Component-wise Application Migration in Bidimensional Cross-cloud Environments

Jose Carrasco, Francisco Durán and Ernesto Pimentel
University of Málaga, Málaga, Spain

Keywords: Cloud Applications, Migration, Cross-cloud, Trans-cloud, Brooklyn, Standards, TOSCA.

Abstract: We propose an algorithm for the migration of cloud applications' components between different providers, possibly changing their service level between IaaS and PaaS. Our solution relies on three of the key ingredients of the *trans-cloud* approach: a unified API, agnostic topology descriptions, and mechanisms for the independent specification of providers. We show how our approach allows us to overcome some of the current interoperability and portability issues of cloud environments to propose a solution for migration, present an implementation of our proposed solution, and illustrate it with a case study and experimental results.

1 INTRODUCTION

In recent years, Cloud Computing (Armbrust et al., 2010) has experienced a growth in the demand of its services (Youseff et al., 2008). As an answer to this demand, vendors have developed their own cloud solutions, offering similar resources through their own APIs, defining their own non-functional requirements, quality of service (QoS) specifications, service level agreements (SLA), and add-ons. This heterogeneity has derived into many interoperability and portability restrictions, and provokes situations where cloud developers are often locked-in specific services from cloud providers.

Recent advances in the management of the connections between components deployed using different technologies and vendors (Kritikos and Plexousakis, 2015; Paraiso et al., 2012; Grozev and Buyya, 2014) have solved most of the interoperability issues. We have witnessed the development of deployment platforms with the ability to distribute modules of applications using services from different providers, with the goal of using the better alternative for each of the individual components and for the operation of our applications. The last step in this direction is the possibility of deploying applications combining services from IaaS and PaaS levels, possibly by different vendors in trans-cloud environments (Carrasco et al., 2016). The selection of the vendor or service level to deploy an application from among the multitude of cloud offerings is indeed a challenge (see, e.g., (Androcec et al., 2015; Moustafa et al., 2016; Brogi et al., 2014)). The decision is indeed non-trivial, and the context and required knowledge may change while applications are running.

Once an application is running, its management requires mechanisms to ensure that the chosen cloud providers are delivering the promised computing resources (Qu et al., 2015; Zheng et al., 2014). For example, developers may decide today to use a PaaS provider for a particular module because it is more cost effective, or because it requires less management effort, but tomorrow they may decide to move some component to IaaS level because their needs or business model may require more control over virtual machines (VM), e.g., for a better integration with their enterprise's infrastructure, or because they need to increase the security level of their services. Unfortunately, moving an application's component between different providers is problematic, and it is more difficult between different abstraction levels, since changes in these decisions require some development efforts (Petcu, 2011; Di Martino, 2014), in order to adapt the components to new service requirements and their integration with other application's components, running in other providers.

A migration process requires the orchestration of the entire cloud environment where the application is running, to accomplish the correct movement of components. This process is even more complicated if the current cloud interoperability and portability problems are taken into account. As a result, migration in the cloud opens a lot of new key issues related to cloud resources and applications' components

Carrasco, J., Durán, F. and Pimentel, E.
Component-wise Application Migration in Bidimensional Cross-cloud Environments.
DOI: 10.5220/0006372302870297
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 259-269
ISBN: 978-989-758-243-1

control. In fact, migration is still an unresolved topic which has been widespread studied by both academia and industry (see, e.g., (Jamshidi et al., 2013a; Zhao and Zhou, 2014)). There have been several proposals for what is usually called *live migration of cloud application's components* (Binz et al., 2011; Durán and Salaün, 2016; Boyer et al., 2013), where components of an application, which are already running in some providers, are moved to different vendors or locations. However, these proposals present significant restrictions, mainly due to the portability of the components between different vendors, but also to their interoperability, which forces to provide ad-hoc solutions. Furthermore, these solutions are limited to one specific service level (cf. (Durán and Salaün, 2016; Boyer et al., 2013; Zeginis et al., 2013)).

Migration of individual components or entire applications may indeed be unavoidable over time, because of changes in the offered services, prices, security policies, or simply because a provider just stops providing its services. Once developers can take advantage of the features of different kinds of services, they will be interested as well in optimizing the cloud resources usage and improve their applications' performance. We propose an orchestration algorithm to reach migration of a component between different providers in an agnostic way, what means that it is not bound to any service level, either IaaS or PaaS, of any particular provider. This allows developers to deal with vendor lock-in issues, facilitating the adaptation of the running applications according to their own needs, by moving just the necessary application's components to another services, independently of the target abstraction level, IaaS or PaaS.

In order to ensure the agnosticity of our proposal, the algorithm is built over *trans-cloud* (or bidimensional cross-cloud) concepts (Carrasco et al., 2016). Specifically, (Carrasco et al., 2016) uses the TOSCA standard to model agnostic applications' topologies, which do not use any specifics of the target providers over which they will be deployed. The information related to the cloud service level, IaaS or PaaS, is added by means of a mechanism independent of the topology description: policies. The trans-cloud environment processes these specifications and uses an homogeneization API, which unifies IaaS and PaaS services of different vendors, to orchestrate the deployment of the application over the required cloud services.

