Blackbox Strategies for Detecting Service Performance Anomalies in Virtualized Environments

Olumuyiwa Ibidunmoye
Erik Elmroth
{*muyi,elmroth*}*@cs.umu.se*

Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden
Email: muyi@cs.umu.se

Abstract

In order to prevent violation of service-level objectives and to guarantee good user experience, detection of symptoms such as slow application response, degraded transaction throughput, and service outages, is crucial. We propose a black-box approach for detecting such symptoms in service performance behaviour without intrusive application instrumentation. In case a known baseline behaviour exists, we employ kernel density estimation to discover deviations from a given set of baseline measurements. Conversely, when no baseline exists, we apply statistical process control charts on prediction errors obtained from Holt-Winter's double exponential smoothing to identify anomalies in metric time-series. We evaluate our methods on tail response times traces collected from experiments conducted in a real testbed under realistic load and fault injections. Results show the applicability of our approach for improving service assurance and also demonstrate how service level anomalies correlate with system-level events such as resource contention and bottlenecks.

1 Introduction

Modern application services are characterized by complex architectures and unpredictable load traffic (such as load spikes and flash-crowd[1] behaviour). These factors often explain performance anomalies such as slow application response, degraded transaction throughput, and outages in cloud-based application services today. Also, the nature of the underlying virtualized infrastructure in which most services are hosted affects service performance in many ways. Cloud infrastructure platforms, such as Amazon EC2, package and provision computational resources in virtual machines (VMs) to numerous service providers in an on-demand manner. Though the initial capital expenditure

[1]Internet phenomenon where a network suddenly receives a huge influx of traffic due to incidents such as breaking news, natural disasters, demise of famous persons, etc.

is greatly minimized for service providers, the co-location of heterogeneous services from multiple providers is sometimes a source of concern for service owners.

According to our previous survey [1], cloud service performance anomalies are often the manifestation of many problems ranging from application-level bugs, workload spikes, and correlated systems faults (e.g. resource contention). Fig. 1 is schematic diagram of the cause-effect relationships between issues from disparate sources and how they together induce capacity bottlenecks and eventually anomalies in service performance.

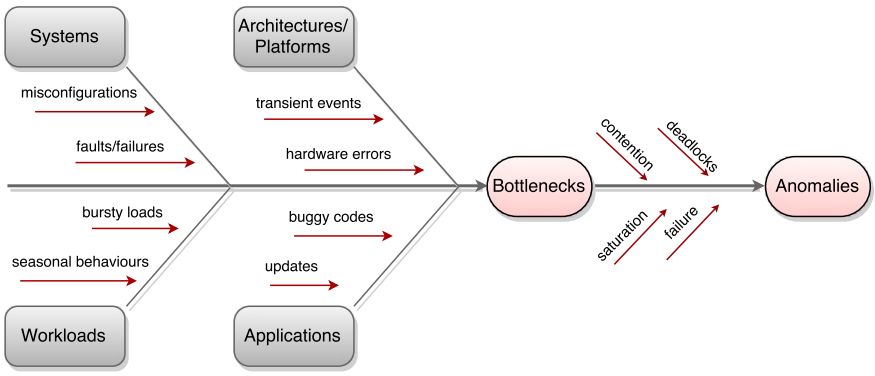


Figure 1: Causes of performance anomalies in systems (source: [1]).

1. *Applications*: Application-level issues such as incorrect parameter settings, buggy codes, and software updates may induce unexpected performance behaviour [2].

2. *Workload*: Bursty workloads, such as caused by flash-crowds, and change in seasonal load patterns may induce undesirable effects such as congested queues, oversubscribed threading resources, and eventually intermittent or sustained spikes in performance measurements.

3. *Architectures and Platforms*: Sometimes transient errors or events in the underlying architecture or operating system–such as the JVM & Intel SpeedStep technology [3], may introduce hard-to-detect transient bottlenecks in systems.

4. *Systems*: System-level issues such as misconfiguration, faulty components, or capacity bottlenecks may induce performance anomalies in many ways. In cloud environments, one major culprit is the inherent performance interference experienced when applications with similar resource needs and workload patterns are co-located on a single server.

Due to the complexity of the runtime environment, diagnosing service performance issues has become a major task in the life-cycle of virtualized services. Service providers generally want to be able to detect unwanted behaviour quickly and in real-time. In our survey [1], we observed that, while there exist many studies focusing on the detection of anomalies in multi-dimensional system-level performance metrics (as done in [4] [5] and [6]), studies on detecting application performance anomalies are few. Existing approaches for detecting anomalies in application performance metrics (e.g. response time and throughput) either resort to simplistic mechanisms based on thresholding (such as in [7], [8], and [9]) or require intrusive application instrumentation (such as tracing request flows in [10] and [11]).

In this paper, we focus on achieving robust anomaly detection in application performance metrics (APM) under two scenarios. In the first scenario, we assume there exist a baseline profile–a collection of *known* or *typical* metric values collected over a period of stable service workload and configuration. The task is to detect deviations from the given baseline. This is informed by trends in the fields of network [12] and database [13] management, where deviations in real-time traffic measurements from known baseline (typical) behaviour are considered as anomalies [14]. In the second scenario, the notion of typical behaviour is unknown a priori and we must detect deviations in prevailing service behaviour. We phrase these two cases in the following questions:

Q1: *How to detect deviations in real-time service performance measurements based on a given baseline profile?*

Q2: *How to detect deviations in real-time service performance measurements when a baseline is a priori unknown ?*

To address *Q1*, we perform a behaviour-based anomaly detection by exploiting the probability distribution of the given baseline profile. A non-parametric kernel density estimation (KDE) technique is used to detect point anomalies in real-time APM measurements. Given that there exist no baseline in case of *Q2*, we perform prediction-based anomaly detection instead. The prediction-based approach performs a one-step ahead forecast of metric value, tracks the residual errors and flags points where statistical properties (such as mean shift) of the residuals change abruptly. Forecasting is done using trend-preserving double exponential filtering while exponentially weighted moving average (EWMA) control chart is used to detect mean shifts.

In Section 2, we layout details of the behaviour-based approach using the KDE model while Section 3 describes how the forecasting and control chart methods are constructed for predictive anomaly detection. We describe the experimental approach for evaluating the techniques in Section 4. Evaluations and results are presented in Section 5. We conclude with some discussions, description of related works and conclusion in sections 6, 7, and 8 respectively.

2 Behaviour-based anomaly detection

Traditionally, APM values are said to be anomalous when they violate a reference point (i.e. threshold). For example, suppose $X = \{x_i\}_{i=1}^{n}$ is a set of n observations of a given performance metric, a threshold-based anomaly detection scheme may define a threshold $\mathcal{T}$ as $\mathcal{T} = k\sigma_X + \mu_X$ where μ_X and σ_X are the mean and standard deviation of values in X respectively and k is a multiplicative factor of the standard deviation. An observations $x_i \in X$ is classified anomalous if it satisfies the condition, $x_i \geq \mathcal{T}$. The idea behind this threshold is that normal values of the performance metric are expected to be within k standard deviations (σ_X) about the mean μ_X. There are some obvious limitations to this approach. First, the assumption is that the distribution of APM values will be Gaussian (i.e. normally distributed) and therefore will result in greater false alarms on data with arbitrary or non-Gaussian distribution. Secondly, thresholds are hard to appropriately specify and may soon become out-dated due to changing runtime environment and workload patterns. It is also not suitable for scenarios where the reference point is a set of baseline measurements rather than a scalar value. Such [13].

