A Model-Based Approach for Optimizing Power Consumption of IaaS

Tam Le Nhan, Gerson Sunyé, Jean-Marc Jézéquel

▶ **To cite this version:**

Tam Le Nhan, Gerson Sunyé, Jean-Marc Jézéquel. A Model-Based Approach for Optimizing Power Consumption of IaaS. IEEE NCCA 2012 IEEE Second Symposium on Network Cloud Computing and Applications, Dec 2012, London, United Kingdom. pp.978-0-7695-4943-9/12, 10.1109/NCCA.2012.22 . hal-00796221

HAL Id: hal-00796221

https://inria.hal.science/hal-00796221

Submitted on 4 Mar 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Model-Based Approach for Optimizing Power Consumption of IaaS

Tam Le Nhan*, Gerson Sunyé† and Jean-Marc Jézéquel*

* *INRIA – University of Rennes 1*
Rennes, France
Email: tam.le_nhan@inria.fr, jezequel@irisa.fr

† *INRIA – University of Nantes*
Nantes, France
Email: gerson.sunye@univ-nantes.fr

***Abstract*—Virtual Machine Image (VMI) provisioning is an important process of Infrastructure as a Service delivery model to provide virtual images in Cloud Computing. The power consumption and energy efficiency of VMI provisioning process depend not only on the hardware infrastructure, but also on the VMI 's configuration, which helps to compose, configure and deploy VMIs in Cloud Computing environments. The major issue of improving the energy efficiency of VMI provisioning process is how to reduce the power consumption while ensuring the compatibility of software components installed in a virtual machine image. This paper describes a model-driven approach to improve the energy efficiency of VMI provisioning in Cloud Computing. This approach considers virtual images as product lines and uses feature models to represent their configurations. It uses model-based techniques to handle VMI specialization, automatic deployment and reconfiguration. The approach aims at minimizing the amount of unneeded software installed in VMIs, and thus to reduce the power consumption of VMI provisioning as well as the data transfer through the network.**

***Keywords*-Model-driven deployment, feature models, cloud computing, virtual image provisioning, power consumption.**

I. Introduction

Cloud Computing [1], [2] is a new architecture which provides dynamically scalable and virtualized resources as services over the Internet. The key feature of Cloud Computing is the virtualization in which all cloud resources become transparent to the user. The virtualization in Cloud Computing combines a set of virtual machine images (VMIs) on top of physical machines. Each virtual image hosts a complete software stack: it includes operating system, middleware, database, development applications, etc. The deployment of a VMI typically involves booting the image, as well as installing and configuring some software packages. Reducing the power consumption of these virtual images is a critical factor to improve the energy efficiency of Cloud systems. In the traditional approach, the creation of a VMI to fit user requirements and its deployment on a 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. Some of these components rarely get used and thus the image and its resource usages are typically larger than necessary. 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 no match is found, then a new one is created and configured to match the request. This traditional approach faces several difficulties that lead to inefficient energy consumption, such as wastage of storage space, memory, CPU usage, operating costs, and waste of network bandwidth when cloning an image and deploying it on the cloud nodes [3]. Therefore, several concerns need to be addressed by the cloud providers to improve the power consumption efficiency: (i) How to create a configuration that minimizes the power consumption of VMIs? (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 that one?

In this paper, we improve our previous work of using a Model-Driven Engineering (MDE) approach for managing VMI in Cloud Computing environments [4]. We extend the feature model to handle features with attributes (e.g. installation time, size of software, etc.) and enhance the reasoning process for finding the optimal configurations of VMI. By adding the attributes to the features, we can evaluate and find the optimal configuration according to each criterion, such as minimum installation time or VMI size. Especially, applying the model-driven approach to reduce the power consumption and adapt to the needs of auto-scaling and self-configuring virtual machine images. In our MDE 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 show that our approach makes the VMIs use less power at runtime, and also significantly reduces data transfer across the network, which is another source of power savings.

We validate this approach with 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. We briefly describe the related backgrounds of Cloud Computing and

Model-based approach in Section II. Section III 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 IV introduces an example about deploying a Java web application development platform and shows the experimental evaluation of our approach. Finally, Section V discusses the related work, and is followed by the conclusion and future work in Section VI.

