A Black-box Approach for Detecting Systems Anomalies in Virtualized Environments

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

Ewnetu Bayuh Lakew
Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden
Email: ewnetu@cs.umu.se

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

***Abstract*—Virtualization technologies allows cloud providers to optimize server utilization and cost by co-locating services in as few servers as possible. Studies have shown how applications in multi-tenant environments are susceptible to systems anomalies such as abnormal resource usage due to performance interference. Effective detection of such anomalies requires techniques that can adapt autonomously with dynamic service workloads, require limited instrumentation to cope with diverse applications services, and infer relationship between anomalies non-intrusively to avoid 'alarm fatigue' due to scale. We propose a black-box framework that includes an unsupervised prediction-based mechanism for automated anomaly detection in multi-dimensional resource behaviour of datacenter nodes and a graph-theoretic technique for ranking anomalous nodes across the datacenter. The proposed framework is evaluated using resource traces of over 100 virtual machines obtained from a production cluster as well as traces obtained from an experimental testbed under realistic service composition. The technique achieve average normalized root mean squared forecast error and R^2 of (0.92, 0.07) across hosts servers and (0.70, 0.39) across virtual machines. Also, the average detection rate is 88% while explaining 62% of SLA violations with an average lead-time of 6 time-points when the testbed is actively perturbed under three contention scenarios.**

I. Introduction

The regular occurrence of service-level degradation and occasional outages in large-scale services in cloud datacenters have great impact on the reliability of services and quality of service (QoS). Hence, according to a Compuware survey [1], 60% of CTOs across 400 IT organizations reported that their biggest concerns is alleviating poor end-user experience due to performance problems in their cloud infrastructures. In fact, service anomalies in the form of sluggish response was reported to account for 60% of the impact of outages in 2013 [2].

Service performance degradation are often accompanied by anomalous indicators at the systems level that point to issues internal to the service itself or in the underlying infrastructure. A typical example of such anomalous indicators is abnormal resource consumption suggesting concrete issues such as DDoS attacks, sporadic changes due to DevOps [3], performance interference, and spikes in service workloads. Performance interference is a major consequent of consolidating application services in the smallest set of servers to reduce operational cost (e.g. energy) and maximize resource utilization [4]. The side-effect of this is the unhealthy contention for limited systems resources (e.g. compute, bandwidth or memory) among co-located applications due to resource overbooking thereby resulting in costly violation of service-level agreements (SLA) for both cloud and application providers [5].

Consequently, the continuous detection of anomalies such as abnormal resource behaviour has become a major objective in cloud monitoring in order to prevent service-level degradation and improve reliability by quick containment of unexpected outages. However, the ever-increasing scale, the architectural complexity of services and infrastructures, coupled with diversity and dynamism of service workloads have made the realization of online anomaly detection a challenging task. The new requirement is that anomaly detection methods should adapt autonomously to changing behaviour without explicit definition of normality or abnormality. In addition, they are also expected to be sufficiently general to cope with multiple levels of systems abstractions and plurality of services and metrics.

While research surveys such as [6], [7] demonstrate the popularity of anomaly detection especially in systems domains, most existing solutions are not fully tailored to meet the above requirements. The classical approach predefines thresholds based on assumed distribution [8], [9] that are sensitive to workload variations and become unwieldy scaling to hundreds of metrics. More sophisticated techniques in [10]–[12] consider individual resource metrics separately thereby ignoring contemporaneous dependencies across metrics. Techniques in [13], [14] are based on modeling the relationship between resource utilization and workload, while [12], [15] rely on the correlation between QoS and resource metrics. Multivariate statistical learning have also been employed to exploit associations between multiple metrics in [16], [17]. The limitations of these proposals is that while the machine learning-based techniques ignore potential auto-correlation (e.g. seasonal or cyclical effect [18]) and cross-correlation (e.g. CPU-IO resource dependency [5]) in systems metrics, the correlation-based approaches are not applicable when QoS and workload metrics are not readily available.

Motivated by the above limitations, we propose a black-

box framework that includes an unsupervised prediction-based mechanism for automated anomaly detection in multi-dimensional resource behaviour of datacenter nodes (e.g. VMs, containers, servers, etc.) as well as a graph-theoretic technique for ranking anomalous nodes across the datacenter. Anomalies are detected by uncovering abnormal changes in the temporal resource consumption pattern of datacenter nodes achieved via a block-based Vector Autoregressive (VAR) model. The ranking is incorporated to deal with large number of simultaneous (perhaps cascade) anomalies by using knowledge of complex dependencies among datacenter components to reduce *alarm fatigue* resulting from and guide anomaly mitigation procedures [19].