In this section we present a non-parametric approach (similar to [15]) to address the aforementioned issues with scalar thresholds. Given a set of baseline measurements, our method estimates a model of the unknown probability distribution of the baseline using Kernel Density Estimation (KDE). We subsequently classify new observations (such as real-time metric measurements) based on the conditional densities estimated by the KDE model. However, the standard KDE algorithm

is known to produce spurious density approximations due to the use of global bandwidths. We implement an adaptive KDE following the procedure in [16] by incorporating local bandwidths to account for local variations in the data. Due to the time complexity of the KDE we also implement both the standard and adaptive KDE using k-dimensional trees [17] and compare their performance later in Section 5.

2.1 Kernel Density Estimation (KDE)

The basic idea of the behaviour-based anomaly detection has to do with exploiting the *probability density function* (PDF) or *density* of the given baseline values. The PDF or density of a continuous random variable is a function that relates the relative likelihood for the random variable to assume a given value in a population [18]. Given comparable capacity configurations and workload characteristics, metric values obtained from a running application should reveal similar metric distribution as the baseline from the same application under normal conditions. Significant deviations from the baseline distribution may indicate an anomaly. Some existing studies have proposed similar approaches in performance studies such as in [19] and [20].

The KDE is a proximity-based technique for detecting novel patterns in data with the assumption that normal instances are far more frequent than anomalous instances [21]. Given a univariate set of metric values $X : \mathbb{R}^{n\times 1}$, the unknown density function $\hat{f}(X)$ can be estimated by aggregating a set of scaled kernel functions as follows;

$$\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K_h(\frac{x - x_i}{h}) \tag{1}$$

Each kernel function $K_h(\cdot)$ is centered at each data point with width controlled by a *bandwidth* parameter h. The bandwidth h, acts as a smoothing factor controlling the variance from one point to another in the data. Since h remains the same for all kernels, it is also referred to as a *global bandwidth*. The scaled kernel $K_h(\cdot)$ of the form $\frac{1}{h}K(\frac{\cdot}{h})$ is any symmetric, function that satisfies [22]

$$\int_{-\infty}^{\infty} K_h(\cdot)d(\cdot) = 1$$

and

$$K_h(\cdot) \geq 0$$

We have used a set of Gaussian kernels for computing the density estimate $\hat{f}(x)$ according to Eq. 1. Though other standard kernels such as the Tophat, Epanechnikov, Exponential, etc., exist, the Gaussian kernel is most commonly used in density estimation due to its ability to produce smooth density estimates [23]. A Gaussian kernel is defined as [21]:

$$K_h(y) = \frac{1}{h\sqrt{2\pi}}\exp(-\frac{y^2}{2h^2}) \tag{2}$$

The choice of kernel $K_h(\cdot)$ does not significantly affect the accuracy of the estimate as much as the bandwidth parameter h. High values of h produce very smooth estimates but may ignore local variance while low values produce less smooth but potentially more accurate estimates [15]. The value of h may be set to an arbitrary value, to a plug-in values such as the AMISE and Silverman's approximations [24], or to values derived through grid-search cross-validation.

2.2 Adaptive KDE (AKDE)

The use of a fixed global bandwidth, h in Eq.1 ignores local variations in density from one data point to another [21]. We follow the procedure in [16] to adapt the amount of smoothing based on the local density in the data (i.e. the bandwidth, h, is allowed to vary from one data point to another). First, a pilot density $\hat{f}^*(x)$, is estimated from the entire dataset using Eq. (1), such that $\hat{f}^*(x_i) > 0, \forall i$. Then individual local density is computed using the square root rule as,

$$\lambda_i = \{\frac{\hat{f}^*(x_i)}{g}\}^{-\alpha}$$

where g is the geometric mean of $\hat{f}^*(x_i)$, that is,

$$\log(g) = n^{-1} \sum_{i=1}^{n} \log(\hat{f}^*(x_i))$$

. The adaptive KDE estimate becomes:

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h\lambda_i} K_h(\frac{x - x_i}{h\lambda_i}) \tag{3}$$

The term $h\lambda_i$ now acts as the local bandwidths such that the width of the kernel placed at each x_i is equal to $h\lambda_i$. Setting $\alpha = 0.5$ is generally recommended as it offers good estimate in most cases [16]. Equation (3) is equivalent to (1) when $\alpha = 0$ since $\lambda_i = 1$. Note that the same value of h used to obtain the pilot, $\hat{f}^*(x_i)$, must be used in computing Eq. (3) so as to ensure that the actual estimate is sensitive to the same scale as the pilot.

2.3 Computational issues with density estimation

The KDE belongs to a class of statistical learning problems called the *generalized n-body problems* with the complexity of a naive solution being $\mathcal{O}(n^2)$. Riegel et al [25] has shown that problems in this category can be efficiently computed in $\mathcal{O}(n\ log\ n)$ using specialized data structures such as the k-dimensional trees [17]. A KD-tree is a spatial data structure for organizing data points in k-dimensional space and readily offer $\mathcal{O}(n \log n)$ k-nearest neighbour computations. The implementation of the KDE models is based on the idea that if a data point $x_0 \in X$ is geometrically distant from a set of points $\vec{y} \subset X$, then it is sufficient to compute the density estimate of x_0 only from the set $X - \vec{y}$, i.e., the k-nearest neighbours of x_0. To implement the standard and adaptive KDE models we used the `KD-Tree` class in `scikit-learn` [26].

2.4 Detecting anomalies based on estimated densities

Given a baseline profile $\mathcal{B}$ containing a set $X = \{x_i\}_{i=1}^n$ of performance metric values. First we learn a model, $\theta_\mathcal{B}$ corresponding to the kernel density estimate, $\hat{f}_\mathcal{B}(X)$, of the baseline values. To classify an observed data point, x_0, we compute a conditional density $p(x_0|\theta_\mathcal{B})$–the likelihood of X taking on the value of x_0 given its probability distribution $\theta_\mathcal{B}$. For a suitably chosen parameter, ϵ, x_0 is classified anomalous if $p(x_0|\theta_\mathcal{B}) < \epsilon$ implying that x_0 is not likely to have been generated by $\theta_\mathcal{B}$. Samples with relatively small $p(x_0|\theta_\mathcal{B})$ are considered anomalous and those for which $p(x_0|\theta_\mathcal{B})$ is high lie in dense regions and thus are normal data.

3 Prediction-based anomaly detection

The approach in Section 2 is based on training a model to recognize new independent data that do not belong to distribution of typical or normal data. In some use cases, it maybe difficult to obtain a baseline or may quickly become obsolete. The challenge is to monitor the service in order to detect unknown events or unexpected changes in service performance behaviour that may suggests workload spikes, system-level faults or resource bottlenecks due to contention among co-located services. The prediction-based anomaly detection exploits the temporal dependency in time series of successive APM measurements to detect abrupt changes in the underlying statistical property of the data (e.g. mean shift). The assumption is that such property is expected to exhibit little or no variation from one time step to another. Consistent shift of a certain magnitude may suggest a change in the time series behaviour and hence an anomaly. In this section we present an online mechanism for achieving predictive anomaly detection using a combination of Holt-Winters forecasting and EWMA control charts.

