Automated Configuration Support for Infrastructure Migration to the Cloud

Jesús García-Galán[a,*], Pablo Trinidad[a], Omer F. Rana[b], Antonio Ruiz-Cortés[a]

[a]*Escuela Técnica Superior de Ingeniería Informática, Avda. Reina Mercedes s/n, 41012, University of Seville, Seville, Spain*
[b]*School of Computer Science and Informatics, Cardiff University, Queen's Buildings, Newport Road, Cardiff CF24 3AA, UK*

Abstract

With an increasing number of cloud computing offerings in the market, migrating an existing computational infrastructure to the cloud requires comparison of different offers in order to find the most suitable configuration. Cloud providers offer many configuration options, such as location, purchasing mode, redundancy, extra storage, etc. Often, the information about such options is not well organised. This leads to large and unstructured configuration spaces, and turns the comparison into a tedious, error-prone search problem for the customers. In this work we focus on supporting customer decision making for selecting the most suitable cloud configuration – in terms of infrastructural requirements and cost. We achieve this by means of variability modelling and analysis techniques. Firstly, we structure the configuration space of an IaaS using feature models, usually employed for the modelling of *variability-intensive* systems, and present the case study of the Amazon EC2. Secondly, we assist the configuration search process. Feature models enable the use of different analysis operations that, among others, automate the search of optimal configurations. Results of our analysis show how our approach, with a negligible analysis time, outperforms commercial approaches in terms of expressiveness and accuracy.

Keywords: EC2, Automated Analysis, Cloud Migration, Feature Model, IaaS

---

1. Introduction

The clear benefits of cloud-based infrastructures are increasing the number of companies that are migrating their private and expensive data centers to the cloud. An *Infrastructure as a Service (IaaS)* enables the dynamic provisioning of computational & data resources (often on-demand), reducing costs (for short term workloads), speeding up the start-up process for many companies, and decreasing resource and power consumption (among other benefits).

Deciding the most suitable provider is often challenging, as each provides a number of possible configurations. As an example of the dimension of this problem, there are over 100 public cloud providers [1], and just for *Elastic Compute Cloud (EC2)* [2], the *Amazon Web Services (AWS)* computing service, we have identified $16,991$ different configurations[1]. As each user/company that plans to use a cloud computing infrastructure is likely to have their own specific requirements, it is necessary to identify the most relevant provider and subsequently the most suitable configuration. Identifying such configuration within a large potential search space is a tedious and error-prone task that requires for an automated support.

In recent times, software tools and research contributions have emerged to support this decision process, but we have found these to have limitations, providing either incomplete configuration spaces or/and imprecise results. On the commercial side, providers such as Amazon or Rackspace provide tools that suggest specific configurations for migrating an on-premise infrastructure [3, 4]. However, such tools ignore some configuration options, forcing for example in the case of Amazon to choose Linux as the only operating system. Other companies, like CloudScreener [5], provide their own comparators to decide which provider and configuration best fit user needs. Nonetheless, such

[*]Corresponding author. Tel.: +34659108324

Email addresses: `jegalan@us.es` (Jesús García-Galán), `ptrinidad@us.es` (Pablo Trinidad), `ranaof@cardiff.ac.uk` (Omer F. Rana), `aruiz@us.es` (Antonio Ruiz-Cortés)

[1]The basis for this number is explained further in this paper.

tools lack information about the configuration space they work with, and some tests have revealed false positives in their optimal results. Recent academic work [6, 7] also suffers from similar concerns, as they generally only consider a small subset of the available configurations of services like EC2 or Azure virtual machines. In order to overcome these limitations, any approach must ensure for each provider to model its complete configuration space.

In this work, we assist the customer in determining the most suitable configuration of an IaaS. For that purpose, we present the case study of AWS EC2. As outlined previously by us [8], AWS is one of the most variable and complex providers in terms of configuration options and pricing. Indeed, understanding EC2 configuration space can be challenging, as it is scattered across several pages, tables and paragraphs. We believe that modelling a complex provider eases the task of modelling simpler providers. Additionally, AWS is one of the most widely used IaaS providers, being present in all the current configuration tools. For focusing on one provider, enables us to check their precision and compare to our approach. Among all the different services AWS offers, we focus on EC2 and *Elastic Block Storage (EBS)*– additional disk for computing instances, which are considered as core infrastructure services.

We interpret an IaaS as a *variability-intensive* system, so we can rely on variability modelling and analysis techniques to support the configuration process. In particular, we propose the modelling of IaaS– and EC2 in concrete – as *Feature Models (FMs)*, a kind of model widely used for variability-intensive systems. In this way, first we represent the configuration space in a complete, structured and compact manner, and second we provide the user with a model to ease the configuration process. This modelling enables the use of the so-named *Automated Analysis of Feature Models (AAFM)*, a set of analysis operations that extracts information from the models, which we subsequently use to assist decision-making. We use some of them to verify the validity and completeness of the FM with respect to the service configuration space, and to determine which configuration is the most suitable for any given requirements. We interpret the most suitable configuration as the one that meets customer's requirements and optimises the cost. Our approach presents two main benefits: first, we consider the complete configuration space, so that the real optimal solution is obtained for given customer requirements; second, assisting the configuration process for such a highly-configurable service like AWS EC2 enables the same approach to be used for other providers, such as Azure or Rackspace.

For evaluation purposes, we *(i)* verify our proposed model, *(ii)* compare our approach to existing commercial applications in term of expressiveness and accuracy, and *(iii)* check and improve the performance of our approach. To verify our model, we describe the EC2 FM using a plain-text language, extract the list of configurations within our model, and check that it matches exactly the available configurations of EC2[2]. The validation of the analysis is performed by means of two different implementations: FaMa Framework – a well-known tool for the AAFM– and a reasoner based on the IBM CPLEX solver. We compare the performance of both approaches, where the latter implementation shows improved and negligible execution times when calculating the most suitable configuration. We also compare the obtained results with the output of CloudScreener, which can be improved by the use of our approach.

This paper extends our previous work [8] in several ways. In particular, we provide i) an explicit description of the configuration problem, ii) a modelling methodology to describe the configuration space of an IaaS as a FM, iii) a verification of the configuration space represented by the FM by means of analysis operations and iv) an evaluation of the expressiveness, accuracy and performance of our approach.

The rest of the paper is structured as follows: Section 2 briefly describes the state of the art in variability modelling and analysis. Section 3 states the problem we tackle in this work. Section 4 describes our modelling methodology for the configuration space of an IaaS, while Section 5 presents a modelling case study with Amazon EC2. Section 6 presents our analysis approach for the search of the optimal configuration, and explains the details of the analysis operations. We present an implementation of our approach in Section 7, and we evaluate it in Section 8. Related work is described in Section 9. Finally, Section 10 discusses our proposal and proposes future directions in our research.

2. Feature Models

Feature Models (FMs) [9] are used to represent all the possible products that can be built in variability-intensive systems such as *Software Product Lines (SPLs)*. FMs are tree-like data structures where each node represents a product

[2]We exclude spot instances and micro instances since they are not intended to be persistent, and EBS optimised instances because we do not consider IOPS provisioning for EBS.

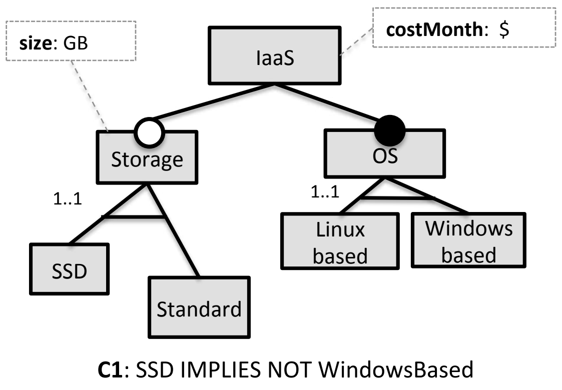

Figure 1: Example of a FM

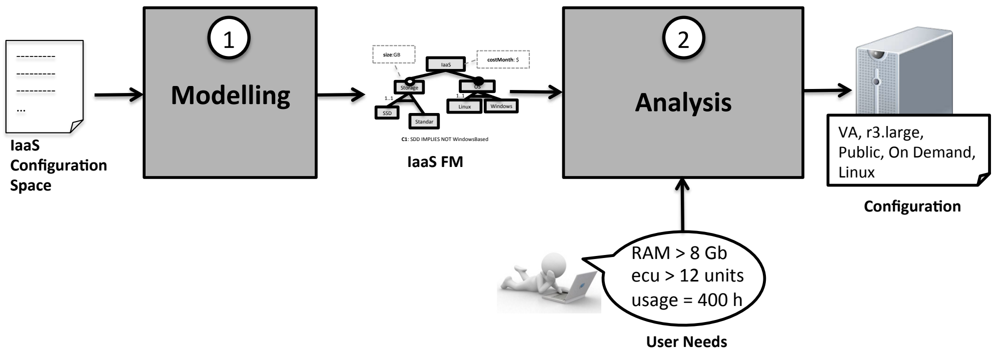

Figure 2: IaaS Configuration Space Description and Analysis as a FM.

