Future Generation Computer Systems 54 (2016) 247–259

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Fault-tolerant Service Level Agreement lifecycle management in clouds using actor system

Kuan Lu [a,b,*], Ramin Yahyapour [a,b], Philipp Wieder [a], Edwin Yaqub [a,b], Monir Abdullah [a,d], Bernd Schloer [a], Constantinos Kotsokalis [c]

[a] Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Germany
[b] The University of Göttingen, Germany
[c] AfterSearch P.C., Greece
[d] Thamar University, Yemen

HIGHLIGHTS

- We elaborate a utility architecture that optimizes resource deployment.
- We provide a mechanism for optimized SLA negotiation.
- Using Actor system as basis, the entire SLA management can be efficiently parallelized.
- We separate the agreement's fault-tolerance strategies into multiple autonomous layers.
- A realistic approach for the automated management of the complete SLA lifecycle.

ARTICLE INFO

Article history:
Received 30 September 2014
Received in revised form
6 March 2015
Accepted 19 March 2015
Available online 1 April 2015

Keywords:
Service Level Agreements lifecycle
Cloud
Actor system

ABSTRACT

Automated Service Level Agreements (SLAs) have been proposed for cloud services as contracts used to record the rights and obligations of service providers and their customers. Automation refers to the electronic formalized representation of SLAs and the management of their lifecycle by autonomous agents. Most recently, SLA automated management is becoming increasingly of importance. In previous work, we have elaborated a utility architecture that optimizes resource deployment according to business policies, as well as a mechanism for optimization in SLA negotiation. We take all that a step further with the application of actor systems as an appropriate theoretical model for fine-grained, yet simplified and practical, monitoring of massive sets of SLAs. We show that this is a realistic approach for the automated management of the complete SLA lifecycle, including negotiation and provisioning, but focus on monitoring as the driver of contemporary scalability requirements. Our proposed work separates the agreement's fault-tolerance concerns and strategies into multiple autonomous layers that can be hierarchically combined into an intuitive, parallelized, effective and efficient management structure.

1. Introduction

Contemporary IT service management (ITSM) expects service level to be managed alongside functional service properties. This requires that arrangements are in place with internal IT support providers and external suppliers in the form of Operational Level Agreements (OLAs) and Underpinning Contracts (UCs), respectively [1]. More recently, automated Service Level Agreements (SLAs) have been proposed as a means to establish a common understanding of expectations and obligations of service providers and their customers. Specifically, automation refers to the electronic formalized representation of SLAs and the management of their lifecycle by autonomous agents. Such proposals have been linked particularly to cloud services.

Previously, we have elaborated a utility architecture that optimizes resource deployment according to business policies and a mechanism for optimized SLA negotiation [2]. In this work, we take all that a step further with the application of actor systems

* Corresponding author at: Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Germany. Tel.: +49 551 3920427; fax: +49 551 2012150.
E-mail addresses: kuan.lu@gwdg.de (K. Lu), ramin.yahyapour@gwdg.de (R. Yahyapour), philipp.wieder@gwdg.de (P. Wieder), edwin.yaqub@gwdg.de (E. Yaqub), monir.kaid@gwdg.de (M. Abdullah), bernd.schloer@gwdg.de (B. Schloer), costas@aftersear.ch (C. Kotsokalis).

http://dx.doi.org/10.1016/j.future.2015.03.016
0167-739X/

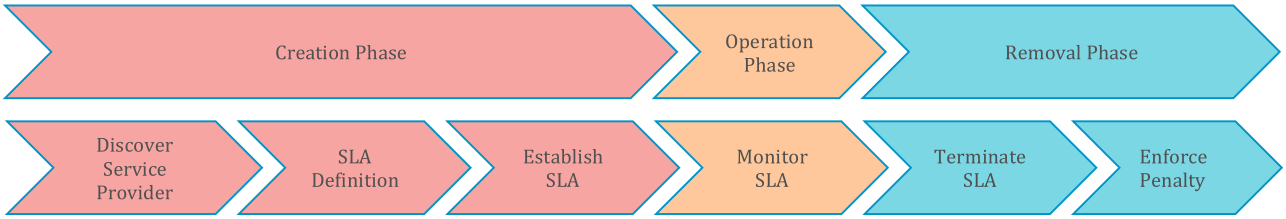

Fig. 1. Evolution of SLA lifecycle management from three steps to six steps. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

