Migrating to the Cloud: a Software Product Line based analysis

Jesús García-Galán[1], Omer Rana[2], Pablo Trinidad[1] and Antonio Ruiz-Cortés[1]
[1]*ETS Ingeniería Informática, Universidad de Sevilla, Seville, Spain*
[2]*School of Computer Science & Informatics, Cardiff University, Cardiff, UK*
jegalan@us.es, o.f.rana@cs.cardiff.ac.uk, ptrinidad@us.es, aruiz@us.es

Keywords: Cloud: IaaS: Variability Management: Feature Model: Automated Analysis: AWS: Modelling

Abstract: Identifying which part of a local system should be migrated to a public Cloud environment is often a difficult and error prone process. With the significant (and increasing) number of commercial Cloud providers, choosing a provider whose capability best meets requirements is also often difficult. Most Cloud service providers offer large amounts of configurable resources, which can be combined in a number of different ways. In the case of small and medium companies, finding a suitable configuration with the minimum cost is often an essential requirement to migrate, or even to initiate the decision process for migration. We interpret this need as a problem associated with variability management and analysis. Variability techniques and models deal with large configuration spaces, and have been proposed previously to support configuration processes in industrial cases. Furthermore, this is a mature field which has a large catalog of analysis operations to extract valuable information in an automated way. Some of these operations can be used and tailored for Cloud environments. We focus in this work on Amazon Cloud services, primarily due to the large number of possible configurations available by this service provider and its popularity. Our approach can also be adapted to other providers offering similar capabilities.

1 INTRODUCTION

Infrastructure as a Service (IaaS) enables the dynamic provisioning of computational & data resources (often on-demand), and as an alternative to private and expensive data centers. Recently, a number of companies are deciding whether to migrate their internal systems to a Cloud environment, or to utilise Cloud-based infrastructure directly for building and deploying new systems. Among other benefits, IaaS reduces costs (for short term workloads), speeds up the start-up process for many companies, and decreases resource and power consumption. However, there are a number of providers currently available – Cloud Harmony (clo,) identifies over 100 public Cloud providers currently on the market. As each user/company intending to make use of Cloud computing infrastructure is likely to have their own (specific) requirements, a process is necessary to identify the most relevant provider and subsequently a suitable configuration that must be used on a particular provider. In this work we focus on the second of these requirements – i.e. better understanding whether a requirement of a user/company can be met through the offerings of a particular provider.

IaaS configuration process is often challenging – due to the variety of possible options that many Cloud providers offer (at different prices). Cloud providers offer highly configurable IaaS capabilities, like computing instance size, operating system, database type or storage features (and recently, availability of specialist accelerators). Therefore, users have to understand and navigate through a very large configuration space to identify whether their particular requirements are likely to be met. For instance, *Amazon Web Services (AWS)* provides, just for computing services (EC2), 1758 different configurations [1]. Identifying compatibility within such a large space of possible options is a tedious and error-prone task. Although online configuration and cost estimation tools are provided – such as the Amazon.com calculator (ama,), they do not let the users specify their preferences to find a suitable configuration. Often within such systems a user needs to look through the various possible offerings and assess whether the capability is suitable (or not) for their own requirements. Other more user friendly approaches are also available, such as (Khajeh-Hosseini et al., 2011) – which make use

[1]Additional information at https://dl.dropbox.com/u/1019151/addinf.pdf

of a spreadsheet for assessing suitability of a particular provider and possible configuration options. Such an approach relies on a dialogue between the various individuals within a company responsible for deciding what and when to migrate to the Cloud, involving managers, system administrators, etc. The assessment outcome is however made manually based on the outcome of the people-based interaction.

We consider an alternative approach in this work, based on the observation that management of large configuration spaces is a common task in the domain of the *Software Product Lines (SPLs)*. SPLs are families of related software products (or configurations), which share a set of common *assets*. Variability models, and specifically *Feature Models (FMs)*, are an extended way to represent commonalities and differences of SPLs in terms of functional features. They can also include additional information, represented as a set of attributes, and identified as being an *Extended Feature Model (EFM)*. In addition to representing variabilities, FMs also contain valuable information about the configuration space they depict, e.g., the set of configurations they represent or the optimal configuration given certain criteria. This information extraction has been defined and organized by means of a catalog of analysis operations through an *Automated Analysis of Feature Models (AAFM)*.

We propose applying Software Product Line-based techniques to the IaaS configuration process, in order to overcome some of the challenges associated with determining whether a particular Cloud provider is able to meet a customer requirement. We model several services of AWS as an FM, focusing on this particular provider due to the widest range of possible configuration options that are available in AWS, and due to the largest user community (of any other Cloud provider) using these services. We have also identified analysis operations of the AAFM to automate the search of suitable AWS configurations. Although works like (Dougherty et al., 2012) or (Schroeter et al., 2012) propose to model Cloud scenarios as FMs, this is the first work to use AAFM to support a user-driven IaaS configuration process. A prototype, the associated analysis operations, a case study and the outcome of analysis are presented in this paper.

Approaching the IaaS configuration process from a SPL perspective has several benefits. The use of EFMs provides us with a compact and easy to manage representation of the whole configuration space. It also allows the user to choose by means of abstract features and non-functional attributes, instead of selecting specific AWS features. For instance, a user would only need to specify a Linux-based operating system and does not need to identify a particular type (i.e. if this is the only requirement he has). Currently, the user would need to identify a particular type, such as Suse or RedHat, for instance. Moreover, the analysis operations of AAFM gives us support to validate user choices and to determine suitable configuration options. Alternatives to FMs also exist, such as the use of spreadsheets (Khajeh-Hosseini et al., 2012), a relational database or the development of a model in the Unified Modelling Language (UML). Both SQL queries and spreadsheets enable the representation of costs depending on the required features. They are a similar approach to the Amazon calculator (ama,). Given a configuration, we could identify the associated cost and other related information. However, the opposite (and more useful) process is not possible, i.e. given constraints about functional features and non-functional properties (based on the requirements of a user), obtaining suitable configurations given the feature set and configuration options available. UML could be an alternative to model the AWS scenario, using class diagrams. However, UML is not oriented to variability modelling and information extraction, and also lacks analysis operations such as those provided in AAFM.

