Future Generation Computer Systems 54 (2016) 1–15

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Self-scalable services in service oriented software for cost-effective data farming

Dariusz Król*, Jacek Kitowski

AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunications, Department of Computer Science, al. A. Mickiewicza 30, Krakow, Poland

HIGHLIGHTS

- We introduce self-scalable services as an extension of Service Oriented Architecture.
- We define scaling rules to express scaling policy for the service.
- Evaluation of the concepts is based on a massively scalable platform for data farming.
- Cost-effectiveness is increased in comparison with management based on fulfilling peak load.

ARTICLE INFO

Article history:
Received 29 October 2014
Received in revised form
24 June 2015
Accepted 1 July 2015
Available online 10 July 2015

Keywords:
Self-scalability
Autonomous computing
Service Oriented Architecture
Data farming

ABSTRACT

Software maintenance is one of the major concerns in service oriented ecosystem with an ever-increasing importance. In many cases, the cost of software maintenance is higher than the cost of software development. In particular, long-lasting services, which operate in a dynamically changing environment, require continuous management and administration. One of the important administration actions is scaling management. The problem lies in responding to workload changes of the hosted services as fast as possible. This is especially important in regard to (but not limited to) cloud environments where unnecessary resource usage leads to unnecessary costs. In this paper, we are introducing the self-scalable services and scaling rules, which are intended to support development of self-scalable systems based on Service Oriented Architecture. We propose a design of a self-scalable service based on some of the well-known software development practices along with a definition of scaling rules, which express scaling policy for the service. Both concepts were evaluated in the context of a massively scalable platform for data farming. The evaluation demonstrates advantages of utilizing the proposed concepts to manage the platform in comparison with traditional platform management strategies based on fulfilling peak load.

1. Introduction

Efficient management of computational resources is one of the primary challenges in today's data centers, and the resources utilization level is an essential metric describing a data center efficiency. Many reports suggest that the level of utilization of a data center's resources today is less than 50% [1–3], which indicates very low cost-efficiency of existing infrastructures. This is especially noticeable in the industry where access to resources is SLA-driven rather than being best-effort as it is in many academic facilities. We argue that one of the main reasons is static configuration of resources, which aims at handling the peak workload. Instead of using only the necessary amount of resources, a static pool of resources is maintained all the time to handle infrequent spikes in the workload. This issue can be addressed at different levels, concerning both hardware and software. Basic mechanisms at the hardware level include powering down unused servers and virtual machines consolidation. At the software level, the main objective is utilizing the minimal amount of resources that satisfies hosted software requirements, services in particular. These requirements heavily depend on the workload generated by service client's requests. Needless to say, the workload changes depend on the service's popularity, hence a combination of static resource configuration and dynamic workload changes (as depicted in [4,5]) often lead to either overutilized or underutilized resources, and both cases are undesirable. Here, resource underutilization refers to a case when a resource is idle more than a defined threshold of some period of time, e.g. more than 10% of time within a month. The threshold may vary on case-by-case basis. This is especially noticeable in the industry where access to resources is SLA-driven instead of being best-effort only as in many academic facilities.

* Corresponding author.
E-mail address: dkrol@agh.edu.pl (D. Król).

http://dx.doi.org/10.1016/j.future.2015.07.003
0167-739X/© 2015 Elsevier B.V. All rights reserved.

Two popular approaches which intend to increase resource utilization rate are services consolidation and dynamic adjustment of the amount of resources utilized by a service to the actual need. Services consolidation involves services migration from underutilized resources to minimize the number of utilized servers. This is mainly the scale-in mechanism, i.e. it relates to decreasing the amount of utilized resources. The second approach – dynamic adjustment of resources – addresses both scale-in and scale-out cases. In a today's data center, services are often scaled manually: upon discovering a change in workload an administrator allocates new resources and reconfigures the service. In most cases, an expected peak load with some overhead is estimated based on administrator's knowledge, and the corresponding amount of resources is allocated upfront. Needless to say, a substantial effort and expertise is required to estimate the necessary amount of resources for a given service. Any changes in the assumed workload impacts the utilization level of the allocated resources. Hence, an online monitoring and adjustment is often required, which can lead to tremendous administration effort taking into account the number of services installed in a data center.

The problem of scalability management of a service is common for a broad range of services used in both scientific and industrial environments. Examples of this kind of services include audio and video file transcoding services [6,7], websites load balancing [8], computation tasks scheduling onto a distributed set of resources, and any optimization problem solving environments.

