The Journal of Systems and Software xxx (2013) xxx–xxx

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

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

Workload-aware anomaly detection for Web applications☆

Tao Wang[a,b,c,*], Jun Wei[a,b], Wenbo Zhang[b], Hua Zhong[b], Tao Huang[a,b]

[a] *State Key Laboratory of Computer Science, Beijing 100190, PR China*
[b] *Institute of Software, Chinese Academy of Sciences, Beijing 100190, PR China*
[c] *University of Chinese Academy of Sciences, Beijing 100049, PR China*

ARTICLE INFO

Article history:
Received 8 October 2012
Received in revised form 3 February 2013
Accepted 15 March 2013
Available online xxx

Keywords:
Anomaly detection
Web applications
Local outlier factor

ABSTRACT

The failure of Web applications often affects a large population of customers, and leads to severe economic loss. Anomaly detection is essential for improving the reliability of Web applications. Current approaches model correlations among metrics, and detect anomalies when the correlations are broken. However, dynamic workloads cause the metric correlations to change over time. Moreover, modeling various metric correlations are difficult in complex Web applications. This paper addresses these problems and proposes an online anomaly detection approach for Web applications. We present an incremental clustering algorithm for training workload patterns online, and employ the local outlier factor (LOF) in the recognized workload pattern to detect anomalies. In addition, we locate the anomalous metrics with the Student's *t*-test method. We evaluated our approach on a testbed running the TPC-W industry-standard benchmark. The experimental results show that our approach is able to (1) capture workload fluctuations accurately, (2) detect typical faults effectively and (3) has advantages over two contemporary ones in accuracy.

1. Introduction

Web applications (e.g., e-commerce application) provide simultaneous services to a large number of users on the Internet. The failure of these services often affects a large population of customers, and leads to severe economic loss. For example, 1 h downtime of Paypal led to more than 7.2 million dollar loss in transactions in August 2009 (Shankland, 2009). Tellme networks estimates that operators take 75% of the failure recovery time to detect faults (Chen et al., 2004). The study also shows that detecting faults in advance prevents 65% of failure occurrences (Oppenheimer et al., 2003). Anomalies, which do not conform to a defined notion of normal behavior, often demonstrate the occurrence of faults (Chandola et al., 2009). Therefore, detecting anomalies is essential for improving the reliability of Web applications.

System operators usually employ management systems (e.g., IBM Tivoli and HP OpenView) to set rules to trigger alerts based on the collected monitoring data. However, it is difficult to manually set suitable thresholds for so many metrics in a complex Web application. Current approaches use specific analytic forms to model correlations among metrics. The analytic forms include linear regression (Jiang et al., 2006b), locally weighted regression (Munawar and Ward, 2007), Gaussian Mixture Model (Zhen et al., 2006)), and so on. These approaches detect anomalies, when the modeled correlations are broken. They can be easily extended to many applications without domain specific knowledge.

However, commercial cloud computing environments are characterized by the extreme diversity of Web applications and dynamic workloads from the Internet. They have raised great challenges for detecting anomalies online. Enterprise cloud such as Google Apps allows more than four million enterprises to deploy their applications for use by more than forty million users (Google, 2012). These applications provide different services, and have various workload characteristics. Thus, the current correlation-based approaches suffer from several shortcomings. First, they use the same mathematical form to model correlations among metrics. However, since there probably exist various complex correlations in kinds of applications, it is difficult to model these correlations with the same assumed mathematical model. More importantly, they assume that the modeled metric correlations hold during normal operations, but these correlations vary as workloads fluctuate.

Recent studies have shown that the workload in the Internet fluctuates over multiple time scales in time-of-day and month-of-year effects (Williams et al., 2005). For example, Arlitt et al. (2001) analyzed the log of a large-scale e-retail website, and demonstrated that the workloads were different during various periods.

☆ This article is an extended version of the paper "Workload-Aware Online Anomaly Detection in Enterprise Applications with Local Outlier Factor" that appeared in Proceedings of the 36th Annual Computer Software and Applications Conference (COMPSAC 2012) (Regular Paper).

* Corresponding author at: Institute of Software, Chinese Academy of Sciences, Beijing 100190, PR China. Tel.: +86 1062661583x637; fax: +86 1062661580.
E-mail addresses: wangtao08@otcaix.iscas.ac.cn, twang2008@gmail.com (T. Wang), wj@otcaix.iscas.ac.cn (J. Wei), zhangwenbo@otcaix.iscas.ac.cn (W. Zhang), zhongh@otcaix.iscas.ac.cn (H. Zhong), tao@otcaix.iscas.ac.cn (T. Huang).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.03.060

Unfortunately, current anomaly detection approaches do not adequately take into account the influence of dynamic workloads. We give two examples to illustrate why considering workloads for the anomaly detection in Web applications is necessary as follows:

Example 1: Workloads influence metrics. For a Web application, the CPU utilization is about 20% with 100 concurrent users, while that is about 70% with 300 concurrent users in the normal situation. How do operators decide whether the system is normal or not when the CPU utilization is about 70%? The system is normal with 300 concurrent users, but probably abnormal with 100 concurrent users. However, the current rule based approaches only set a threshold for the CPU utilization without taking into account the number of concurrent users.