The rest of the paper is structured as follows: Section 2 briefly describes the various Cloud services that Amazon provides, while in Section 3 we present these services modelled as a FM. Section 4 describes the process we propose to extract information from the AWS model. A study case is presented in Section 5 to demonstrate how our proposed approach can be used in practice. Related work is described in Section 6, and Section 7 concludes the paper with a discussion of lessons learned and possible future work.

1.1 SPL background

SPLs (Clements and Northrop, 2002) are a software engineering and development paradigm focused on re-utilization and cost optimisation of software products. In essence, a SPL is a family of related software products which share a set of common assets (capabilities), but each of which can also include a number of variations. It may be viewed as a way to achieve mass customization, the next step after mass production in software industry. SPLs have been applied to multiple industrial experiences and research (Rabiser et al., 2010), including Cloud environments (Dougherty et al., 2012). Assets, also named features, of a SPL are represented using variability models. The most common variability model used in SPL is the FM, proposed in 1990 by K.Kang (Kang et al., 1990). A FM is a tree-like data structure in which

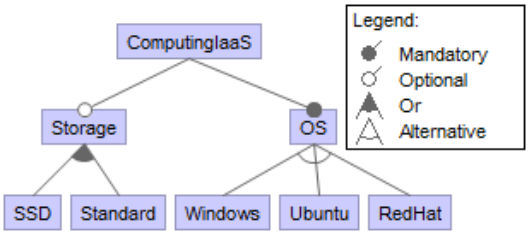

Figure 1: Example of a FM

each node represents a feature. All nodes taken together represent all the possible types of products (also named configurations or variants) of the SPL. However, not every feature has to correspond with a specific functionality. There may be abstract features (Thum et al., 2011), which represent domain decisions, but not concrete functionalities made available within a particular product. A FM also may contain attributes linked to features, representing non-functional properties. Figure 1 is a FM that represents general features of an IaaS provider. `ComputingIaaS` is the root node, which has two children, an optional feature (white circle) named `Storage`, and a mandatory feature (filled/black circle) named `OS`. Both features present group relationships, but `Storage` has an *Or* relationship, where one or more features should be selected, while `OS` has an *Alternative* relationship, where exactly one child must be selected.

FMs contain valuable information about the configurations and the whole product line. From the FM of a SPL we can deduce a number of possible outcomes, such as the total list of possible products, the set of common features of every product, the set of products that meet a given criteria and the product with the minimum associated cost. However, analysing the FM manually is a tedious and error-prone task, and for computers can become computationally intractable in case of large sized FMs. Hence, various research has focused on the AAFM (Benavides et al., 2005) (Benavides et al., 2010), using modelling and constraint programming techniques. Various analysis operations have been proposed and implemented since 2005 (Benavides et al., 2010), several of them usable within our approach.

2 AMAZON WEB SERVICES

Amazon Web Services (AWS) [2] provides a number of capabilities, such as the ability to execute and store software/data in the Amazon Cloud, and on pay-per-use basis. Amazon generally provides a per instance price and has recently also started to offer a "spot" price option (to make better use of spare capacity). Although in the last few years the number and types of Cloud service providers has increased (Rackspace/OpenStack, Flexiant, Microsoft Azure are good examples), AWS still has the largest, most configurable & mature services available on the market – especially for IaaS Clouds. For this reason, several PaaS and SaaS providers, like Heroku [3] or Netflix [4], run over AWS. Among others, AWS provides services for computation, storage, databases, clusters or content delivery. Moreover, users can choose options like datacenter location, resource reservation or managed support.

In this paper, we have focused on four of the most well-known Amazon services: EC2 (Elastic Compute Cloud), EBS (Elastic Block Storage) which makes use of EC2 instances, S3 (Simple Storage Service) for common storage, and RDS (Relational Database Service). EC2 provides resizable computing capacity, available at different *instance* sizes. Instances can be reserved, or run on demand, and several versions of Windows and Linux are available as OS. Instance size can vary in granularity from small to extra large. Additionally, there are instances for special needs, which have boosted CPU, RAM, or IO performance, and even clusters of instances. Options like detailed monitoring of instances, data center location or load balancing are also available. EBS provides storage linked to the EC2 instances. Furthermore, the user can also configure the storage as on-demand or provisioning IOPS (Input Output Operations per Second). For simple storage, we can configure S3, which presents two types of storage: standard (more durable) or reduced redundancy (less durable than standard). Finally, RDS provides different DB instances, in a similar way to EC2 providing compute/CPU instances. We can choose between different DB engines, like Oracle, SQLServer or MySQL; instances sizes, from a small standard instance to a high memory quadruple extra large instance; deployment type, between single or multi-availability zone deployment; configurable storage and reserved instances.

The two main concerns a user has when relying on IaaS in general and AWS in particular are cost and risk of failure. Cost is easier to measure and control. All the configurable options have an impact on the cost. That impact could be looked up at the AWS website [5]. However, risk is harder to measure and control. AWS provides a Service Level Agreement for some of its services, which guarantees a monthly

