. The approach also supports synchronization of maintenance of the deployed VMIs at runtime. This mechanism allows users to update, remove, and add new components to running VMIs, without image shutdown and re-deployment. It is more flexible than the traditional approach. **Comment**: *END-EDIT*.

In our approach, we create models that drive the creation of VMIs instance on demand. Every time a new virtual machine is created on the cloud node, the cloud provider selects features of VMI, generates configurations and applies the model to it. Figure 4 describes an overview architecture of our approach.

- **VMI Repository**
  The VMI Repository contains basic virtual machine images that are used as the initial VMIs: e.g., `Ubuntu11.10.img`, `fedora15.0.img`. These are standard VMIs with minimum configuration, such as operating system and assistance software, like monitoring tools.
- **VMI Configuration Manager**
  The VMI Configuration Manager is responsible for the creation and the management of configurations of virtual machine image to fulfill requested requirements. By using the VMI configuration manager, users can easily select the required software for creating the appropriate virtual machine. It also helps the cloud providers to manage the preparation and provision of resources as per client requirements.
- **Execution Model**
  The Execution Model is responsible for reserving cloud nodes, deploying virtual machines, and executing the corresponding configuration that resulted

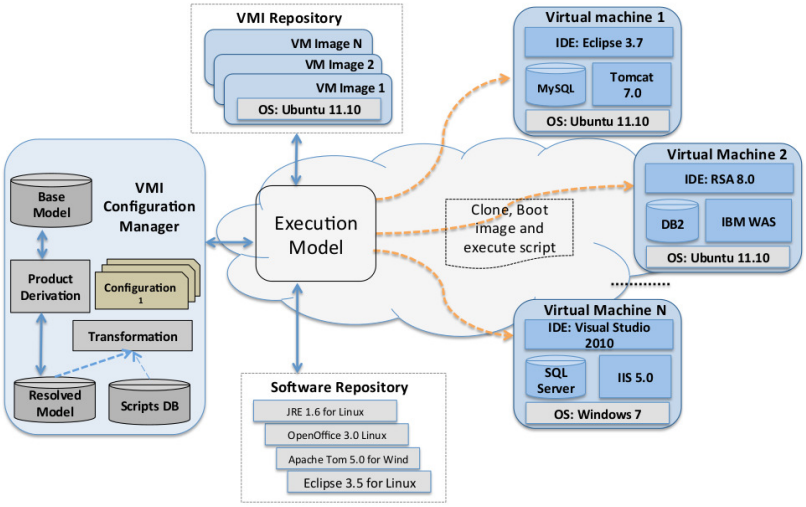

Fig. 4. An Overall Architecture of Model-Driven Approach for VMI Deployment

from the reasoning of VMI Configuration Manager. It is an encapsulation of Ruby and shell script files.

- **Cloud Nodes**
  Cloud Nodes are reserved nodes in the cloud infrastructure for hosting and running virtual machines.
- **Software Repository**
  The Software Repository stores software packages used to compose a VMI. It can be a file server inside the cloud infrastructure or other repositories from the Internet, such as the Debian repository.

2.3 Model-Based Deployment Process

The deployment process deals with the VMI Configuration Manager, the Execution Model, the Software Repositories, and the Cloud Nodes. It includes the following steps:

- **Create a VMI configuration.**
  In this step, cloud users interact with the VMI Configuration Manager to select configuration options from the base model. The VMI Configuration Manager analyzes the user's choices and generates a resolved model (i.e., a valid configuration of a VMI).
- **Generate a deployment script file.**
  A resolved model is transformed into a deployment script file for automatic deployment and configuration. In the current implementation, we use Chef[4]