Our migration algorithm relies on the trans-cloud infrastructure for the management of each application module and the interaction with the used cloud services, to *stop*, *restart* or *move* the necessary components independently of the service level, IaaS or PaaS, the cloud technology or any other dependencies. In order to preserve the architecture consistency and avoid unexpected situations during the migration, a certain strategy has to be applied while the components are being operated.

The rest of this paper is structured as follows. Preliminaries about trans-cloud deployment and its current implementation are presented in Section 2. The proposed migration algorithm is described in Section 3. Details on the implementation of the algorithm are presented in Section 4, together with some experimental results. In Section 5, we compare our proposal to other related work. Finally, Section 6 conclude the paper and presents some plans for future work.

2 AN OVERVIEW OF TRANS-CLOUD

In this section we provide an overview of trans-cloud and its main capabilities. These concepts are illustrated with a running case study, which will be later used to show the use of the our proposal in Section 3 and to evaluate it in Section 4.

2.1 The Softcare Case Study

Softcare is an application for the social inclusion of elderly people and for the management of their medical problems. The application was developed by Atos Spain in the context of the SeaClouds project (Brogi et al., 2015). Softcare is a cloud-based clinical, educational, and social application, based on state-of-the-art technology.

As depicted in Figure 1, the application is composed of seven modules: four web modules over respective Tomcat servers, namely SoftwareDashboard, SoftwareWS, Multimedia, and Forum (note the Tomcat icons), and three MySQL databases, namely SoftcareDB, MultimediaDB, and ForumDB (note the database icons). The Softcare Dashboard component provides the main graphical user interface, which depends on the Forum, Multimedia and SoftcareWS modules. Forum adds a forum service to the web platform, Multimedia is responsible for managing the offered multimedia content, and SoftcareWS contains the application's business logic. The databases ForumDB, MultimediaDB and SoftcareDB store, respectively, the forum's messages, the multimedia content, and the rest of the application's data.

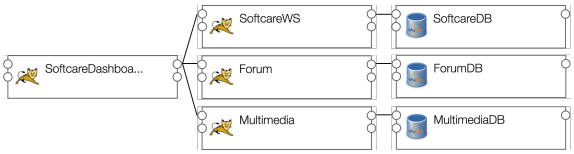


Figure 1: Brooklyn-TOSCA Softcare's topology.

2.2 Trans-Cloud Concepts

The main goal of the trans-cloud management is to allow developers to deploy their applications by using available services and resources offered by different providers, at IaaS or PaaS levels, accordingly each applications needs and preferences.

Trans-cloud reduces significantly the portability and interoperability related issues, liberating developers from the usual infrastructure limitations while defining their applications. The approach relies on three main ideas:

Agnostic Topology Description. The knowledge about applications' components, configurations, interrelations, etc. is specified using agnostic descriptions, without details about cloud providers.

Target Services Specification. Target providers are specified independently of topologies, which allows maintaining agnostic and reusable topology descriptions. It provides a key flexibility to define what services, IaaS or PaaS, will be used to deploy each specific component.

Unified API. Trans-cloud defines an homogeneization API where IaaS and PaaS services management is unified under a common interface. This API mitigates the cloud heterogeneity and provides a vendor general solution, without depending on tools, frameworks or technologies to manage IaaS and PaaS services. Then, once a topology has been defined and the target providers have been specified, the API uses different mechanisms, clients, drivers, etc., to operate with the selected services, IaaS or PaaS, to carry out the application deployment.

These concepts are not just useful for enabling a unified cloud management, they provides an essential basis for carrying out the migration of components: Target locations can be added in a later phase to an application description, and the underlying API is in charge of the management of the necessary resources (using drivers for the cloud technologies and connectors).

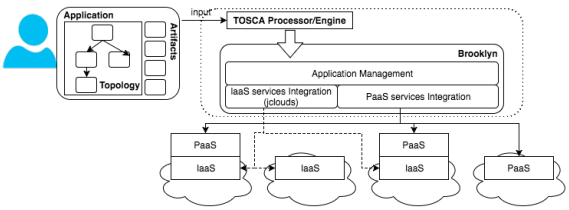


Figure 2: Trans-cloud approach.

2.3 A Trans-Cloud Implementation

The trans-cloud tool presented in (Carrasco et al., 2016) is based on the TOSCA standard[1] for the description of agnostic topologies. Specifically, it builds on the Brooklyn-TOSCA open project for enabling an independent specification of the used services, and on the Apache Brooklyn project to provide a common API for the unified management of IaaS and PaaS services. Figure 2 shows an overview of the proposal in (Carrasco et al., 2016).

The open-source Apache Brooklyn project is a multi-cloud application management platform for the management of the provisioning and deployment of cloud applications. It provides a common API that enables cross-computing features through a unified API. Although the current official release of Brooklyn only handles IaaS services, as can be seen in Figure 2, Brooklyn's API was extended in (Carrasco et al., 2016) with new mechanisms for the management of PaaS services. Behind its current API, we have allocated the behavior to describe PaaS providers and mechanisms to control application components on them. Our extension to provide support for PaaS services builds on the genericity and flexibility of Brooklyns API, which has the independency between application descriptions and cloud services used in their operation as one of its goals. Initially, CloudFoundry-based platforms, like Pivotal Web Services, IBM Bluemix, etc., were integrated, to prototype the PaaS support, and to allow components to be deployed using both IaaS and PaaS.