[2] http://aws.amazon.com/products/

[3] www.heroku.com

[4] www.netflix.com

[5] http://aws.amazon.com/pricing/

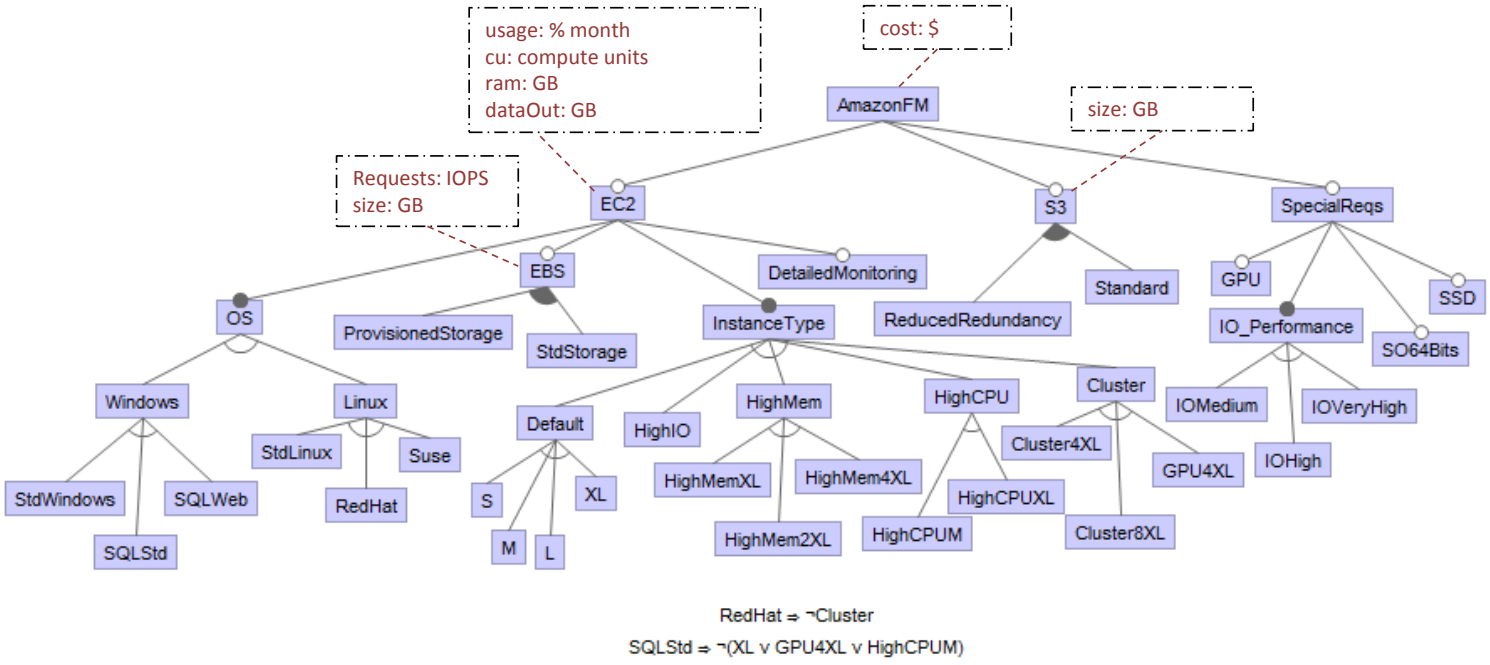

Figure 2: AWS Extended Feature Model: EC2, EBS and S3

uptime percentage and penalties in case these uptime guarantees cannot be met. For detailed information about the availability, a website [6] with the current and historical status of services is provided by AWS.

3 MODELLING AMAZON WEB SERVICES

In this work, we have modelled Amazon EC2, EBS, S3 and RDS as an EFM, as Figure 2 and Figure 3 show. A set of configurable aspects of the previous services is depicted at Figure 4. Functional aspects, like OS or instance type, have been modelled as features, while non-functional aspects, such as cost or usage, have been modelled as attributes. Both features and attributes present constraints to model the real behaviour of AWS. However, just some of them have been represented in the figures to make these figures readable. A working version of the AWS EFM is presented, in FaMa notation format, in Section 4.1.

The EC2 EFM is shown in Figure 2. Two main sets of features, *instance type* and *OS* represent most of the variability in EC2. The OS is composed by Windows and Linux variants, while instance types contain features such as standard, high mem, high CPU, high IO and cluster instances. Both instance type and OS feature groups are modelled as alternative features, i.e. only one feature within this group must be selected. Detailed monitoring is also present as an optional feature. EC2 feature has four attributes: `usage`, `ram`, `cu`, and `data out`. Ram represents the memory in GB, usage the percentage of use per month, cu the Amazon Compute Units[7], and data out the amount of data transferred to outside AWS. Although all of them are configurable, the value of cost, ram and cu depends on the instance type and the OS. Cost also depends on the detailed monitoring, and on the usage attribute. These dependencies between attributes and features are modelled using constraints.

EBS and S3 services are also represented in Figure 2. *EBS*, a child feature of EC2 in the model, is the storage linked to EC2 instances. Its features represent the two types of storage, standard or provisioned. The associated attributes include cost (in dollars), storage (in GB) size and IOPS (Input/Ouput Operations Per Second). *S3* is a child of the root feature, at the same level of EC2. For this service, the two storage alterna-