3.1 Residual generation using Holt-Winters forecasting

We consider a sequence of APM measurements as a time series of the form

$$X = \{x_1, ..., x_{t-1}, x_t, x_{t+1}, ...\}$$

. Exponential smoothing is a common method in statistical forecasting literature and have been successfully used in diverse domains of applications. The basic idea is to forecast future values using weighted averages of past observations, with weights decaying exponentially over time [27]. Holt-Winters [28] forecasting extends basic exponential smoothing to account for trends and seasonality in a time series data. Holt-Winters is based on the assumption that observed time series can be decomposed into three components: a *level* component (the intercept of a signal), ℓ_t, a linear *trend* (the long term direction or slope), b_t, and a *seasonal* (the systematic, calendar related movement) component, s_t. Studies such as in [29] has shown the basic component form of the Holt-Winters model to not produce good estimates for the level and seasonal components. Their work suggests methods for correcting estimates produced by the method. According to Hyndman et al in [27], the additive and error correction form of the smoothing equations is given as:

$$\ell_t = \ell_{t-1} + b_{t-1} + \alpha e_t \tag{4}$$

$$b_t = b_{t-1} + \alpha\beta e_t \tag{5}$$

$$s_t = s_{t-m} + \gamma e_t \tag{6}$$

where α, β and γ are smoothing parameters for the level, trend, and seasonal components respectively and are set to values in the range $[0, 1]$. The index of seasonality, m, denotes the length of a season (e.g. $m = 12$ for monthly data with yearly trend). The one-step-ahead forecast at time t using previous forecast of the level, trend and seasonality at time $t - 1$ is thus

$$\hat{x}_{t|t-1} = \ell_{t-1} + b_{t-1} + s_{t-m}$$

. The corresponding forecast error or residual, e_t, is computed as $e_t = x_t - \hat{x}_{t|t-1}$, where x_t is the observed value at time t and $\hat{x}_{t|t-1}$ is the predicted value using the most recent smoothed components from time $t - 1$. The statistical property of the time series of residuals,

$$E = \{e_1, ..., e_{t-1}, e_t, e_{t+1}, ...\}$$

will be exploited in the next section for anomaly detection.

The Mean Absolute Percentage Error (MAPE) is a commonly used measure of forecast accuracy [27]. The MAPE is defined as

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\frac{|y_t - \hat{y}_t|}{y_t} \times 100$$

where y_t is the observation at time t and $\hat{y}_t$ is the forecast of y_t. The optimal settings for parameters α, β, and γ are those that minimize the MAPE. We will use this strategy to determine optimal parameter settings in Section 5.

3.2 Detecting deviations using EWMA control chart

Statistical process control (SPC) is a collection of tools widely used to achieve stability and monitor variability in production processes [18]. A control chart is a graphical display showing individual values of a quality characteristic, Y, against the sample number or time. The chart contains a center line (CL), the mean of Y under normal conditions. The upper control limit (UCL) and lower control limits (LCL) are horizontal lines drawn above and below the CL respectively [18]. Generally, control charts operate under the premises that output of the monitored process are normally distributed and so appropriates values for the LCL and UCL can be determined from a normal bell curve. Therefore they are not suitable to be applied directly on time series of APM measurements with unknown distribution, trend, or seasonal variation. Instead we apply control charts on forecast residuals obtained in Section 3.1.

The most commonly used control charts are the Shewhart's $\bar{x}$, s, and R control charts. However, these charts are not suitable for APM measurements as they either assume normality or that measurements arrive in batches. We employ the Exponentially Weighted Moving Average (EWMA) control chart. EWMA control charts have been shown in studies [30] to be robust against non-normality and are particularly suitable for case where measurements arrive sequentially rather than in batches [18]. Given forecast residuals $e_1, ..., e_{t-1}, e_t, e_{t+1}, ...$, we construct an EWMA chart of the form:

$$e'_t = \lambda e_t + (1-\lambda)e'_{t-1} \tag{7}$$

Equation (7) exponentially smooth the forecast residuals over time controlled by parameter λ ($0 < \lambda \leq 1$). The center line and control limits for the EWMA control charts is then defined as follows [18]:

$$(UCL, LCL) = \mu_0 \pm L\sigma\sqrt{\frac{\lambda}{(2-\lambda)}\left[1-(1-\lambda)^{2t}\right]} \tag{8}$$

where λ and L are design parameters for the EWMA charts, while λ controls the rate of smoothing, L is a multiple of the standard deviation and controls how sensitive the chart is to shifts around the center line. Parameter L, is generally set to 3 in literature [31]. Suppose the residuals e_i are independent random variables with mean $\mu_0 = \frac{1}{T}\sum_{t=1}^{T} e_t$, the CL, and with variance σ^2, then the variance of each e'_i is given as

$$\sigma^2_{e'_i} = \sigma^2\frac{\lambda}{(2-\lambda)}\left[1-(1-\lambda)^{2t}\right]$$

.

The limitation of this formulation is the assumption that entire time series measurements are available so that we can easily estimate parameters μ_0 and σ a priori before setting up the control

chart. This is not always possible when dealing with dynamic service performance data with storage constraints. Variation in service workload may cause the center line and in turn the upper and lower limits, to shift over time. To overcome this issue we modify Eq. 8 so that parameters μ and σ are computed adaptively as measurements arrive using the Welford algorithm [32]. The Welford technique iteratively updates the mean and variance online using data observed up to each time step. The implication of this is that the control limits UCL and LCL will also vary along with the center line and variance. However, as t grows the term $[1-(1-\lambda)^{2t}]$ approaches unity and the control limits will converge to asymptotic (steady-state) limits $\mu_0 \pm L\sigma\sqrt{\frac{\lambda}{(2-\lambda)}}$ [18].

Observations that violate the control limits are in statistical literature said to be *out-of-control*. Such observation are what we call anomalies since they are indications of changes in the time series behaviour due to load spikes, bottlenecks, bugs or faults. To detect anomalies, we monitor consecutive values of the smoothed forecast errors (e'_t), and out-of-control observations are classified as anomalies. Specifically, out-of-control observations are those for which the predicates $e'_t > UCL$ or $e'_t < LCL$ hold.

A general measure of the effectiveness of control charts is the *Average Run Length* (ARL). The ARL is the average number of points that must be plotted before a point indicates an out-of-control condition [18]. For EWMA control charts there exists a trade-off between the number of out-of-control detection and the ARL based on the value of parameter λ. Lower values of λ yield lower ARL and in turn higher of out-of-control anomalies. Conversely, when λ is close to unity a high ARL is observed and as a result fewer anomalies are recorded. In general, λ should be high enough so as to reduce the false alarm rate and to ensure greater sensitivity to anomalies.

We describe data specific issues such as parameter initialization in Section 5 when we evaluate the algorithms on specific datasets.

4 Experimental Design

The goal of our experiments is to evaluate the two anomaly detection schemes (in sections 2 and 3) on real performance traces obtained from a benchmark web application under realistic load and fault injections deployed in a virtualized environment. In this section we describe the experimental testbed, the test application, workload emulation and the experimental methodology.

4.1 The Testbed

The experiments were conducted on a HP ProLiant physical machine (PM) equipped with a total of 32 CPU cores[2], 56 GB of memory, four 500G SATA disks in RAID10 and Gigabyte Ethernet. The server is virtualized using the Xen Hypervisor [33]. The testbed (in Fig. 2) consists 3 virtual machines, $\text{VM}_{httpmon}$ running the load emulator, VM_{rubis} (configured with 16 VCPUs, 4GB memory and 25GB storage capacity), hosting RUbiS benchmark application, and an antagonist, $\text{VM}_{neighbour}$ (6 VCPUs, 4GB memory and 25GB storage), used to inject faults into the environment. We deploy VM_{rubis} and $\text{VM}_{httpmon}$ on the same PM with compute-level isolation in order to reduce network delays. Compute isolation is achieved by partitioning physical CPUs into two non-overlapping pools, `A` (10 cores) and `B` (22 cores) using Xen's CPUpools policies[3]. Each pool may have entirely different scheduler and VMs are assigned to pools on creation and can be moved from one pool to another. $\text{VM}_{httpmon}$ is allocated to `Zone A` while the other two are allocated to `Zone B`.