Brooklyn-TOSCA is an open project with the goal of extending Brooklyn with the capability of deploying and managing applications and cloud resources through TOSCA concepts. Brooklyn-TOSCA promotes an initiative to build agnostic TOSCA topologies by means of expressing the target services using TOSCA policies.

Listing 1 shows Softcare's TOSCA YAML topology schema, where just some elements are described

[1] TOSCA (Topology and Orchestration Specification for Cloud Applications) is an OASIS standard for the description of cloud applications, the corresponding services and their relationships.

to illustrate the agnostic-based application description. As we can see in lines 29–37, this definition follows the Brooklyn-TOSCA initiative that allows target cloud services to be specified by means of policies (brooklyn.location). In this case, we can see how two groups have been defined to be deployed on IaaS, specifically on AWS (Ireland's cluster) and SoftLayer (London's cluster). This mechanism allows the separation between topology description and the providers specification. In fact, if we decided to re-deploy the application, but using different providers, we would just need to change the corresponding locations, without modifying the original topology, as we can see in Listing 2, where Pivotal (PaaS) is used to deploy some of the components.

As we will see in the following section, the trans-cloud approach provides a set of useful basic mechanisms to build an agnostic algorithm to reach the migration of application's component.

3 MIGRATION ALGORITHM

In this section, we present our algorithm for the agnostic reconfiguration of cloud applications' components. It effectively connects the components as required, stopping, starting, releasing and provisioning necessary cloud resources and respecting the functional dependencies.

3.1 Description of the Algorithm

Before presenting the algorithm itself in Section 3.2, we provide in this section some insights into how the algorithm works on our case study. Specifically, we describe its operation step by step by looking at how the Forum component of the Softcare case study can be migrated.

Let us assume that the Softcare application has been described and deployed following the trans-cloud approach. That is, there is an agnostic description of it and its components, e.g., using TOSCA, as in Section 2. In that description, the only reference to the concrete locations on which the components were to be deployed, or even whether they were using IaaS or PaaS services, were given as policies in the specification of the groups of the components. The trans-cloud infrastructure is in charge of hiding the management of the application's components and the vendors and the abstraction levels. In fact, this infrastructure does not only manage the module to be deployed or migrated, but also the related cloud resources. It identifies interdependencies between components and is in charge of handling them, both in the deployment

```
1  tosca_definitions_version:
       tosca_simple_yaml_1_0_0_wd03
2  ...
3  topology_template:
4   node_templates:
5    SoftcareDashboard:
6     type: org.apache.brooklyn.entity.webapp.
            tomcat.TomcatServer
7      ...
8     requirements:
9      - endpoint_configuration:
10         node: SoftcareWS
11         ...
12     - endpoint_configuration:
13         node: Forum
14         ...
15     - endpoint_configuration:
16         node: Multimedia
17         ...
18   SoftcareWS:
19    type: org.apache.brooklyn.entity.webapp.
            tomcat.TomcatServer
20     ...
21    requirements:
22     - endpoint_configuration:
23         node: SoftcareDB
24         ...
25   SoftcareDB:
26    type: org.apache.brooklyn.entity.database.
            mysql.MySqlNode
27    ...
28  ...
29  groups:
30   add_compute_locations:
31    members: [SoftcareDB, ForumDB,
            MultimediaDB, Forum]
32    policies:
33    - brooklyn.location: aws-ec2:eu-west-1
34   add_web_locations:
35    members: [SoftcareDashboard, SoftcareWS,
            Multimedia]
36    policies:
37    - brooklyn.location: softlayer:lon02
```

Code 1: Softcare's TOSCA description.

and the migration process, ensuring the integrity of the topology.

Given the deployment plan used for the Softcare application in Section 2.3, let us assume that the Forum component is running in AWS EC2 (IaaS). Now, assume we want to move it to Pivotal Web Services (PaaS). The sequence of steps followed by the migration process are depicted in the diagram in Figure 3. The diagram shows a Migration Orchestrator element that receives a migration request to move the Forum component to Pivotal Web Services, and is in charge of controlling the migration process.

Once the migration request is received, the migration orchestrator stops all elements that have functional dependencies with the Forum component—

```
...
groups:
  add_compute_locations:
    members: [SoftcareDB, ForumDB,
          MultimediaDB, Forum]
    policies:
    - brooklyn.location: aws-ec2:eu-west-1
  add_web_locations:
    members: [SoftcareDashboard, SoftcareWS,
          Multimedia]
    policies:
    - brooklyn.location: pivotal-ws
```

Code 2: Adding new locations to web modules.

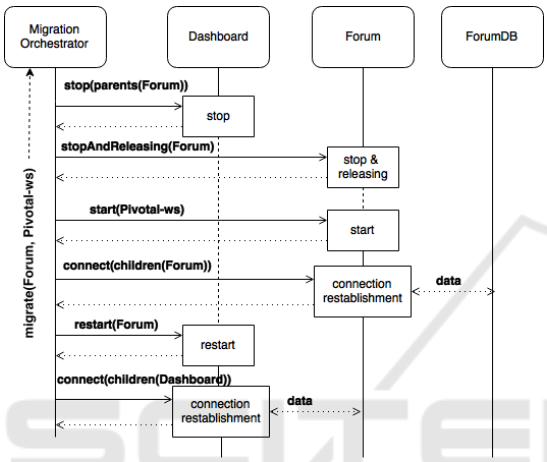


Figure 3: Forum migration process.