feature. Figure 1 shows a FM that represents general features of a fictional IaaS provider. Features are bound by means of hierarchical (mandatory, optional and set) and cross-tree relationships. These relationships define how features can be combined in a product, defining the configuration space of the system. In a FM, a feature does not necessarily represent a specific functionality but can be used as abstract features [10] which represent domain decisions such as `Linux based` feature (Figure 1). `IaaS` is the root feature that represents the overall functionality of the system. It has two children, an optional feature (white circle) named `Storage`, and a mandatory feature (black circle) named `OS`. Both features present set relationships whose cardinality indicates the number of child features that can be chosen at the same time.

FMs can also have attributes that represent non-functional properties, leading to attributed FMs. These attributes are linked to a specific feature. In the FM sample, the size attribute is linked to the `Storage` feature, and monthly cost attribute is linked to the root feature. Optionally, it is possible to define constraints that describe the relationships among attributes.

FMs contain valuable information about all the possible configurations and their properties. From a FM we can deduce a number of possible outcomes, such as the total list of possible products, the set of common features among products, the set of products that meet a given criterion and the product with a minimum cost. Analysing the FM manually is a tedious and error-prone task. Many researchers have focused on the AAFM [11] [12], that currently offers over 30 different analysis operations, each of them solved by means of different declarative approaches such as constraint programming, SAT problems or binary decision diagrams.

3. Problem

Migrating existing applications to the cloud has received significant interest recently. Cloud benefits, like cost savings, scalability and on-demand features, make large companies and *Small and Medium Enterprises (SME)* want to embrace the cloud. However, this process requires facing a number of issues, such as the need to carry out feasibility studies, provider selection or code/ application modifications. Recently, a *Cloud Reference Migration Model* has been proposed [13] to address such issues. This reference model encompasses three main phases: planning, execution and evaluation.

Selecting the most suitable provider and its configuration are relevant decisions in the migration planning process. Each provider offers several configuration options, each with up to dozens configuration values, which may be slightly or very different from one vendor to another. Moreover, the information and constraints about the configuration space are often poorly organised. For instance, the configuration options and values of EC2 are scattered among three different pages, making it difficult to manually search for suitable configurations. Rackspace presents the configuration space of its servers also in a similar way. The size, interconnections and organisation of the configuration space makes configuration a tedious and error-prone process requiring an automated solution.

In this work, we identify and tackle two challenges, as depicted in Figure 2

1. *Model the configuration space of a cloud provider in a structured and compact way.* Configuration information is commonly described in an unstructured way using natural language, and therefore subject to ambiguity. It is necessary to model and structure it to enable ease of use and understanding of the services by an end-user, and to enable automated support. Most of the academic works that have approached migration issues [6, 7, 14, 15, 16, 17, 18] deal with reduced subsets of the providers' configuration spaces. However, when we plan for cloud adoption (or migration), we have to address the whole configuration space to find the most suitable configuration.

2. *Assist the search of the most suitable configuration.* Even with a well-structured configuration space, most of the providers offer thousands of different configurations. Assisting the search for the most suitable configuration would save time and effort for the users, and could ensure that the selected configuration really fits customer needs. In this paper, we define the most suitable configuration as the one that fulfils the customer requirements and minimises the total cost.

3.1. Scenario

Table 1: Migration case study

Purpose	**# instances**	**Cores**	**RAM**	**Disk**	**Location**	**OS**	**Hours / Months**
Production	2	2	4	2 TB	Europe	Linux-based	730 / 12
Pre-production	1	2	4	250 GB	-	Linux-based	160 / 12
Perf. testing	1	8	4	1 TB	-	RedHat	40 / 12

We present a migration scenario of a SME, which has multiple application servers and a server for pre-production tasks. They plan to migrate to the the AWS cloud, and they look for servers with the characteristics shown in Table 1. In addition to the the two application and one pre-production servers, they are currently involved in new developments that require performance testing. Therefore, they also need a large compute-intensive instance for testing. The application instances must be running 24/7, while the pre-production instance is needed 40 hours per week. The performance testing instance will be used around 40 hours per month.

We have chosen AWS and EC2 as the provider and service under study for several reasons. First, AWS is one of the most widely used providers of infrastructure services. Second, the number of configuration options and values of EC2 leads to thousands of different configurations that necessitates automated assistance. Finally, the complex price policy turns the modelling into an even more challenging task.

4. Modelling

In this section, we describe our approach to model an IaaS configuration space as a FM. First, we present a brief taxonomy of the typical IaaS configuration options offered by commercial providers. We then propose a modelling methodology to describe such configuration options as a FM.

4.1. IaaS Configuration Options

IaaS commercial providers aim to address the needs of different kinds of customers. From those looking for on-demand virtual machines to big companies such as Netflix requiring massive computation and storage, IaaS providers offer them multiple configuration options to meet their requirements. For this taxonomy of the IaaS configuration space, we have studied four main providers: Google [19], Amazon [2], Microsoft [20] and Rackspace [21]. Most of the providers enable configuration options based on the following factors:

- *Instance type*. Each instance type determines the basic characteristics of the computing instance – RAM and number of cores. Depending on the provider, a default storage, linked to the instance, is also offered. The instance type range is usually broad, from small instances with a single core to large clusters.
- *Operating system*. Most of the providers offer different flavours of Windows and Linux OS.
- *Storage*. Although some providers offer a default storage depending on the instance type, all of them allow to hire additional storage linked to the instance. This storage is usually limited to a number of GBs per instance, and can be based on Solid State Disks (SSDs).
- *Geographic location*. The infrastructure services offered by IaaS providers are delivered through different datacenters, spread across multiple geographic locations around the world.
- *Purchasing mode*. While all the providers offer their services on-demand, usually billing the use per hour, most of them also provide additional commitment plans or purchasing modes to hold their customers while they get significant savings.

4.2. Modelling Methodology

In order to describe the aforementioned configuration options as a FM, we provide a description about the method we follow. Lee *et al.* [22] propose some guidelines to identify and organise features that we extend to support attribute modelling as follows:

Table 2: IaaS mapping to FM

IaaS Aspect	**Example**	**FM**
Config point	Location	Abstract feature
Config value	US Virginia	Leaf feature
Usage data	Months	Attribute
Service att.	Cost hour	Attribute

1. *Tree definition*. The tree structure of the FM, i.e. the features and their relationships, is defined at first.
   (a) *Features definition*. A feature [9] is a tree node which can be selected or removed. Traditionally, two types of features have been considered: leaf features to express the configuration values that define the configuration space, and abstract features to organise semantically related features. We interpret all the values that make any two IaaS configurations to be different as leaf features in the FM. We also identify a number of abstract features, that we will define as the configuration options. These features may also be grouped, such OS, geographical location, or instance type – see Table 2.

(b) *Relationships definition*. Once we have defined the leaf features and the configuration options and its possible values from the provider documentation, we create the tree like structure by means of relationships. The root feature is placed at the top representing the whole configuration space of the IaaS. At a second level, all configuration options are placed as children of the root feature, having either a mandatory or an optional relationship depending on the optional character of the feature. At a third level, all the leaf features are placed as child of the corresponding configuration option. Since only one leaf feature can be selected at a time for each configuration option, they are grouped by means of a set relationships with a 1..1 cardinality. Once all the leaf features are placed in the model, we introduce some intermediate abstract features to ease configuration and to represent higher-level decisions.

2. *Attributes definition*. At this point, there exist elements that cannot be modelled as features, because their domain is not boolean and/or because their values is the result of a calculation from other attributes or features. For example, an instance cost per hour is clearly a non-boolean value that depends on the cost of the selected features. Such elements are modelled as attributes – with units and domain – and are linked to their related feature. We also identify two kinds of attributes, as shown in Table 2:

   - *Usage data:* information about the usage which is provided by the customer, e.g. instance hours per month.
   - *Service derived attributes:* attributes whose values are obtained from other decisions, like the RAM size or number of cores of an instance, which depend on the instance type.