II. Background

A. Cloud Computing

Cloud Computing services are classified according to the service delivery model of cloud providers. There are three major Cloud Computing service layers [5]: Infrastructure as a Service, Platform as a Service, and Software as a Service.

- **Infrastructure as a Service (IaaS)**: provides a virtualized infrastructure to the user, comprising storage, network, processing, and other computing resources. Cloud users can deploy and run software, which include operating system and applications.
- **Platform as a Service (PaaS)**: offers development platforms where the developers can design, develop, deploy and test their applications. Several IT vendors have developed new PaaS systems.
- **Software as a Service (SaaS)**: provides services or composition of services to users. These services run on a cloud infrastructure and are accessible from various client devices through a client interface such as Web browsers (e.g,. Web Mail, Google Docs.) or client applications (e.g., iTunes, Picasa).

B. Model-Driven Engineering

Model-driven engineering refers to the systematic use of models as primary engineering artifacts throughout the engineering lifecycle. Models are considered as first class entities. MDE can be applied to software, system, and data engineering. It offers a promising approach to address the inability of third generation languages to reduce the complexity of platforms and express domain concepts effectively [6]. Applying MDE approach for the deployment process helps to encapsulate the deployment and management of a cloud system into a series of procedural operations.

C. Feature Modeling

In software development, feature models [7] represent all products of a Software Product Line (SPL). It supports for specifying and reasoning about the commonality and variability of a product line [8]. 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.

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.

III. Model-Based 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.

A. Feature Modeling for VMI Configuration Management

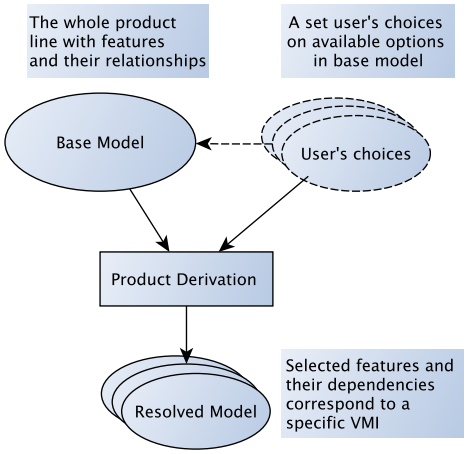

Figure 1. Feature Modeling Approach

Our approach uses feature modeling 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 specify how the choice of a given component restricts the choice of other components, in the same Virtual Machine Image.