The proposed framework is evaluated using resource traces of over 100 virtual machines obtained from a production cluster as well as traces obtained from an experimental testbed under realistic service composition. The technique achieve average normalized root mean squared forecast error and R^2 of (0.92, 0.07) across hosts servers and (0.70, 0.39) across virtual machines. Also, the average detection rate is 88% while explaining 62% of SLA violations with an average lead-time of 6 time-points when the testbed is actively perturbed under three contention scenarios.

The rest of the paper is organized as follows; we discuss problem background and overview of proposed approach in Section II. Algorithms are discussed in detail in Section III and Section IV. Design of experimental evaluation is presented in Section V, and in Section VI we demonstrate the efficacy of the proposed approach by answering a number of important research questions. We discuss relevant research contributions in Section VII and conclude in Section VIII.

II. Background and Overview

In Fig. 1, we demonstrate a small segment of a typical virtualized cloud datacenter. It shows composition of three service workloads–a media streaming service, an e-commerce website, and an analytics back-end–and the mapping between service components, deployed in virtual machines (VM) at the virtual layer, to nodes in the physical layer, that is physical machines (PM) and network devices, representing intra- and inter-layer dependencies in the datacenter. Each VM typically encapsulate a set of system resources (e.g. hardware cache, CPU, Memory, Disk I/O, and Network bandwidth) that rely on shared resources in physical components through virtualization technology.

Each tenant typically express their QoS in one or combination of performance metrics such as request or query *latency*, *frame/sec*, *transaction/sec*, or *availability*, etc. Temporary or prolong changes in these QoS metric are perceived by end-users as performance degradation and may result in violation of SLA.

The ensuing performance interference in this scenario can induce abnormal resource usage in different ways. There are cases where a service is not able to fully utilize provisioned resources due to multiplexing of a PM's resources among a set of VMs as total demand becomes greater than available

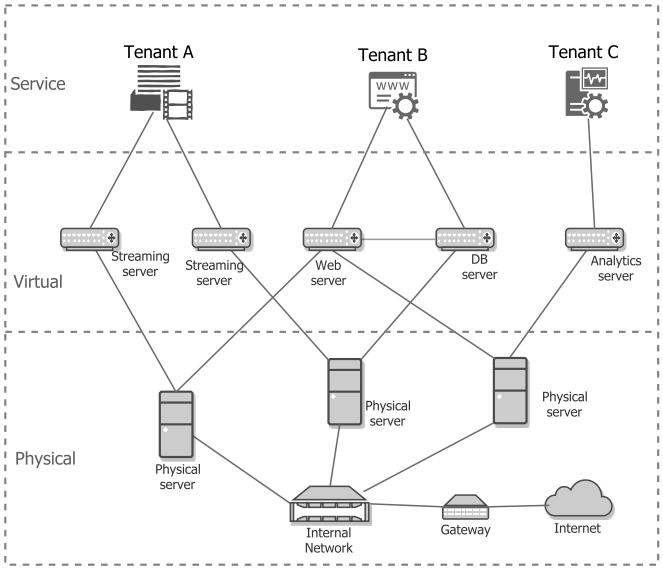


Fig. 1. A typical virtualized environment

capacity. Colocated VMs may also thrash the hardware cache at peak times. Similarly, a combination of resources (e.g. CPU and disk I/O) may exhibit interdependence due to the nature of workloads e.g., serving a request may require fetching disk-bound pages while the CPU waits. Events such as periodic IO background operations (e.g. backups or rolling updates) on a PM can also hurt hosted services if they coincides with VM's IO workloads.

Besides performance degradation, these may affect customers' perception of the cloud's ability to meet performance guarantees thereby limiting performance-critical applications. For example, Fig. 2 shows the impact of resource contention on QoS of an interactive web service in a testbed similar to Fig. 1. By examining resource utilization metrics of the concerned VM, we attribute the level shift between the 36th and 42nd minute to CPU contention from another co-located VM while that beginning from the 47th minute is due to background I/O noise in the physical server hosting the database component of the service. The tail latency suggests that performance has degraded from around 150ms to over 400ms for at least 5% of all end-users, which may worsen at peak times. For a cloud service with 300K hourly users, this implies about 15K may be dissatisfied during an hour-long contention-induced degradation.