3. *Constraints definition*. Finally, we define constrains on the features and attributes of the FM to adequately represent the IaaS configuration space. These constraints encompass the instance cost hour, availability, characteristics and additional dependencies. For most of the cases, the only source to obtain constraints associated with the configuration space is the cloud provider's website. While the extraction of some constraints is trivial, others have to be inferred from the description of the service in natural language. An example of the former is the pricing of the instances, usually structured in tables, such as in Amazon, Rackspace or Microsoft Azure. For instance, consider the following constraint:

   ```
   (M3.large AND OneYear AND Sydney AND RedHat AND Public) IMPLIES (upfrontCost==249
                                AND costHour==0.235)
   ```

   In order to automate the extraction of this kind of constraints, we can use web scraping techniques. It is important to note that the configuration space of commercial providers evolves often, so manual adjustments in the scraper or advanced scraping techniques may be required in such cases. However, the extraction of other constraints, such the total cost per year of the infrastructure

   ```
   Total Cost = Cost Hour × Hours Per Month × Number Of Months + Upfront Cost + EBS
                                     cost
   ```

   cannot be automated, and must be formulated manually.

5. Modelling Case Study: Amazon EC2

In this section, we exemplify our modelling approach by means of a case study: the description of Amazon EC2 as a FM. First, we briefly describe the main aspects and configuration options of EC2. Second, we present the EC2 FM, result of the methodology described in the previous section. Finally, we suggest some guidelines for the configuration of an EC2 instance using the EC2 FM.

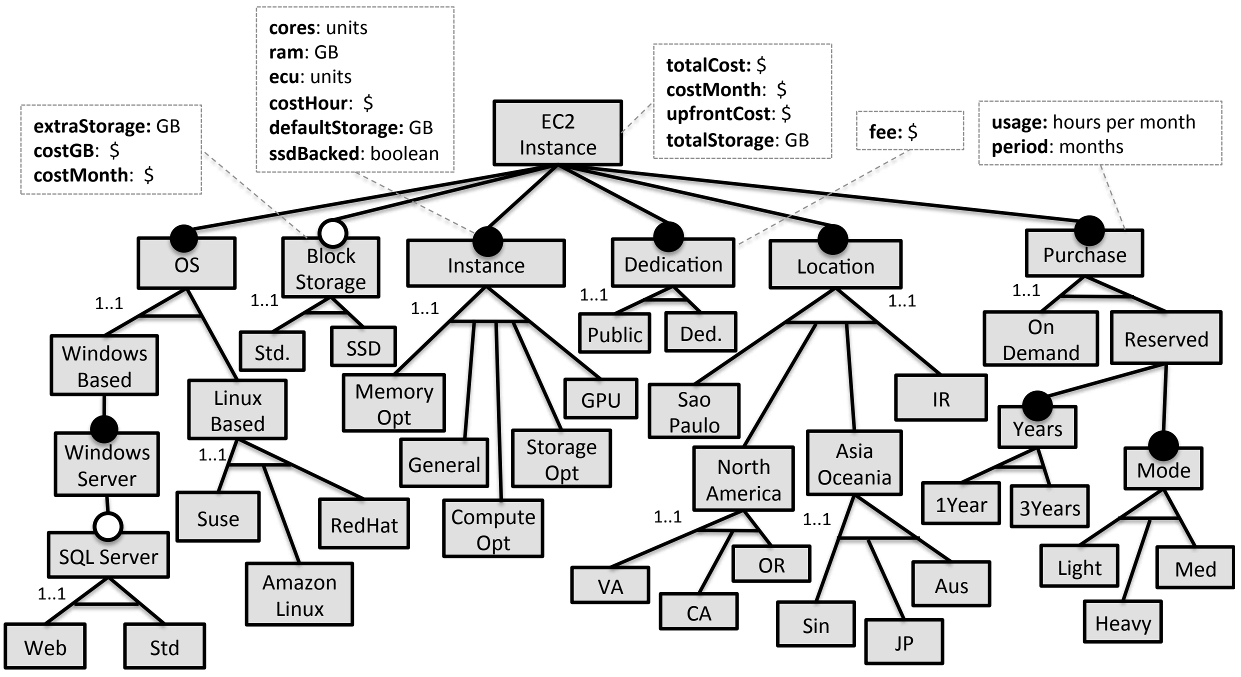

Figure 3: Feature Model of EC2 and EBS

5.1. AWS Elastic Compute Cloud

AWS provides services for computation, storage, databases, clusters or content delivery among others. Due to the wide range of services, several PaaS and SaaS providers like Heroku or Netflix run over AWS. As defined by Amazon, EC2 "provides resizable compute capacity in the cloud" [2] on pay-per-use basis. In this paper, we work with the AWS snapshot for EC2 and EBS of 12th June 2014. On this date, Amazon provides five main configuration options for EC2, which match the general configuration options of IaaS providers depicted in the previous section:

- *Instance type*. EC2 offers 32 different instance types, grouped under different categories, depending on the purpose: General Purpose, Compute Optimized, GPU Instances, Memory Optimized and Storage Optimized, distinguishing also between current and previous generations. Each instance type has a specific RAM size, a number of cores and disk storage. In this point, we ignore micro instances, since they are intended for short CPU burst purposes. Consequently, they lack disk storage, and their performance is highly variable and unpredictable.

- *Operating system*. Three different Linux distributions (Amazon Linux, Suse and Red Hat) and a Windows version (Windows Server) with an optional SQL Server in different flavours (Express, Web and Standard) are available.

- *Storage*. AWS provides a fixed disk size for EC2 instances, depending on each instance type. If the default storage needs to be extended, the EBS service provides additional storage. In this sense, we must say that we do not consider provisioned IOPS (input/output operations per second) EBS, and neither EBS optimised instances, that "*enable EC2 instances to fully use the provisioned on an EBS volume*".

- *Geographic location*. Amazon offers 8 geographic locations, distributed among different areas of North America, Europe, Asia and South America.

- *Purchasing mode*. EC2 instances may be purchased on demand or in a reservation way. While users pay per use in both modes, cost hour is lower for reserved instances in exchange for an upfront payment. In total, there are seven different purchasing modes. We exclude here spot instances, because Amazon determines their price and availability dynamically based on supply and demand. Indeed, AWS recommend the use of spot instances only for time-flexible and fault-tolerant tasks.

Additionally, an EC2 instance may run on a dedicated machine, guaranteeing an additional isolation in exchange of an additional cost.

5.2. EC2 Feature Model

The resulting EC2 FM is defined in a plain-text language and presents 81 features, 17 attributes and more than 20 000 constraints [3]. It is based on the description of EC2 of the day 12 June 2014 [2]. The huge amount of constraints needed to represent pricing and instances availability makes necessary to automate their extraction by means of a web scraper, whose details are explained in Section 7. Due to the difficulty of representing the complete model using feature diagrams, only an excerpt is shown in Figure 3.

The root feature, *EC2 Instance*, groups the configuration options and defines 4 attributes: *totalCost*, *costMonth*, *upfrontCost* and *totalStorage*. There are 6 configuration options where only one leaf feature can be selected at the same time: *(i) OS* is composed by Linux and Windows variants. We include some abstract features and SQL Server options for more flexibility in the decisions. Since SQL Sever Express is included by default in any Windows variant, we exclude it from the FM. *(ii)* The additional *Block Storage* is defined as optional, and can be SSD or magnetic based. It has three attributes: *extraStorage* for the size, *costGB* for the GB cost per month, and *costMonth*, which value is calculated as $extraStorage \times costGB$. *(iii) Instance* contains general purpose, high mem, high CPU, high IO and GPU features. Due to spatial constraints the specific instances types are shown separately in Table 3. We define 6 attributes for this configuration option: *cores* for the number of cores of the instance, *ram* for the memory, *ecu* for the EC2 Compute Units, *costHour*, *defaultStorage* for the default storage of the instance, and *ssdBacked* to show if the instance storage is SSD backed. *(iv) Dedication* determines if the EC2 instance is based on shared or dedicated hardware resources. This kind of isolation implies additional cost, so we have defined an attribute, named *fee*, to represent it. *(v) Location* groups all the available locations for EC2 instances by continent and state/country. Finally, *(vi) Purchase* represents the purchasing options, and also defines use hours and number of months, by means of *usage* and *period* attributes.