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

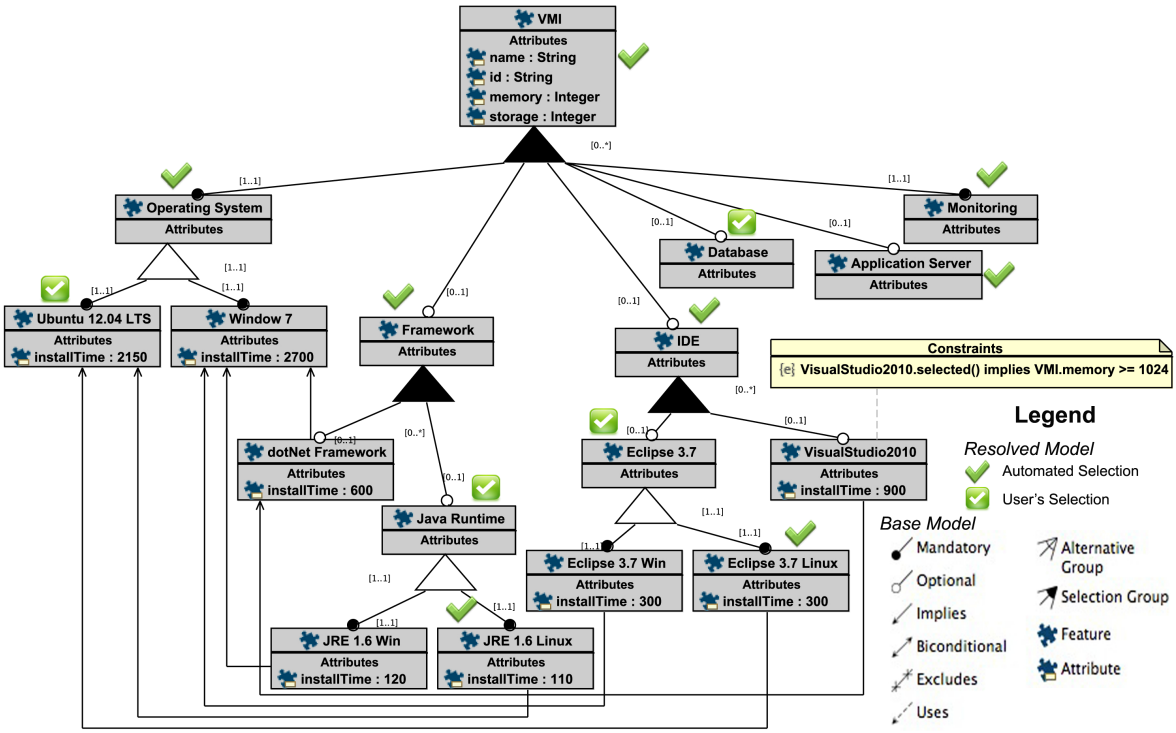

Figure 2. Feature Diagram represents a Base Model with user's choices and auto-selected features

process, it contains selected features and their dependencies.

1) 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 12.04 LTS≫ must be selected.
- If the feature ≪Ubuntu 12.04 LTS≫ is selected, then features that require ≪Windows 7≫ cannot be selected, for instance: ≪VisualStudio2010≫, ≪JRE 1.6 Windows≫, etc.

We can classify the features in a base model into two types: *category feature* and *package feature*. A *category feature* classify features, similarly to a folder in file systems. A *package feature* corresponds to a software package which used for installation, similarly to a file in file systems. It keeps detail information of a software package, such as: installation time, size of packages, etc. For example in Figure 2, ≪Operating System≫, ≪IDE≫, ≪Database≫ are category features, while ≪Eclipse 3.7 Linux≫, ≪VisualStudio2010≫, ≪JRE 1.6 Win≫ are package features.

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 models relies on the correctness of the feature model that represents them. Many approaches and tools were proposed to automate analysis of feature models [9], [10], [11]. 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.

2) Product Derivation: Product Derivation is a process that is responsible for the creation of the final configuration. It supports the derivation of VMI configurations from the base model [12]. 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 that the selection is valid. When a feature is selected, the Product Derivation process checks its relationships and constraints. Constraints are defined by using Object Constraint

Language (OCL)[1]. For example, in Figure 2, there is a constraint of the selection of feature ≪VisualStudio2010≫: "Visual Studio 2010 needs at least 1GB RAM". It is defined in OCL as following: $VisualStudio2010.selected()$ $implies\ VMI.memory \geq 1024$.

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. Sometimes, a cloud user does not have any specific requirement with respect to given a feature selection. In these cases, the Product Derivation process needs a mechanism to generate an optimal solution according to some criteria, such as VMI size, installation time, etc. If the cloud provider wants to focus on the response time, then the Product Derivation process will find a solution based on the installation time of software packages. Algorithm 1 presents the Product Derivation process to select automatically the best solution in terms of installation time. It is a recursive procedure that examines the total installation time of the candidate features and their dependencies. If a user selects a category feature, and this feature has more than one child feature, which are the candidates for the selection, then the Product Derivation process examines and selects the child feature that has a minimum installation time.

Algorithm 1 Algorithm to select a feature and its dependencies with minimum installation time

INPUT: F is a feature, and S is a set of selected features at the current step

OUTPUT: minimum installation time if the feature F is selected, and X is a set of selected features(includes and F's dependent features).

SELECTFEATURE(F) // return an integer value - a minimum installation time when F is selected

```
time = 0
X is empty
if F has no child features then
    time = time + F.installTime;
    FR is a set of features which required by F
    for all FR_i in FR do
        if FR_i is not in S and X then
            time = time + SELECTFEATURE(FR_i); and
            add FR_i to X
        end if
    end for
else
    FC is a set of available child features of F
    find a feature FC_i where SELECTFEATURE(FC_i) =
    Min(SELECTFEATURE(FC_j)) with all FC_j in FC
    time = time + SELECTFEATURE(FC_i); and add
    FC_i to X
end if
add X to S
return time
```

[1]http://www.omg.org/spec/OCL/

3) 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. The Product Derivation process is responsible to validate user selections and auto-select dependencies based on the feature representation in the base model. A resolved model corresponds to a specific configuration of a Virtual Machine Image. Figure 2 presents an example of a resolved model that is derived from the base model. It represents the user 's selections: operating system is ≪Ubuntu 12.04 LTS≫, integrated development environment is ≪Eclipse 3.7≫, and ≪Apache Tomcat 5.5≫ for application server. According to the base model, the dependent features of the user 's selections are also selected by the Product Derivation process (e.g. ≪JRE 1.6 Linux≫, ≪IDE≫, ≪Monitoring≫, etc.)