[4] http://wiki.opscode.com/display/chef/About

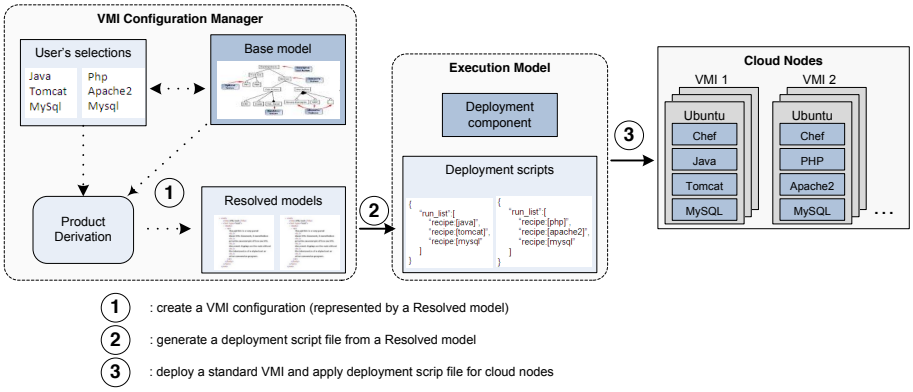

Fig. 5. Model-Driven Process

to automatic install and configure software on a virtual machine. Chef is an installation software that cloud providers use to deploy, install, and configure software stacks on the cloud nodes at runtime. Chef requires an input file, describing the node configuration: the required software, as well as their role. Actually, the input file is a Ruby or JavaScript Object Notation[5] (JSON) source code.

– **Deploy a standard VMI and apply the deployment script file to the cloud nodes.**
The Execution Model, based on the resolved model and deployment script file, selects a standard VMI and launches it on the reserved nodes. After that, it transfers the deployment script to the nodes and executes Chef. Finally, it returns the successful nodes to the cloud user.

3 An example of the VMI for Java Web Application

To illustrate our approach, we introduce an example of VMI provisioning for the Java Web Application Development platform. The configuration of this VMI includes an operating system, a web application server, a database management system, and a programming language compiler.

Cloud users select the required features on the base model by the using VMI Configuration Manager, for example: **Ubuntu**, **Eclipse**, **Apache Tomcat**, and **Database**. Figure 6 represents the selection of configuration options from the base model.

A VMI includes only one operating system, so the choice of Ubuntu feature is mutual exclusive with other operating systems and their dependencies. For example, the users can select neither the SQL Server nor the Eclipse for Windows because both features require Windows, which is a mutual exclusive feature of

[5] http://www.json.org/

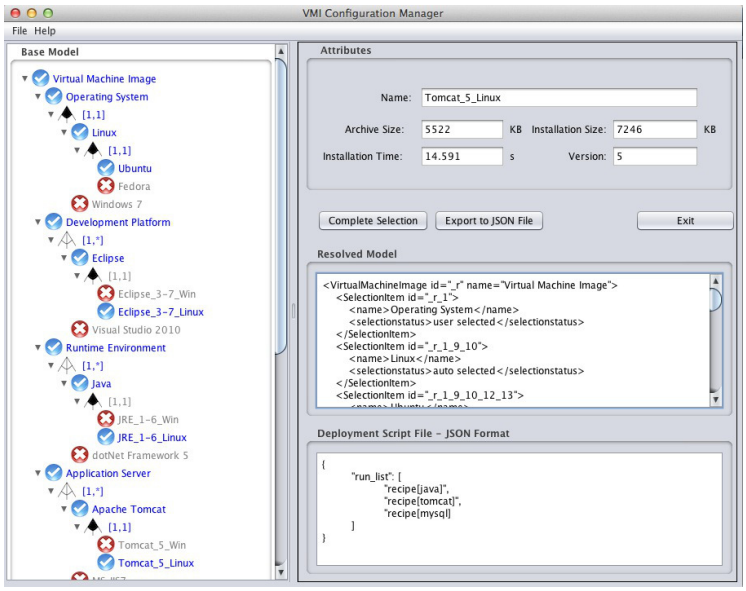

Fig. 6. Example of VMI Configuration Manager

Linux Ubuntu. The features JRE 1.6 for Linux, Chef-Linux are auto-selected because Apache Tomcat requires JRE 1.6 for Linux and Chef-Linux is a mandatory feature.