Table 3: EC2 instance categories and specific types

Category	Specific types
General purpose	M3.medium, M3.large, M3.xlarge, M3.2xlarge, M1.small, M1.medium, M1.large, M1.xlarge
Compute opt	C3.large, C3.xlarge, C3.2xlarge, C3.4xlarge, C3.8xlarge, C1.medium, C1.xlarge, CC2.8xlarge
Memory Opt	M2.xlarge, M2.2xlarge, M2.4xlarge, CR1.8xlarge, R3.large, R3.xlarge, R3.2xlarge, R3.4xlarge, R3.8xlarge
Storage Opt	I2.xlarge, I2.2xlarge, I2.4xlarge, I2.8xlarge, HS1.8xlarge, HI1.4xlarge
GPU	G2.2xlarge, CG1.4xlarge

5.3. Customer Requirements on EC2 FM

Representing the EC2 configuration space as a FM has several benefits. Our model represents the EC2 configuration space in a compact and well-structured way, so it is easy for the customer to see the big picture. Abstract features

[3]Available at https://dl.dropboxusercontent.com/u/1019151/EC2FM.pdf

Table 4: Customer requirements for the case study

Instance	Location	OS	Dedication	Cores	RAM	Total storage	Usage	Period
Production	IR	LinuxBased	Public	2	4	2,000	730	12
Pre-production	-	LinuxBased	Public	2	4	250	160	12
Testing	-	RedHat	Public	8	4	1,000	40	12

let users make high-level decisions, and attributes like RAM, storage and period add the ability to make decisions about user terms besides EC2 configuration terms. Moreover, FMs enable the use of the AAFM analysis operations to assist the decision-making.

We propose to represent customers' infrastructure requirements relying on the EC2 FM. Customers can make decisions on the different configurations points of our model. They can decide whether to select a feature or not, and specify preferences about attributes. At least, customers should make the next mandatory decisions:

- Assign values to attributes *period* and *usage*.

Optionally, we recommend costumers to make decisions on the following elements:

- Select a child feature of *OS*, *Dedication* and *Location*.
- Remove or select a child feature of *EBS*.
- Assign values to *ram*, *cores/ecu* (one of them) and *ssdBacked*.

We take the case study of Section 3.1 and Table 1 as an example. Table 4 shows the decision making we propose for each server. As we can see, we select abstract feature for some cases, which give us higher-level decisions, and specific features in others. For attributes, the value is constrained with a greater or equal relational constraint (≥), since it is possible that EC2 cannot match exactly the value.

6. FM Analysis

Modelling the configuration space of an IaaS as a FM enables the extraction of information by means of AAFM operations. Benavides *et al.* [12] define the AAFM as "*the process of extracting information from FMs using automated mechanisms*". The AAFM provides a catalogue of analysis operations, where each operation retrieves different kinds of data from the model. An analysis operation can be interpreted as a black-box procedure that receives a FM and any operation-specific parameter as inputs and retrieves a different kind of output data depending on the operation. Examples of AAFM operations are products listing (that enumerates all the products described by a FM), valid configuration (that checks whether it is possible to find at least one product given user requirements), or error checking (that searches for different kinds of semantic errors).

The main goal of this work is to automate the search of the most suitable configuration of a given IaaS provider – AWS EC2 in our case – that meets a set of user requirements. Prior to configuration search, we need verify that the FM correctly represents the configuration space of service. For that purpose, we identify two tasks to perform that can be solved relying on AAFM capabilities: (1) error checking, to ensure the absence of semantic errors [23] in the FM and (2) product listing, to obtain a list of all the possible configurations. In this section we detail the use of AAFM operations in order to accomplish these tasks.

6.1. Error Checking

As Trinidad et al. propose [23, 24], it is a good practice to check that a FM is free of errors prior to perform any further analysis operation. When cross-tree constraints and attributes are used, contradictory information can be introduced in a FM provoking undesirable effects and making the FM not to represent the configuration space correctly. The AAFM provides the *Error Checking* operation (Figure 4.a) that determines if a FM has any of four kinds of errors:

1. Void FM: a FM that describes no product at all due to contradictory relationships or constraints.

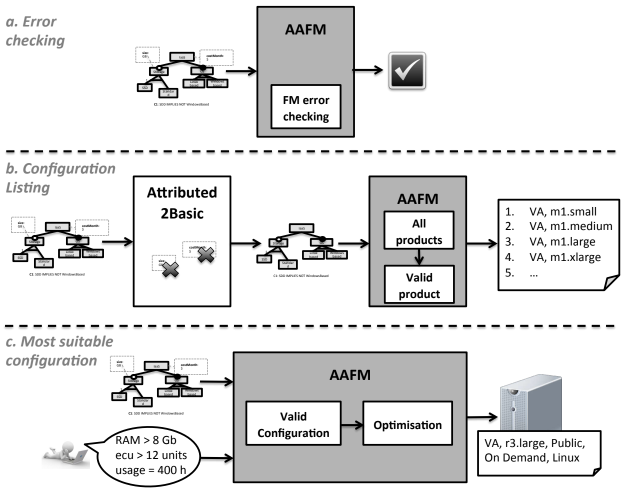

Figure 4: AAFM operations support for EC2 configuration

2. Dead features: a feature that cannot be selected and therefore appears in no product.
3. False-optional features: a feature that despite of being modelled as optional, must be selected as far as its parent feature is selected being a *de facto* mandatory feature.
4. Wrong cardinals: any number of child features in a set relationship that cannot be selected at the same time.

Just in case any error were found, the AAFM provide an error explanation operation that proposes different reasons why the errors appear in a model. These explanations can be used to identify the source of the errors and repair them manually.

6.2. Configurations Listing

Despite error checking is a good practice that increases the confidence in the correctness of a FM, it does not guarantee that the FM represents all and just all the products in a configuration space. We need to list all the available configurations in the IaaS FM and check each of them against the retrieved information of the IaaS provider. We explain how we check them or the particular case of EC2 in Section 7.

Throughout this paper, we have been referring to the enormous size of the EC2 configuration space. In particular, we have referred to the potential number of configurations (22,848 for EC2), i.e. all the possible combinations of the different values of its configuration points, and to the available configurations (16,991). The reason is that not all the potential configurations are really available. For example, some locations and purchasing modes exclude the availability of several instance types. In order to obtain the list of available configurations, we use the *All Products* operation [12]. This operation, as its name denotes, returns the list of products that a FM represents. But first we still need to match the concepts of IaaS configuration and FM product.

We consider a *configuration* as the set of required decisions to make in order to run a cloud service instance. In the case of an IaaS computing instance, such decisions match the configuration points previously defined: OS, instance type, dedication, location and purchasing mode. Additionally, for EC2 we have to decide also on the dedication. In a FM, a product is a set of selected features that satisfies all the relationships and constraints [12]. It is necessary to remark that this definition matches for basic FMs that contain no attributes. This makes necessary to review the product concept for attributed FMs.

In an attributed FM, users can not only make any decision about features, but also about attribute values. A product is defined when a decision is made about all the features and yet all the relationships and constraints are satisfied. Due to the reasoning techniques used to perform the AAFM, it is not possible to list the products defined by an attributed FM with precision. An approximated way to perform this operation consists of removing attributes and their constraints from the model to obtain a basic FM just with features and relationships. Using this basic FM, we can perform the All Products operation to get the list of the available products of an attributed FM.

In an attributed FM, attribute constraints could discard certain combinations of features because of the values of their linked attributes. When attributes and constraints are removed from the FM, these combinations are available again, so we cannot expect that the set of basic products coincides with the set of attributed products. In order to check if a basic product is also a valid attributed product, the *Valid Configuration* operation can be used to check if it satisfies all the constraints in the attributed FM. Repeating this operation for each basic product in the list, we obtain the list of attributed products that satisfies all the relationships and constraints in an IaaS FM.

Figure 4.b depicts the process to obtain the list of IaaS configurations. First, the IaaS FM is transformed into a basic FM, removing all the attributes and their related constraints. In the specific case of EC2, EBS feature is also removed, since in this point we are only interested in EC2 configurations. Afterwards, we invoke All Products operation, which returns the list of available basic products, and for each of them, we check if it is valid for the attributed model. In this way, we obtain the number of total EC2 configuration: 16,991

6.3. Most Suitable Configuration

Once the FM is checked to define the IaaS configuration space correctly, we can perform any analysis operation with confidence. The main goal of this work is assisting the users in the search of the most suitable configuration. Previously, we have defined the most suitable configuration as the one that satisfies customer requirements while the cost is minimised. This problem can be interpreted as an optimisation problem, where user requirements are hard constraints and the optimisation criterion is minimising the cost.