[6] http://status.aws.amazon.com/

[7] A EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron

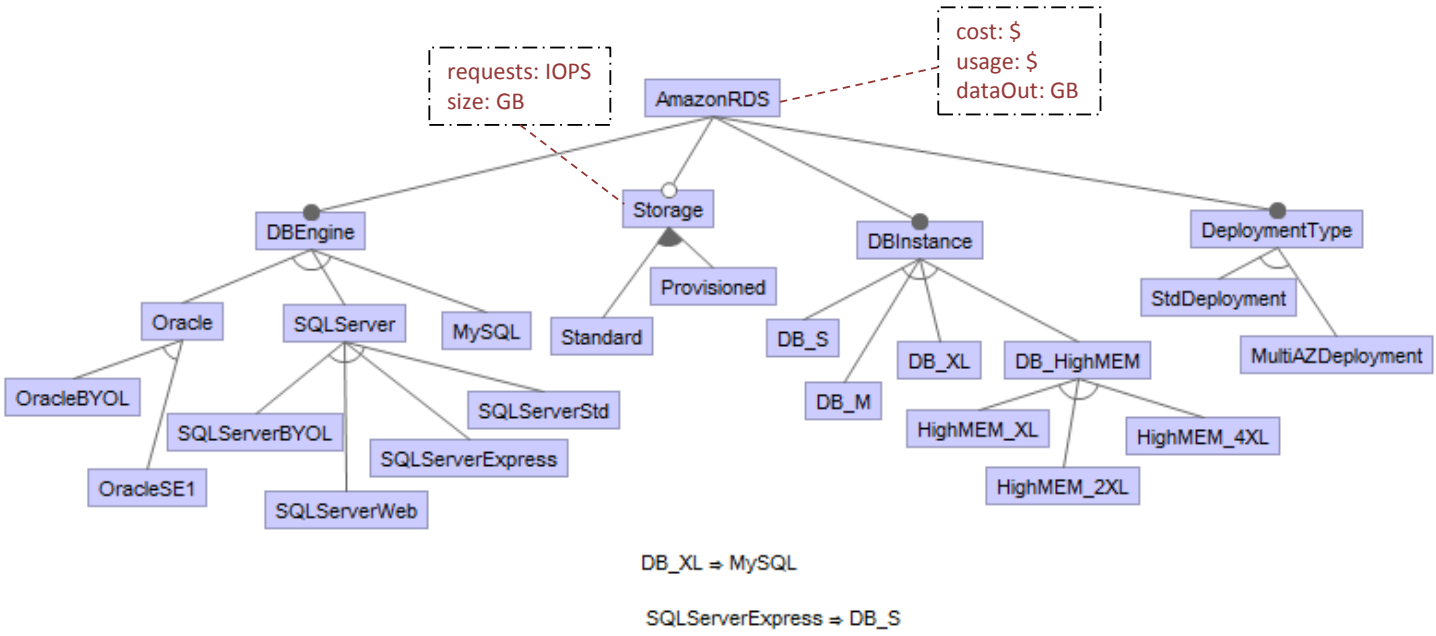

Figure 3: AWS Extended Feature Model: RDS

tives, standard storage and reduced redundancy storage, are represented. Cost of the S3 service, modelled as an attribute, depends on the configured storage size, also an attribute, of each of the alternatives.

The last group of features of Figure 2, named *SpecialReqs*, shows a set of options to satisfy specific user needs. They do not correspond directly with any AWS service instance, but their selection/de-selection implies addition/removal of other features. Preferences about IO performance, 64/32 bit OS, Solid State Disk (SSD) storage or GPU capabilities are provided within this group.

RDS (shown in Figure 3) contains four main configurable parameters: DB engine, instance class, deployment type and storage capacity. Three different engines (MySQL, Oracle ad SQLServer) are available with different licenses, and even a Bringing Your Own License (BYOL) capability has been included. Instances have been split into two groups: standard class (small, large and extra large) and high memory class (extra large, double extra large and quadruple extra large). Two alternative deployments are available: a standard deployment, and a multi-zone availability deployment, which is maintained for planned or unplanned outages. The storage has the same options as EBS, standard or provisioned IOPS. For the non-functional aspects of RDS, we have defined cost, usage, and out data attributes for the whole RDS service, and size and IO operation attributes for the RDS storage.

We show in Figure 4 the extra-functional aspects related to several of the Amazon services. Both EC2 and RDS instances could be reserved for periods of one to three years. This supposes a fixed cost per year, but a reduced cost per hour of usage. Such a fixed price option may be suitable for users who have a heavy computational requirement over a longer time frame. The location of the data center is another aspect that increases or decreases the price of the services, and which we use to measure the risk associated with deployment.

Both cost and risk have been taken into account when defining this model. Cost is represented by cost attributes associated with EC2 and RDS. It is calculated by means of constraints, based on the information at the AWS website, and it depends on the selected features and attribute values. Risk is modelled by considering the location, the service type and the AWS status website. Status website has a historical availability status (and therefore the associated downtime) for the last month for each service and location. We use this data to infer the values of the EC2, RDS and S3 issue rate attribute.

Attributes and abstract features (features which do not correspond directly to any AWS instance) are the main reasons for making our model suitable for user configuration. Using abstract features and attributes, a user can express general preferences without specifying concrete AWS service instances. For example, a user who needs an EC2 instance with high IO

performance can express it by just selecting EC2 and HighIoPerformance features. He can also express his RAM and CPU requirements, by identifying suitable constraints associated with the ram and cu attributes. In the analysis phase that follows such a requirement description, we can search and identify the most suitable EC2 instance for the user.

3.1 Configuring the AWS Feature Model

Usually, users need to configure more than one instance of some of the previously presented services, example, different EC2 (e.g. standard or cluster) or RDS instances. Often, just one variant is not enough. We allow a user to configure several instances, as many as they need, of the AWS EFM. However, this implies the total cost is now the sum of the cost associated with every instance. We define a new attribute named `global cost`, with a global scope, to store and configure this value.

To identify a configuration that meets the user requirements, it is necessary to select features, identify constraints associated with attribute values and define an optimisation criterion. Each configuration instance is composed of a set of feature selections/deselections, and constraints associated with attribute values. If a certain feature is needed, it should be marked as selected. If it is forbidden, it should be marked as removed. However, if a user does not care about a feature, it can be left unmodified in the model. Constraints are expressed using relational operators ($<, \leq, >, \geq, =$, and $\neq$) to exclude or force the selection of certain values. For example, our requirements is to find, a linux cluster with at least 15 cu, with a cost constraint of 400 dollars, to maximise the usage. We have to mark as selected the cluster feature, in the group of instance type, and the linux feature, in the group of OS type. We must also constrain the cost to be less than 400 dollars, and set the EC2 usage attribute as the maximisation criterion. If we also want another EC2 instance (in addition and different to the linux cluster), we have to configure another instance.

In addition to the global cost attribute, all instances are linked by the optimisation criteria. Hence, the optimisation criteria is what determines which configuration is the most suitable for a user. This optimisation maximises or minimises the value of a certain attribute, which could be an instance specific attribute, like the usage or the cu, or the global cost.

