Expert Systems With Applications 89 (2017) 188–204

Contents lists available at ScienceDirect

Expert Systems With Applications

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

An unsupervised approach to online noisy-neighbor detection in cloud data centers

Tania Lorido-Botran[a,*], Sergio Huerta[a], Luis Tomás[b,1], Johan Tordsson[b], Borja Sanz[a]

[a] DeustoTech – Computing, University of Deusto, Avenida de las Universidades 24, Bilbao 48007, Spain
[b] Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden

ARTICLE INFO

Article history:
Received 31 January 2017
Revised 22 July 2017
Accepted 24 July 2017
Available online 25 July 2017

Keywords:
Anomaly detection
Virtual machine
Cloud computing
DPGMM
Noisy-neighbor effect
Similarity distances

ABSTRACT

Resource sharing is an inherent characteristic of cloud data centers. Virtual Machines (VMs) and/or Containers that are co-located in the same physical server often compete for resources leading to interference. The noisy neighbor's effect refers to an anomaly caused by a VM/container limiting resources accessed by another one. Our main contribution is an online, lightweight and application-agnostic solution for anomaly detection, that follows an unsupervised approach. It is based on comparing models for different lags: Dirichlet Process Gaussian Mixture Models to characterize the resource usage profile of the application, and distance measures to score the similarity among models. An alarm is raised when there is an abrupt change in short-term lag (i.e. high distance score for short-term models), while the long-term state remains constant. We test the algorithm for different cloud workloads: websites, periodic batch applications, Spark-based applications, and Memcached server. We are able to detect anomalies in the CPU and memory resource usage with up to 82–96% accuracy (recall) depending on the scenario. Compared to other baseline methods, our approach is able to detect anomalies successfully, while raising low number of false positives, even in the case of applications with unusual normal behavior (e.g. periodic). Experiments show that our proposed algorithm is a lightweight and effective solution to detect noisy neighbor effect without any historical info about the application, that could also be potentially applied to other kind of anomalies.

1. Introduction

1.1. Motivation

Cloud data centers are able to run millions of applications (Foster, Zhao, Raicu, & Lu, 2008). Each application service or task is typically encapsulated in a Virtual Machine (VM) or container. VMs or containers with different resource usage needs may be co-located in the same physical machine. Resource sharing (including CPU, memory or cache) may cause resource contention bottleneck, i.e., two VMs (or group of containers) compete for the same resources, but the resource capacity is not enough for both of them. This leads to *anomalies* in the resource usage of the application that may penalize application performance.

Resource management in virtualized environments typically makes use of consolidation or overbooking techniques, which just increments the risk of VM interference. Application malfunctioning translates directly into financial penalties: end-users will be discouraged from using the application, or cloud providers must compensate the client for SLA (Service Level Agreement) violations. These are some real numbers from large corporations (Huang, Maltz, Li, & Greenberg, 2011): Amazon suffers from 1% decrease in sales for additional 100 ms delay in response time, while Google reports a 20% drop in traffic due to 500 ms delay in response time. Thus, detecting performance issues is mandatory from the cloud provider's perspective, in order to avoid performance degradation that might cause significant economical losses.

1.2. Problem statement and goal

VMs/containers in the same physical server share some resources, such as CPU, memory, or cache hierarchies. Resource sharing may lead to VMs/containers affecting or being affected by other co-located VMs/containers. The *noisy neighbors* effect is an analogy for this interference (Pu et al., 2010). It is reflected as anomalous

* Corresponding author.
E-mail addresses: tania.lorido@deusto.es (T. Lorido-Botran), shuerta@deusto.es (S. Huerta), luis@cs.umu.se (L. Tomás), tordsson@cs.umu.se (J. Tordsson), borja.sanz@deusto.es (B. Sanz).
[1] Currently at Red Hat, Spain.

http://dx.doi.org/10.1016/j.eswa.2017.07.038
0957-4174/

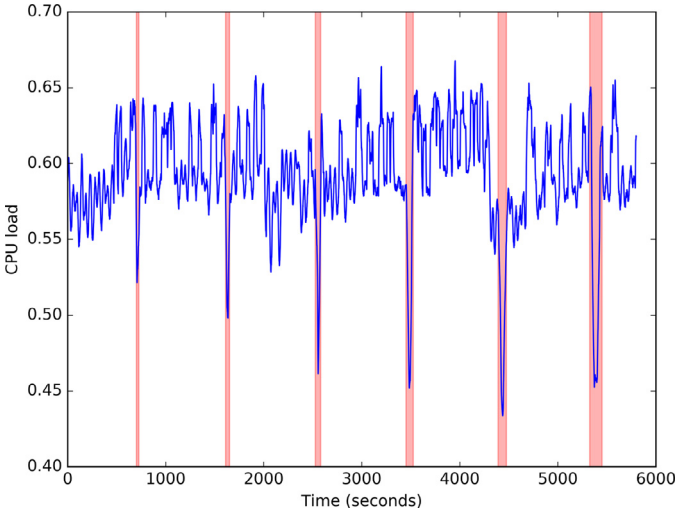

Fig. 1. Sample noisy neighbor effect on CPU load. Anomalies are shaded areas (in red color). Data has been collected from a benchmark application (a static website) running on a Docker container. Anomalies represents interference caused by another process or container; the benchmark application cannot access to the CPU share that was initially allocated to it. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