Forum's parents—to avoid scenarios where a component is working without some of its dependencies. Finding all parents of a module might be a complex and expensive task, since functionalities are defined as connections which are configured and stablished using different mechanisms, for example, environment variables, configuration files, etc. However, in the trans-cloud approach, there is a complete description of the application's topology, where all the relations and dependencies are specified.

Since each of the components of an application may be running on services of different providers, either IaaS or PaaS, an operation on them will be dependent on its specific case. For example, to stop a server that is running on a VM in IaaS, probably a specific command should be executed on the VM using ssh, or, if the piece of software is running on a PaaS environment, a concrete REST web service of the platforms API should be called to stop the piece of software. However, thanks to its unified API, which hides the cloud heterogeneity, the trans-cloud approach greatly simplifies the management of different cloud services.

Once all parents have been stopped, the next step is stopping the component that has to be migrated, the Forum component in this case (Step 2 in Figure 3). However, only stopping the component is not enough, since the resources that are being used by the Forum component in AWS will not be used again once the component is moved to its new location, and therefore it is also necessary to release all the associated resources. Again, the trans-cloud capabilities allows our algorithm to be designed without worrying about the risks of managing different providers, since it allows us to stop the application component and release the associated cloud resources independently of the vendor or the abstraction.

Once the component is stopped, and the cloud resources have been released, a new instance of the Forum component is started in the new location (Step 3). Once more, the trans-cloud mechanisms are key to accomplish this task in a very straightforward way. The agnostic topology of the application contains all the information about the structure of the application, which is used, together with the information on the target providers, by the unified API, to deploy the component: The description of any component (Forum) is provided by the original topology, and the new target location (Pivotal Web Service) is given as an argument of the operation.

With the component started in its new location, it is still necessary to re-establish the connections of its functional dependencies to maintain the structural integrity of the application (Step 4). In our example, once the Forum component in running in Pivotal, it is necessary to re-establish its connection with the ForumDB component. The trans-cloud environment is able to analyze the application topology, find the necessary relations for the newly migrated component and re-establish the connections with the other components in the topology independently of the cloud environments where the components are running. For our example, since Pivotal offers PaaS services, connections between components are modeled using environment variables.

The last step consist in restarting all application's elements stopped in Step 1. The trans-cloud mechanisms makes the restarting of the necessary components and the re-establishment of their connections (Steps 5 and 6) straightforward. The topology of the application is analized and the parents of the Forum component (Dashboard) are restarted. The new Forum component's endpoint, which is provided now by Pivotal Web Services, is used to re-establish the connection.

3.2 Algorithm Specification

As we have seen in the previous section, the trans-cloud approach provides the necessary ingredients for the definition of an agnostic migration algorithm, where the diversity of the cloud and the complex management of applications' components is delegated to the different mechanisms provided by the trans-cloud infrastructure.

Our migration algorithm is specified in Algorithm 1. It is completely agnostic, it is just a process orchestrator. Given an application, the component to be migrated, and the target location for such component, the migration orchestrator defines an operational plan for the migration process, delegating the management of each concrete application's component and bound cloud resource to trans-cloud mechanisms.

Algorithm 1: Migration Algorithm.

```
Input: a : application
Input: c : component to migrate
Input: l : new location for the component
 1: procedure MIGRATE(a, c, l)
 2:     for parent: parents(a,c) do
 3:         STOPPARENTS(a,parent)
 4:     stopAndReleaseResources(a,c)
 5:     start(a,c,l)
 6:     for child : children(a,c) do
 7:         restablishRelations(a,c,child)
 8:     for parent: parents(a,c) do
 9:         RESTARTPARENTS(a,parent)
10: procedure STOPPARENTS(a, c)
11:     for parent: parents(a,c) do
12:         STOPPARENTS(a,parent)
13:     stop(a,c)
14: procedure RESTARTPARENTS(a, c)
15:     restart(a,c)
16:     for child : children(a,c) do
17:         reestablishRelations(a,c,child)
18:     for parent: parents(a,c) do
19:         RESTARTPARENTS(a,parent)
```

The operation MIGRATE(a,c,l) receives as input the application to operate on a, the component to be migrated c, and the target location l. Before migrating a component, it is necessary to stop all its input dependencies (lines 2-3). STOPPARENTS (lines 10-13) is a recursive procedure that stops all the ancestors of a given component following a top-down strategy, that is, it stops a component once all its parents have been previously stopped. The $stop(a,c)$ operation (line 15), provided by the trans-cloud infrastructure, stops an application's component.

Once all parents have been stopped, the component to be migrated is stopped and all bound resources are released using the $stopAndReleaseResources(a,c)$ operation provided by the trans-cloud infrastructure (line 4). Then, the target component is started in its new location and all its connections are re-established (lines 6–8). Thanks to the trans-cloud operations $start(a,c,l)$ and $reestablishRelations(a,c,child)$, the new component can be deployed and started in its location, hiding the complexity of managing the different services, and inspecting the application's topology to find and manage the relations in order to accomplish the reconnection between the target component and its children.