Example 2: Workloads influence metric correlations. For an e-commerce website, customers always rush to purchase commodities for special events such as new advertising campaigns, special promotions and approach of holidays like Christmas, while they mostly just browse in ordinary days. Thus, different users' access patterns lead to different correlations between browsing and purchasing related metrics, such as reading and writing operations in database. However, the current metric correlation based approaches describe the correlations among metrics with the same model.

This paper addresses the above problems and proposes an online anomaly detection approach for Web applications. We present a method for training workload patterns with an incremental clustering algorithm, and detect anomalies in the recognized workload pattern with the local outlier factor (LOF). In addition, we employ the Student's *t*-test method to locate anomalous metrics.

The main contributions of this paper are as following:

- We online train and recognize workload patterns considering both the access pattern and request volume with an incremental clustering algorithm through grouping similar workload vectors together.
- We detect anomalies in a specific workload pattern, which provides a significant advantage over current works without considering the influence of workload on anomaly detection.
- We employ LOF to detect anomalies, instead of modeling correlations with assumed mathematical forms in the existing works.
- We evaluate our approach on the TPC-W benchmark. The experimental results demonstrate that our approach is able to capture workload fluctuations accurately, can detect typical faults effectively, and performs better than two contemporary approaches in accuracy.

The remainder of this paper is organized as follows. Section 2 presents an incremental clustering algorithm for recognizing workload patterns. Section 3 presents our online anomaly detection approach in detail. Section 4 presents the experimental results. Section 5 compares our work with related work. Section 6 discusses the limitations of our work and possible extensions. Section 7 concludes the paper.

2. Workload pattern recognition with incremental clustering

2.1. Characterization of workload in Web applications

We characterize the workload of Web applications in this subsection. The object of our study is the component-based transactional Web applications. The most typical representative of these applications is the JEE application. Web applications are composed of components, and process transactional requests. They are usually deployed on a typical multi-tier infrastructure to provide services for online users with web interface. The infrastructure is composed of a web server, an application server and a database server. The web server provides interaction interfaces to users for presenting data in the front tier; the application server supports application logic in the middle tier; the database is used to store persistent data in the back tier. Customers interact with the Web applications through a series of HTTP requests; the object of a request is commonly a web page composed of HTML files generated from web components (e.g., Java Servlet, Java Server Page and EJB); the application server invokes web components to process these requests. Thus, the invocation sequences of web components reflect various workloads, and consume system physical resources (e.g., CPU, memory and network bandwidth).

Customers always access websites to do different operations during different periods. We regard different users' behaviors in accessing a website as different access patterns. We take the industry standard e-business benchmark TPC-W (Menasc, 2002) as an example to describe access patterns. TPC-W simulates the activities of accessing an online bookstore website, and describes 14 different web pages. These pages are divided into six browsing pages and eight ordering pages. TPC-W specifies three different mixes of web interactions according to the ratio of browsing related pages to ordering related pages. Thus, we regard the mixes (including browsing mix, shopping mix and ordering mix) as three access patterns; different access patterns lead to different kinds of resource utilization.

To characterize workloads, the traditional methods always choose metrics (e.g. hits/s, pages/s, bits/s), and manually set their thresholds to differentiate various workloads. For example, a workload is classified as type one when the speed of data transfer is from 5 M to 10 M bits/s. However, because the workload is associated with many factors (e.g., the number of concurrent requests, users' access patterns and request types), the traditional methods based on metric threshold are not suitable for current transactional Web applications.

First of all, we consider the request density, which is decided by the request arrival rate (i.e., the number of requests arrived per second) and the think-time (i.e., the interval between requests). Moreover, a component has different resource utilization under various access sequences (Urgaonkar et al., 2005); different customers exhibit different navigational patterns (Menasc et al., 1999). Thus, we ought to take into account the behavior of customers with similar navigational patterns. We introduce the request matrix (*rm*) to represent a workload, which considers both the request density and the users' access pattern. Given *k* components in an application, we get a *k*-order matrix. The *k* is the number of web components (e.g., servlet and Jsp) in a Web application. For example, if a Web application is composed of three servlet and four Jsp files, the *k* is seven in our method. A state represents a request type; the frequency of state transitions during a period describes the users' access pattern. Let's denote:

$$rm = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1k} \\ p_{21} & p_{22} & \cdots & p_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ p_{k1} & p_{k2} & \cdots & p_{kk} \end{bmatrix} \tag{1}$$

where p_{ij} represents the frequency of transitions from state i to state j, and k is the number of components.

For convenience, we introduce workload vector (*wv*) transformed from *rm* to characterize the runtime workload. Let us denote

$$wv = \{e_1, e_2, \ldots, e_m, \ldots, e_{k\times k}\}, \tag{2}$$

where $e_m = p_{ij}$, $m = (i-1) \times k + j$, and p_{ij} is an element in *rm*.