Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Architectural reliability analysis of framework-intensive applications: A web service case study

M. Rahmani, A. Azadmanesh*, H. Siy

College of Information Science and Technology, University of Nebraska-Omaha, Omaha, NE 68182, USA

ARTICLE INFO

Article history:
Received 25 September 2012
Received in revised form 19 January 2014
Accepted 23 March 2014
Available online 4 April 2014

Keywords:
Architecture-based software reliability
Petri Net
Service oriented architecture

ABSTRACT

A novel methodology for modeling the reliability and performance of web services (WSs) is presented. To present the methodology, an experimental environment is developed in house, where WSs are treated as atomic entities but the underlying middleware is partitioned into layers. WSs are deployed in JBoss AS. Web service requests are generated to a remote middleware on which JBoss runs, and important performance parameters under various configurations are collected. In addition, a modularized simulation model in Petri net is developed from the architecture of the middleware and run-time behavior of the WSs. The results show that (1) the simulation model provides for measuring the performance and reliability of WSs under different loads and conditions that may be of great interest to WS designers and the professionals involved; (2) configuration parameters have substantial impact on the overall performance; (3) the simulation model provides a basis for aggregating the modules (layers), nullifying modules, or to include additional aspects of the WS architecture; and (4) the model is beneficial to predict the performance of WSs for those cases that are difficult to replicate in a field study.

1. Introduction

Internet globalization has provided the unprecedented opportunity for enterprises to develop and deliver electronic services such as online shopping and banking services. These services, called web services (WSs) are becoming a promising technology for building distributed and complex systems. World Wide Web Consortium (W3C) defines a *web service* a software system designed to support interoperable machine-to-machine interaction over a network (Booth, 2004a).

WSs are commonly delivered through an infrastructure based on the Service-Oriented Architecture (SOA) (Booth, 2004b). WSs are deployed using an *application server* (AS), part of the software ecosystem comprising the *middleware* connecting service requesters to providers. The middleware is a complex and intertwined combination of various libraries, components and their interfaces, forming an *application framework* upon which web services can be deployed. Because the majority of code execution of a web service normally occurs within the components of this framework, web applications are a type of *framework-intensive applications* (Dufour et al., 2007). Due to the growing complexity of such framework-intensive applications as well as society's increasing dependence on their services, the reliability of these systems has become critical. In this study, WSs are treated as atomic entities but the middleware is partitioned into layers. WSs are developed in house and the application server (AS) considered is JBoss AS (JBoss, 2012). An experimental model is developed that anchors the analytical results to empirical observations. In parallel, a simulation model is developed that provides an operational view of the behavior and inner working of the overall distributed system. The architecture of the simulation model follows a hierarchical structure, modularized in accordance to the layers and their interactions. As the web services are often executed in a complex and distributed environment, the modularized approach enables a user to observe and investigate the performance of the entire system under various conditions. The architecture-based analysis, which takes into account the components, e.g. middleware layers, and their interactions, is a form of *white-box* analysis. In contrast, most approaches to WSs analyses are monolithic in that the entire system is treated as a *black-box*.

Although this study complements some of the research work in literature (Cao et al., 2003; Souza et al., 2006; Xiao and Dohi, 2010; Wells et al., 2001), the contribution of this study can be summarized as follows:

- One cannot underestimate the importance of middleware in the overall reliability of the applications. This study conducts

* Corresponding author. Tel.: +1 402 554 3976; fax: +1 402 554 3284.
E-mail address: azad@unomaha.edu (A. Azadmanesh).

http://dx.doi.org/10.1016/j.jss.2014.03.070
0164-1212/© 2014 Elsevier Inc. All rights reserved.

performance analysis of the middleware based on its major architectural layers. To the best of our knowledge there is no solid architecture-based reliability analysis of the middleware.

- A large number of studies consider theoretical analysis without investigating the real-world applications and tackling the challenges of complex software systems (Cheung, 1980; Singh et al., 2001; Zhong and Qi, 2006). Since many of these approaches validate their models using simulation-based analysis or simple applications, the applicability of these approaches for real-world complex software systems is unknown. This study uses a combination of analytical, experimental, and simulation models.
- Configuration errors occur when operators define incorrect settings for system parameters, e.g., specifying insufficient number of threads to service user requests entering the web server. Configuration parameters play an important role in the performance of web services, but not much attention has been made in the failure analysis of web services caused by these error types. Research in Pertet and Narasimhan (2005) has shown misconfiguration of web-based and framework-intensive software can contribute to nearly forty percent of failures in web applications. This study pays a close attention to these types of failures.
- The software system under study by a number of research papers is often simple and rudimentary (Cao et al., 2003; Singh et al., 2001; Souza et al., 2006; Wells et al., 2001; Zhong and Qi, 2006). In addition, the common model presentation used is Markov chains, which may lead to state explosion for large systems. Instead, this study presents an architectural model using a specific type of Petri Nets called *Stochastic Activity Network* (SAN) (Sanders and Meyer, 2001). The model is simple enough to build, which is based on the static and dynamic code analysis of the middleware. Static analysis is used to determine the main layers and configuration parameters of the middleware. Dynamic analysis assists in the extraction of the timing information of the layers and the call-graph of a particular web service execution.