Finally, all components that were stopped in previous steps have to be restarted (lines 9–10). Again, a recursive function, RESTARTPARENTS$(a,parent)$ (lines 17–22), which follows a bottom-up strategy to avoid unexpected behaviors and wrong results, is in charge of re-starting all the stopped ancestors. This procedure ensures that all dependencies of a component are available before restarting it, and thus concluding the migration process.

4 THE TOOL IN PRACTICE

We explain in this section how our migration algorithm has been integrated into our trans-cloud infrastructure as an effector of cloud entities. We evaluate it by focusing in two aspects: the effort required for moving one element to a new location and the times taken by the execution of two different migration scenarios.

4.1 Algorithm Implementation

As explained in Section 3.2, our migration algorithm is treated like an autonomous element inside the trans-cloud approach. More specifically, Algorithm 1 has been developed and integrated as a new part of the customized Brooklyn described in Section 2.3. In this way, the algorithm can be accessed as part of the other available trans-cloud mechanisms, to manage providers and cloud resources, to operate individually with application components and carrying out operations like stopping, starting, restarting, etc. The trans-cloud extended Brooklyn tool, its documentation and examples is available in github from `https://goo.gl/DzXcXr`.

To be able to provide support for the management of connections and dependencies, key for the migration algorithm, our trans-cloud infrastructure had to be extended to enable explicit management of the functional relations of application's components.

Brooklyn, as well as the customized Brooklyn presented in Section 2.3, have a limited support on the management of relations. They read the TOSCA relationships specified in the topology of applications and configure components, for example, using environment variables, or configuration files, to establish the connection between them. However, the explicit knowledge about the relations is not shared with the trans-cloud API, which means that it does not offer the operations to identify, and manage the relations and functional dependencies between application's component, e.g., for re-establishment the connections, to find all component which depend of another (parents), or retrieve all the dependencies of one of them (children), which are required to develop the proposed algorithm.

In our extended implementation of Brooklyn, when TOSCA relationships are processed by Brooklyn-TOSCA to configure the component's relations, this information is added to the trans-cloud API. For that, we have added new mechanisms, based on the official Brooklyn API, to enable the management of relationships and to implement operations to find the parents and children of any component.

One of the key issues we have to solve when implementing the support for functional dependencies was the handling of Brooklyn's composition relations. Composition relations are used, for example, to model the relation between a cluster and the servers it controls. For composition relations, a component is in charge of the management of its sub-components. Thus, for example, if an operation, like stopping or starting, is applied to a cluster, it has to apply the same operation (stop or start) to all its children, in order to maintain the consistence of the topology. Then, composed elements, like clusters or elastic components can be understood and managed as a bundle, which represent a set of sub-components. Fortunately, the bundle behavior is perfectly compatible with our algorithm, since, if an operation must be carried out on a bundle, this one will ensure that the same operation is applied to all its sub-components. For example, if the migration algorithm requires to stop a cluster, this cluster itself ensures that all the servers, which compose it, will also be stopped. This maintains the topology integrity during the process, and ensures that the algorithm orchestration is delivered to all application's components.

4.2 On the Effort for Migration

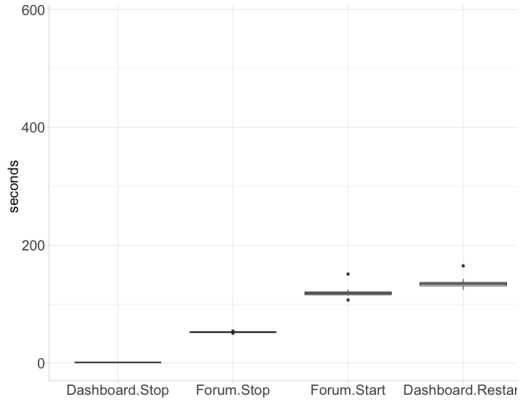
Whereas the migration from on-premise applications to the cloud has been studied by many researchers (Jamshidi et al., 2013b), not much work has been published on changes in target providers for migration. Besides, although there is no consensus in the literature on how to measure alternative deployments, the one used in (Kolb et al., 2015) is, to the best of our knowledge, possibly the most interesting so far. They use their proposal to compare and analyze the feasibility of the migration of an application using seven different vendors in terms of portability and effort. For an application composed of modules similar to those used in our case study, in the analysis in (Kolb et al., 2015), the deployment steps needed for a given set of PaaS providers are very different. Although these steps are semantically similar among vendors, they are carried out by proprietary tools, which do not permit them to be carried out in a standardized way. Their experiments showed that, on average, a migration may require an effort of 17 actions with a maximum spread of 14 and a standard deviation of 5. A low number of steps is usually offset by a complex configuration of the initial code repository, and it makes the initial deployment of an application a non-elementary task.