[2]Two 2.1GHz AMD Opteron$^{\text{TM}}$ 6272 processors, with 16 cores each.
[3]`http://wiki.xenproject.org/wiki/Cpupools_Howto`

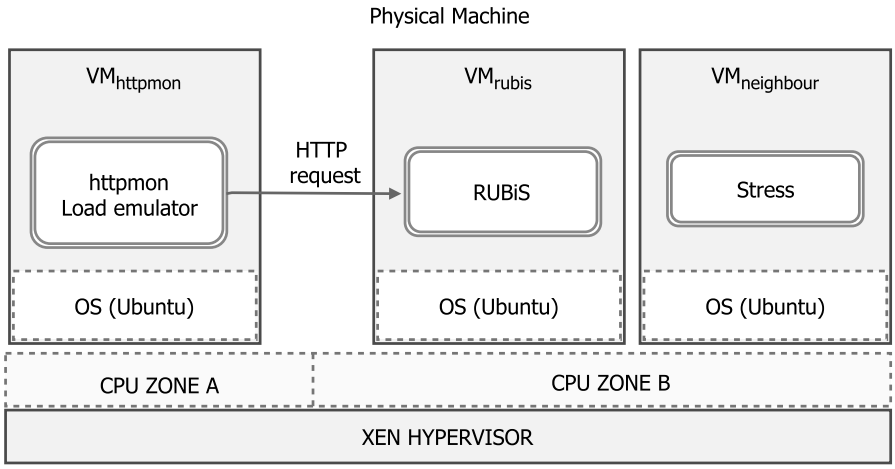


Figure 2: Experimental testbed.

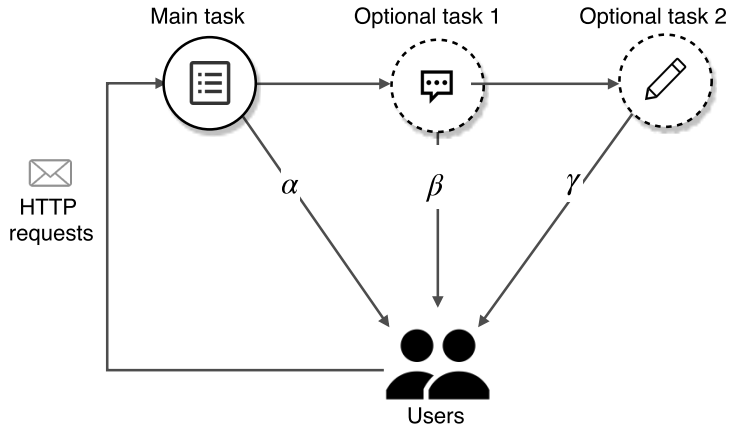


Figure 3: A model of workload mix generated by `httpmon`

4.1.1 Benchmark Applications

The benchmark web application used in our experiments is *RUBiS*[4]. RUBiS is an eBay-like web-based e-commerce application that provides selling, browsing and bidding functionalities. The PHP variant of RUBiS is deployed in VM_{rubis}. The VM is running `Apache 2.0` as its web server (with module `Apache MPM prefork` enabled for thread safety and request isolation), and `MySQL 5.0` as the database server. The `MaxClients`, `ServerLimit` parameters for `Apache` have been set to relatively high values to accommodate high load. Also, to introduce compute and IO perturbations into the environment, we used the open-source `stress`[5] tool deployed in $\text{VM}_{neighbour}$.

[4] `http://rubis.ow2.org/index.html`
[5] `http://people.seas.harvard.edu/~apw/stress/`

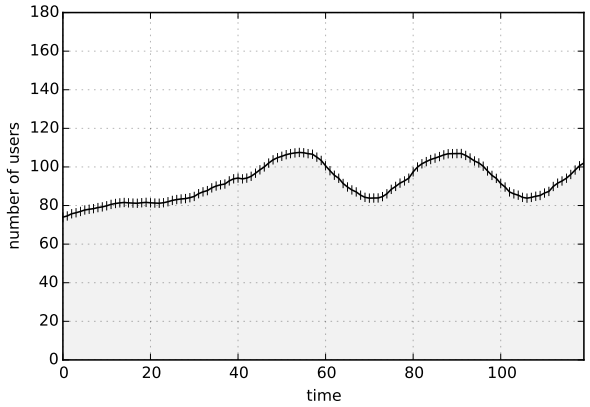
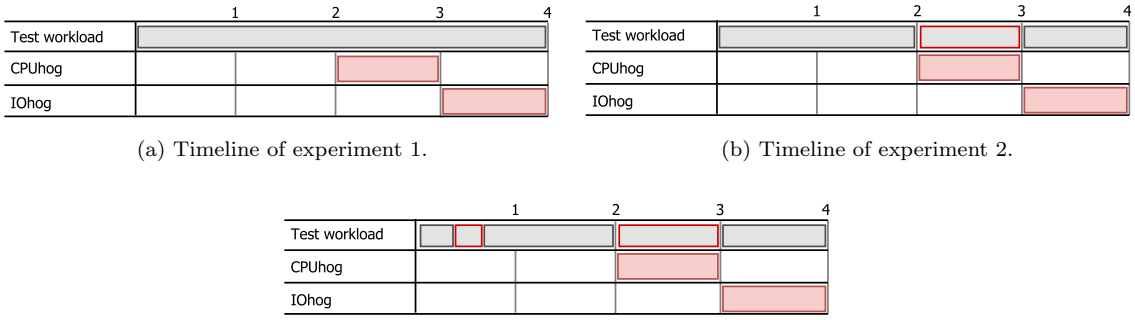


Figure 4: Workload intensity



(a) Timeline of experiment 1.

(b) Timeline of experiment 2.

(c) Timeline of experiment 3.

Figure 5: Ghant charts showing the timeline of experiments and the injection of anomaly-inducing events. Gray boxes in red border correspond to periods where workload spikes have been injected into the test workload while the red corresponds to injection of CPU and IO contention.

4.1.2 Workload and Performance metric

Fig. 3 is a simple model of how the workload mix used in our experiments are is generated. It shows the pattern of interactions between emulated clients and RUBiS. The interaction consist of a dynamic set of users concurrently sending HTTP requests to a main task such as fetching a random product page. With probability α, they exit the application or proceed to perform an optional task (e.g. requesting all past comments about the product) with probability β and/or perform another optional task (e.g. submitting a remark about the product) with probability γ. End-users are emulated using, `httpmon`[6], a versatile HTTP (`GET` or `POST`) traffic generator. We adapted `httpmon` to support the use of arbitrary workload intensity (Fig. 4) and mix (Fig. 3). The `thinktime` parameter of `httpmon` is set to `900`ms in `OPEN` loop mode for all experiments, while the `concurrency` is held constant for one minute per observation from the workload intensity profile. This technique allows us to stage realistic workload and application behaviour.

The performance of the benchmark application is assessed by the Response Time (RT) or latency

[6] `https://github.com/cloud-control/httpmon`

of individual HTTP (`GET/POST`) request. RT is the time elapsed from when the first byte of a request is sent to the time the last byte of its response is received. `HTTPMON` measures request latencies and aggregates by taking the 95^{th} percentile latency of all requests observed over a given time interval. The choice of tail (95^{th} percentile) response time (P_{95}-RT) is because (a) RT metric are commonly used to reason about user satisfaction and quality of experience in Internet services (b) 95^{th} percentile aggregation enables consistent latency measurements [34] and correlates more with violation of service-level objectives (SLO) than averages (which is easily skewed by spikes).

