The Journal of Systems and Software 000 (2016) 1–18

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

ScapeGoat: Spotting abnormal resource usage in component-based reconfigurable software systems

I. Gonzalez-Herrera[a,*], J. Bourcier[a], E. Daubert[a,d], W. Rudametkin[b], O. Barais[a], F. Fouquet[c], J.M. Jézéquel[a], B. Baudry[e]

[a] University of Rennes 1-IRISA, Campus de Beaulieu, 35042 Rennes, France
[b] University of Lille 1, Cite Scientifique, 59655 Villeneuve d'Ascq Cedex, France
[c] University of Luxembourg, Security and Trust Lab, 1457 Luxembourg, Luxembourg
[d] CTO Energiency, 2 rue de la Châtaigneraie, 35510 Cesson-Sé vigné, France
[e] INRIA, Campus de Beaulieu, 35042 Rennes, France

ARTICLE INFO

Article history:
Received 20 October 2014
Revised 23 December 2015
Accepted 20 February 2016
Available online xxx

Keywords:
Resource monitoring
Component
Models@Run.Time

ABSTRACT

Modern component frameworks support continuous deployment and simultaneous execution of multiple software components on top of the same virtual machine. However, isolation between the various components is limited. A faulty version of any one of the software components can compromise the whole system by consuming all available resources. In this paper, we address the problem of efficiently identifying faulty software components running simultaneously in a single virtual machine. Current solutions that perform permanent and extensive monitoring to detect anomalies induce high overhead on the system, and can, by themselves, make the system unstable. In this paper we present an optimistic adaptive monitoring system to determine the faulty components of an application. Suspected components are finely analyzed by the monitoring system, but only when required. Unsuspected components are left untouched and execute normally. Thus, we perform localized *just-in-time* monitoring that decreases the accumulated overhead of the monitoring system. We evaluate our approach on two case studies against a state-of-the-art monitoring system and show that our technique correctly detects faulty components, while reducing overhead by an average of 93%.

1. Introduction

Many modern computing systems require precise performance monitoring to deliver satisfying end user services. For example, in pervasive, ubiquitous, building management or Internet of Things systems, devices that are part of the system have limited resources. The precise exploitation of these limited resources is important to take the most out of these systems, therefore requiring a fine and dynamic resource monitoring. Many other examples exist in the area of cloud computing where the notion of elasticity and on demand deployment is key to enable dynamic adaptation to user demand (Galante and Bona, 2012). Indeed, the elasticity management mechanism in charge of allocating and shrinking computing resources to match user demand, requires a precise performance monitoring of the application to determine when to increase or decrease the amount of allocated resources. Another example can be found in the cloud computing domain at the Software as a Service level when services with various Quality of Services (QoS) are offered to end users. The differentiation of QoS between two users of the same services requires a precise performance monitoring of the system to cope with the specified QoS.

To implement such behavior, these modern computing systems must exhibit new properties, such as the dynamic adaptation of the system to its environment (Rouvoy et al., 2009) and the adaptation of the system to available resources (Poladian et al., 2004). Modern Component based frameworks have been designed to ease the developers' tasks of assembling, deploying and adapting a distributed system. By providing introspection, reconfiguration, advanced technical services, among other facilities (Garlan et al., 2004), component frameworks are good candidate to assist software developers in developing resource-aware system. These frameworks provide extensible middleware and assist developers in managing technical issues such as security, transaction

* Corresponding authors. Tel.: +33 0687195442.
E-mail addresses: inti.glez@gmail.com, inti.gonzalez_herrera@irisa.fr (I. Gonzalez-Herrera), johann.bourcier@irisa.fr (J. Bourcier), erwan.daubert@irisa.fr (E. Daubert), walter.rudametkin@inria.fr (W. Rudametkin), barais@irisa.fr (O. Barais), francois.fouquet@uni.lu (F. Fouquet), jezequel@irisa.fr (J.M. Jézéquel), benoit.baudry@inria.fr (B. Baudry).

http://dx.doi.org/10.1016/j.jss.2016.02.027
0164-1212/

management, or distributed computing. They also support the simultaneous execution of multiple software components on the same virtual machine (The OSGi Alliance, 2012; Fouquet et al., 2012; Bruneton et al., 2006).

Current monitoring mechanisms (Frénot and Stefan, 2004; Kreger et al., 2003; Binder and Hulaas, 2006) continuously interact with the monitored application to obtain precise data regarding the amount of memory and I/O used, the time spent executing a particular component or the number of call to a particular interface. Despite their precision, these monitoring mechanisms induce high overhead on the application; this prevents their use in production environments. The overhead of a monitoring mechanism can be up to a factor of 4.3 as shown in the results presented in Binder et al. (2009). As we discuss in Section 4 the performance overhead grows with the size of the monitored software. Thus, overhead greatly limits the scalability and usage of monitoring mechanisms.