The Product Derivation process generates a resolved model from the user's selections. The transformation from a resolved model into a script file helps to automatic install and configure software stacks that are selected in the resolved model. Figure 6 also shows the example of a resolved model and a deployment script file, which are corresponding to the user's selection from the base model. The Deployment script is a JSON file, named **deployscript.json**. The Execution model uses the script file for automatic installing and configuring software into the selected virtual image. Listing 1 presents a partial Ruby code for executing the script file on cloud nodes.

4 Experiment Evaluation

In this section, we present an experimental evaluation of our approach on the easiness of manipulation and the performance of deployment, in terms of data transfer and deployment duration. The experiment is executed on Grid5000[6], a virtualization infrastructure for research in France. We use Grid5000 's tools to reserve nodes and deploy VMIs to the nodes.

[6] https://www.grid5000.fr/mediawiki/index.php

Listing 1. A Partial Code in Ruby of the Execution on the Cloud Nodes

```
Net::SSH::Multi.start do |session|
  # access servers via a gateway
  session.via STRGATEWAY, CONFIG['username']
  deployment["nodes"].each do |node|
    session.use "root@#{node}"
  end
    url = 'http://public.grenoble.grid5000.fr/~tlenhan/TamlnChefScripts/'
    session.exec 'hostname'
    session.loop
    session.exec 'mkdir -p /tmp/chef-solo'
    session.loop
    session.exec 'wget ' url 'deployscript.json'
    session.loop
    session.exec 'chef-solo -j deployscript.json -r ' url 'cookbooks.tgz'
    session.loop
  end
```

4.1 Scenario Description

Our simple scenario deployment generates a VMI that includes selected software stacks in the previous example (Java, Tomcat, MySQL). We deploy this VMI to the reserved nodes on Grid5000. We compare our approach to the traditional approach in terms of time for setting up the environment, amount of data transfer through the network, and operating steps. We evaluate the traditional approach in two cases:

- Case 1: There is no existing VMI that fits the requirements. The cloud provider needs to create a new VMI containing Java, Tomcat and MySQL.
- Case 2: There is an existing VMI that fits the requirements. It is used as a standard VMI for deploying on the cloud nodes. However, for meeting different user requirements, it also contains software that may not be used: Java, Tomcat, MySQL, Apache2, Jetty, PHP5, Emacs, PostgreSQL, DB2-Express C, Jetty, LibreOffice, etc.

4.2 Traditional Approach vs Model-Driven Approach

Time and Operations of the Deployment In the traditional approach most decisions are taken by experts, because they require the knowledge of underlying systems and software dependencies. Our approach provides a graphical interface, the VMI Configuration Manager, which guides cloud users in the selection of a set of configuration options. After that, the Configuration Manager deploys the new VMI on cloud nodes, making easy to update and to maintain the running VMI. Table 1 shows a comparison between the traditional and the model-driven approaches, in terms of operations and deployment duration. Experiments show that the deployment duration of our approach is slightly better that the traditional approach, if there is an existing VMI that fits the requirements. However, if there is no appropriate VMI and the cloud provider creates a new one, then our approach is faster than the traditional approach.

Traditional Approach			**Model-driven Approach**		
Operations	*Handled by*	*Estimated Time*	*Operations*	*Handled by*	*Estimated Time*
1. Find the existing VMI in repository that fit the requirements. • If found: go to operation 3, • If not found: create a new one or modify an existing VMI	Expert (Manually)	1 minute	1. Create a VMI configuration & Generate a deployment script file	Expert & Non-expert (Manually)	2 minutes
2. Create a new VMI • Boot a clean VMI • Install and configure software • Save to an image	Expert (Manually)	11 minutes	2. Reserve 50 nodes & deploy the standard VMI to the nodes	Automatic	8 minutes
3. Reserve 50 cloud nodes & deploy the VMI to the nodes	Expert (Manually)	9 / 12 minutes (*)	3. Copy the deployment file to the running nodes & execute it to install, configure software	Automatic	2 minutes
Total time		21 / 13 minutes (**)	**Total time**		14 minutes

(*) : 9 minutes for case 1, and 12 minutes for case 2
(**): 21 minutes for case 1, and 13 minutes for case 2