Traditional services often require constant monitoring and possible manual reconfiguration when the workload pattern changes, e.g. when the number of clients increases or resource usage efficiency drops as a result of competition with other services running on the same host. In many situations a human administrator needs to be present all the time simply to ensure that a critical service continues to operate. System malfunctions resulting from excessive client load often incur significant costs — up to millions of dollars per hour of downtime [9]. Scientists tackled this problem with self-adaptive services [10,11], which aimed at dealing with unpredictable and dynamic load conditions without any human interaction. Once configured, such a service should adjust itself to different workload patterns dynamically. Self-scalability is a special feature of self-adaptive services, which addresses the problem of maintaining multiple instances of a service in an automatic manner. Moreover, by using monitoring data, self-scalable services can be more efficient than their manually operated counterparts due to faster reaction time. This is especially important in dynamically changing environments when the workload pattern cannot be determined or predicted beforehand.

Building a self-scalable service can be a challenging task. Such functionality is typically implemented in a separate module, often referred to as the management module, responsible for analyzing service load based on monitoring data and executing scaling actions, e.g. starting a new service instance on a different server. The management module typically implements an adaptation loop [11], which involves the following four activities:

- online monitoring, i.e. collecting online data about service workload,
- detection of events that trigger a scaling procedure,
- resource discovery encompasses identification of resources that can be used during scaling actions; this step does not assume that all available resources are known a priori, e.g. resources can be added or removed dynamically within a single data center, e.g. due to maintenance or transient errors,
- scaling actions execution, which involves acquisition of additional resources by the service in case of scale out (or releasing the utilized resources in case of scale-in).

Besides implementing these features, self-scalable services require knowledge about events that should trigger the scaling procedure. This knowledge can assume the form of rules which define conditions under which the management module should perform certain actions. Such rules are often gathered by observing the service in real-life scenarios and may be difficult to generate automatically. Thus, the decision to enhance an existing service with self-scalability features is not an obvious one.

In this paper, we address the issue of self-scalable software development by introducing two concepts: scaling rules and self-scalable services. The concept of scaling rules provides a formal way of expressing scaling management knowledge. For a given platform scaling rules describe how the system should rescale itself in response to various conditions. Such rules can be predefined by domain experts and then utilized automatically by computer systems. These rules are technology agnostic but are intended to support service oriented software, in particular different hosted services can have different rules defined. In order to address the scalability requirements of modern services, an extension of Service Oriented Architecture, called self-scalable services, is proposed, acknowledging the scalability property as a first-class citizen of software architectures. A self-scalable service extends the meaning of a software modularization unit with built-in self-scalability. We propose an architecture of self-scalable services, which follows the best software development practices. We do not address any particular SOA-related technology stack, instead we intend to operate in a generic service-oriented space, with a technology-agnostic sample implementation. To evaluate both concepts, we developed a platform for data farming called Scalarm, which is a complete solution for performing large-scale simulation-based virtual experiments using a heterogeneous computational infrastructure, i.e. an infrastructure consisting of different environments – clusters, clouds, and grids – each of which having a different access method.

The remaining part of the paper is organized as follows: Section 2 introduces related work on self-scalable software development and algorithms. In Section 3 we propose the concepts of scaling rules and self-scalable services in regard to their definition, design and main features. Section 4 describes a sample implementation of both introduced concepts in the context of the data farming methodology, while Section 5 provides more information about utilizing the introduced concepts to manage scalability of Scalarm in an automatic manner. Section 6 includes evaluation of the self-scalability of the implemented platform. Section 7 concludes the paper and outlines future work.

2. State of the art

In this section, we overview recent efforts related to self-scalable service development. In particular, we aim at design approaches to self-scalable and highly scalable software, auto-scaling algorithms, sample implementations of self-scalable software and infrastructure elasticity supporting self-scalability.

In the context of Service Oriented Architecture, a service is the basic modularization unit, which exists as an independent software program with distinct functional context and is comprised of a set of capabilities related to this context [12]. The most important features of services include: standardized service contract, loose coupling, abstraction, autonomy, reusability, statelessness, discoverability, composability, discoverability, and granularity. In our opinion this list lacks the self-scalability feature, which is more and more important today, but it was not as important when the feature list was introduced. However, it is worth noticing that the list includes already important aspects regarding scalability like statelessness, composability and autonomy.