Prior to solving any optimisation problem, we should check that the user requirements are valid and define at least one configuration in the configuration space. The AAFM provides the *Valid Partial Configuration* operation that takes a FM and a set of user decisions as inputs, and returns a boolean value indicating if these decisions are consistent with the FM and therefore it is possible to find at least one product for such requirements.

Once the requirements are checked for validity, an optimisation problem can be defined and solved using the *Optimisation* operation. The optimisation operation takes as inputs a FM, a set of user decisions and an objective function and returns a configuration that satisfies the decisions and optimises the function. In our case, the objective function refers to the minimal `TotalCost` attribute value. Figure 4.c shows our proposed approach.

7. Implementation and Verification

In this section, we describe the implementation and verification details of our approach. First, present the implementation of the analysis operations of Section 6. Second, we briefly motivate and describe the web scraper we employ to build the EC2 FM. And finally, we detail the implementation and verification of the EC2 FM.

7.1. Analysis Operations Implementation

The analysis operations proposed in Section 6 are implemented in the *FaMa Framework*. FaMa Framework[4] [25] is an AAFM open-source tool that supports more than 20 different analysis operations with 4 different reasoners. What we had to develop was the proposed transformation from attributed FMs to basic FMs in Section 6, in order to obtain the number of configurations of an IaaS FM.

FaMa employs different logical paradigms, such as propositional logic or CSP, to perform the AAFM. This means that the FM is translated to logic variables and constraints, which are interpreted and resolved by off-the-shelf solvers. Features and attributes are mapped as variables, while relationships and constraints are mapped as different kinds of constraints. Depending on the AAFM operation, additional inputs can be mapped, and different operations are invoked on the solver. More details about the AAFM and its mappings, and details about the operations are available in [12].

However, the traditional mapping of the AAFM– and consequently FaMa – presents performance issues for specific operations, and especially for the Optimisation operation. For this reason, we have decided to add a new reasoner

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

to FaMa, based on the CPLEX CP Optimizer solver [5], to tackle these performance problems. This new reasoner implements the Optimisation operation, and present three main changes:

- *Alternative FM mapping.* Traditionally, the default mapping to CSP that solvers proposed in the literature maps each feature to a boolean variable [11]. This is the mapping that all the FaMa reasoners adopt. In our alternative implementation, however, we have mapped each of the six configuration options of the EC2 FM as an integer variable. Each variable has as many values as leaf features the configuration option has.
- *CPLEX tuples.* By default, every FM constraint is translated to a solver constraint by the FaMa reasoners. Nonetheless, in models with thousands of constraints this may lead to an overhead in the solvers. For this kind of models, CPLEX provides tuples, which improve dramatically the performance by describing all the elements within a set one by one. Our alternative implementation represents the availability and pricing constraints using CPLEX tuples. We also modelled attributes internally as integers, since CPLEX tuples lack support for real numbers. Anyway, the impact in the accuracy of the total cost is negligible, as Table 8 shows.
- *Search strategy.* We use a customised search strategy, defining the configuration options of EC2 as the decision variables of the problem.

7.2. EC2 Web Scraper

As stated in Section 5, the EC2 FM presents more than 20,000 constraints. Most of such constraints determine the exact cost hour of each configuration, and the configurations that are not available for different reasons. The constraints are presented by AWS in tables in its EC2 website, but its number make its manual processing tedious and error-prone.

The retrieval of the pricing and availability constraints from Amazon EC2 can be considered as an additional challenge. There is no standard interface to retrieve this information directly from the provider. This requires the use of automated techniques, such as web scraping [26] on the provider's website, to get the data. Although we perform this technique to build the model, it is an implementation aspect to verify our proposal – it is a means to our end but not the end itself. Another issue related to the information retrieval is the evolution or variation of the configuration space. This happens often in commercial cloud services, and requires manual work to adjust the scraper and or update manually the model. We are aware of this limitation, and therefore in this paper we work with a snapshot of Amazon EC2 from 12th June 2014.

We have developed a web scraper to automatically extract the constraints from the EC2 website. We have to remark that this is a basic and non-adaptable scraper, i.e. it is not intended to adapt itself for the changes and evolution of the EC2 website, but to automate the constraints extraction. The scraper is developed in Java using the jSoup library [6] and takes as input the FM tree and attributes, and their correspondence with the elements in the EC2 website. It processes the characteristics and pricing tables, and produces as output the characteristics of each instance type – RAM, ecu and cores – and the pricing tables of EC2 as FM constraints in the FaMa plain text format – see next subsection for this format. In the case that a configuration does not appear in the pricing tables it means that is unavailable, so the scraper generates a constraint to remove such configuration from the EC2 FM.

This scraper only extracts instance characteristics, pricing and availability constraints. Changes in the instance pricing or availability are supported by our EC2 scraper. However, any change in the configuration options or their values, or in the presentation or format of the EC2 website would require manual fixes on the scraper. Indeed, during the realisation of this paper, we have made adjustments on the scraper two times due to changes in the website format and the addition of new instances.

7.3. EC2 FM

Although there are several formats to define FMs, we have chosen the FaMa plain text format [25, 27]. The most well known notation is the so-named feature diagram, employed for the excerpt of Figure 3. This format is adequate for illustration purposes and simple FMs. However, in our case, our EC2 FM is large and complex enough to use a text

[5]`http://www-01.ibm.com/software/commerce/optimization/cplex-cp-optimizer/`
[6]`http://jsoup.org/`

format. There are different alternatives for FM text formats [28], but only a few of them provide automated analysis support. Since we take advantage of the analysis operations already provided by FaMa, we use the FaMa notation to define the EC2 FM in plain text.

The proposed methodology to build an IaaS FM relies on a deep understanding of the service and on automated information retrieval mechanisms. While in this paper we have implemented a basic scraper for the latter, some errors may appear due to the former during the definition of the tree, the attributes or some constraints. Consequently, we find useful to employ some of the analysis operations of Section 6 to verify the model. On the one side, we want to calculate the completeness and correctness of the EC2 FM with respect to the retrieved configuration space information. On the other side, we want to ensure that the EC2 FM is free of FM errors, as described also in Section 6. Note that finding errors in the FM may not imply that there are errors on the building on the model, but on the configuration space of the service.

Completeness and Correctness

We need to ensure that the EC2 FM gathers all the available EC2 configurations, but no more. That is, we need to check the completeness and correctness of the EC2 FM. For this purpose, we employ several of the operations explained in Section 6 and Figure 4. First, we parse the tables of the EC2 website to get the EC2 available configurations. Then, we check that each and every retrieved configuration is contained in the EC2 FM, by means of the Valid Configuration operation. In this way, we measure the completeness of the EC2 FM. If all the valid EC2 configurations are represented in the FM, we get the number of configurations of the FM, and compare with the number of configurations retrieved from the tables. If the number is the same, we can say that our EC2 FM is correct and complete.

The EC2 FM required several iterations to correctly represent the configuration space. Initially, the model represented several spurious configurations, due to some unavailable configurations are just excluded from the EC2 pricing tables instead of be marked as "N/A". After three rounds of fixes, we made the necessary changes to remove the fake configurations and consider all the available ones.

Error Checking

In this step, we verify, using FaMa and its analysis operations, the validity of the EC2 FM. As stated in Section 6.1, the error checking operation detects behaviours that suggest a wrong modelling, or errors in the configuration space. The execution of this operation in FaMa detects no errors.

8. Evaluation

In this section we compare, in terms of optimal cost and expressiveness, the results obtained by our approach with the results obtained in commercial tools, in particular CloudScreener (CS) and Amazon TCO. Besides, we present an experimental study to check the performance of our prototype.

8.1. Comparison to Other Approaches

Our first comparison study shows that, in terms of expressiveness, our EC2 FM provides the users more freedom to express decisions than two "competitors": AWS TCO tool and CloudScreener. AWS TCO [3] is a tool provided by Amazon to "compare the cost of running your applications in an on-premises or collocation environment to AWS". CloudScreener [5] is a web application to "*easily identify your need in terms of infrastructure and compare available offers upon many criteria*". While both of them provide web interfaces, we use the EC2 FM for expressing user requirements. Table 5 shows a comparison about the main elements of the three approaches. Except for choosing among different providers, our approach presents more configuration options for the measured configuration aspects. For instance, AWS TCO ignores all the OS but Linux. About CloudScreener, for example, they only offer four fixed values to express hours per month information.

