Science of Computer Programming 117 (2016) 1–16

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Formal design of dynamic reconfiguration protocol for cloud applications

Rim Abid *, Gwen Salaün, Noel De Palma

University of Grenoble Alpes, France

ARTICLE INFO

Article history:
Received 23 May 2014
Received in revised form 30 October 2015
Accepted 2 December 2015
Available online 10 December 2015

Keywords:
Cloud computing
Dynamic reconfiguration
Distributed applications
Fault-tolerance
Verification

ABSTRACT

Cloud applications are complex applications composed of a set of interconnected software components running on different virtual machines, hosted on remote physical servers. Deploying and reconfiguring this kind of applications are very complicated tasks especially when one or multiple virtual machines fail when achieving these tasks. Hence, there is a need for protocols that can dynamically reconfigure and manage running distributed applications. In this article, we present a novel protocol, which aims at reconfiguring cloud applications. This protocol is able to ensure communication between virtual machines and resolve dependencies by exchanging messages, (dis)connecting, and starting/stopping components in a specific order. The interaction between machines is assured via a publish-subscribe messaging system. Each machine reconfigures itself in a decentralized way. The protocol supports virtual machine failures, and the reconfiguration always terminates successfully even in the presence of a finite number of failures. Due to the high degree of parallelism inherent to these applications, the protocol was specified using the LNT value-passing process algebra and verified using the model checking tools available in the CADP toolbox. The use of formal specification languages and tools helped to detect several bugs and to improve the protocol.

1. Introduction

Cloud computing leverages hosting platforms based on virtualization and provides resources and software applications as service over the network (such as the Internet). It allows users to benefit from these services without requiring expertise in each of them. For service providers, this gives the opportunity to develop, deploy, and sell cloud applications worldwide without having to invest upfront in expensive IT infrastructure.

Cloud applications are intricate distributed applications composed of a set of virtual machines (VMs) running a set of interconnected software components. Cloud users need to (re)configure and monitor applications during their lifetime for elasticity or maintenance purposes. Therefore, after deployment of these applications, some reconfiguration operations are required for setting up new virtual machines, replicating some of them, destroying or adding virtual machines, handling VM failures, and adding or removing components hosted on a VM. Some of these tasks are executed in parallel, which complicates their correct execution.

Existing protocols [15,17,31] focus mainly on self-deployment issues where the model of application (the number of virtual machines, components, ports, and connections between components) is known before the application execution.

* Corresponding author.
E-mail addresses: Rim.Abid@inria.fr (R. Abid), Gwen.Salaun@inria.fr (G. Salaün), Noel.Depalma@imag.fr (N. De Palma).

http://dx.doi.org/10.1016/j.scico.2015.12.001
0167-6423/

These approaches manage static applications which do not require to be changed after the deployment phase. Existing deployment solutions barely take into account configuration parameters. Unlike these static applications, cloud applications need to be reconfigured in order to include new requirements, to fulfill the users expectations, or to perform failure recovery. More specifically, cloud users need protocols that are not only limited to deploy specific applications, but that are also able to modify applications during their execution and take into account the changes that can occur such as the failure of some virtual machine.

In this article, we introduce a novel protocol which aims at automatically deploying and (re)configuring applications in the cloud. These applications are composed of multiple and interconnected software components hosted on separate virtual machines. A reconfiguration manager guides the reconfiguration tasks by instantiating new VMs or destroying/repairing existing VMs. The reconfiguration manager may also apply a number of architectural changes to the application by adding new components or removing existing components hosted on a specific VM. After the creation of a component due to the instantiation of a VM or to a component addition request, the protocol is responsible for starting all components in the correct order according to the architectural dependencies. Each VM embeds a local reconfiguration agent that interacts with the other remote agents. For each component, the VM agent tries to satisfy its required services by connecting them to their providers in order to start the component. The component cannot be started before the components it depends on. The provider of the service can be hosted on the same VM or on another VM. When a VM receives a VM destruction or a component removal request from the reconfiguration manager, it tries to stop and unbind each component. A component cannot stop before all partner components connected to it have unbound themselves. In order to exchange messages and bind/start/unbind/stop components, VMs communicate together through a publish-subscribe messaging system. The protocol is also able to detect VM failures that occur to a running application. When a VM failure occurs, the protocol notifies the VMs that are impacted. The protocol supports multiple failures. It always succeeds in finally reconfiguring the application at hand and stopping/starting all components.

Our management protocol implies a high degree of parallelism. Hence, we decided to use formal techniques and tools to specify, verify the protocol, and ensure that it preserves important architectural invariants (*e.g., a started component cannot be connected to a stopped component*) and satisfies certain properties (*e.g., each VM failure is followed by a new creation of that VM*). The protocol was specified using the specification language LOTOS NT (LNT for short) [12], which is an improved version of LOTOS [20], and verified with the CADP verification toolbox [18]. For verification purposes, we used 600 hand-crafted examples (application models and reconfiguration scenarios). For each example, we generated the Labeled Transition System (LTS) from the LNT specification and we checked on them about 40 properties that must be respected by the protocol during its application. To do so, we used the model checking tools available in the CADP toolbox. These formal techniques and tools helped us improve the protocol by (i) detecting several issues and bugs, and by (ii) correcting them systematically in the corresponding Java implementation.

Our main contributions with respect to related approaches are the following:

- We propose a novel protocol, which reconfigures cloud applications consisting of a set of interconnected software components distributed over remote virtual machines.
- The protocol is fault-tolerant in the sense that it is able to detect and repair VM failures.
- The protocol was verified using model checking techniques and is implemented in Java.

The outline of this article is as follows. Section 2 introduces the reconfiguration protocol and presents how it works on some concrete applications. We present in Section 3 the LNT specification of the protocol and its verification using CADP. Section 4 reviews related work and Section 5 concludes the article.

2. Reconfiguration protocol

This section successively presents the application model, the protocol participants, the protocol itself including the different possible reconfiguration operations, and an overview of the Java implementation.

2.1. Application model

In this section, we present an abstraction of the model, which is sufficient for explaining the protocol principles. The real model exhibits more details such as port numbers, URLs, and other implementation details. The model we use here is used for verifying the soundness of the protocol but its primary role is to keep track of the VMs and components currently deployed in a cloud application.

An application is composed of a set of interconnected software components hosted on different virtual machines (VMs). Each component may provide services via exports and require services via imports. Ports are typed and match when they share the same type. One export can provide its service to several components and can thus be connected to several imports. An import must be connected to one export only provided by a component hosted on the same machine (local binding) or a component hosted on another machine (remote binding). When many exports provide the same service, the import is bound to one of them which is randomly chosen by the publish-subscribe system. The import can be mandatory or optional. A mandatory import represents a service required by the component to be functional. Therefore, if the component needs mandatory imports, it cannot be started before all its imports are satisfied (*i.e.*, all mandatory imports are connected