Despite the differences in effort between the different alternatives shown in (Kolb et al., 2015), this effort is reduced to 1 when the proposed algorithm uses bidimensional cross-cloud approach to orchestrate the migration and interact with the different providers. This is true in our case also, not only for migrations between the same kind of abstraction livels, such as PaaS, but for any combination of IaaS and PaaS vendors used for each of our application's modules, since the knowledge to interact with a specific provider and to handle application topologies, like relations, is encapsulated inside the customized Brooklyn. So, this allows the effort required for the migration of an application component to be only 1. Thus, the benefit is obvious. We only need an initial effort to specify the topology of our application in TOSCA, and bidimensional-based algorithm allows any application's component to be migrated between different providers, which is comparable with—even simpler than—the effort needed to do the orchestrate the migration process by hand.

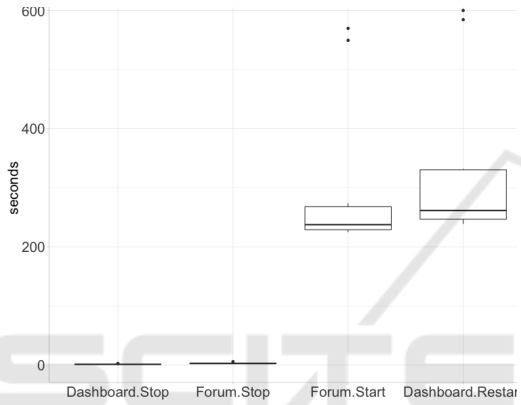
4.3 Deployment Times

A second important issue to be illustrated is as regards the quantitative level. Comparing the performance or reliability of providers or abstraction levels is not the goal of this work, it would require a more exhaustive analysis. However, our experiments show that we have not lost performance or reliability by using a trans-cloud deployment.

We have carried out experiments with two differ-



(a) Aws-to-Pivotal



(b) Pivotal-to-Aws

Figure 4: Forum migration process.

ent scenarios. Starting with the Softcare application deployed with the trans-cloud approach using different services, then the Forum component is migrated between different providers:

Aws-to-Pivotal Forum is moved from AWS EC2 to Pivotal Web Services.

Pivotal-to-Aws Forum is moved from Pivotal Web Services to AWS EC2.

Each of these two migration scenarios has been executed 10 times using our trans-cloud-based migration algorithm. The tool was instrumentalized to gather information at each sub-task of the process for each module. Specifically, and following the process explained in Section 3.2, we identify in both cases tasks Dashboard.stop, Forum.stop, Forum.start, Dashboard.restart, and gather the times at which they were completed (all time amounts are in seconds).

Charts in Figure 4a and 4b show box plots for the migration times in scenarios Aws-to-Pivotal and Pivotal-to-Aws, respectively. Both of them show similar times for the Dashboard component stopping, but we can see clear delays in the stopping and releasing of the Forum component (Forum.stop). This delay is explained by the different character of the kind of the cloud resources used, since releasing of cloud resources on IaaS (AWS EC2) (Figure 4a) is more expensive than releasing of PaaS services (Figure 4b).

The event Forum.start represents the redeployment of the Forum component in its new location and the reconnection of its dependencies. In Figure 4a the time values for Forum.start present smaller values and smaller data dispersion than in Figure 4b. The reason for this is that when the Forum component is deployed on IaaS (AWS EC2), it requires provisioning and configuring a new VM, and then executing the necessary commands to deploying the component, etc. The process is much simpler in PaaS.

Finally, the Dashboard component is restarted. Although the component is running on AWS EC2 in both cases, Dashboard.restart shows a greater delay in Figure 4b due to the accumulated dispersion in previous steps. The restart operation is in charge of reconnecting its dependencies.

5 RELATED WORK

There is some research on the study of the portability and interoperability issues in cloud computing, but not much specific on component migration. Indeed, depending on what kind of services are involved and how they are managed, we identify in the literature different ways to understand the term *migration* (see, e.g., (Jamshidi et al., 2013a; Zhao and Zhou, 2014)): migration of legacy apps, VM migration, and the migration of app components.

The most common form of migration is the *migration of legacy applications to the cloud*, where an entire application has to be moved to cloud environment in order to take advantage of cloud features (Armbrust et al., 2010), such as elasticity and scalability, payments strategies, or on-demand provisioning of services. Several studies are focused on the adaptation (Andrikopoulos et al., 2013) and the predictive cloud selections (Brogi et al., 2014; Qu et al., 2015; Vu and Asal, 2012) for the efficient and robust deployment of legacy systems on cloud environments (Cai et al., 2016; Papazoglou et al., 2007).

The second use of the term migration is in the context of the *management of virtualized resources on IaaS contexts*, like virtual machine migration (Clark et al., 2005). VM migration enables the movement from an online server to a new location, allowing an efficient usage and allocation of resources on-demand, ensuring the accomplishment of SLAs and

minimizing costs. Some proposals have tried to mitigate these issues by providing algorithms (Kashyap et al., 2014; Zhang et al., 2014; Lu et al., 2015), and new metrics (Deshpande et al., 2014) for the evaluation, optimization and scheduling of VM migration.

Finally, as a natural evolution of the previous migration scenarios, we identify a more abstract migration process, the *live migration of cloud application's components*. In this context, different algorithms for the robust movement of components among clouds are proposed (Durán and Salaün, 2016; Boyer et al., 2013). These solutions are limited to one specific service level. For instance, the approach presented in (Durán and Salaün, 2016) focuses on IaaS, whereas Cloud4SOA is devoted to the management of PaaS environments.