Table 1. The Operations of Model-Driven Approach and Traditional Approach

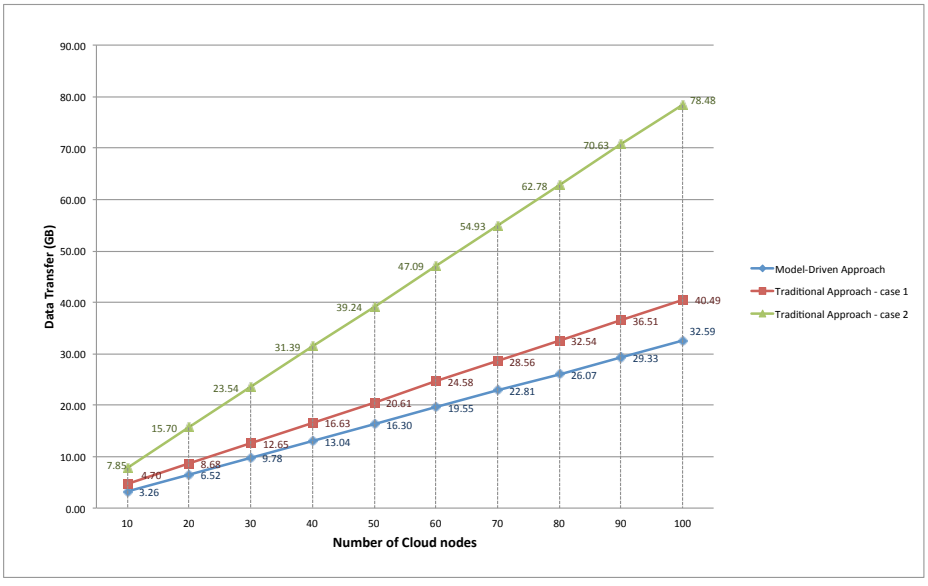

Fig. 7. Data Transfer Through the Network of the VMI Deployment

Data Transfer Through the Network In our experiment, we use a clean image **Squeeze-x64-nsf**[7] (333.587 MB), which is available on the Grid5000 's repository. This is also the standard VMI for the case 1 of the traditional approach. In our approach, we use minimal configuration images, only containing an installation software and its dependencies (i.e., Chef). **Comment**: *Begin-Edit* After the installation of the minimal software, the image size is 339.955 MB. In the case 2, unused software is installed for adapting different requirements from users. This makes the size of a standard VMI is much bigger, 803.60 MB. **Comment**: *End-Edit* Figure 7 shows that in both cases, the model-driven approach transfers less data than the traditional approach. Especially when the pre-packaged VMI contains more software installed, and deploy to a large number of cloud nodes. In this example, when we deploy 100 cloud nodes, the traditional approach transfers 40.49GB of data for case 1, and 78.48GB of data for case 2, while the model-driven approach only transfers 32.59GB of data. The traditional approach reduces the amount of data in 19.5% and 58.47%, comparing to cases 1 and 2, respectively.

5 Related Work

Our work is related to two areas: Configuration management and the deployment of VMI in cloud environment.

5.1 Virtual Machine Image Configuration Management

Some research efforts use feature models to capture configuration options of complex systems [8, 17]. Wenzel et al. [17] explain how feature models help to simplify the selection of configuration options. Similarly to our approach, the authors use feature models and cross cutting constraints for managing the configuration. This can reduce requirement elicitation errors and support automated choice propagation [10]. Nevertheless, they focus on creating the database of the configuration management system, while we used feature model to manage the configuration of VMIs for supporting the automatic deployment process in cloud computing.