4 ANALYSIS METHODOLOGY

A key objective of this work is to identify the most suitable AWS configuration that meets a set of user requirements. When undertaken manually, the user has to navigate through a very large search space – a task that becomes untenable as the complexity of the search space increases. In addition, it is also useful to identify (as a boolean outcome) whether a given provider (in this case AWS) is able to support a required capability. For example, a user requires a Red Hat Cluster instance, but AWS does not provide support these at all. Subsequently, it is also necessary to evaluate functional (features) and non-functional (attributes and optimisation) information, and look for the variant which best suit user requirements. To overcome this challenges, we propose the use of AAFM techniques.

(Benavides et al., 2010) define AAFM as the process of "extracting information from feature models using automated mechanisms". As input, a FM, an analysis operation (e.g. filter, list, etc) and optionally one or more configurations, are provided. Each analysis operation leads to different types of information being retrieved from the model. Some examples are the set of products that a FM represents, checking if a configuration over the FM is valid, or checking if the FM contains errors. Subsequently, depending on the input parameters, the FM is translated into a specific logical paradigm, like Propositional Logic, Satisfiability problem (SAT) or Constraint Satisfaction Problem (CSP), and mapped to a solver. Finally, the result is mapped again to the FM domain to present it in a comprehensible way. The choice of the solver depends on the type of analysis required over the FM– for instance a SAT solver would be used to undertake boolean satisfiability checking, etc.

The AAFM can include over 17 analysis operations, therefore our first step is checking if some of these operations are suitable for our analysis requirements (which include configuration validation and optimisation). Fortunately, *Valid partial configuration* and *Optimisation analysis* operations provide us the analysis we need. Valid partial configuration takes a FM and a partial configuration as input and returns a value indicating whether the partial configuration (identified by a user) meets the FM. When the outcome is negative, we could also identify to the user what changes need to be made to make the configuration valid, as suggested by (White et al., 2010). The optimisation operation takes a FM and an objective function (involving the maximisation or minimisation of an attribute value – such as memory, number of CPUs, etc) as inputs and returns the configuration ful-

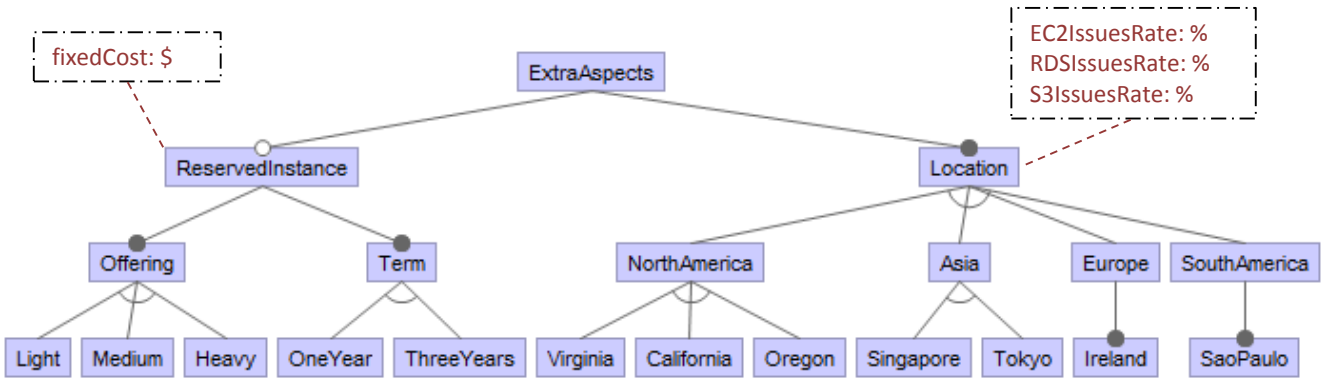

Figure 4: AWS Extended Feature Model: Reserved instances and location

filling the criteria identified by the objective function. Figure 5 shows our proposed approach, based on the use of AAFM, and with the support of the aforementioned operations. The inputs are the AWS model, the user preferences (as partial configurations) and the optimisation criteria. Using the AAFM operations, first we ensure the user preferences are valid. If they are, we use the optimisation operation to obtain the most suitable configurations. Finally, the model configurations are translated to AWS configurations and returned to the user.

The standard optimisation process in AAFM takes as inputs a FM and a optimisation criteria, and optionally a partial variant too – and generally returns a single outcome. However, a user configuring AWS may need more than one instance – for example if he needs two EC2 or RDS instances. It is therefore necessary to enable the user to configure as many instances as needed as part of the optimisation operation. These instances are related by attribute constraints and optimisation criteria. We have an optimisation where the user provides several related instances which we must identify, at the same time, and in the same search space.

Due to the relatively large size of the search space (involving 1758 possible configurations in AWS), the performance of the optimisation operation is a key issue. We propose a pre-processing step to obtain the subset of variants which are able to support user requirements in terms of features. Those variants which cannot fulfill user needs are removed. For this task, we use the *filter* AAFM operation, which takes as input a FM and a partial variant, and returns the set of variants including the input variant that can be derived from the model. This subset is then used within the optimisation operation.

We propose using Boolean Satisfiability Problems (SAT) techniques for the pre-processing phase and Constraint Satisfaction Optimization Problems (CSOP) techniques – both paradigms have been extensively used in AAFM. SAT provides a better performance than CSP, but is limited to the use of boolean variables. For the optimisation, we use CSOP, a variant of the classic Constraint Satisfaction Problem (CSP) with optimisation capabilities. A CSP is a three tuple composed of a finite set of variables, domains (for each variable) and constraints. CSOP has been chosen because it has a high degree of expressivity about variable domains and operators, and a large tools support. Additional discussion and mapping over SAT and CSP can be found in (Benavides et al., 2005) and (Benavides et al., 2010).