In this paper, we address excessive overhead in monitoring approaches by introducing an optimistic adaptive monitoring system that provides lightweight global monitoring under normal conditions, and precise and localized monitoring when problems are detected. Although our approach reduces the accumulated amount of overhead in the system, it also introduces a delay in finding the source of a faulty behavior. Our objective is to provide an acceptable trade-off between the overhead and the delay to identify the source of faulty behavior in the system. Our approach targets component-models written in object-oriented languages, and it is only able to monitor the resource consumption of components running atop a single execution environment. In this paper, we discuss how we can leverage our proposal to provide the foundations for resource consumption monitoring in distributed environments.

Our optimistic adaptive monitoring system is based on the following principles:

- **Contract-based resource usage.** The monitoring system follows component-based software engineering principles. Each component is augmented with a contract that specifies their expected or previously calculated resource usage (Beugnard et al., 1999). The contracts specify how a component uses memory, I/O and CPU resources.
- **Localized just-in-time injection and activation of monitoring probes.** Under normal conditions our monitoring system performs a lightweight global monitoring of the system. When a problem is detected at the global level, our system activates local monitoring probes on specific components in order to identify the source of the faulty behavior. The probes are specifically synthesized according to the component's contract to limit their overhead. Thus, only the required data are monitored (e.g., only memory usage is monitored when a memory problem is detected), and only when needed.
- **Heuristic-guided search of the problem source.** We use a heuristic to reduce the delay of locating a faulty component while maintaining an acceptable overhead. This heuristic is used to inject and activate monitoring probes on the suspected components. However, overhead and latency in finding the faulty component are greatly impacted by the precision of the heuristic. A heuristic that quickly locates faulty components will reduce both delays and the accumulated overhead of the monitoring system. We propose using Models@run.time techniques in order to build an efficient heuristic.

The evaluation of our optimistic adaptive monitoring system shows that, in comparison to other state-of-the-art approaches, the overhead of the monitoring system is reduced by up to 93%. Regarding latency, our heuristic reduces the delay to identify the faulty component when changing from global, lightweight monitoring to localized, just-in-time monitoring. We also present a use case to highlight the possibility of using ScapeGoat on a real application, that shows how to automatically find buggy components on a scalable modular web application.

An early version of the ScapeGoat monitoring framework is presented in Gonzalez-Herrera et al. (2014). This paper introduces four new majors contributions:

- The paper includes **a new mechanism to monitor memory consumption that can be turned on/off**. In Gonzalez-Herrera et al. (2014), memory monitoring cannot be turned off. As a consequence, the probes used to account for memory consumption are always activated. This impacts the performance of the system even when in global monitoring mode. In this paper, we propose a mechanism to avoid any kind of overhead when in global monitoring mode thank to the new monitoring mechanism. Using this new mechanism we reduce even more the performance overhead in terms of CPU consumption and we avoid overhead in terms of memory consumption.
- New **experiments to assess the performance impact** of the proposed mechanism **to compute memory consumption monitoring**.
- In addition to the experiments proposed in Gonzalez-Herrera et al. (2014), a **new use case is used to evaluate the monitoring framework**.
- We show how to generalize the approach to deal with properties other than CPU, memory and related resources.

The remainder of this paper is organized as follows. Section 2 presents the background on Models@run.time and motivates our work through a case study which is used to validate the approach. Section 3 provides an overview of the ScapeGoat framework. It highlights how the component contracts are specified, how monitoring probes are injected and activated on-demand, how the ScapeGoat framework enables the definition of heuristics to detect faulty components without activating all the probes, and how we benefit from Models@run.time to build efficient heuristics. Section 4 validates the approach through a comparison of detection precision and detection speed with other approaches. Section 5 presents a use case based on an online web application[1] which leverages software diversity for safety and security purposes. In Section 6 highlights interesting points and ways to apply our approach to other contexts. Finally, Section 7 discusses related work and Section 8 discusses the approach and presents our conclusion and future work.

2. Background and motivating example

2.1. Motivating example

In this section we present a motivating example for the use of an optimistic adaptive monitoring process in the context of a real-time crisis management system in a fire department. During a dangerous event, many firefighters are present and need to collaborate to achieve common goals. Firefighters have to coordinate among themselves and commanding officers need to have an accurate real-time view of the system.

The Daum project[2] provides a software application that supports firefighters in these situations. The application runs on devices with limited computational resources because it must be mobile and taken on-site. It provides numerous services for firefighters depending on their role in the crisis. In this paper we focus on the two following roles:

[1] http://cloud.diversify-project.eu/.
[2] https://github.com/daumproject.

Please cite this article as: I. Gonzalez-Herrera et al., ScapeGoat: Spotting abnormal resource usage in component-based reconfigurable software systems, The Journal of Systems and Software (2016), http://dx.doi.org/10.1016/j.jss.2016.02.027