Therefore, cloud operators invest great effort to monitor resources metrics and apply various techniques to uncover abnormal consumption patterns. However, the inherent complexity of virtualized datacenters impose new requirements on anomaly detection techniques that (a) are non-intrusive (i.e. *black-box*) and having no need for application instrumentation since providers are typically oblivious of tenants source codes and workloads (b) function in an unsupervised manner, being able to handle unlabeled data that have not being categorized as *normal* or *anomaly* a priori (c) are sufficiently online that

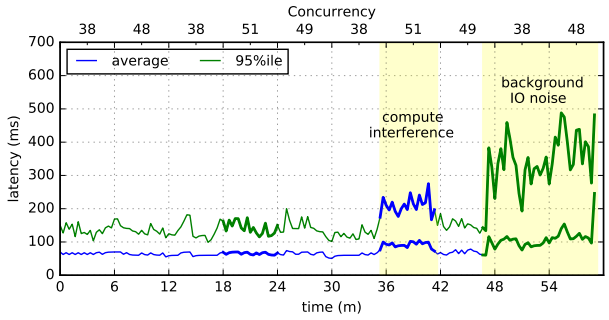


Fig. 2. Impact of resource contention on average and tail latency.

autonomously adapt to changing systems behaviour without need for extensive off-line training and human intervention (d) operate independently of application QoS since developers cannot be so trusted especially if they feel it is a burden sharing such information.

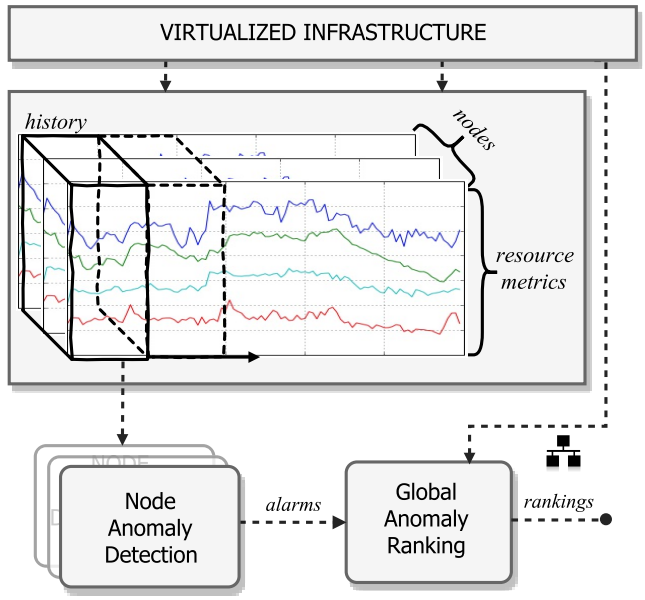


Fig. 3. System overview

A. System Overview

Based on these requirements, we present in Figure 3 an overview of the propose approach.

The system rely on real-time utilization measurement of compute, memory, disk IO and network resources collected by distributed and light-weight monitoring agents deployed in each node (VMs and PMs) of the virtual infrastructure and sampled at fixed time interval (e.g. 5min).

The *Node Anomaly Detection (NAD)* module is deployed as an independent agent in individual node with the aim of adaptively detecting time-points where the node experiences abnormal resource usage either temporarily or for an extended period of time. To achieve this aim, NAD works in three steps. First it estimates a continuous temporal profile of *normal* behaviour from historical data using an adaptive block-based Vector Autoregressive (VAR) model [20] that exploits contemporaneous associations between the metrics. The profile estimate is used as *baseline* to predict expected behaviour in the immediate future. The statistical distance of the forecast residual of the expectation from that of the baseline in multi-dimensional space is estimated using Robust Mahalanobis Distance [21] and serves as the basis for identifying anomalous data points. NAD raises an alarm when an anomaly is discovered.

The *Global Anomaly Ranking (GAR)* module is a separate module with cluster-wide observability. Its function is to coalesce all alarms from distributed nodes and at regular time point rank anomaly alarms or nodes based on spatial dependencies across datacenter components to produce an associative mapping of how observed anomalies relate or how anomaly in one node may potentially induce other anomalies observed within same period of time. The aim of this is to facilitate root-cause analysis and anomaly mitigation procedures especially for large number of alarms. We assume that the datacenter topology is accessible and continuously updated after reconfigurations using tools such as Dynatrace[1] or Apex Lake [22].