4.1 Prototype

We have developed a prototype of the proposed process, supporting the Valid Partial Configuration and Optimization operations. To develop this prototype, we have extended FaMa[8] (Trinidad et al., 2008), a Java-based tool for AAFM. FaMa includes several plugins around a central core. The tool supports different variability metamodels (FMs, EFMs, Orthogonal Variability Models, etc) and also reasoners, which implement most of the defined AAFM analysis operations.

Valid Partial Configuration for EFMs is supported by default in FaMa, we have therefore developed the optimization operation (without pre-processing) for

[8] www.isa.us.es/fama

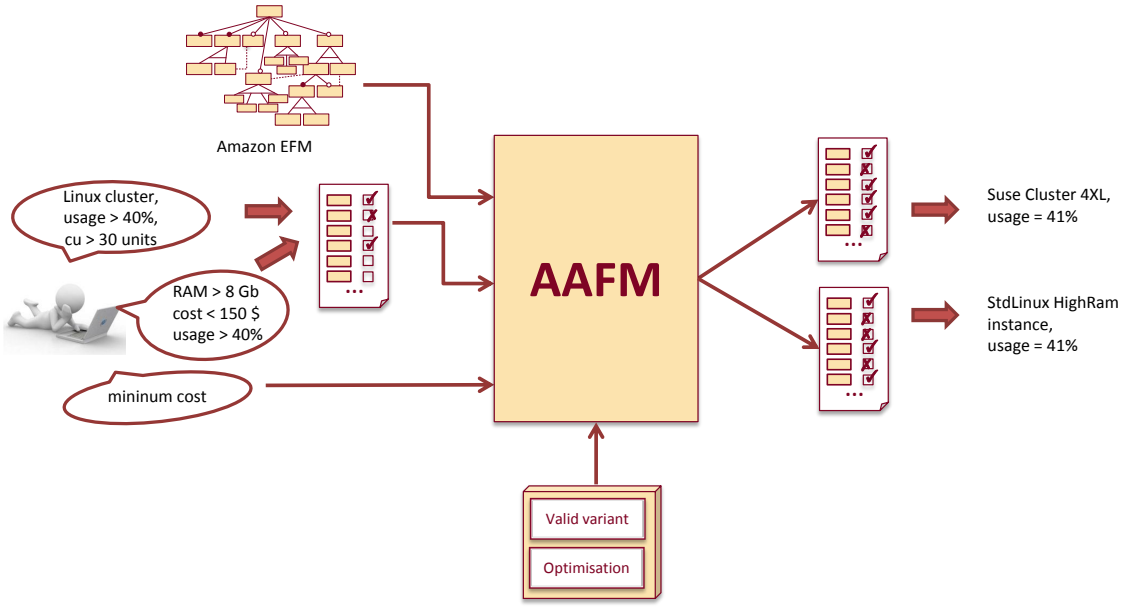

Figure 5: Automated analysis support for AWS configuration

multiple variants. This analysis operation receives the AmazonEFM, a set of partial variants, an optimisation criteria, and optionally an execution time limit as inputs, and returns the set of optimal AWS variants found. Choco [9], one of the FaMa reasoners, is the CSP solver we have used for the implementation. It provides a set of heuristics, which are useful to customise the search in complex problems like this.

The AmazonEFM is provided in FaMa text format [10]. Syntax and details about the format are available in the FaMa user guide [11]. We have simplified the model removing the content of figure 4, it means, removing location and reserved instances. Attributes have been modelled as integer variables, and cost variations (depending on features and attributes selected) have been modelled as constraints. User configuration, optimisation criteria and time limit are specified using the FaMa programmatic interface.

5 CASE STUDY

In order to validate our configuration and analysis approach, we consider a case study involving the migration of computational resources at a University research group interested in: (i) expanding their existing computational capability; (ii) using additional resources to support a demand peak; (iii) understanding the cost implications of using a Cloud environment for hosting their computational infrastructure (with reference to upgrading in-house resources). Fully understanding cost implications – for (iii) – is often a difficult task in practice – as cost associated with administrative staff has to be considered over a particular time frame. This scenario has been chosen as it is representative of a small to medium scale organisation, often operating under a tight budget, and represents on-demand computing requirements. Our intention is to check if our approach is useful in this context and if AWS is a realistic choice for these cases.

The computing infrastructure is shown in Table 1. Although we have four machines, we want to migrate just two of them, the non-critical ones. These computers (labs and clinker) are used for pre-production tasks, so their usage is on-demand. Furthermore, the group has requirements for using a cluster to execute large experiments. Since the budget is limited, our objective is to find the minimal cost solution that satisfies these requirements.

Table 2 shows the configuration of the machines over the AWS FM. Labs and Clinker both perform with RedHat OS, while the cluster would require a Linux distribution. To configure the computing capacity, a conversion to Amazon Compute Units is required. Dividing each desired CPU capacity by the standard cu, we obtain the values in the table. RAM,

[9] http://www.emn.fr/z-info/choco-solver/
[10] https://dl.dropbox.com/u/1019151/AWS.afm
[11] http://famats.googlecode.com/files/FaMa%20Manual.pdf

Name	Processor	RAM	Disk	OS	DB	Usage
Labs	2.8 GHZ (1 core)	2 GB	20 GB	Red hat	MySQL	on demand
Clinker	2.8 GHZ (1 core)	4 GB	40 GB	Red hat	MySQL	on demand
Cluster	2.33 GHZ (8 cores)	10 GB	500 GB	Linux	MySQL	on demand

Table 1: Computational infrastructure of the research group

Name	OS	Instance.cu	Instance.ram	Instance.usage	Storage.size	Storage.IOPS
Labs	RedHat	$\geq 3cu$	$\geq 2GB$	$\geq 30\%$	$\geq 20GB$	$\geq 200M$
Clinker	RedHat	$\geq 3cu$	$\geq 4GB$	$\geq 30\%$	$\geq 40GB$	$\geq 200M$
Cluster	Linux	$\geq 16cu$	$\geq 10GB$	$\geq 10\%$	$\geq 500GB$	$\geq 300M$