B. Model-Based Deployment Architecture

Unlike the traditional approach, where software packages are installed and configured together when the VMI template is created, the model-driven deployment approach installs and configures software packages at deployment time, when a template VMI is booted. The approach also supports synchronization of maintenance of the deployed VMIs at runtime.

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 3 gives an overview of our approach.

- **VMI Repository**
  The VMI Repository contains basic virtual machine images that are used as the initial VMIs. 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 virtual machine image configurations 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 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 configuration that resulted 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

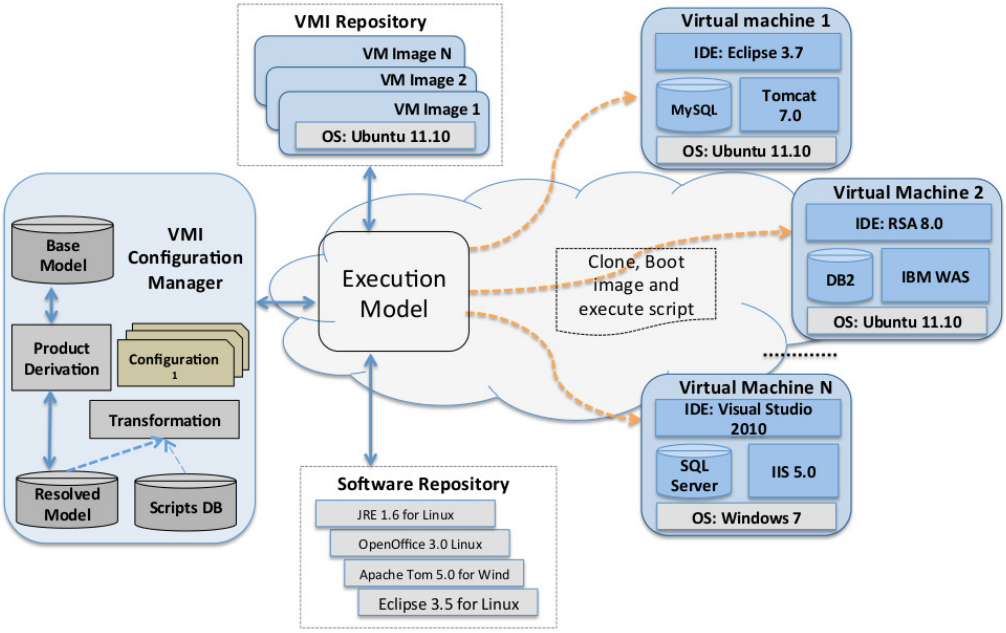

Figure 3. An Overall Architecture of Model-Driven Approach for VMI Deployment

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.

C. 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:

1) **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 selections and generates a resolved model (i.e., a valid configuration of a VMI).
2) **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[2] 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[3] (JSON) source code.
3) **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.

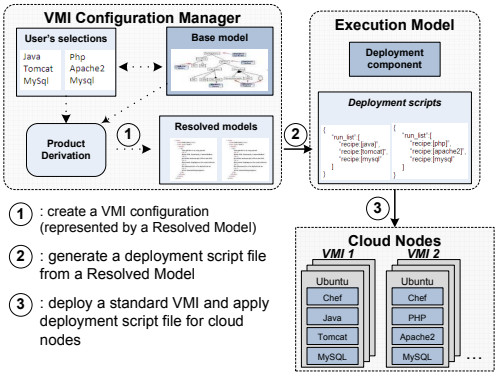

Figure 4. Model-Driven Deployment Process

IV. Experimental Evaluation

In this section, we present an experimental evaluation of the performance of our approach, in terms of data transfer and power consumption of virtual image deployment. Experiments concerning the evaluation of the deployment time were presented in previous work [4]. The experiment is executed on Grid5000[4], a virtualization infrastructure for research in France. We use Grid5000 's tools to reserve nodes and deploy VMIs to the nodes.