4.1.3 Experiment Process

The first step is to generate the baseline used for behaviour-based anomaly detection in subsequent experiments. This is accomplished by running the benchmark (VM_{rubis}) for about an hour without $VM_{neighbour}$ in the setup in order to eliminate unwanted noise. We injected the synthetic workload intensity and mix according to Fig. 4 and Fig. 3 respectively. The workload mix probabilities are initialized as $\alpha = 0.60, \beta = 0.35, \gamma = 0.05$. We randomly sample the P_{95}-RT over consecutive three-minutes windows to obtain 21 baseline values. The distribution of values in the baseline is shown in Fig. 6. The green region is shown for mere aesthetic since negative latency values are unreasonable. In the evaluation section, we will describe how the SLO-violation annotation is used to quantify the accuracy of the KDE models. We perform three similar experiments to emulate

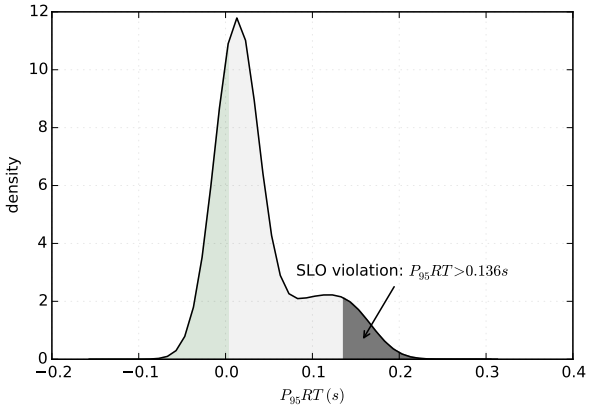


Figure 6: Distribution of the baseline P_{95}-RT.

how load spikes and resource contention due to VM co-location cause anomalies in service-level performance metric behaviour. Each experiment runs for two hours following the same workload profile as the baseline experiment. The P_{95}-RT of the application is sampled at 20 seconds interval. A Ghant chart describing the timeline and composition of the testbed as well as the injection of anomaly-inducing events for each experiment is shown in Fig. 5. The entire duration is divided into four 30-minutes long periods. The `test workload` event refers to running VM_{rubis} while the `CPUhog` and `IOhog` events refer to running $VM_{neighbour}$ to emulate compute and IO contention in the execution environment. For compute contention, we configured `stress` so that it aggressively consumes as much CPU as available in `Zone B`. Also, `stress` is used to induce IO contention by writing blocks of bytes into the disk without disk caching. In addition, we injected workload spikes (by arbitrarily increasing the number of users) between period two to three in experiment 2 and 3, as well as within period one and two in experiment 2.

Table 1: Description of acronyms and symbols

Acronym / Symbol	Method	**Description**
KDE, AKDE	Behaviour-based	Standard and adaptive kernel density estimation
KDETree, AKDETree	Behaviour-based	k-dimensional tree variants of KDE and AKDE
EWMA	Prediction-based	Exponentially weighted moving average control chart
ARL	Prediction-based	Average run length of the control chart
CL, UCL, LCL	Prediction-based	Center line, upper and lower control limits
τ	Behaviour-based	SLO violation threshold (Section 5.1)
h	Behaviour-based	The global kernel bandwidth (Eq. 1 & 3)
ϵ	Behaviour-based	The conditional density threshold (Section 2.4)
α, β	Prediction-based	Holt-Winter's smoothing parameters (Eq. 4 - 6)
ℓ_t, b_t	Prediction-based	Time series level and trend components (Eq. 4 - 6)
λ	Prediction-based	Smoothing parameter for EWMA chart (Eq. 7)
L, σ	Prediction-based	Control limit parameters (Eq. 8)
e_t, e'_t	Prediction-based	Forecast and smoothed errors (Eq. 7)

5 Evaluation

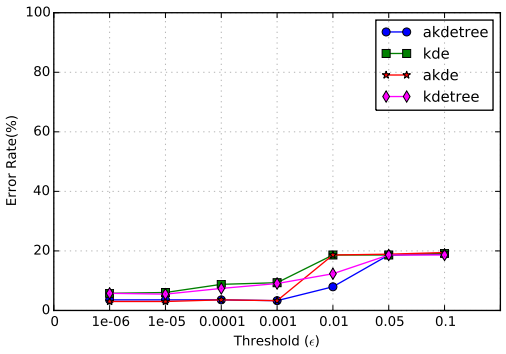
In this section, we evaluate the proposed anomaly detection schemes based on three experiments described in the previous section. To enhance readability, we describe acronyms and symbols used so far in Table 1.

5.1 Behaviour-based anomaly detection

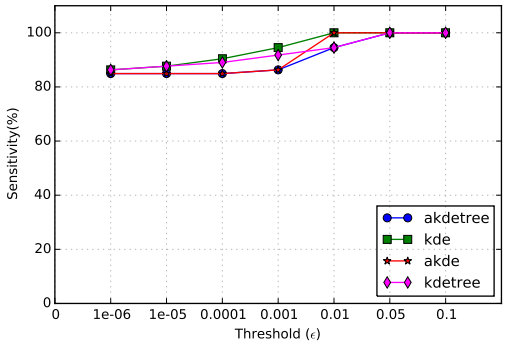
The KDE models described in this paper and the raw data from our experiments are unsupervised in nature, hence it is not so straightforward to quantify the accuracy of the models. We follow the approach in supervised anomaly detection where a label is provided for each observation in the dataset distinguishing which ones are anomalous and which are not [21]. To label our data, we define a threshold τ equal to the 95^{th} percentile of the values in the baseline (see Fig. 6). The dataset is then labeled according to the following rule:

$$y_i = \begin{cases} 0, & \text{if } x_i < \tau, \\ 1, & x_i \geq \tau. \end{cases} \tag{9}$$

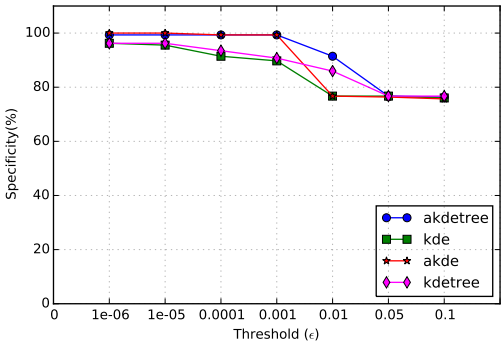
x_i and y_i are the value and class label of observation i respectively. Observations with $y_i = 0$ are normal (below the SLO violation threshold) while $y_i = 1$ are anomalies (above the SLO violation threshold). Fitting a KDE model as described in Section 2 requires setting global bandwidth h to an optimal value. It also requires choosing the conditional density threshold ϵ that guarantees good model accuracy. To select the right parameters we tried different combinations of h and ϵ. We used grid search cross-validation to optimize the value of h which resulted in $h = 10$, a rather high value compared to the Silverman's [23] approximation, $h = \frac{3n}{4}^{-\frac{1}{5}} \approx 0.57$ (recall $n = 21$). We tried different h within the range $(0.5, 10)$ while varying ϵ as well. The observation is that higher bandwidths ($h > 1.5$) result in over-fitting leading the model to mis-classify anomalies as normal observations.



(a) Misclassification rate



(b) Sensitivity



(c) Specificity

Figure 7: Performance of the KDE models for behaviour-based anomaly detection in terms of (a) the overall misclassification rate, (b) ability to correctly identify anomalies, and (c) ability to correctly distinguish normal observations.

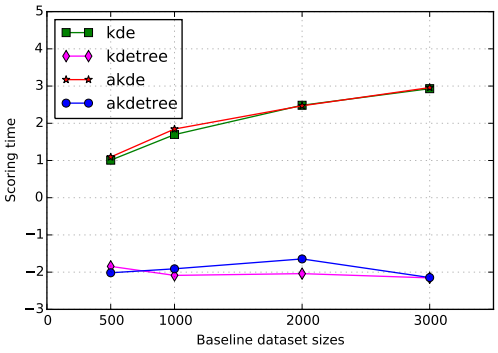


Figure 8: Time requirement of KDE models at varying test data sizes (in log scale).

