Information Sciences 264 (2014) 340–348

Contents lists available at SciVerse ScienceDirect

Information Sciences

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

Online reliability computing of composite services based on program invariants

Zuohua Ding [a,*], Mei-Hwa Chen [b], Xiaoxue Li [a]

[a] Lab. of Intelligent Computing and Software Engineering, Zhejiang Sci-Tech University, Hangzhou 310018, PR China
[b] Computer Science Department, University at Albany-State University of New York, Albany, NY 12222, USA

ARTICLE INFO

Article history:
Available online 31 May 2013

Keywords:
Reliability computing
Prot-based reliability model
Program invariants
Service composition

ABSTRACT

Reliability is an essential software quality requirement, especially for online service software. Without an accurate prediction of service reliability, any unexpected failure can disrupt service. The majority of existing models use static data collected prior to the release of the software. These types of models may predict the reliability of the software as it was during the data collection phase. However, online service software is continuously evolving, and their behaviors can be changed by the runtime usage. Thus, the prediction made by static data can be inaccurate. We present an approach to tackle this challenge by taking into account software runtime behavior in our reliability prediction. We used a data mining tool, Daikon, to collect likely invariants of the software to capture its states in the runtime. This runtime information is then used to compute the reliability of the software by using our port-based reliability model.

© 2013 Elsevier Inc. All rights reserved.

1. Introduction

Service-oriented software has gained emerging popularity among broad areas of application domains. The reliability of service-oriented software is an important metric for measuring the quality of the service. Most existing techniques for measuring software reliability rely on static data collected prior to the release of the software to predict the reliability of the software in operation. For service-oriented software, these techniques may not be able to predict service reliability accurately. The reasons are in follows.

- First, the discrepancy between the reliability estimates and the actual reliability may be derived from the differences between the behaviors of the software under development and those of the deployed software. The former are in a controlled environment, while the latter can be dynamically changed by the usage.
- Second, service-oriented software normally is used by a number of users simultaneously and ubiquitously, which is different from the traditional standalone software. Consequently, the nature of their failure can be different.
- Third, a service may experience multiple failures at the same instant of time due to its concurrent uses, and the time of a failure may be delayed for an asynchronous service. Moreover, a service may use a number of ports, each of which provides a set of operations, which is different from the traditional single entry and single exit programs.

Therefore, to better predict the reliability of operational service software, we first need to cope with these differences. Some efforts have been done to handle a program's dynamics. For example, Dai et al. [1] presented a solution to online improve the software reliability. It is based on consequence-oriented diagnosis and healing. Consequence-oriented diagnosis

* Corresponding author. Tel./fax: +86 571 86843245.
E-mail address: zouhuading@hotmail.com (Z. Ding).

0020-0255/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.05.020

offers the prediction of the consequence from the symptoms instead of detecting the causes that lead to the symptoms. Consequence-oriented healing, then, prevents the system from the consequence, running extra instructions. A hybrid diagnosis tool that combines the Multivariate Decision Diagram, Fuzzy Logic, and Neural Network is developed. Romanazzi et al. [11] proposed a prediction model that can be used to make predictions as to the execution time of a given job on a given computational resource of parallel multigrid software running on a distributed memory architecture. The model is based on their developed partition method, namely strip partition for multigrid.

In this paper, we present an approach that leverages runtime information on program execution states to compute reliability of online services. We used a data mining tool, Daikon,[1] to detect programs' invariants that hold a certain property (properties) at program executions. An unexpected value of a program property can be used to detect abnormal program behaviors. Thus, we determine a service failure by comparing the values of the invariants collected at runtime with their expected ones. Furthermore, we adopt a port-based reliability model [4] that we developed to compute reliability of online services, taking into account the nature of the service requests and provisions. The port-based model, on one hand, formally describes the behavior model for Service Component Architecture (SCA)[2] (for details, see our previous work [3]); on the other hand, it can automatically compute overall system reliability by cooperating Nonhomogeneous Poisson Process (NHPP) as the analysis model to each port, based on the rules of ports and the behavior model.

This paper is organized as the follows. Section 2 gives an overview of the port-based reliability model. Section 3 shows how to obtain program invariants and to compute reliability with invariants. Section 4 presents a case study of using an Online Shop to illustrate our method. Section 5 discusses the related work. Section 6 concludes the paper.

2. Port-based reliability model

For the completeness, we give an overview of the port-based reliability. For details, we refer to our previous work [4].

2.1. Port-based component model

In SCA, a component provides or requires a service through a port. Ports represent the addressable interfaces of the component and the requirement that the component has on a service provided by other component.

Definition 1 (*Port*). A port p is a tuple (M, t, c), where M is a finite set of methods in p, t is the port type that can be provided or required, and c is the communication type that can be synchronous or asynchronous.

Definition 2 (*Component*). A component *Com* is a tuple (P_p, P_r, G, W), in which P_p is a finite set of provided ports, P_r is a finite set of required ports, G is a finite sub component set, $W \subseteq TP \times \bigcup_{C \in G}(C.P_p \cup C.P_r)$ is the port relation that is non-reflexive, where $TP = P_p \cup P_r \cup \bigcup_{C \in G} C.P_r$, $C.P_p$ and $C.P_r$ denote the provided and required port sets of the sub component C respectively.

A composite contains assemblies of service components. The composite diagram is shown in Fig. 1. SCA *wire* within a composite connects source component required port to target component provided port. If a link is from provided port to provided port or from required port to required port, then it is called *promote*.

We use port activities to describe the component dynamic behavior, the basic activity of which is assumed to be the message exchanging between two ports. Let m be the message. $p\bullet_m$ represents that the port p sends the message m synchronously. $p_1\bullet_m p_2$ represents that the port p_1 synchronously sends the message m to the port p_2. $p\blacklozenge_m$ represents that the port p sends the message m asynchronously. $p_1\blacklozenge_m p_2$ represents that the port p_1 asynchronously sends the message m to the port p_2. $p \circ_m$ means that port p synchronously receives the message and $p\Diamond_m$ means that port p asynchronously receives the message. $BE[p_1 \to p_2]$ is used to specify the wiring operation which is the syntactic transformation.

2.2. Reliability computing

2.2.1. Port reliability

Let p be a port of a component. At each time to be visited, port p will perform $p[p_1 \to p_2]$ for some wiring or promoting operation $p_1 \to p_2$. For a given test suite, we assume that the tests are executed one by one, thus all the test executions together have a fixed execution time. It follows that each port has a fixed time to be visited. Let $m_p(t)$ be the expected number of faults detected by time t at port p, and $\lambda_p(t)$ is the corresponding failure intensity function. Assume that the total test execution time is T. Hence the reliability of port p with operation $p_1 \to p_2$ in the time interval $[0,T]$ can be defined as:

$$r(p[p_1 \to p_2]) \triangleq e^{-\int_0^T \lambda_p(\tau)d\tau}.$$

This is called Non-Homogeneous Poisson Process (NHPP) model that has been widely used to track reliability improvement during software testing. In our testing bed, we choose Goel and Okumoto model[7] for the computing.

1 http://groups.csail.mit.edu/pag/daikon/
2 http://www.osoa.org/display/Main/Home

Download English Version:

https://daneshyari.com/en/article/392834

Download Persian Version:

https://daneshyari.com/article/392834

Daneshyari.com