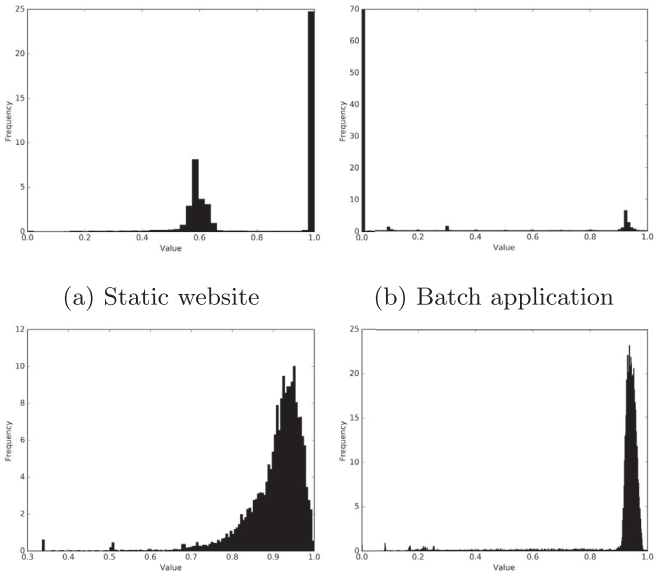

Fig. 2. Histogram of CPU load for different applications.

resource usage from the service running inside the VM or container (see Fig. 1). Its detection imposes several challenges: (1) the normal resource usage pattern is application-specific, with unknown distribution, and (2) prone to change due to workload variations; and (3) even the *anomaly* definition is application-specific and it might appear under different forms.

The *goal* of this paper is to propose an online, lightweight and application-agnostic solution to detect anomalies caused by the noisy neighbor problem.

1.3. Existing solutions

There are different alternatives to address the noisy neighbor problem. Some lie under the mitigation or avoidance approaches (Beloglazov & Buyya, 2013; Bobroff, Kochut, & Beaty, 2007; Wood, Shenoy, Venkataramani, & Yousif, 2009), others in the detection side (Bodik, Goldszmidt, Fox, Woodard, & Andersen, 2010; Silvestre, Sauvanaud, Kaâniche, & Kanoun, 2015; Wang, Talwar, Schwan, & Ranganathan, 2010).

The first obvious technique is to try avoiding the problem. Several solutions have been proposed to deal with performance issues coming caused by resource interference. There exist techniques to provide particular resource isolation (mitigation), i.e. CPU pinning (IBM, 2012). Schedulers might try to select applications with compatible profiles: e.g. CPU-intensive with memory-intensive one. Still, cloud data centers are highly dynamic due to different factors e.g. rapid elasticity (Jula, Sundararajan, & Othman, 2014), VM migrations, varying incoming workloads. Thus, anomalies in performance will happen, and detection algorithms are a *real* need in data centers.

There are two main strategies in any detection problem, supervised and unsupervised approaches. Supervised learning algorithms are suited for recognizing well-known anomalies, but they require labeled datasets. It is difficult, if not totally impossible, to obtain labeled training data (i.e., measurement samples associated with normal or abnormal labels) from production cloud systems (Dean, Nguyen, & Gu, 2012). The need for well-labeled training data greatly limits the scope of their application for real-time use Ibidunmoye, Hernández-Rodriguez, and Elmroth (2015). In contrast, unsupervised learning algorithms require no labeled data. In addition to this, such algorithms are particularly suitable for detecting unknown anomalies in cloud data centers where precise definition of anomaly characteristics may not always exist (Ibidunmoye et al., 2015).

Anomaly detection methods can be classified into two subcategories, which are threshold-based and statistical methods (Wang et al., 2010). Threshold-based approaches are too simple and usually ineffective when anomaly properties vary over time. In contrast, statistical methods are often used for anomaly detection in the cloud (Dean et al., 2012; Sauvanaud, Silvestre, Kaâniche, & Kanoun, 2015; Wang et al., 2010), but they usually suffer from high computing overheads and often require prior knowledge about application (Wang et al., 2010).

CPI2 (Zhang et al., 2013) approach falls under this category of statistical detection methods. Authors propose adjusting a distribution probability to the CPI metric (generalized extreme vale in their case), and take values that exceed 2σ (or 3σ) from the mean as outliers. However, we can easily see in Fig. 2 that resource usage (CPU in this case) of different applications may show different probability distributions. Therefore, an anomaly detection algorithm cannot just assume a concrete probability distribution.

In conclusion, both avoidance and detection techniques may (and should) be used together. The current manuscript focuses on solutions for anomaly detection, more specifically, for the noisy neighbor's effect. From the literature (Ibidunmoye et al., 2015), we can conclude that an unsupervised approach would be suitable in an environment where there is no a-priori knowledge, and also that an statistical approach would be effective, provided that we find a way to model any resource usage pattern (probability distribution) while keeping its computational cost low.

1.4. Our approach

Given the literature, we have come up with the following requirements for our algorithm:

- *Application-agnostic:* That is, the algorithm should not require any a-priori knowledge of the application. We have seen that each application has different resource usage patterns (check Fig. 2).