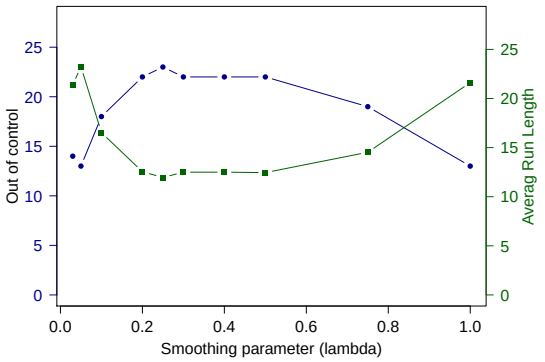
Fig. 7 shows the performance of the KDE models as well as their tree implementations for $h = 1.5$ with varying ϵ while Fig. 7a is the misclassification rate of the model. Fig. 7b is the represents the percentage of actual anomalies in the test dataset that are correctly identified and Fig. 7c is the percentage of normal data instances that are correctly identified by the model. It can be observed that the AKDE model and the tree variants offer consistently low error rates at thresholds below 0.01 after which only the tree variants remain low. Though Fig. 7b indicates that the tree variants are not able to correctly detect anomalies as much as the non-tree variants at lower thresholds, Fig. 7c shows that they are however better at identifying normal data even at higher thresholds. For the given baseline, a threshold of 0.01 seems to offer the best trade-off between model accuracy and sensitivity to anomalies. At this threshold, the tree implementation of AKDE is the preferred model because it offers the best trade-off between mode accuracy and ability to correctly classify both anomalies and normal samples. Furthermore, we fit the models on data obtained from experiments 2 and 3 with parameter settings $h = 1.5$ and $\epsilon = 0.01$. The tree-based AKDE model also performed better than the rest at error rates of 8% and 14%, and sensitivity of 95% and 99% for experiments 2 and 3 respectively.

Fig. 8 confirms the $\mathcal{O}(n^2)$ complexity of the standard KDE model under varying test data sizes. The plot illustrate the number of seconds required (in log scale) to completely compute the conditional density of observations in test datasets of varying sizes. With their $\mathcal{O}(n \ \log \ n)$ time complexity, the tree variants offer greater speed.The tree models thus become scalable and faster as the data size increase in comparison to the standard implementations. However, the AKDE model demands additional time requirement for computing local density bandwidth using the pilot estimate which can get somewhat expensive for large baseline data.

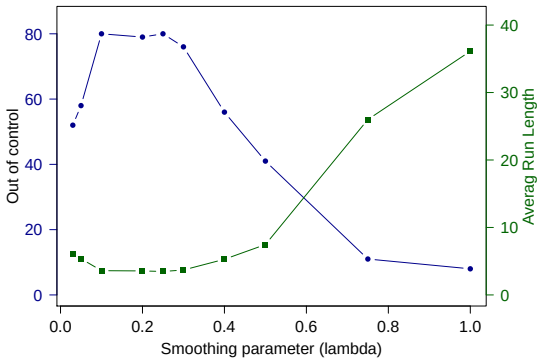
5.2 Prediction-based anomaly detection

In this section, we use the same datasets from experiments 1, 2 and 3 as in Fig. 5 to evaluate the capability of the predictive anomaly detection scheme to detect anomalies in service performance. The most important part of this is parameterizing the Holt-Winter's forecasting algorithm and the EWMA control chart. The Holt-Winter's forecasting scheme described in Section 3 incorporates seasonality in the data through the s_t term of Eq. 6, controlled by parameter γ. Since datasets from our experiments do not exhibit any seasonality, we set γ to zero. This reduces the problem to only finding the optimal values for α and β needed to control the time series level and trend respectively. Recall that lower values of α gives more importance to observations in the past than the recent. The same applies to β, values close to zero means that trends in the distant past affect the current forecast than the recent trends. We searched over a grid of possible combinations of α and β, and for each pair we fit the Holt-Winter's model and select the pair with low mean absolute percentage error (MAPE). Hence, we obtained $\alpha = 0.9$ and $\beta = 0.3$ for the Holt-Winter's forecasts. To initialize Holt-Winter's forecasting we set $\ell_0 = x_1$ and $b_0 = x_2 - x_1$, where x_1 and x_2 are the first and second observation of the time series respectfully.

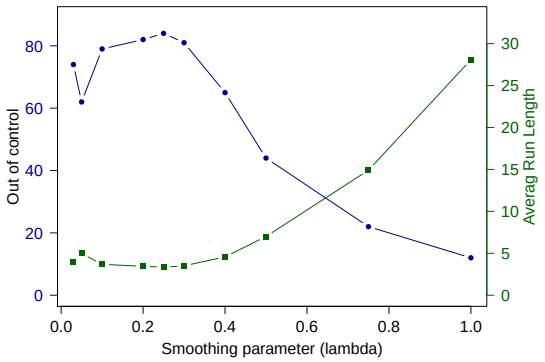
For each test data, the Holt-Winter's method is used to generate forecast residuals on which the EWMA control chart is applied. The parameter L of Eq. 8 controls how wide the control limits are and in turn the sensitivity to shifts in the behaviour of measurements. We set L to the value of 3 (i.e. 3σ) as commonly done in literature [18] [30]. Equation 7 requires that parameter λ is set appropriately to control both the rate at which out-of-control alarms are raised and the average run length (ARL). In Fig. 9 we illustrate the trade-off between number of out-of-control (anomalies) detection and the ARL when λ is varied between 0 and 1. Control charts with higher ARL are generally known to have fewer out-of-control detections and false alarms. Hence, λ should be chosen to yield lower ARL while sensitive enough to shifts of varying magnitude in the time series behaviour.



(a) Experiment 1



(b) Experiment 2

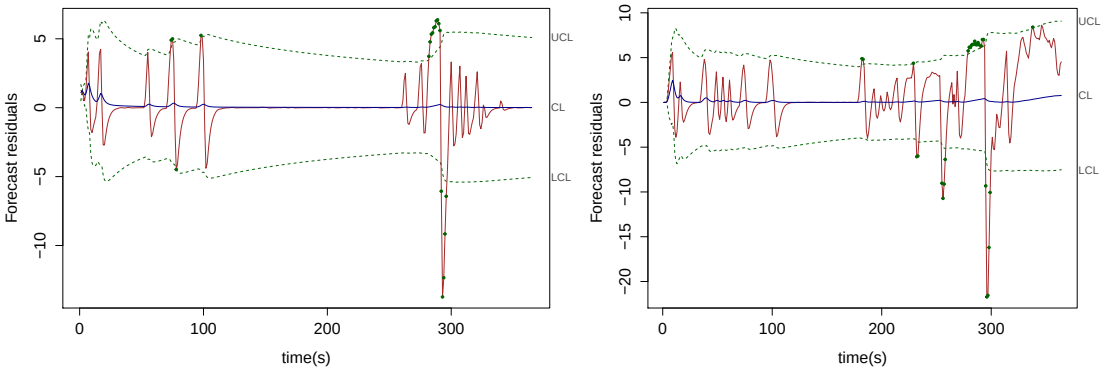


(c) Experiment 3

Figure 9: Predictive anomaly detection: Plots showing how the number of anomalies detected and the Average Run Length (ARL) vary with respect to λ at $L = 3$.

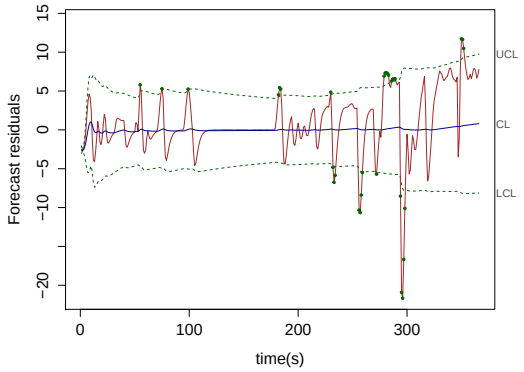
Using Fig. 9, we set the value of λ to 0.81, 0.6, and 0.61 for experiments 1, 2 and 3 respectively. These values correspond to the intersection of the growing ARL line and the decaying out-of-control line that partitions the plots into two regions; a region of low ARL and high out-of-control detection and a region of high ARL but low out-of-control detection. To initialize the EWMA control chart, we set $e'_0 = e_1$, CL to the mean of the first $k = 3$ residuals and σ^2 to the variance of this mean as described in Section 3.