We outline in detail both NAD and GAR modules in the following Section III and IV.

III. Node Anomaly Detection

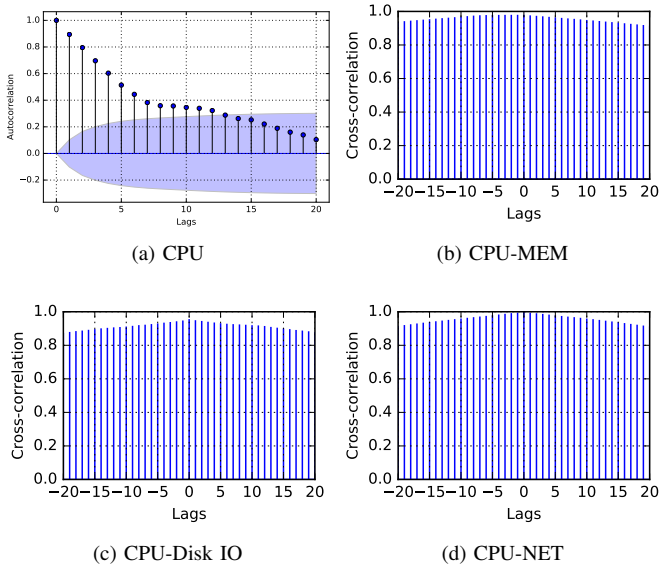
The goal is to detect time-points where abrupt or gradual deviation from historical patterns in resource consumption is observed. First, we hypothesize that the aggregate resource behaviour of a node is a function of the historical consumption pattern of individual resource (e.g. CPU utilization) and contemporaneous association between other resources (e.g. CPU vs NET utilization). To demonstrate that is the case, we clarify using temporal correlation plots of resource metrics for a web service in Fig. 4. Figure 4a reveals that future CPU utilization values are likely dependent on the past (up to 12 lags) while the high positive correlation in figures 4b, 4c, and 4d suggests a strong association between historical utilization values of CPU and other system resources. This insight suggests that we might be able to predict future utilization values of each resource as a linear combination of on its lagged values (recent past) and lagged value of utilization values of other resources. This insight is often ignored in existing systems thereby limiting applicability under complex heterogeneous workloads.

In this section, we acknowledge and exploit this temporal and spatial property for prediction-based anomaly detection in multivariate resource utilization metrics.

A. Resource Profile Estimation

We propose a block-based adaptive Vector Autoregressive (VAR) to characterize the normal resource consumption pattern of datacenter nodes from monitoring data based on

[1]https://www.dynatrace.com



(a) CPU (b) CPU-MEM

(c) CPU-Disk IO (d) CPU-NET

Fig. 4. Auto-correlation (temporal dependency) and cross-correlation (spatio-temporal dependency) in resource consumption behaviour of a web application.

sequential sliding windows. The temporal profile is then used to estimate expected resource consumption pattern of a given node in the immediate future. A significant deviation of the expectation from the true behaviour is a sign of abnormal resource usage pattern. The VAR model generalizes the univariate autoregressive model for characterizing the evolution of and interdependencies between a set of time-series over a given sample period based on the assumption that the variables influence each other equally [23]. We chose a VAR-based model over alternatives such as Artificial Neural Networks (ANN) and ARIMA models since it does not require the extensive trial-and-error training required by the former to determine optimal network structures and complex transformation required to handle multivariate time-series as well as the offline parameterization procedure of the later.

Let $X_t = (x_{1t}, x_{2t}, ..., x_{nt})$ be a $(1 \times n)$ vector of measurement produced by a node's monitoring agent at time t where x_{mt} refer to the value of metric m at time t and n is the number of monitored metrics. Though $n = 4$ in our case, utilization metrics of 4 system resources, our system is not limited to this since our model can incorporate more metrics. The vector X_t can be expressed as a linear combination of its historical observations using a VAR(p) model of the form [23]:

$$X_t = \theta_0 + \sum_{i=1}^{p} \theta_i X_{t-i} + \epsilon_t \tag{1}$$

where p is the order of the model or number of lags, defining how far in past to consider. Hence, X_{t-j} is the jth-lag of X. Note that Eq. (1) comprises n equations, one per monitored metric, representing a linear function of its own lagged values as well as the lagged values of other metrics. Parameter θ_0 is a $(n \times 1)$ vector of constants and θ_i, $(i > 0)$ are $(n \times n)$ coefficient matrices and ϵ_t is a $(n \times 1)$ vector of white noise.

