Future Generation Computer Systems 53 (2015) 63–76

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Allocating resources for customizable multi-tenant applications in clouds using dynamic feature placement

Hendrik Moens *, Bart Dhoedt, Filip De Turck

Ghent University – iMinds, Department of Information Technology, Gaston Crommenlaan 8/201, B-9050 Gent, Belgium

HIGHLIGHTS

- We model customizable SaaS applications using feature modeling.
- A dynamic, migration-aware management approach is presented.
- Two ILP-based algorithms and a heuristic algorithm are compared.
- The dynamic algorithms reduce migrations and remain within 3% of the optimal cost.

ARTICLE INFO

Article history:
Received 5 May 2014
Received in revised form
8 May 2015
Accepted 13 May 2015
Available online 9 June 2015

Keywords:
Cloud resource management
Software product line engineering
Dynamic application placement
Feature placement

ABSTRACT

Multi-tenancy, where multiple end users make use of the same application instance, is often used in clouds to reduce hosting costs. A disadvantage of multi-tenancy is however that it makes it difficult to create customizable applications, as all end users use the same application instance. In this article, we describe an approach for the development and management of highly customizable multi-tenant cloud applications. We apply software product line engineering techniques to cloud applications, and use an approach where applications are composed of multiple interacting components, referred to as application features. Using this approach, multiple features can be shared between different applications. Allocating resources for these feature-based applications is complex, as relations between components must be taken into account, and is referred to as the feature placement problem.

In this article, we describe dynamic feature placement algorithms that minimize migrations between subsequent invocations, and evaluate them in dynamic scenarios where applications are added and removed throughout the evaluation scenario. We find that the developed algorithm achieves a low cost, while resulting in few resource migrations. In our evaluations, we observe that adding migration-awareness to the management algorithms reduces the number of instance migrations by more than 77% and reduces the load moved between instances by more than 96% when compared to a static management approach. Despite this reduction in number of migrations, a cost that is on average less than 3% more than the optimal cost is achieved.

1. Introduction

In recent years there has been a growing interest in using cloud computing as a means of offloading applications and reducing costs. An efficient way in which costs of cloud deployments may be reduced is through multi-tenancy. In a traditional model, every client is provided with a separate application instance. In multi-tenant environments, however, a single instance can be used by multiple clients. Every client of the application is referred to as a *tenant* and is considered to be an organization with its own end users. The major advantage of this approach is that it makes it possible to use fewer application instances to provision the service to each of these tenants, reducing the cost of offering the service. Additionally, this approach makes it easier to scale applications, as sudden increases in the number of users results in smaller increases of the number of required instances. Spikes in the number of end users of one tenant can also be compensated by decreasing numbers of end users of other tenants.

Building customizable multi-tenant applications is however difficult, and it is often hard to make changes that are not just cosmetic configuration changes. Therefore, multi-tenant applications are often offered as a take it or leave it package, with only limited customizability. This approach works well for many

* Corresponding author.
E-mail address: hendrik.moens@intec.ugent.be (H. Moens).

http://dx.doi.org/10.1016/j.future.2015.05.017
0167-739X/

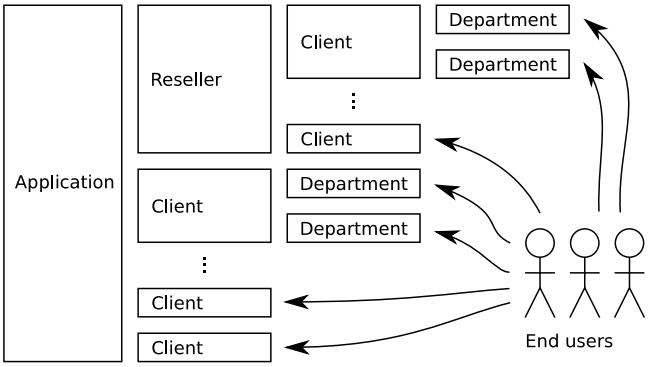

Fig. 1. An illustration of a scenario where the application is offered to end users by a hierarchy of the three types of tenants: resellers, clients, and client departments. Resellers can also sell the application to other resellers, and departments may also be further divided into smaller departments. At every level, different application customizations may be required.