Dougherty et al. [8] present a technique to minimize the number of idle VMs in an auto-scaling queue. The technique helps to reduce the energy consumption and the operating cost and satisfies the constraint of response time. Their work defines a method to represent VMI configuration options by using feature models with constraints in the form of Constraint Solving Problems (CSP). It uses an auto-scaling queue to store created images in idle status. This leads to an improvement of response time when the request matches the available image in the queue. T.Zhang et al. [18] present the concept of typical virtual appliances (TVA) and their management. A TVA contains popular software, and the system can provide a set of frequently used virtual appliances. It helps to minimize the

[7] https://www.grid5000.fr/mediawiki/index.php/Squeeze-x64-nfs-1.1

transformation time from an existing virtual appliance to a new one that fits the request. However, in both approaches, the composition of virtual image occurs at design time and at the administrator side, before the system copy and deploys it into cloud nodes. This makes it difficult to synchronize the maintenance and modification of the running images as needed when the amount of running cloud nodes is large. For example, upgrading the software version, or installing a new software package on the running virtual machines. Our approach composes VMI at runtime, when the standard VMI is deployed on cloud nodes. We put the configuration file into the running cloud nodes (they clone the standard VMI), and the installation and configuration occurs inside these nodes. Therefore, it is easy to maintain or modify the running images.

5.2 Virtual Machine Image Deployment

Konstantinou *et al.* [12] describe a model-driven engineering approach for virtual image deployment in virtualized environments. They focus on reusable virtual images and their composition. The authors introduce the concept of virtual solution models. This concept defines the solution as a composition of multiple configurable virtual images. The virtual solution model is an abstract deployment plan and it is platform-independent. According to the specific cloud platform, the model can be transformed into an executable deployment plan[9]. Chieu *et al.* [6, 7] and Arnold *et al.* [3] propose the use of virtual image templates. Their approaches describe a provisioning system that provide pre-installed virtual images according to the deployment scenario. M. Sethi *et al.* [15] present an approach for automated modification of dependency configuration in SOA deployment. In their work, the software stacks are installed and configured at deployment time, transferring smaller VMIs through the cloud network. Sun Microsystems [1] proposes an approach to deploy applications in cloud computing environment. Similarly to our approach, their approach uses shell-script files to execute on running cloud nodes at runtime. However, both approaches need experts on virtual image provisioning. By using feature models to represent the configuration options, our approach can support both experts and non-experts, who lack knowledge about virtual image provisioning and underlying software systems and dependencies. It can reduce errors and improve the consistency of configurations during the composing of VMIs.

6 Conclusion and Future Work

In this paper, we presented a model-driven approach to manage and create configurations, as well as deploy images for virtual machine image provisioning in Cloud Computing. We consider virtual images as product lines, use feature models to capture their configurations, and use model-based techniques for automatic deployment of virtual images. This approach makes the management of virtual image more flexible and easier to use than the traditional approach. On the implementation side, we developed a prototype for validating the approach. It

helps cloud users to select configuration options, to create virtual images and to deploy them on cloud nodes. We used Grid5000 as a Cloud Computing environment testbed for deploying virtual images.

The framework includes two major parts: the VMI Configuration Manager and the Execution Model. The VMI Configuration Manager helps cloud users to select configuration options, create a valid configuration of a VMI through a graphical user interface. It also generates deployment script files. The Execution Model uses these files to automatically deploy and configure software into cloud nodes at runtime without any manual intervention.

We compared our approach to the traditional cloud deployment approach in two different scenarios, using an existing compatible VMI and creating a new one. Experiments showed that the model-driven approach helps cloud users to create the configurations and deploy VMIs on demand easily. It minimizes error-prone manual operations. Additionally, our approach reduces the network data transfer, comparing to the traditional approach. Especially, if a pre-packed VMI contains unwanted software. In this case, experiments showed that our approach reduces the data transfer up to 58.47%. It saves network resources during VMIs provisioning in Cloud Computing.

Our framework could be extended to support cloud users for estimating the deployment time and operational costs as needed. Therefore, it could improve the performance of virtual machine image provisioning. However, the reasoning engine of our Product Derivation process is still limited with simple constraints of the configuration. It is a challenge to deal with more elaborated configurations that have optimal requirements on the complex constraints of multiple parameters. In the future, we plan to improve the reasoning engine of the Product Derivation process, to deal with more complex configuration options and constraints. We believe that a reasoning engine could enhance the performance of the Product Derivation process in the VMI configuration management. **Comment**: *BEGIN-EDIT* Currently, our current prototype only works in the Grid5000 environment. We are improving the prototype to have the ability to work with some open-source cloud platforms, such as OpenNebula, Nimbus, etc. **Comment**: *END-EDIT*