The second comparison study, about the optimality of the results, shows how our analysis approach improves the results of CloudScreener, which sometimes even violates the user requirements. For this comparison, we have taken 5 test cases. We have used them as input for our analysis, and also for the web application of CloudScreener using the "*expert mode*". The inputs and results of the comparison are shown in Table 6, and a record of the process is also

Table 5: Expressiveness comparison among EC2 FM, Amazon TCO and CloudScreener.

	EC2 FM	AWS TCO	CS
Multi-provider	No	No	Yes
OS	Generic & specific	Linux	Generic
Location	State & Continent	AWS areas	Area
Dedication	Yes	No	No
Usage data	hours/ month	% of 3 years	4 values
Period	1 to 48 months	% of 3 years	1, 3, 6, 12, 24, 36 months
RAM	GB	GB	GB
Comp. unit	ecu, cores	processors, cores	CS units
Storage	GB	GB	GB

Table 6: Comparison of results with CloudScreener. m3.large instances present 7.5 GB of RAM, while c3.xlarge instances present 14 ECU.

					Cloud Screener			Our approach	
	OS	**Area**	**ECU/CSPU**	**RAM**	**Config**	**Cost/h**	**Sat. reqs**	**Config**	**Cost/h**
C1	Linux	Virginia / US East	12/6	8	m3.large	0.14 $	No	m3.xlarge	0.28 $
C2	Linux	South America	4/2	9	m3.xlarge	0.381 $	Yes	m2.xlarge	0.323 $
C3	Windows	Singapore	8/4	40	hs1.8xlarge	5.901 $	Yes	r3.2xlarge	1.292 $
C4	Linux	California / US West	68/34	160	r3.8xlarge	3.12 $	Yes	r3.8xlarge	3.12 $
C5	Windows	Europe	16/8	5	c3.xlarge	0.376 $	No	c3.2xlarge	0.752 $

available [7]. In order to facilitate the comparison, we have even set as constant the following properties: Storage ≥ 0, period = 1, usage = 24 h/day = 730 hours/month, and dedication = public, since CloudScreener ignores dedicated instances. CloudScreener uses an own unit, CSPU, to define the computing power, as "an equivalent to EC2 unit (ECU)". Therefore, we divide by 2 the ECU of our inputs when translating to CloudScreener. As Table 6 shows, our approach improves CloudScreener results for four of the five test cases. CloudScreener results for C1 and C5 are not even valid, since they violate customer requirements – RAM for C1 and ECU for C5 –. For C2, and specially for C3, our analysis improves the instance selection, obtaining a cheaper instance that still satisfies user needs.

Table 7: Experimental settings groups and ranges.

Cases	*ECU*	*RAM*	*Disk*	*Usage*	*Period*	*SSD*	*OS*	*Location*	*Dedicated*
Group 1	[1,20]	[1,10]	[1, 1 000]	[1, 730]	[1, 36]	bool	Enum	Enum	bool
Group 2	[6,60]	[10,40]							
Group 3	[20,108]	[40,244]	[1 000, 10 000]						

8.2. *Performance Study*

Although the migration is not a real-time task, we want to ensure that our approach performs the analysis in a reasonable time.

[7] `http://youtu.be/apQmFV5ilUA`

With the intention of testing our analysis approach and its performance, we have implemented an automated generator of user requirements for EC2. This generator produces decisions about the main configuration options described at the end of Section 5, that is, location, OS, dedication, instance characteristics and usage data. Most of the decisions are randomised on the domain of the configuration option, as Table 7 shows. For instance, location value may be a specific area like Virginia, or a group of areas like NorthAmerica. In contrast, we have managed in a different way RAM, ECU and storage values. Such properties have large domains, but instance types are distributed exponentially along the domain, as Figure 5 shows with its logarithmic scale. Consequently, we have defined three different groups of test cases, in order to aim for areas where different instances can satisfy customer requirements. We have generated 200 test cases for area, for a total of 600 test cases.

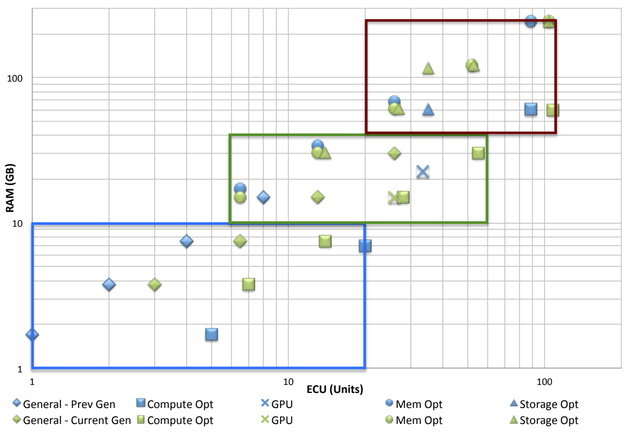

Figure 5: Instance types distribution in terms of RAM and ECU (log10 scale). Each coloured rectangle denotes a different area of test cases.

Both analysis approaches – FaMa and CPLEX-based reasoner – produce the same output configurations, but in terms of performance, our CPLEX-based reasoner outperforms FaMa up to several orders of magnitude. While FaMa lasts for minutes in most of the cases, the alternative implementation outputs the same result in less than a second. As shown in Table 8, the average difference in the analysis time is three orders of magnitude (10^3), while the standard deviation follows the same scale. Total cost errors, due to the round of real variables to integer variables, are negligible, since average and standard deviation values are around cents of dollars.

Table 8: Comparison between FaMa based impl and our alternative impl in terms of performance and output

	Performance (ms)		**Total cost diff.($)**	
Impl.	Avg. (μ)	SD (σ)	Avg. (μ)	SD (σ)
FaMa	1.73×10^5	1.9×10^5	0.4	0.33
Alternative	5.19×10^2	2.15×10^2		

9. Related work

9.1. Cloud Migration

Jamshidi *et al.* [13] present a systematic literature review of existing research works (up to 23) about planning, executing and validating migration of legacy systems towards cloud-based software. The authors identify a reference model for the migration process, where they classify the different works depending on their scope. They also identify a general lack of tool support for the migration process, which we try to palliate in this paper. Frey et al. [29, 6, 30] propose an overall migration approach, based on the concept of CDO (Cloud Deployment Option). A CDO represents "a combination of a specific cloud environment, deployment architecture, and runtime reconfiguration rules for dynamic resource scaling". This approach provides CDOSim [29], a simulator to evaluate CDOs costs and response times, and CDOXplorer [6], an evolutionary algorithm to search for well-suited CDOs in terms of response time, cost and SLA violations. Both approaches are integrated in CloudMig [30], a suite to assist the migration of applications to the cloud. Differently from our work, which is focused on infrastructure migration, they are concerned about migrating applications. Khajeh-Hosseini et al. [31, 32] describe the challenges the users face when they plan to adopt the cloud. They introduce the cloud adoption toolkit to tackle such challenges, which provides a framework to assist the users in the decision process. This tool provides support to analyse, among others, technology suitability, energy consumption and cost prediction of providers and configurations. However, they provide exhaustive information rather than automating the search for the most suitable configuration. CloudGenius framework, an approach by Menzel and Ranjan [33], provides a process and decision support for migrating to the cloud. Their approach comprises a formal mathematical model and a migration process, whose goal is to lead to a VM image and cloud infrastructure selections. However, the tool support is still in a preliminary stage. Beserra et al. [14] present CloudStep, a decision process to support the migration of legacy applications to the cloud. CloudStep relies on template-based profiles of the migrating company, its legacy application and the candidate providers. Such profiles are cross analysed to identify and solve constraints, and to create a migration strategy. Kwon and Tilevich [34] propose an automated transitioning for applications to use cloud-based services. This approach can be seen as a crosscutting concern of the migration to the cloud, and in particular of our proposal to assist the configuration.

There is a number of works that focus on algorithms to automatically evaluate or rank cloud services configurations. Truong and Dustdar [35] present a service for estimating, monitoring and analysing costs associated with scientific applications in the cloud. They rely on cost models and experiment with real-world applications. Trummer et al. [17] interpret the outsourcing of part of the IT-stack as a constraint optimisation problem, that they solve using existing solvers. Tsai et al. [18] consider 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 [16] propose indexing and ranking cloud providers using a set of algorithms based on user preferences. Venticinque et al. [36] describe an approach to collect cloud resources from different providers that continuously meet requirements of user applications. The related work of Borgetto et al. [37] is oriented to software reallocation in different virtual machines in order to decrease energy consumption.