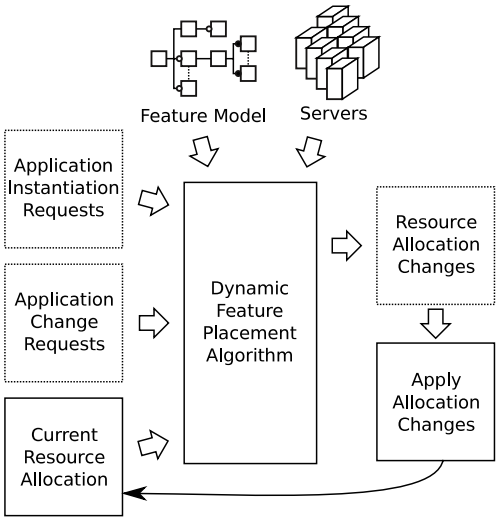

Fig. 2. The dynamic feature placement, its inputs and its function within a management system.

application types, especially when tenant needs are very similar, but there are use cases where a very high degree of customizability is required. This is the case in various domains, such as for example document processing, medical communications and medical information management. These application cases are all characterized by the fact that the offered platform is used by a relatively small number of large tenants that each have a large number of end users. Each of these tenants may request its own customizations to the application platform, and as many tenants are large, it is difficult to deny these requests. Currently such customizations are often developed on an ad-hoc basis. This however poses difficulties concerning the management of these customizations and as separate tenants have custom tailored codebases, it becomes impossible to share resources between end users. This problem becomes even more complex when clients are also split up into multiple departments that each require specific customizations and when the application platform is offered to other clients using resellers. An illustration of the various tenant types is shown in Fig. 1.

Using feature modeling [1], this issue can be addressed. Feature modeling is an approach where the variability of an application is modeled using a *feature model*. The customizability of the application is represented by a collection of features, a representation of specific functionality that may or may not be added to the application, and their relations. Features can be implemented using aspect oriented programming [2], as configuration changes, or as custom code modules. While feature modeling is an interesting approach for managing the codebase of customizable applications, this still results in customized application binaries, making it impossible to use multi-tenancy in the resulting applications. We previously proposed an approach where applications are separated into multiple interacting components, effectively making sure every feature is implemented in its own service [3]. The entire application is then composed from the various components, thus forming a service oriented architecture. As every code module is itself multi-tenant, the advantages of multi-tenancy can be attained.

Splitting applications into multiple components however impacts the performance of the applications, complicating cloud management. Additionally, the chosen features should be taken into account by the management system. It may e.g. be cheaper to use an existing high-performance instance for a tenant that does not pay for such an instance rather than to allocate a low-performance instance specifically for this tenant. We previously addressed resource allocation taking this information into account, referred to as feature placement, in [4,5], but the approach however resulted in a static resource allocation, that has to be recomputed periodically. In doing so, the number of migrations is not taken into account, which adversely impacts the performance of the system when services are migrated. Furthermore, adding applications is relatively expensive and slow as they can only be added whenever the algorithm is invoked rather than immediately when they are added.

In this article, we focus on dynamic feature placement algorithms that relocate and reconfigure features when changes occur. In computing these changes, the previous state of the system is taken into account, minimizing the number of application changes and instance migrations. We present both ILP-based algorithms and a heuristic algorithm, the Dynamic Feature Placement Algorithm (DFPA). Fig. 2 shows the algorithm inputs and how it functions within a cloud management system.

The remainder of this article is structured as follows. In the next section we discuss related work. Afterwards, in Section 3 we describe how the system in which the feature placement algorithm is executed is structured, and how feature modeling is used within the approach. A formal problem representation is presented in Section 4. In Section 5, we present the DFPA. The evaluation setup is presented in Section 6, and the algorithms are then evaluated in Section 7. Finally, we state our conclusions in Section 8.

2. Related work

To manage variability when building applications, Software Product Line Engineering (SPLE) [6] techniques are used. Instead of managing multiple codebases for different application variants, a single codebase is used, and different variants are generated using SPLE tools. In traditional SPLE applications, the application configuration is however generally decided at compile-time, making it ill-suited for cloud environments. Dynamic SPLE [7] can be used to configure and reconfigure software variants at runtime, making it more suited for cloud environments. This makes it possible to characterize runtime variability and reconfigure applications at runtime. SPLE has been used in cloud environments [8–10], but the approaches tend to focus mostly on development, deployment and configuration. We however focus specifically on runtime resource allocation for customizable SPLE applications by adding awareness of application variability to the cloud management algorithms. Similarly, other work [11–14] focuses on the design-time variability of the applications rather than on their runtime management, the latter being the focus of this article.

In this article, we focus on cloud resource allocation [15] and design dynamic management algorithms that are aware of application customizability. In particular, we focus on extending the