To fit the VAR(p) in Eq. (1), we need to determine how many lags (p) to include in the model in a data-driven way. Determining the right value of p is essential due to its impact on model complexity and how errors enter into model from one prediction to another. We also want to do this in an online manner since the lag value may change overtime. Hence, we build a baseline training model by segmenting historical measurement of resource utilizations using a sequence of temporal k-length windows that advances one step at a time. Hence, at time t, there exist a window w containing the most recent k observations $X^w = \{X_{t-k}, X_{t-k+1}, ..., X_{t-1}\}$ over interval $[(t-k), (t-1)]$. Note that it is assumed that X^w is sufficiently free of anomalies and represents a model of most recent normal behaviour.

To prevent the baseline model from becoming unwieldy, we constrain the space of exploration by defining a maximal lag $p_{\max}$ so that we exhaustively search for p in the range $[0, p_{\max}]$ that minimizes an Information Criterion (IC). This is done using the principle of least-square where parameters θ_0 and θ_i $(1 + pn$ coefficients in total) are also estimated to minimize forecast errors ϵ_t. The IC is typically selected from one of Akaike (AIC), Schwarz-Bayesian (BIC), Hannan-Quinn [20]:

$$\text{AIC}(p) = ln|\tilde{\Sigma}(p)| + \frac{2}{T}pn^2$$
$$\text{BIC}(p) = ln|\tilde{\Sigma}(p)| + \frac{lnT}{T}pn^2$$
$$\text{HIC}(p) = ln|\tilde{\Sigma}(p)| + \frac{2lnlnT}{T}pn^2$$

where n remains number of metrics, $T = |X|$ is the size of training set, and $\tilde{\Sigma}(p) = T^{-1}\sum_{t=1}^{T} \hat{\epsilon}_t \hat{\epsilon}_t{}'$ is the estimated covariance matrix of residuals. We chose the HIC(p) over the AIC and BIC is based on its ability to select intermediate models that balances the tradeoff between high-order models of the AIC and low-order models of the BIC. It is also known to consistent when the true p lies within $[1, p_{max}]$. The optimal p given the training set X is determined as

$$p^* = \underset{p}{\operatorname{argmin}}\, \text{HIC}(p) \tag{2}$$

Associated with the selected p^* are parameters $\hat{\theta}_0$ and $\hat{\theta}_i$, $1 \leq i \leq p^*$ as well as $\hat{E} = (\hat{\epsilon}_1, ..., \hat{\epsilon}_T)$ the set of error terms when the model is used to evaluate the training set. $\hat{E}$ typically satisfies that errors are serially uncorrelated and are multivariate normal with zero mean and full rank covariance matrix $\tilde{\Sigma}$. Finally, we define $\mathcal{P}^w$ as the profile of normal behaviour based on the most recent observations is defined as a tuple $\mathcal{P}^w = (X^w, p^*, \{\hat{\theta}_i\}_{i=0}^{p^*}, \hat{E})$.

B. Detecting Abnormal Resource Consumption

At each time point, we exploit the baseline profile $\mathcal{P}_w$ for detecting abnormal time-points. First we derive a forecast of

expected resource consumption in the next time step (i.e. one-step ahead) conditioned on $\mathcal{P}_w$ as following:

$$\hat{X}_{t+1} = \hat{\theta}_0 + \sum_{i=1}^{p^*} \hat{\theta}_i X_{t-i} \tag{3}$$

Next we compute the forecast error as the difference between true observation (X_{t+1}) and the forecast, $\hat{\epsilon}_{t+1} = \hat{X}_{t+1} - X_{t+1}$, where the error is of the form: $\hat{\epsilon}_{t+1} = (\hat{\epsilon}_{1,t+1}, \hat{\epsilon}_{2,t+1}, ..., \hat{\epsilon}_{n,t+1})$