Table 2: Machines configuration over AmazonEFM

Name	EC2 Instance	EC2 OS	EC2.cu	EC2.ram	EC2.storage	RDS instance	Cost
Labs	High CPU - M	Red Hat	$5cu$	$2GB$	$350GB$	Small	118
Clinker	Standard - L	Red Hat	$5cu$	$8GB$	$850GB$	Small	152
Cluster	Cluster - 4XL	Linux	$34cu$	$23GB$	$1690GB$	Small	148

Table 3: Case study analysis results

storage, and DB engine properties are translated directly. To estimate usage we calculate the hours per month we need to make use of these resources (as follows): 40 staff working hours per week, divided by 168 hours per week, is approx. 24%. We use a value of 30% to cover extraordinary events. For the cluster, we estimate that the demand for experiments could be about 10%. Finally, we have set IO operations per month to 100 million.

Analysis results are shown in Table 3. The optimal cost per month, obtained after 308 seconds, is 118 + 152 + 148 = 418 $. Additional decisions also need to be made, such as the selection of a High CPU medium instance for labs, instead of a standard large one. Although we require just 16 cu, we can only use (as a minimum) an instance with 34 cu. A similar case holds for storage capacity for EC2 instances.

Currently, the configuration process has to be manually specified by a user – we do not provide a user interface for interacting with the prototype. It is useful to note that identifying the values for data out, usage or IOPS need prior evaluation by a user. Determining what the values for these should be is often a non-trivial process.

5.1 Performance preliminary results

To evaluate the feasibility of our prototype for more realistic (larger scale) scenarios, we calculate the overall performance associated with undertaking the analysis on the FM. Our experiment involves running optimisation operations from one to four configurations and measuring the associated time for finding the optimal configuration(s). Experiments have been executed on a Core 2 Duo 2.00 Ghz laptop, with 2GB Ram and Windows 7 Business Edition OS.

Table 4 shows the preliminary performance results. As we can see, the execution time growth is exponential for the selected optimisation operation. For one configuration, the time is less than 6000 ms, to 32000 ms for two configurations, and 370000 ms for three configurations. However, identifying Valid Partial Configuration shows a different, linear time growth. Therefore, efforts must be focused on improving the Optimisation performance.

6 RELATED WORK

Configuring and analysing Cloud plaftorms/providers is continuing to receive significant attention from both the research and business community. For instance, CloudHarmony [12] is a startup which looks for obtaining metrics about cloud providers performance, and provides a comparison framework for many services providers. PlanForCloud [13] is another startup, focused on configuring and simulating cost of several cloud platforms, like Amazon, Azure or Rackspace. They provide interesting options, like creating elastic demand patterns, and filtering by options like OS or computing needs. Increasingly, there has also been a recognition that Cloud performance can vary significantly over time (Iosup et al., 2011) – based on the workload currently running on a particular provider.

Several research efforts focus on optimising deployment over a cloud infrastructure. (Clark et al., 2012) propose an Intelligent Cloud Resource Allocation Service to evaluate the most suitable configuration given consumer preferences. (Tsai et al., 2012)

[12] http://cloudharmony.com/

[13] http://www.planforcloud.com/

considers a similar approach, choosing between different Cloud providers using data mining and trend analysis techniques, and looking for minimising cost. In a related research, (Sundareswaran et al., 2012) propose indexing and ranking Cloud providers using a set of algorithms based on user preferences. (Venticinque et al., 2011) describe an approach to collect Cloud resources from different providers that continuously meet requirements of user applications. The related work of (Borgetto et al., 2012) is oriented to software reallocation in different Virtual Machines in order to decrease energy consumption. Several ontologies have been also proposed in the last years for cloud services discovery and selection (Androcec et al., 2012).

Applying SPL techniques to Cloud services has also been considered – for instance, (Quinten et al., 2012) propose using FMs to configure IaaS and PaaS, and also use the AAFM to extract information from the model. In a more specific work, (Schroeter et al., 2012) use extended FMs to configure IaaS, PaaS and SaaS, and also present a process to manage the configuration of several stakeholders at the same time. (Dougherty et al., 2012) also uses FMs to model IaaS, but the goal in this case is reducing energy cost and energy consumption, towards the development of a "Green Cloud". In a different way, (Cavalcante et al., 2012) proposes the extension of traditional SPL with Cloud computing aspects. Our work differs from these approaches in two key ways: (i) we focus on one specific provider – to better understand the range of capabilities that this provider offers. We have chosen AWS as it is the most widely used and configurable provider currently on the market; (ii) we focus on using specialist analysis approaches – such as use of SAT and CSP solvers – in order to automatically analyse the resulting feature model.

The optimisation operation has been the focus of several research works related to AAFM. Recently, (Roos-Frantz et al., 2012) propose the optimisation of a radio frequency warner system using Orthogonal Variability Models (OVM), an alternative to FMs. (Guo et al., 2011) approach the optimisation for EFMs from a more general perspective, using genetic algorithms.

7 CONCLUSIONS AND FUTURE WORK

In this work, we have modelled several AWS services using FMs. Modelling the configuration space of a large Cloud provider like AWS provides a useful and compact representation to express user preferences.

	Time in ms			
	1 conf	2 confs	3 confs	4 confs
Optimisation	5615	31799	363578	1h30+
Valid Conf.	135	260	267	302

Table 4: Average execution time of operations for 1, 2, 3 and 4 configurations

The resulting model still contains lots of information and required data, it is therefore necessary for be clear about their requirements and needs. Therefore although the AWS FM eases the configuration process, it is still necessary to have an indepth understanding about the infrastructure requirements to interpret the analysis results. However, this kind of information is near to the users domain, and allows them to express their problem in a way more conducive to their own understanding of it, and not in the terms of a specific provider catalogue.