Federated multi-clouds (Paraiso et al., 2012) defines a PaaS federated platform to manage applications on IaaS and PaaS providers. It uses an OASIS standard, the Service Component Arquitecture, enabling the management over IaaS and PaaS levels of different providers. They also offer a standard-based unified provider-management, and allow the description of application architectures. However, they do not offer a robust unified modeling mechanism to represent the knowledge about applications, which makes the addressing of migration and elasticity more difficult.

Some research projects and initiatives, like jClouds, COAPS (Sellami et al., 2013) or Nucleous (Kolb and Röck, 2016) have developed generic APIs to manage services of different providers and their own models to represent application components and cloud services, in order to address the vendor lock-in problem and facilitate the migration of components. These approaches focus on the simplification of concrete abstraction levels, e.g., jClouds unifies IaaS services, and COAPS is centered on PaaS. We provide a level-agnostic solution by using trans-cloud's homogeneization mechanisms.

Projects like Roboconf (Pham et al., 2015) and SCALES (Ranabahu et al., 2015) provide frameworks for distributed application orchestration to define applications and the target providers to manage applications in multi-cloud platforms. Like us, both of them provide a generic and extensible DSL-based infrastructure where IaaS and PaaS providers can be integrated. Unfortunately, to deploy an application, in these solutions developers must provide a number of additional elements, to specify the steps necessary to manage the application over the target providers. Application descriptions and target services are very interdependent, which makes it very difficult to modify the target providers without affecting the application models. Like all above-mentioned approaches, they offer some early mechanisms to facilitate the migration of applications, but they do not formalize any algorithm to orchestrate and automatize this process.

6 CONCLUSIONS

We have presented an agnostic algorithm to orchestrate the migration process for application's stateless component. Since it is based on trans-cloud concepts, the algorithm is vendor, technology and service-level neutral.

The proposed algorithm takes advantage of these capabilities, simplifying the management of the different cloud providers solutions, reducing the portability and interoperability issues related to vendor lock-in. In fact, the algorithm is totally agnostic, and it can be applied to any stateless application component, either it is using IaaS or PaaS services to run. The migration process is fully automated, the only required external intervention to carry out the migration is just a migration request to initialize the process, indicating the component which has to be migrated and the target location. Indeed, it would be treated like an autonomous element, because it cannot need be integrated in the application modeling or its management lifecycle. It can be described as a self-governing orchestrator, whether facilitate the implementation and its integration in a trans-cloud system.

The current algorithm is intended for the migration of a single component of an application. We will in the near future work on the migration of several components of the same application, maybe all of them, in parallel. Moreover, we plan to study the possibility of using flexibility and scalability mechanisms for the support of autoscaling techniques to improve the reconfiguration skills of the presented work.

ACKNOWLEDGEMENTS

We are grateful to our partners in the SeaClouds project, and in particular to our colleagues Alex Heneveld, Andrea Turli, and the rest of Cloudsoft, and Francesco D'Andria and Roi Sucasas from Atos Spain. This work has been partially supported by MINECO/FEDER projects TIN2014-52034-R and TIN2015-67083-R, and Universidad de Málaga, Campus de Excelencia Internacional Andalucía Tech.

REFERENCES

Andrikopoulos, V., Binz, T., Leymann, F., and Strauch, S. (2013). How to adapt applications for the cloud environment. *Computing*, 95(6):493–535.

Androcec, D., Vrcek, N., and Kungas, P. (2015). Service-level interoperability issues of platform as a service. In *World Congress on Services (SERVICES)*, pages 349–356.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. (2010). A view of cloud computing. *Communications of the ACM*, 53(4):50–58.

Binz, T., Leymann, F., and Schumm, D. (2011). Cmotion: A framework for migration of applications into and between clouds. In *Intl. Conf. on Service-Oriented Computing and Applications (SOCA)*, pages 1–4. IEEE.

Boyer, F., Gruber, O., and Pous, D. (2013). Robust reconfigurations of component assemblies. In *Intl. Conf. on Software Engineering (ICSE)*, pages 13–22.

Brogi, A., Carrasco, J., Cubo, J., Nitto, E. D., Durán, F., Fazzolari, M., Ibrahim, A., Pimentel, E., Soldani, J., Wang, P., and D'Andria, F. (2015). Adaptive management of applications across multiple clouds: The SeaClouds approach. *CLEI Electron. J.*, 18(1).

Brogi, A., Ibrahim, A., Soldani, J., Carrasco, J., Cubo, J., Pimentel, E., and D'Andria, F. (2014). SeaClouds: a European project on seamless management of multi-cloud applications. *ACM SIGSOFT Sw. Eng. Notes*, 39(1):1–4.

Cai, Z., Li, X., and Gupta, J. N. (2016). Heuristics for provisioning services to workflows in XaaS clouds. *IEEE Trans. on Services Computing*, 9(2):250–263.

Carrasco, J., Cubo, J., Durán, F., and Pimentel, E. (2016). Bidimensional cross-cloud management with TOSCA and brooklyn. In *9th IEEE International Conference on Cloud Computing (CLOUD)*, pages 951–955.

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. (2005). Live migration of virtual machines. In *Conf. on Networked Systems Design & Implementation (NSDI)*, pages 273–286.