The multilayer and the reliability approaches proposed in this study have the following advantages:

- The application server contains a large number of internal applications, utilities, and enormous number of Java classes. The proposed multilayer approach offers a more manageable model in order to analyze the reliability of a web service.
- As failure rate of each layer and configuration parameters are estimated separately, analyzing the reliability of the system would be more insightful to better understand the effect of each layer on the overall reliability of the system.
- The analytical, experimental and the simulation-based models can validate each other to ensure the correctness of the results. Hence, the simulation model can be used to test the performance of the web environment under various conditions.

The rest of the paper is organized as follows. Section 2 provides some background on web service reliability and prior approaches to reliability modeling. Section 3 describes the three proposed models. Section 4 provides the performance results of the three models and their comparison. Section 5 shows how system reliability might be characterized based on inputs from the simulation model. Section 6 provides some concluding remarks and some avenues for extending the current research.

2. Background and literature review

WSs follow a client/server paradigm, where the middleware running on a server connects clients to the desired WSs. The middleware is formed by multiple layers. The *application server* layer, e.g. JBoss AS, is the main engine of the middleware that provides the environment for running applications regardless of what the application might be. One of the major tasks of the server is to hide the intrinsic details of common programming tasks such as security, persistent storage, and queuing requests, so that application developers can concentrate on the business logic rather than on periphery aspects. It is through this layer that web services can communicate with other layers. The *web server* layer, e.g. Apache (Apache, 2012), is the front end software that delivers requests to the application server on which web services are deployed. The *database* layer stores data relevant to web services.

2.1. Web service reliability

The reliability of a software system is a probabilistic measure of correctly delivering services during a period of time. The measured reliability depends on fault types considered, such as crash faults, software faults, exception faults, time-outs, resource exhaustion, misconfiguration of the underlying shared resources, etc. For example, one may want to obtain the reliability of the system based only on resource exhaustion. On the other, as fault sources are often numerous, to make reliability analysis manageable, attempts can be made to categorize the faults based on their impacts rather than their origin. For instance, an expected message not received in the client side might be due to a fault in the network, or a fault in the server side software component or because the configuration parameters in the middleware are not set correctly. So the spectrum of faults with the impact of not receiving messages can be categorized as omission faults. In general, a possible categorization of faults based on their impacts that collectively captures all kinds of faults is timing faults, omission faults, and arbitrary faults (Azadmanesh et al., 2008; Srinivasan and Azadmanesh, 2013). A timing fault occurs when an expected correct message is not received within a predetermined period of time. An omission fault happens when no response is received. Finally, an arbitrary fault is the one that behaves in any manner other than timing or omission faults. For instance an arbitrary fault can cause an erroneous value to be received or when two conflicting values are transmitted as response to the same input values. This study is mostly concerned with the timing (time-out) faults of web services. More specifically, this study follows the definition presented by Hwang et al. (2008), which defines the web service reliability as “the probability the web service successfully responds within a reasonable period of time”. This form of timing faults can be caused by many factors such as delays in the middleware, inadequate setting of configuration parameters, or network problems. A study (Pertet and Narasimhan, 2005), which used information from Google, IBM, Intel, Microsoft, Hewlett-Packard, Oracle, and Sun, indicates that the main cause of failures in web applications is not logical or computational errors. Rather, the most frequent causes of failure are due to system overload, configuration errors, resource exhaustion, human/operator errors, and lack of resource sharing. Furthermore, to the best of our knowledge, many architecture-based reliability studies do not consider these failures (Grassi, 2005; Zhong and Qi, 2006). Although some studies (Grassi and Patella, 2006; Grassi, 2005) claim that architecture-based reliability approaches can be applied to service-oriented computing applications, no solid work exists to confirm this. For instance, Grassi (2005) presents an approach to the reliability prediction of an assembly of services that uses the architecture-based reliability analysis. The approach does not provide a definition of failure in a service-oriented