Fig. 10 are the control charts obtained when the adaptive EWMA control chart scheme is applied to Holt-Winter's forecast residuals from the experiments. Notice how the control limits (UCL, CL and LCL) follows the trend and shift in the signal. The out-of-control points (i.e. anomalies) are the indicated green points on the plots. The three control charts clearly detect more anomalies at points corresponding to between periods 2 and 4 respectively of the time-line in Fig. 5, indicated as a period of unstable shifts in the time series around the center line. The workload spikes injected in experiments 2 and 3 does not seem to induce as much shift in the data as resource contention between



(a) Experiment 1 ($\lambda = 0.81$)

(b) Experiment 2 ($\lambda = 0.6$)



(c) Experiment 3 ($\lambda = 0.61$)

Figure 10: Predictive anomaly detection: Adaptive EWMA control chart applied to smoothed Holt-Winter's forecast errors using $\alpha = 0.9$, and $\beta = 0.3$. The indicated control limits are obtained using $L = 3$ (i.e. 3σ). Anomalies are the out-of-control points in green.

periods 2 and 4. Hence, fewer anomalies are detected within that period. This behaviour suggests that sometimes low-level events such as resource saturation may not readily manifest as anomalies in service performance. By manual observation of the pages accessed by requests to VM_{rubis} and its system metrics, most of the spikes between period 2 and 4 in the charts is due to CPU saturation and database reads and writes that coincide with IO noise from $\text{VM}_{neighbour}$. In addition, spikes in the beginning of the charts are due to initial disk activity before the disk cache is sufficiently warmed up and the initial phase through which the control charts adjusts to the steady behaviour of the time series.

6 Discussions

The behaviour-based scheme is useful for two scenarios (a) when a baseline profile is available and the task is to identify anomalies in previously unseen APM measurements and (b) when a baseline is not explicitly provided but the entire set of APM dataset is and the task is to identify anomalous values in the given data off-line. The first scenario, then the algorithm described in Section 2 is applicable directly. In the second scenario, the task is to identify anomalous measurements from the entire data off-line. The behaviour-based technique is still applicable by using the entire data as both the baseline and test data. First we fit the AKDE model on the data and an appropriate value is determined for the density threshold ϵ. In one more pass over the data, the model is used to classify each observation as anomaly or normal. The prediction-based approach on the hand is best for the second scenario and most suitable when measurements arrive sequentially in a pre-defined interval.

In comparison, the EWMA control chart technique generally detects fewer anomalies and false alarms than the KDE models due to its adaptive nature. The exponential smoothing of the residuals means that spurious spikes are smoothed out and alarms are only raised for sustained anomalies. The KDE model however detects more anomalies accurately than the control chart scheme some of which are false alarms. To reduce occasional spurious alarms due to spiky data, on method is to define a hanging window within which we set a threshold on the maximum number of consecutive anomalies. For example, let $\mathcal{A}$, be the number of sequential anomalies detected within a k-length window, $\mathcal{W}$, an alarm is raised when $\mathcal{A} > \mathcal{L}$, where $\mathcal{L}$ is the maximum number of permissible consecutive anomalies.

For longer experiment periods, we would expect the performance of the KDE to worsen in time as the baseline becomes obsolete. The technique can however be adapted for use in an on-line manner like the predictive scheme by taking recent windows of observations as the baseline and using the trained model to predict in the next window. The classification threshold ϵ can also be set to reflect changing SLO thresholds. However, deciding whether to use the tree variants of the standard KDE or adaptive KDE will depend on the trade-off between speed and accuracy of detection. On one hand, the adaptive KDE will yield fast $\mathcal{O}(n \ \log \ n)$ complexity but require significant time for computing local bandwidths through pilot density estimation especially for large baseline data. On the other hand, the tree implementation of the KDE will offer fast classification without prior pilot estimate at the expense of accuracy.

Though the prediction-based technique is easily amenable to real time deployment, it does require getting the parameter settings right. Also the fact that out-of-control samples are also included in computing the control limits may also cause the control limits to be too wide at certain points that it masks actual anomalies. Moreover, the fact that few to no anomalies were generated in some periods (e.g. between period 1 to 2) of experiment 3 is interesting. It suggests that because a service has not yet exhibited anomalous behaviour does not mean there are no suspicious events at the system-level that may soon portend harm.

7 Related Work

In our previous survey in [1], we have carried out extensive review of of performance anomaly detection and bottleneck identification research in distributed and cloud environments. In this section, we describe some related works from the survey in addition to recent studies. Performance anomaly Detection (PAD) is concerned with identifying problem symptoms and anomalies in performance metrics of computer systems. Many recent work in this area generally focus on detecting problems in system-level metrics using predominantly statistical learning techniques. SEAD [35] adapts

Support Vector Machines (SVM) to achieve a self-evolving mechanism for detecting anomalies in system metrics. EbAT [36], analyses entropy time-series of metric distributions to automatically identify anomalies in datacenter servers. Dean et al [5] propose UBL, an unsupervised framework for identifying performance anomalies in systems metrics using Self-Organizing Maps (SOMs). A workload-aware technique for localizing anomalous requests and metrics in web applications using Local Outlier Factor (LOF) and the Student-t test is presented in [6]. The limitation of these works is that they ignore the cause-effect relationship and correlation between anomalies at the service level and system-level events such as capacity bottlenecks and faults.

Moreover, existing approaches for detecting anomalies in application performance metrics (e.g. response time and throughput) either resort to simplistic mechanisms based on thresholding or require intrusive application instrumentation (such as tracing request flows in [10] and [11]). Iqbal et al in [7] [37] employ hard SLA thresholds to detect anomalies in service response times. CloudPD [8] performs automated end-to-end problem detection, diagnosis and classification of faults in cloud environments. CloudPD detects anomalous response time and throughput measurements using percentage-based thresholds. E2EProf [10] analyzes causal paths of service requests to identify end-to-end requests delays and faulty components in distributed services. Spikes in requests delays are detected by point maxima (e.g. $3\sigma + \mu$) thresholds. Zhang et al [19] detects anomaly behaviour in jobs (applications) running in a Google web-search cluster based on a 2σ deviation in the job's cycle-per-instruction (CPI) metric distribution.

The behaviour-based anomaly detection method proposed in this paper is closely related to approaches proposed in [14] and [15] for intrusion detection in network traffic. The predictive anomaly detection scheme is similar to the work of Munz et al [30] and Ye et al [38] focusing on the use of statistical control charts in anomaly detection. Time series forecasting have also been used in dynamic resource provisioning in cloud environments such as in [39] and [40].

8 Conclusion

We have proposed and evaluated two schemes for detecting anomalies in real-time service performance metric measurements. In use cases where deviations from a known metric baseline is classified as anomalous, we have presented a technique that automatically detects deviations from the given baseline using non-parametric statistical learning. To address cases where a baseline is unknown, we have exploited the temporal dependency in service performance metric measurements to identify anomalies. Deviations in service performance behaviour are detected by applying adaptive control limits on residuals of Holt-Winter's forecasts using EWMA control charts. Our evaluation is based on tail service latency traces obtained from experiments in a real virtualized testbed under realistic load and fault injections. Results show the applicability of our approach for anomaly detection and also suggest that many service performance anomalies may be explained by system-level events such as resource contention and capacity bottlenecks. In the future, we plan to extend our approach for automatic root-cause analysis. By correlating system-level events with service-level anomaly detection, the resulting system may be able to diagnose observed anomalies, identify potential root-causes and remediations. Also, we plan to further evaluate accuracy of the techniques using labeled datasets from public system traces.