Deshpande, U., You, Y., Chan, D., Bila, N., and Gopalan, K. (2014). Fast server deprovisioning through scatter-gather live migration of virtual machines. In *Intl. Conf. on Cloud Computing (CLOUD)*, pages 376–383. IEEE.

Di Martino, B. (2014). Applications portability and services interoperability among multiple clouds. *IEEE Trans. on Cloud Computing*, 1(1):74–77.

Durán, F. and Salaün, G. (2016). Robust and reliable reconfiguration of cloud applications. *J. of Systems and Software*, 122:524–537.

Grozev, N. and Buyya, R. (2014). Inter-cloud architectures and application brokering: taxonomy and survey. *Softw., Pract. Exper.*, 44(3):369–390.

Jamshidi, P., Ahmad, A., and Pahl, C. (2013a). Cloud migration research: a systematic review. *IEEE Trans. on Cloud Computing*, 1(2):142–157.

Jamshidi, P., Ahmad, A., and Pahl, C. (2013b). Cloud migration research: A systematic review. *IEEE Trans. on Cloud Computing*, 1(2).

Kashyap, S., Dhillon, J. S., and Purini, S. (2014). Rlc-a reliable approach to fast and efficient live migration of virtual machines in the clouds. In *Intl. Conf. on Cloud Computing (CLOUD)*, pages 360–367. IEEE.

Kolb, S., Lenhard, J., and Wirtz, G. (2015). Application migration effort in the cloud. In *Intl. Conf. on Cloud Computing (CLOUD)*, pages 41–48.

Kolb, S. and Röck, C. (2016). Unified cloud application management. In *World Congress on Services Computing (SERVICES)*, pages 1–8.

Kritikos, K. and Plexousakis, D. (2015). Multi-cloud application design through cloud service composition. In *Intl. Conf. on Cloud Computing (CLOUD)*, pages 686–693.

Lu, H., Xu, C., Cheng, C., Kompella, R., and Xu, D. (2015). vhaul: Towards optimal scheduling of live multi-vm migration for multi-tier applications. In *Intl. Conf. on Cloud Computing (CLOUD)*, pages 453–460.

Moustafa, A., Zhang, M., and Bai, Q. (2016). Trustworthy stigmergic service composition and adaptation in decentralized environments. *IEEE Trans. on Services Computing*, 9(2):317–329.

Papazoglou, M. P., Traverso, P., Dustdar, S., and Leymann, F. (2007). Service-oriented computing: State of the art and research challenges. *Computer*, 40(11).

Paraiso, F., Haderer, N., Merle, P., Rouvoy, R., and Seinturier, L. (2012). A federated multi-cloud PaaS infrastructure. In *Intl. Conf. on Cloud Computing (CLOUD)*, pages 392–399.

Petcu, D. (2011). Portability and interoperability between clouds: challenges and case study. In *Towards a Service-Based Internet*, pages 62–74.

Pham, L. M., Tchana, A., Donsez, D., De Palma, N., Zurczak, V., and Gibello, P.-Y. (2015). Roboconf: a hybrid cloud orchestrator to deploy complex applications. In *Intl. Conf. on Cloud Computing (CLOUD)*, pages 365–372.

Qu, L., Wang, Y., Orgun, M. A., Liu, L., Liu, H., and Bouguettaya, A. (2015). CCCloud: Context-aware and credible cloud service selection based on subjective assessment and objective assessment. *IEEE Trans. on Services Computing*, 8(3):369–383.

Ranabahu, A., Maximilien, E. M., Sheth, A., and Thirunarayan, K. (2015). Application portability in Cloud Computing: An abstraction-driven perspective. *IEEE Trans. on Services Computing*, 8(6):945–957.

Sellami, M., Yangui, S., Mohamed, M., and Tata, S. (2013). PaaS-independent provisioning and management of applications in the cloud. In *Intl. Conf. on Cloud Computing (CLOUD)*, pages 693–700.

Vu, Q. H. and Asal, R. (2012). Legacy application migration to the cloud: Practicability and methodology. In *World Congress on Services (SERVICES)*, pages 270–277. IEEE.

Youseff, L., Butrico, M., and Silva, D. D. (2008). Toward a unified ontology of cloud computing. In *IEEE Grid*

Computing Environments Workshop (GCE), pages 1–10.

Zeginis, D., D'Andria, F., Bocconi, S., Cruz, J. G., Martin, O. C., Gouvas, P., Ledakis, G., and Tarabanis, K. A. (2013). A user-centric multi-paas application management solution for hybrid multi-cloud scenarios. *Scalable Computing: Pract. and Exp.*, 14(1).

Zhang, W., Lam, K. T., and Wang, C. L. (2014). Adaptive live vm migration over a wan: Modeling and implementation. In *Intl. Conf. on Cloud Computing (CLOUD)*, pages 368–375. IEEE.

Zhao, J.-F. and Zhou, J.-T. (2014). Strategies and methods for cloud migration. *Intl. J. of Automation and Computing*, 11(2):143–152.

Zheng, Z., Zhang, Y., and Lyu, M. R. (2014). Investigating QoS of real-world web services. *IEEE Trans. on Services Computing*, 7(1):32–39.