as an appropriate theoretical model for fine-grained, yet simplified and practical monitoring of massive sets of SLAs. Using actor system as an implementation basis, the whole SLA management can be efficiently parallelized. In this work, we are using the actor system in Akka framework [3] for JVM-based actors. Comparing to other event-driven programming languages such as Esper [4] and Storm [5], actor systems build hierarchies of actors using strong parent–child relationships to achieve fault-tolerance. In this setting, each actor may have one or more child-actors that are responsible for their own specific functionality. When failure occurs in one of the child-actors, the failure will be propagated upwards until it is handled by predefined solutions [6]. Our approach separates the agreement's fault-tolerance concerns and strategies, i.e. resource (re)-scheduling, outsourcing, (re)-negotiation, into multiple autonomous layers that can be hierarchically combined into an intuitive, parallelized, effective and efficient management structure. We show that this is a realistic approach for the automated management of the complete SLA lifecycle, including negotiation and provisioning, but also focus on monitoring as the driver of contemporary scalability requirements.

The remainder of this paper is structured as follows. In Section 2, the derivation and the classification of SLA lifecycle management will be discussed. In Section 3, we will discuss prior art. Then the model of the problem is given in Section 4. In Section 5, a formal model of SLA negotiation scenario is provided. Section 6 gives a description of a formal model of SLA monitoring scenario. In Section 7, we validate our proposal via discrete event simulations. Finally, we conclude the paper in Section 8.

2. SLA lifecycle management

SLAs relate by their very nature to different phases of a service lifecycle. Therefore, it is of fundamental importance to agree on a common reference model for such a service lifecycle which details the various phases, their stakeholders and their expected outputs in a common way without compromising the general design space for SLA management architecture.

The overall lifecycle model has been influenced to a dramatic degree and is broadly in-line with the ITIL framework [7]. In general, the service lifecycle management considers six main phases, which are: design and development, service offering, service negotiation, service provisioning, service operations and service decommissioning. SLAs also play a central role in the service lifecycle, because by capturing service expectations and entity responsibilities they drive both engineering decisions at conception level (during for example service design) and operational decisions (during service usage and delivery) [8].

Although there is no standard definition, in recent years, various projects, research activities and companies provide the foundation for the state-of-the-art in SLA lifecycle management. For instance, Ron et al. define the SLA lifecycle in three phases [9]. Later on, more detailed classifications are proposed and outlined individually by SLA@SOI project [10], by Sun Microsystems [11] and by the TeleManagement Forum [12]. In general, SLA lifecycle management consists of three phases with corresponding color on top of the Fig. 1, namely creation (red), operation (orange) and removal phases (blue), each of which can be further expanded to sub-phases with the same color. The SLA creation includes three sub-steps, i.e. discover service provider, SLA definition and SLA establishment. Once service providers are discovered, customers have to be aware of the detailed capacity of the service providers. Therefore, the service providers describe and define their services properly and deliver the definition of their services to the customers. Then, the customers further establish the agreement(s) with one or more service providers based on the service definition through a process of SLA negotiation.

3. Related works

3.1. Service fault-tolerant monitoring

Currently, there are various kinds of service fault-tolerant monitoring tools, such as CloudWatch [13] from Amazon that provides monitoring for AWS cloud resources and the applications. Developers or system administrators can use it to collect and track metrics, gain insight and react immediately to keep their applications and businesses running smoothly. Likewise, by configuring Nagios [14], the entire IT infrastructure components could be monitored, traced, including various system metrics, applications, services and so on. In the end, Nagios can send alerts to administrators when critical infrastructure components fail. Similarly, through OpenTSDB [15], users are able to fetch the historical data of many system metrics, such as CPU, memory and hard disk utilization. However, all above tools are only designed for reporting a historical record of outages or failures and the failures however have to be resolved manually. Eventually, in any business that depends on computers, creating an infrastructure with clear procedures and preventive maintenance can reduce the likelihood of failures. In other words, how does the system understand the different types of possible failures and respond to the potential violation alarm. Therefore, recovery from failures must be planned and an autonomous service violation self-heal system is of paramount importance [16].

Some related fault-tolerant SLA management frameworks, such as [17], which presents a dependable QoS monitoring system called "QoSMONaaS", which relies on the as a Service paradigm, and can thus be made available to virtually all cloud users in a seamless way; [18] introduced a framework "SLAMonADA" for monitoring and explaining violations of WS-agreement-compliant documents; and [19] introduced an SLA-aware service virtualization architecture that provides non-functional guarantees in the form of SLAs and consists of a three layered infrastructure including agreement negotiation, service brokering and on demand deployment. However, they all only focused on SLA monitoring system to detect the SLA violations and corresponding penalty cost without considering the reaction process how to avoid the SLA violations. Furthermore, many works about SLA monitoring self-healing and autonomous adaption emerge as the times require, such as [20–26]. In detail, works [20,21], engaged the monitoring, analysis, planning and execution cycle to react against violations in