Some other works propose processes and methodologies to assist the migration, and identify factors and tasks. The work of Mohagheghi and Sather [38] presents some software engineering challenges related to migrating legacy systems to cloud services. They identify application modernization and understanding cloud technologies as the main issues. Tak et al. [7] identify cost key factors when migrating to the cloud. Different deployment alternatives, based in Azure and Amazon services, are benchmarked and detailed in terms of cost. Lloyd et al. [39] discuss which factors should be accounted when deploying to a cloud. The authors focus on bottlenecks which appear when scaling applications, and the impact of provisioning variation. Zardari and Bahsoon [40] rely on goal oriented requirements engineering (GORE) and specific tasks to assist users in the adoption of cloud services.

9.2. Variability, Ontologies and Cloud Services

Applying Software Product Lines and variability techniques to cloud services is attracting attention. Quinton et al. [15] propose a software product lines based approach that supports stakeholders while configuring a cloud environment and automates the deployment of such configurations. They also use FMs for the configuration process, but while they consider features as deployable artefacts of cloud environments, we consider features as configuration points and specific values of commercial cloud services. In the same line, Schroeter et al. [41] use 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. [42] 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. [43] proposes the extension of traditional software product lines with cloud computing aspects. Our work differs from these approaches in a key way: while they focus on deployable cloud environments, and consider features as deployable artefacts, we focus on commercial cloud services, and consider features as configuration points and specific values of the services. Wittern et al. [44, 45] present the so-named service feature modelling, for the representation of service design concerns, and to capture their potential combinations. They also also provides a method [45] to rank service design alternatives based on stakeholder preferences. While they focus on capturing diverse aspects of cloud services in a general way, we focus on the complete modelling and analysis of a particular service, EC2, and compare our approach against commercial applications.

The use of ontologies has been proposed by several authors to assist the modelling and selection of cloud services. Han et al. [46] construct a cloud ontology consisting of a taxonomy of concepts of Cloud services, in order to support a cloud service discovery system. Dastjerdi et al. [47] propose an ontology-based discovery to provide QoS aware deployment of appliances on Cloud service providers. Ngan et al. [48] also present a semantic cloud service discovery and selection system, based on OWL-S. The proposed system supports dynamic semantic matching of cloud services described with complex constraints. Garcia-Rodriguez et al. [49] propose a semantically-enhanced platform to assist the process of discovering the cloud services, stored in a semantic repository, that best match user needs. While ontologies are an excellent option for a modelling approach, they present performance issues and reasoning lacks on integer and real variables, which are inherent to the problem we tackle in this paper.

9.3. Commercial Approaches

Configuring and analysing cloud platforms/providers is continuing to receive significant attention also from the business community. Besides previously referred CloudScreener [5], there are a number of companies and start-ups providing comparisons, benchmarks and configuration support for cloud services. Cloudorado [50] also provides a cloud services price comparison engine. CloudHarmony [1] is a startup which looks for obtaining metrics about cloud providers performance, and provides a comparison framework for many services providers. PlanForCloud [51] is another start-up, 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.

As shown in the evaluation section, our proposal improves the expressiveness and configuration results returned by some of these approaches. In particular, CloudScreener returned false positives for the search of the best configuration that we improved with our analysis. However, we still lack cross-provider support. In future works, where we will consider different cloud providers and metrics, we may extend our evaluation with some of these approaches.

10. Conclusions

We have presented an approach to assist the configuration selection when migrating an in-house computing infrastructure to the cloud. In terms of expressiveness, our modelling approach provides greater degree of freedom for the customer when making decisions about such configuration, compared to commercial applications like AWS TCO and CloudScreener. In terms of accuracy, we have proved that our analysis approach improves results compared to CloudScreener, which sometimes even violates user requirements. In terms of performance, we have implemented an analysis prototype to obtain the most suitable configuration whose execution times are negligible, being most of the times less than a second.

For such purposes, we have presented a modelling methodology to describe an IaaS as a FM. Modelling the configuration space of a large cloud provider, such as AWS, as a FM provides a well-structured and compact representation to express user infrastructure requirements. The resulting model still contains, for the particular case of EC2, lots of information, with a huge number of constraints – more than 20,000. The addition of abstract features and attributes eases the decision making, since information like RAM, period of commitment or total storage is closer to the user than specific instance types, purchasing options or EBS extra storage.

We have also proposed the use of the AAFM to assist the configuration when migrating to an IaaS. With the support of the AAFM operations, user requirements can be validated and employed to obtain suitable EC2 configurations. From the variability management perspective, we have (1) demonstrated the applicability of the AAFM for the configuration cloud services, (2) defined the product listing for attributed FMs, and (3) proved that off-the-shelf AAFM mappings, such as the one provided by FaMa framework, can be improved to dramatically reduce execution times.

Our approach can be generalised to other scenarios – including those considered in this work, primarily focusing on a migration to the cloud. In a recent work [52], we propose the adaptation of a multi-tenant service to satisfy the changing needs of the users. In this sense, our analysis approach could be applied to IaaS reconfiguration scenarios where the user needs to change and the current configuration becomes unsuitable. Our analysis approach could also be integrated with existing tools. An integration, for instance with the AWS calculator, would provide us with a reliable way to get the exact pricing of EC2, and it would provide the calculator with capabilities for the automated search of the most suitable configuration for given needs. The modelling methodology has also the potential for a cross-provider analysis. However, in this case, we would have to propose a common IaaS FM, and also address issues related to *ontology alignment* between the concepts of different providers.

However, our approach still has limitations and room for improvement, that may motivate new works in this area. In addition to a cross-provider analysis, we think that FMs and our modelling methodology could be employed for different cloud services. We also consider defining a metamodel for the configuration of cloud services, based on FMs, but with specific mechanisms to get beyond some limitations of FMs and the current language. For instance, there should be a better and more compact way to define price policies and temporal-aware requirements [53, 54]. It is also necessary to enable a way to express constraints over several instances of the same services, such as the discount policies for multiple hired services, which is a current limitation of FMs. On the customer's side, FMs lack mechanisms to express fuzzy needs or preferences [55] as well as requirements. An extension would provide users with greater capability to specify their needs. The aforementioned AWS calculator could be used also to verify that we have described the EC2 pricing policy correctly. Finally, we think that we can derive additional benefits from the AAFM. It is necessary to analyse in depth the applicability of all the existing operations of AAFM to the services field.

ACKNOWLEDGEMENTS

Special thanks to Pablo Fernandez for his support in the tooling of the paper. This work was partially supported by the European Commission (FEDER), the Spanish and the Andalusian R&D programmes (grants TIN2012-32273 (TAPAS), P12-TIC-1867 (COPAS) and TIC-5906 (THEOS)).