Acknowledgment

This work is supported by the Swedish Research Council (VR) under contract number C0590801 for the Cloud Control project, the Swedish Strategic Research Program eSSENCE, and the European

Unions Seventh Framework Programme under grant agreement 610711 (CACTOS).

References

[1] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth, "Performance Anomaly Detection and Bottleneck Identification," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 4:1–4:35, Jul. 2015. [Online]. Available: http://doi.acm.org/10.1145/2791120

[2] T. Kelly, "Detecting Performance Anomalies in Global Applications," in *WORLDS*, vol. 5, 2005, pp. 42–47.

[3] Q. Wang, Y. Kanemasa, J. Li, D. Jayasinghe, T. Shimizu, M. Matsubara, M. Kawaba, and C. Pu, "Detecting Transient Bottlenecks in n-tier Applications through Fine-grained Analysis," in *33rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2013, pp. 31–40.

[4] Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan, "PREPARE: Predictive Performance Anomaly Prevention for Virtualized Cloud Systems," in *32nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2012, pp. 285–294.

[5] D. J. Dean, H. Nguyen, and X. Gu, "UBL: Unsupervised Behavior Learning for Predicting Performance Anomalies in Virtualized Cloud Systems," in *Proceedings of the 9th international conference on Autonomic computing*. ACM, 2012, pp. 191–200.

[6] T. Wang, J. Wei, W. Zhang, H. Zhong, and T. Huang, "Workload-aware Anomaly Detection for Web Applications," *Journal of Systems and Software*, vol. 89, pp. 19–32, 2014.

[7] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Sla-driven Automatic Bottleneck Detection and Resolution for Read Intensive Multi-tier Applications Hosted on a Cloud," in *Advances in Grid and Pervasive Computing*. Springer, 2010, pp. 37–46.

[8] B. Sharma, P. Jayachandran, A. Verma, and C. R. Das, "CloudPD: Problem Determination and Diagnosis in Shared Dynamic Clouds," in *43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2013, pp. 1–12.

[9] D. J. Dean, H. Nguyen, P. Wang, and X. Gu, "PerfCompass: Toward Runtime Performance Anomaly Fault Localization for Infrastructure-as-a-Service Clouds," in *Proceedings of the 6th USENIX conference on Hot Topics in Cloud Computing*. USENIX Association, 2014, pp. 16–16.

[10] S. Agarwala, F. Alegre, K. Schwan, and J. Mehalingham, "E2EProf: Automated End-to-End Performance Management for Enterprise Systems," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2007, pp. 749–758.

[11] R. Filipe, S. Boychenko, and F. Araujo, "On Client-Side Bottleneck Identification in HTTP Servers," in *10th International Conference on Internet and Web Applications and Services*. IARIA, 2015, pp. 22–27.

[12] P. Carden. (2013) Network Design Manual: Network Baselining and Performance Management. [Online]. Available: http://www.networkcomputing.com/netdesign/base1.html

[13] N. Bhatia, "Proactive Detection of Performance Problems Using Adaptive Thresholds," 2010. [Online]. Available: https://neerajbhatia.files.wordpress.com/2010/10/Adaptive-thresholds.pdf

[14] Y. Gu, A. McCallum, and D. Towsley, "Detecting Anomalies in Network Traffic Using Maximum Entropy Estimation," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 2005, pp. 32–32.

[15] D.-Y. Yeung and C. Chow, "Parzen-window Network Intrusion Detectors," in *Proceedings of the 16th International Conference on Pattern Recognition*, vol. 4. IEEE, 2002, pp. 385–388.

[16] F. J. G. Gisbert, "Weighted Samples, Kernel Density Estimators and Convergence," *Empirical Economics*, vol. 28, no. 2, pp. 335–351, 2003.

[17] A. Moore, "A Tutorial on KD-Trees," http://www.cs.cmu.edu/*sim*awm/papers.html, 1991, Extract from PhD Thesis.

[18] D. C. Montgomery, *Introduction to Statistical Quality Control*. John Wiley & Sons, 2007.

[19] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "CPI2: CPU Performance Isolation for Shared Compute Clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 379–391.

[20] S. Malkowski, M. Hedwig, and C. Pu, "Experimental Evaluation of N-tier Systems: Observation and Analysis of Multi-bottlenecks," in *International Symposium on Workload Characterization, IISWC*. IEEE, 2009, pp. 118–127.

[21] C. C. Aggarwal, *Outlier Analysis*. Springer Science & Business Media, 2013.

[22] B. E. Hansen, "Lecture Notes on Nonparametrics," 2009. [Online]. Available: http://www.ssc.wisc.edu/~bhansen/718/NonParametrics1.pdf

[23] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*. CRC press, 1986, vol. 26.

[24] B. E. Hansen, "Bandwidth Selection for Nonparametric Distribution Estimation," *Manuscript, University of Wisconsin*, 2004.

[25] R. N. Riegel, "Generalized N-body problems: a Framework for Scalable Computation," Ph.D. dissertation, Georgia Institute of Technology, 2013.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[27] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*. OTexts, 2014, accessed: 2016-02-10.

[28] P. R. Winters, "Forecasting Sales by Exponentially Weighted Moving Averages," *Management Science*, vol. 6, no. 3, pp. 324–342, 1960.

[29] R. Lawton, "How Should Additive Holt–Winters Estimates be Corrected?" *International Journal of Forecasting*, vol. 14, no. 3, pp. 393–403, 1998.

[30] G. Munz and G. Carle, "Application of Forecasting Techniques and Control Charts for Traffic Anomaly Detection," in *Proceedings of the 19th ITC Specialist Seminar on Network Usage and Traffic*. Citeseer, 2008.

[31] NIST/SEMATECH, "e-Handbook of Statistical Methods," 2012, [Online; accessed 10-February-2016]. [Online]. Available: http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc431.htm

[32] B. Welford, "Note on a Method for Calculating Corrected Sums of Squares and Products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[33] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[34] D. Desmeurs, C. Klein, A. V. Papadopoulos, and J. Tordsson, "Event-Driven Application Brownout: Reconciling High Utilization and Low Tail Response Times," in *International Conference on Cloud and Autonomic Computing (ICCAC)*, 2015.

[35] H. S. Pannu, J. Liu, and S. Fu, "A Self-evolving Anomaly Detection Framework for Developing Highly Dependable Utility Clouds," in *Global Communications Conference (GLOBECOM)*. IEEE, 2012, pp. 1605–1610.

[36] C. Wang, V. Talwar, K. Schwan, and P. Ranganathan, "Online Detection of Utility Cloud Anomalies Using Metric Distributions," in *Network Operations and Management Symposium (NOMS), 2010 IEEE*. IEEE, 2010, pp. 96–103.

[37] W. Iqbal, M. Dailey, and D. Carrera, "SLA-driven Adaptive Resource Management for Web Applications on a Heterogeneous Compute Cloud," in *Cloud Computing*. Springer, 2009, pp. 243–253.

[38] N. Ye, S. Vilbert, and Q. Chen, "Computer Intrusion Detection through EWMA for Autocorrelated and Uncorrelated Data," *IEEE Transactions on Reliability*, vol. 52, no. 1, pp. 75–82, 2003.

[39] H. Engelbrecht and M. van Greunen, "Forecasting Methods for Cloud Hosted Resources, a Comparison," in *11th International Conference on Network and Service Management (CNSM)*. IEEE, 2015, pp. 29–35.

[40] C. Vazquez, R. Krishnan, and E. John, "Time Series Forecasting of Cloud Data Center Workloads for Dynamic Resource Provisioning," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, vol. 6, no. 3, pp. 87–110, 2015.