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

[2]http://wiki.opscode.com/display/chef/About
[3]http://www.json.org/
[4]https://www.grid5000.fr/mediawiki/index.php

management system, and a programming language compiler.

Cloud users use the VMI Configuration Manager to select the required features on the base model, for example: **Ubuntu**, **Eclipse**, **Apache Tomcat**, and **Database**.

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 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. There is more than one option for database because the cloud user does not have any specific requirement of a database implementation. The Base model proposes either MySQL or DB2 Express-C. However, the Product Derivation process selects MySQL because of its installation time is smaller than the one of DB2 Express-C.

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. 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.

A. Data Transfer Through the Network

In terms of power consumption, the amount of data transfer through the network is not as important as I/O or CPU usage. However, it is a factor that needs to be considered for reducing power consumption, since transferring more data implies consuming more network bandwidth [13]. This means that the network equipments (e.g. switches, routers.) consume more power. Reducing the VMI size help to reduce the amount of data transfer through the network. It make to decrease the energy consumption of the transferring virtual machines. We evaluated the amount of data transfer through the network of VMI provisioning process in our previous work [4]. The scenario describes the generation of a VMI that includes selected software stacks in the preceding example (Java, Tomcat, MySQL), and the deployment of this VMI to the Grid5000 reserved nodes. We compare our approach to the traditional approach in terms of amount of data transfer through the network, and power consumption of virtual machine images. 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.

The evaluation results show that the model-driven approach transfers 58.47% less data than the traditional approach [4]. This means that the power consumption of network equipments (e.g. switches, routers, etc.) can be reduced.

B. Power Consumption Comparison

We consider the power consumption of the model-driven approach and the traditional approach in two ways: First, power measurement from inside the running VMs and then on the the cloud nodes which host the running VMIs. In our approach, we create and deploy specific VMIs, as in the above example, while the images used for the traditional approach contain the unneeded software. This software is also booted and executed when the VMI is running, meaning that the unneeded software use computing resources (e.g. CPU, RAM.). We simulate programmers doing Java web application programming, showing how the computing resources are used at runtime. In this simulation, we use a script to auto re-compile a Java program and update it to the Tomcat web server. The small Java program emulates some complex deterministic computation by generating the *shortest addition chains*[5] of a number N by a recursive method and writes the results to a MySQL database; the results can be displayed on a JSP web page which is executed by Apache Tomcat web server. We run this program several times with different values for N.

1) Power Measurement from Inside the Running VMIS: To estimate the power consumption from inside the running VMIs, we deploy these virtual images on the same node of *Graphene* cluster on the Nancy side of Grid5000. In this experiment, we consider two aspects: First, we access to the power monitoring hardware installed on this cluster to get information on PDUs (Power Distribution Unit) and then we use a resource statistic tool – Dstat[6] which captures information of CPU usage percentage. According to the SNMP MIB[7], description the *outletWatts* give a unique value (in Watts) for the active power sensor attached to the outlet [14]. We consider the power consumption of the running VMIs in 1h with the interval request of 10 seconds. We run this scenario five times and get the mean values of power consumption and CPU usage. After that, we measure the average power consumption of the virtual image in the traditional approach is of **87.43** Watts and this value in the MDE approach is **78.25** Watts while the average CPU usages is **49.93%** and **42.71%** for the traditional approach and MDE approach, respectively.

2) Power Measurement of Cloud Nodes: We deploy virtual images on the same node *sagittaire-53.lyon.grid5000.fr* of the cluster *sagittaire* on the Lyon

[5]http://wwwhomes.uni-bielefeld.de/achim/addition_chain.html
[6]http://dag.wieers.com/home-made/dstat/
[7]http://www.net-snmp.org/