[1] CloudHarmony, Cloud Harmony, http://cloudharmony.com/ (2014).
[2] A. W. Services, Amazon Elastic Compute Cloud (EC2), http://aws.amazon.com/ec2 (2014).
[3] Amazon, AWS Total Cost of Ownership (TCO) Calculator, https://awstcocalculator.com (2014).
[4] Rackspace, Rackspace Solutions Configurator, http://www.rackspace.co.uk/solutions-configurator (2014).
[5] CloudScreener, Cloud Screener, http://www.cloudscreener.com/ (2014).
[6] S. Frey, F. Fittkau, W. Hasselbring, Search-based genetic optimization for deployment and reconfiguration of software in the cloud, in: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, 2013, pp. 512–521.
[7] B. C. Tak, B. Urgaonkar, A. Sivasubramaniam, To move or not to move: The economics of cloud computing, in: Proceedings of the 3rd USENIX conference on Hot topics in cloud computing, USENIX Association, 2011, pp. 5–5.
[8] J. García-Galán, O. F. Rana, P. Trinidad, A. Ruiz-Cortés, Migrating to the Cloud: a Software Product Line based analysis, in: 3rd International Conference on Cloud Computing and Services Science (CLOSER), 2013, pp. 416–426.
[9] K. Kang, S. Cohen, J. Hess, W. Nowak, S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Software Engineering Institute, 1990.
[10] T. Thum, C. Kastner, S. Erdweg, N. Siegmund, Abstract Features in Feature Modeling, in: Proceedings of the 2011 15th International Software Product Line Conference, 2011, p. 10.
[11] D. Benavides, P. Trinidad, A. Ruiz-Cortés, Automated reasoning on feature models, in: Conference on Advanced Information Systems Engineering (CAiSE), Vol. 01, Springer, 2005, pp. 491–503.
[12] D. Benavides, S. Segura, A. Ruiz-Cortes, Automated analysis of feature models 20 years later: A literature review, Information Systems (2010) 615–636.
[13] P. Jamshidi, A. Ahmad, C. Pahl, Cloud migration research: A systematic review, Cloud Computing, IEEE Transactions on 1 (2) (2013) 142–157. doi:10.1109/TCC.2013.10.
[14] P. V. Beserra, A. Camara, R. Ximenes, A. B. Albuquerque, N. C. Mendonca, Cloudstep: A step-by-step decision process to support legacy application migration to the cloud, in: Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2012 IEEE 6th International Workshop on the, IEEE, 2012, pp. 7–16.
[15] C. Quinton, D. Romero, L. Duchien, Automated selection and configuration of cloud environments using software product lines principles, in: IEEE CLOUD 2014, 2014, p. 8.
[16] S. Sundareswaran, A. Squicciarini, D. Lin, A Brokerage-Based Approach for Cloud Service Selection, in: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, 2012, pp. 558–565.
[17] I. Trummer, F. Leymann, R. Mietzner, W. Binder, Cost-optimal outsourcing of applications into the clouds, in: Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, IEEE, 2010, pp. 135–142.
[18] W.-T. Tsai, G. Qi, Y. Chen, A Cost-Effective Intelligent Configuration Model in Cloud Computing, in: Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on, 2012, pp. 400–408.
[19] Google Inc., Google Compute engine, https://cloud.google.com/compute/ (2014).
[20] Microsoft, Azure Virtual Machines, http://azure.microsoft.com/en-us/services/virtual-machines/ (2014).
[21] Rackspace Inc., Rackspace Servers, http://www.rackspace.com/cloud/servers/ (2014).
[22] K. Lee, K. C. Kang, J. Lee, Concepts and guidelines of feature modeling for product line software engineering, in: Software Reuse: Methods, Techniques, and Tools, Springer, 2002, pp. 62–77.

[23] P. Trinidad, D. Benavides, A. Durán, A. Ruiz-Cortés, M. Toro, Automated error analysis for the agilization of feature modeling, Journal of Systems and Software 81 (6) (2008) 883–896. doi:10.1016/j.jss.2007.10.030.

[24] J. White, D. Benavides, D. C. Schmidt, P. Trinidad, B. Dougherty, Ruiz-Cortes, Automated diagnosis of feature model configurations, Journal of Systems and Software doi:10.1016/j.jss.2010.02.017.

[25] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, A. Jimenez, FAMA Framework, in: 12th Software Product Lines Conference (SPLC), 2008, pp. 359–359.

[26] R. B. Penman, T. Baldwin, D. Martinez, Web scraping made simple with sitescraper (2009).

[27] ISA Research group, Fama tool suite, `http://www.isa.us.es/fama/` (2014).

[28] H. Eichelberger, K. Schmid, A systematic analysis of textual variability modeling languages, in: Proceedings of the 17th International Software Product Line Conference, ACM, 2013, pp. 12–21.

[29] F. Fittkau, S. Frey, W. Hasselbring, Cdosim: Simulating cloud deployment options for software migration support, in: Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2012 IEEE 6th International Workshop on the, IEEE, 2012, pp. 37–46.

[30] S. Frey, W. Hasselbring, The cloudmig approach: Model-based migration of software systems to cloud-optimized applications, International Journal on Advances in Software 4 (3 and 4) (2011) 342–353.

[31] A. Khajeh-Hosseini, D. Greenwood, J. W. Smith, I. Sommerville, The Cloud Adoption Toolkit: supporting cloud adoption decisions in the enterprise, Softw., Pract. Exper. (2012) 447–465.

[32] A. Khajeh-Hosseini, I. Sommerville, J. Bogaerts, P. B. Teregowda, Decision Support Tools for Cloud Migration in the Enterprise, in: IEEE CLOUD Conference, 2011, pp. 541–548.

[33] M. Menzel, R. Ranjan, Cloudgenius: decision support for web server cloud migration, in: Proceedings of the 21st international conference on World Wide Web, ACM, 2012, pp. 979–988.

[34] Y.-W. Kwon, E. Tilevich, Cloud refactoring: automated transitioning to cloud-based services, Automated Software Engineering 21 (3) (2014) 345–372.

[35] H.-L. Truong, S. Dustdar, Composable cost estimation and monitoring for computational applications in cloud computing environments, Procedia Computer Science 1 (1) (2010) 2175–2184.

[36] S. Venticinque, R. Aversa, B. Di Martino, D. Petcu, 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, 2011, pp. 184–191.

[37] D. Borgetto, M. Maurer, G. Da-Costa, J.-M. Pierson, I. Brandic, 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, 2012, p. 10.

[38] P. Mohagheghi, T. Sæther, Software engineering challenges for migration to the service cloud paradigm: Ongoing work in the remics project, in: Services (SERVICES), 2011 IEEE World Congress on, IEEE, 2011, pp. 507–514.

[39] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, K. Rojas, Migration of multi-tier applications to infrastructure-as-a-service clouds: An investigation using kernel-based virtual machines, in: Grid Computing (GRID), 2011 12th IEEE/ACM International Conference on, IEEE, 2011, pp. 137–144.

[40] S. Zardari, R. Bahsoon, Cloud adoption: a goal-oriented requirements engineering approach, in: Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing, ACM, 2011, pp. 29–35.

[41] J. Schroeter, P. Mucha, M. Muth, K. Jugel, M. Lochau, Dynamic configuration management of cloud-based applications, in: Proceedings of the 16th International Software Product Line Conference - Volume 2, 2012, pp. 171–178.

[42] B. Dougherty, J. White, D. C. Schmidt, Model-driven auto-scaling of green cloud computing infrastructure, Future Generation Computer Systems.

[43] E. Cavalcante, A. Almeida, T. Batista, Exploiting software product lines to develop cloud computing applications, in: Software Product Line Conference, 2012, pp. 179–187.

[44] E. Wittern, J. Kuhlenkamp, M. Menzel, Cloud service selection based on variability modeling, in: Service-Oriented Computing, Springer, 2012, pp. 127–141.

[45] E. Wittern, C. Zirpins, Service feature modeling: modeling and participatory ranking of service design alternatives, Software & Systems Modeling (2014) 1–26.

[46] T. Han, K. M. Sim, An ontology-enhanced cloud service discovery system, in: Proc. of the International MultiConference of Engineers and Computer Scientists, Hong Kong, 2010, pp. 17–19.

[47] A. V. Dastjerdi, S. G. Tabatabaei, R. Buyya, An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery, in: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010, pp. 104–112.

[48] L. D. Ngan, R. Kanagasabai, Owl-s based semantic cloud service broker, in: Web Services (ICWS), 2012 IEEE 19th International Conference on, IEEE, 2012, pp. 560–567.

[49] M. Á. Rodríguez-García, R. Valencia-García, F. García-Sánchez, J. J. Samper-Zapater, Ontology-based annotation and retrieval of services in the cloud, Knowledge-Based Systems 56 (2014) 15–25.

[50] Cloudorado, Cloudorado, http://www.cloudorado.com/ (2014).

[51] PlanForCloud, Plan for cloud, http://www.planforcloud.com/ (2014).

[52] J. García-Galán, L. Pasquale, P. Trinidad, A. R. Cortés, User-centric adaptation of multi-tenant services: preference-based analysis for service reconfiguration., in: SEAMS, 2014, pp. 65–74.

[53] O. Martín-Díaz, A. Ruiz-Cortés, A. Durán, C. Müller, An approach to temporal-aware procurement of web services, in: Service-Oriented Computing-ICSOC 2005, Springer, 2005, pp. 170–184.

[54] C. Müller, O. Martín-Díaz, A. Ruiz-Cortés, M. Resinas, P. Fernandez, Improving temporal-awareness of WS-agreement, Springer, 2007.

[55] J. M. García, D. Ruiz, A. Ruiz-Cortés, A model of user preferences for semantic services discovery and ranking, in: The Semantic Web: Research and Applications, Springer, 2010, pp. 1–14.