References

[1] Model-driven application deployment for cloud computing environments. White Paper, Sun Microsystem Inc. (January 2010), `http://www.techrepublic.com/whitepapers/model-driven-application-deployment-for-cloud-computing-environments/1829151`, available online (18 pages)

[2] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A Berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley (2009)

[3] Arnold, W., Eilam, T., Kalantar, M.H., Konstantinou, A.V., Totok, A.: Automatic realization of SOA deployment patterns in distributed environments. In:

Service-Oriented Computing - ICSOC 2008, 6th International Conference, Sydney, Australia, December 1-5, 2008. Proceedings. pp. 162–179 (2008)
[4] Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated reasoning on feature models. Lecture Notes in Computer Science 3520, 381–390 (2005)
[5] Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems 25(6), 599 – 616 (2009)
[6] Chieu, T., Mohindra, A., Karve, A., Segal, A.: Solution-based deployment of complex application services on a cloud. In: Service Operations and Logistics and Informatics (SOLI), 2010 IEEE International Conference on. pp. 282 –287 (july 2010)
[7] Chieu, T., Mohindra, A., Karve, A., Segal, A.: Dynamic scaling of web applications in a virtualized cloud computing environment. In: e-Business Engineering, 2009. ICEBE '09. IEEE International Conference on. pp. 281 –286 (oct 2009)
[8] Dougherty, B., White, J., Schmidt, D.C.: Model-driven auto-scaling of green cloud computing infrastructure. Future Generation Computer Systems 28(2), 371 – 378 (2012)
[9] Han, R., Guo, L., Guo, Y., He, S.: A deployment platform for dynamically scaling applications in the cloud. In: Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on. pp. 506 –510 (29 2011-dec 1 2011)
[10] Hubaux, A., Classen, A., Mendonca, M., Heymans, P.: A preliminary review on the application of feature diagrams in practice. In: 4th International Workshop on Variability Modelling of Software-Intensive Systems - VaMOS. Proceedings (2010)
[11] Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Tech. rep., Carnegie-Mellon University Software Engineering Institute (November 1990)
[12] Konstantinou, A.V., Eilam, T., Kalantar, M., Totok, A.A., Arnold, W., Snible, E.: An architecture for virtual solution composition and deployment in infrastructure clouds. In: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing. pp. 9–18. VTDC '09, ACM, New York, NY, USA (2009)
[13] Mell, P., Grance, T.: The nist definition of cloud computing. Tech. rep., National Institute of Standard and Technology - NIST (2011)
[14] Mendonça, M., Wasowski, A., Czarnecki, K.: Sat-based analysis of feature models is easy. In: 13th International Conference on Software Product Lines (SPLC 2009), San Francisco, CA, USA (2009)
[15] Sethi, M., Kannan, K., Sachindran, N., Gupta, M.: Rapid deployment of SOA solutions via automated image replication and reconfiguration. In: Services Computing, 2008. SCC '08. IEEE International Conference on. vol. 1, pp. 155 –162 (july 2008)
[16] Thüm, T., Batory, D.S., Kästner, C.: Reasoning about edits to feature models. In: ICSE. pp. 254–264. IEEE (2009)
[17] Wenzel, S., Berger, T., Riechert, T.: How to configure a configuration management system – an approach based on feature modeling. In: 1st International Workshop on Model-driven Approaches in Software Product Line Engineering (MAPLE 2009) at SPLC 2009. San Francisco, CA (Aug 2009)
[18] Zhang, T., Du, Z., Chen, Y., Ji, X., Wang, X.: Typical virtual appliances: An optimized mechanism for virtual appliances provisioning and management. Journal of Systems and Software 84(3), 377 – 387 (2011)

[19] Ziadi, T., Jézéquel, J.M.: Software product line engineering with the UML: Deriving products. In: Käkölä, T., Dueñas, J.C. (eds.) Software Product Lines, pp. 557–588. Springer (2006)