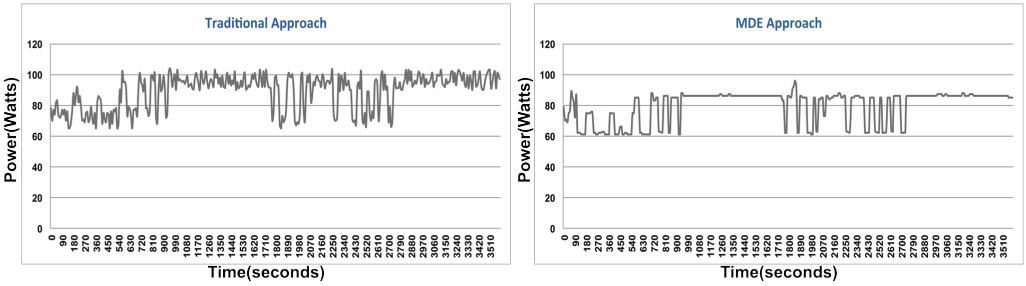

Figure 5. Power Measurement from inside the VMIs

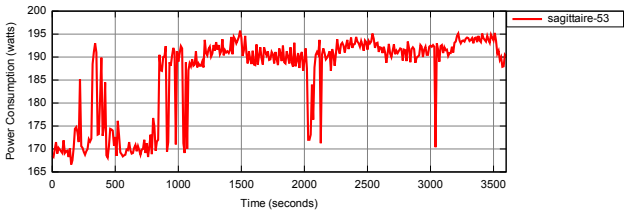

Figure 6. Power Measurement of a VMI by the Traditional Approach

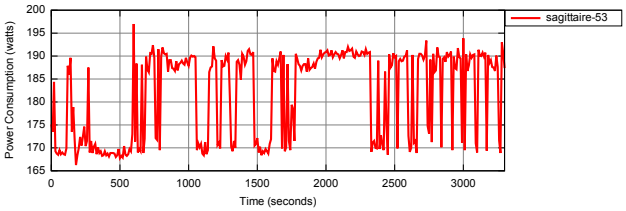

Figure 7. Power Measurement of a VMI by the Model-Driven Approach

site of the Grid5000. To estimate the power consumption of the cloud nodes, we use a live monitoring tool developed by the Green-Net[8] research team. This tool uses watt-meters provided by the SM Omegawatt[9] and it helps to measure the electrical consumption of all nodes on Grid5000 's Lyon site in a real-time manner [15]. Figure 7 and Figure 6 show the visualization of power consumption of both approaches model-driven and traditional. The experiment results show that the virtual image created by the traditional approach consumes **187.3** Watt-hours, while the image deployed by the model-driven approach consumes **179.6** Watt-hours.

From the above experiment, we can see that the difference of power consumption of two approaches is not much when we consider only one cloud node within 1h. However, it is much different when we consider on the private cloud system with a large number of nodes running during a long time. Let us think about an example of software companies, they have their own private clouds and provide a platform as a service for hundreds developers working for several months according to their projects. If they have 50 developers working during six months in the above scenario, the model-driven approach helps to reduce from **1.66** to **1.98** Megawatts comparing to the traditional approach. Therefore, the companies can save a considerable amount of electrical expense.

[8]http://www.ens-lyon.fr/LIP/RESO/Projects/green-net/
[9]http://omegawatt.fr/gb/index.php

The above two comparisons give different results. This, because the first one is a measurement the power consumption of software running from inside the VMI, while the second one is a measurement of a cluster node that hosts the running VMI. Therefore, it is also influenced by other software used to monitor the running VMI.

V. Related Work

Konstantinou *et al.* [16] 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. Chieu *et al.* [17], [18] and Arnold *et al.* [19] 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.* [20] 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 [3] 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.

Some other research efforts use feature models to capture configuration options of complex systems [21], [22]. Wenzel et al. [22] 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 [23]. Nevertheless, they focus on creating the database of the configuration management

system, while we use feature model to manage the configuration of VMIs for supporting the automatic deployment process in cloud computing.

Dougherty et al. [21] 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. [24] 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 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.
Power measurement of VMIs is also considered in [13], [25], [26]. Their work mainly focus on the analysis and the profiling the power consumption of VMIs. It reduces the power consumption of virtualized systems on the VMI scheduling and migrating process instead of provisioning process.

VI. 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 and minimizes error-prone manual operations. It reduces the network data transfer when deploying the VMIs, and the power consumption of VMIs at runtime, 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**% [4], it uses fewer network resources during VMIs provisioning in Cloud Computing. Especially, our approach saves around **8** Watt-hours per cloud node. It is not too much, however, when the number of cloud nodes is considerable and the system runs in a long time, then the amount of saved power consumption would be significant.

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.

Acknowledgment

Experiments presented in this paper were carried out using the Grid5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

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

[2] P. Mell and T. Grance, "The nist definition of cloud computing," National Institute of Standard and Technology - NIST, Tech. Rep., 2011.