We have also proposed the use of several analysis operations of the AAFM, some of them tailored to assist the user in the configuration process. With the support of these operations, user preferences (functional and non-functional) can be validated, and used to obtain suitable AWS configurations. From the variability management perspective, we have demonstrated the applicability of AAFM to Cloud services.

Finally, we have presented a prototype of the analysis operations, based in the FaMa tool, and a case study. Although the prototype and its performance are at a premliminary stage, additional work is being carried out to improve their performance. The prototype has revealed the need for improving performance by using pre-processing and heuristics to execute the optimisation operation in a reasonable time.

Our approach could be extended and generalised to providers other than AWS, like Rackspace/OpenStack or Microsoft Azure. Most of the AWS FM structure is reusable, enabling us to adapt some of the attributes and cost to other providers. We believe there is a need to create a FM of an *abstract Cloud provider* which has a minimal core set of features and attributes common across providers, and where existing cloud services ontologies could be considered for a better result. Subsequently, FMs can be specialised based on the particular provider being considered. This would also provide a useful basis for comparing between providers. A friendly way to configure the AWS FM is also required, by means of a Domain Specific Language or at a minimal the development of a user interface. In this sense, an integration with a platform like `www.planforcloud.com` would be really interesting. Applying metaheuristic techniques, such as (Guo et al., 2011), could also be used as a basis to

improve the performance in real-time scenarios. A deeper experimentation is also required. Modelling and analysing an small-medium company case, and comparing performance and optimality of different implementation techniques are mandatory tasks for further work.

ACKNOWLEDGEMENTS

This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project SETI (TIN2009-07366) and by the Andalusian Government under ISABEL (TIC-2533) and THEOS (TIC-5906) projects.

REFERENCES

Amazon ec2 calculator. http://calculator.s3.amazonaws.com/calc5.html. Last accessed: November 2012.

Cloud harmony. http://cloudharmony.com/. Last accessed: November 2012.

Androcec, D., Vrcek, N., and Seva, J. (2012). Cloud computing ontologies: A systematic review. In *MOPAS 2012, The Third International Conference on Models and Ontology-based Design of Protocols, Architectures and Services*.

Benavides, D., Ruiz-Cortés, A., and Trinidad, P. (2005). Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*.

Benavides, D., Segura, S., and Ruiz-Cortes, A. (2010). Automated analysis of feature models 20 years later: A literature review. *Information Systems*.

Borgetto, D., Maurer, M., Da-Costa, G., Pierson, J.-M., and Brandic, I. (2012). Energy-efficient and sla-aware management of iaas clouds. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*.

Cavalcante, E., Almeida, A., and Batista, T. (2012). Exploiting software product lines to develop cloud computing applications. In *Software Product Line Conference*.

Clark, K., Warnier, M., and Brazier, F. (2012). An intelligent cloud resource allocation service: Agent-based automated cloud resource allocation using micro-agreement. In *CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science*.

Clements, P. and Northrop, L. M. (2002). *Software Product Lines: Practices and Patterns*. Addison-Wesley.

Dougherty, B., White, J., and Schmidt, D. C. (2012). Model-driven auto-scaling of green cloud computing infrastructure. *Future Generation Computer Systems*.

Guo, J., White, J., Wang, G., Li, J., and Wang, Y. (2011). A genetic algorithm for optimized feature selection with resource constraints in software product lines. *J. Syst. Softw.*

Iosup, A., Yigitbasi, N., and Epema, D. H. J. (2011). On the performance variability of production cloud services. In *Proceedings of CCGRID*.

Kang, K., Cohen, S., Hess, J., Nowak, W., and Peterson, S. (1990). *Feature-Oriented Domain Analysis (FODA) Feasibility Study*.

Khajeh-Hosseini, A., Greenwood, D., Smith, J. W., and Sommerville, I. (2012). The cloud adoption toolkit: supporting cloud adoption decisions in the enterprise. *Softw., Pract. Exper.*

Khajeh-Hosseini, A., Sommerville, I., Bogaerts, J., and Teregowda, P. B. (2011). Decision support tools for cloud migration in the enterprise. In *IEEE CLOUD Conference*.

Quinten, C., Duchien, L., Heymans, P., Mouton, S., and Charlier, E. (2012). Using feature modelling and automations to select among cloud solutions. In *2012 3rd International Workshop on Product LinE Approaches in Software Engineering, PLEASE 2012 - Proceedings*.

Rabiser, R., Grnbacher, P., and Dhungana, D. (2010). Requirements for product derivation support: Results from a systematic literature review and an expert survey. *Information and Software Technology*.

Roos-Frantz, F., Benavides, D., Ruiz-Corts, A., Heuer, A., and Lauenroth, K. (2012). Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Journal*.

Schroeter, J., Mucha, P., Muth, M., Jugel, K., and Lochau, M. (2012). Dynamic configuration management of cloud-based applications. In *Proceedings of the 16th International Software Product Line Conference - Volume 2*.

Sundareswaran, S., Squicciarini, A., and Lin, D. (2012). A brokerage-based approach for cloud service selection. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*.

Thum, T., Kastner, C., Erdweg, S., and Siegmund, N. (2011). Abstract features in feature modeling. In *Proceedings of the 2011 15th International Software Product Line Conference*.

Trinidad, P., Benavides, D., Ruiz-Cortés, A., Segura, S., and A.Jimenez (2008). Fama framework. In *12th Software Product Lines Conference (SPLC)*.

Tsai, W.-T., Qi, G., and Chen, Y. (2012). A cost-effective intelligent configuration model in cloud computing. In *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*.

Venticinque, S., Aversa, R., Di Martino, B., and Petcu, D. (2011). Agent based cloud provisioning and management: Design and prototypal implementation. In *CLOSER 2011 - Proceedings of the 1st International Conference on Cloud Computing and Services Science*.

White, J., Benavides, D., Schmidt, D., Trinidad, P., Dougherty, B., and Ruiz-Cortes, A. (2010). Automated diagnosis of feature model configurations. *Journal of Systems and Software*.