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

[4] L. N. Tam, G. Sunyé, and J.-M. Jézéquel, "A model-driven approach for virtual machine image provisioning in cloud computing," in *European Conference on Service-Oriented and Cloud Computing*, ser. Lecture Notes in Computer Science, vol. 7592, 2012, pp. 107–121.

[5] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 50–55, December 2008.

[6] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon University Software Engineering Institute, Tech. Rep., November 1990.

[8] M. Acher, "Managing multiple feature models: Foundations, language and applications," Ph.D. dissertation, Université de Nice–Sophia Antipolis, 2011.

[9] T. Thüm, D. S. Batory, and C. Kästner, "Reasoning about edits to feature models," in *ICSE*. IEEE, 2009, pp. 254–264.

[10] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, "Automated reasoning on feature models," *Lecture Notes in Computer Science*, vol. 3520, pp. 381–390, 2005.

[11] M. Mendonça, A. Wasowski, and K. Czarnecki, "Sat-based analysis of feature models is easy," in *13th International Conference on Software Product Lines (SPLC 2009), San Francisco, CA, USA*, 2009.

[12] T. Ziadi and J.-M. Jézéquel, "Software product line engineering with the UML: Deriving products," in *Software Product Lines*, T. Käkölä and J. C. Dueñas, Eds. Springer, 2006, pp. 557–588.

[13] A. J. Younge, G. von Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers, "Efficient resource management for cloud computing environments," *International Conference on Green Computing*, vol. 0, pp. 357–364, 2010.

[14] Power measurement in Nancy. [Online]. Available: http://www.grid5000.fr/mediawiki/index.php/Power_Measurements_in_Nancy

[15] M. Dias De Assuncao, J.-P. Gelas, L. Lefèvre, and A.-C. Orgerie, "The Green Grid5000: Instrumenting a Grid with Energy Sensors," in *INGRID'2010 : 5th International Workshop on Distributed Cooperative Laboratories: Instrumenting the Grid*. Springer, 2012, pp. 25–42.

[16] A. V. Konstantinou, T. Eilam, M. Kalantar, A. A. Totok, W. Arnold, and E. Snible, "An architecture for virtual solution composition and deployment in infrastructure clouds," in *Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, ser. VTDC '09. New York, NY, USA: ACM, 2009, pp. 9–18.

[17] T. Chieu, A. Mohindra, A. Karve, and A. Segal, "Solution-based deployment of complex application services on a cloud," in *Service Operations and Logistics and Informatics (SOLI), 2010 IEEE International Conference on*, july 2010, pp. 282 –287.

[18] T. Chieu, M. A., A. Karve, and A. Sega, "Dynamic scaling of web applications in a virtualized cloud computing environment," in *e-Business Engineering, 2009. ICEBE '09. IEEE International Conference on*, oct. 2009, pp. 281 –286.

[19] W. Arnold, T. Eilam, M. H. Kalantar, A. V. Konstantinou, and A. Totok, "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*, 2008, pp. 162–179.

[20] M. Sethi, K. Kannan, N. Sachindran, and M. Gupta, "Rapid deployment of SOA solutions via automated image replication and reconfiguration," in *Services Computing, 2008. SCC '08. IEEE International Conference on*, vol. 1, july 2008, pp. 155 –162.

[21] B. Dougherty, J. White, and D. C. Schmidt, "Model-driven auto-scaling of green cloud computing infrastructure," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 371 – 378, 2012.

[22] S. Wenzel, T. Berger, and T. Riechert, "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.

[23] A. Hubaux, A. Classen, M. Mendonca, and P. Heymans, "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.

[24] T. Zhang, Z. Du, Y. Chen, X. Ji, and X. Wang, "Typical virtual appliances: An optimized mechanism for virtual appliances provisioning and management," *Journal of Systems and Software*, vol. 84, no. 3, pp. 377 – 387, 2011.

[25] Q. Huang, F. Gao, R. Wang, and Z. Qi, "Power consumption of virtual machine live migration in clouds," in *Communications and Mobile Computing (CMC), 2011 Third International Conference on*, april 2011, pp. 122 – 125.

[26] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual machine power metering and provisioning," in *Proceedings of the 1st ACM symposium on Cloud computing*, ser. SoCC '10. New York, USA: ACM, 2010, pp. 39–50.