The problem of detecting abnormal resource consumption can then be cast as multivariate proximity-based outlier detection based on the statistical distance of the forecast error, $\hat{\epsilon}_{t+1}$, to the centroid of the training residuals, $\hat{E}$, in multidimensional space using robust Mahalanobis Distance (*m*-distance) [24] as follows:

$$d_{t+1} = \sqrt{(\hat{\epsilon}_{t+1} - \mu_{\hat{E}})' S_{\hat{E}}^{-1} (\hat{\epsilon}_{t+1} - \mu_{\hat{E}})} \tag{4}$$

where $\mu_{\hat{E}}$ and $S_{\hat{E}}$ are the mean vector and covariance matrix derived from $\hat{E}$ respectively. To make distance computation more robust to outliers, we used Minimum Covariance Determinant (MCD) [21] to estimate the covariance matrix $S_{\hat{E}}$ instead of the maximum likelihood estimate. Also, since monitored metrics different with varying magnitude, we scale both $\hat{E}$ and $\hat{\epsilon}_{t+1}$ to the same units using *min-max* normalization.

The quantity d_{t+1} can be interpreted as the 'rarity' of $\hat{\epsilon}_{t+1}$ relative to $\hat{E}$. Let $D_{\hat{E}}$ denote the set of distances of training residuals ($\hat{E}$), then unusually values such that

$$d_{t+1} > (\mu_{D_{\hat{E}}} + L\sigma_{D_{\hat{E}}}) \tag{5}$$

indicate significant deviation from the expected behaviour and so X_{t+1} is considered an anomaly. Parameter L is set to the 99th percentile z-score of the distance distribution of training residuals ($\hat{E}$).

C. Learning Adaptation

A set of VAR(p) models are built using a sequence of k-length sliding windows each containing the most recent measurements to estimate time-varying parameters such as the optimal lag p^* and other coefficients (Section III-A). Then, the model is used to predict one step ahead and on the basis of the residual error, the true observation is classified normal or abnormal (Section III-B). Since the objective is to correctly learn normal behaviour sequentially, we persist temporal continuity by replacing anomalous samples with forecasts at respective time points. Also, the residual at such points are the expected value of training residuals. The sliding window is shifted one step and we repeat the whole process in an interleaved train-test cycle (rolling forecast and detection). This strategy offer two technical benefits (a) it facilitates online learning from only a small set of recent measurements rather than the entire measurements, and (b) the short forecast horizon enables the model to better track trends and small shifts in the data, which is useful to quickly detect onset of performance degradation.

The size of the sliding window, k, is important for the reliability of our block-based technique. Larger windows may miss transient changes in the data due to increased temporal smoothing of model coefficients while smaller windows slows down smoothing rate but may miss interesting dynamics in the data if temporal dependency among variables do not occur within the short window. Hence, k should be large enough to capture interesting temporal changes as well as dynamic dependencies in the data. Generally, a total of n^2p coefficients are to be estimated for a VAR(p) model with p lags and n variables [23], it is therefore recommended to let $k >> n^2p$, to ensure sufficient data points are available to correctly fit the model.

Finally, since each distributed NAD agent build models for only one node, the runtime requirement is reasonable for online deployments. Also, the procedure can easily be parallelized to eliminate computational bottlenecks. In Section VI we demonstrate the impact of window size on forecast performance, achievable lead-time, and runtime requirement per train-test cycle.

Our implementation of the block-based adaptive VAR model rely on the `statsmodels`[2] library, while the distance computation is based on modules from `scikit-learn`[3].

D. Baseline Multivariate Anomaly Detection

In order to compare our VAR-based technique we consider an alternative approach for addressing similar problem is the Multivariate Exponentially Weighted Moving Average (MEWMA) control chart [25]. The main properties of the MEWMA chart include its ability to easily incorporate historical measurements and correlation between variables as well as its relative sensitivity to small variations in the data. The MEWMA is based on its on the simple exponential smoothing involving a single variable and defined as:

$$Z_t = \Lambda X_t + (I - \Lambda) Z_{t-1} \tag{6}$$

where X_t is the vector of measurements at time t, Z_t is the smoothed and Z_0 is initialized the mean vector from historical samples and $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_n)$ is a diagonal matrix and n is the number of variables ($n = 4$ in our case). A MEWMA control chart plots the quantity $T_t^2 = Z_t \Sigma_{Z_t}^{-1} Z_t$, where Σ_{Z_t} is the covariance matrix of Z_t. Time points and observations where $T_t^2 > H$ are flagged as anomalous. The value of control limit H is determined based on theoretical limits [25] with respect to the diagonal value of Λ to achieve a desired level of in-control average run length (ARL)–the average number of measurements before another anomaly is observed. We use a constant 0.5 main diagnonals of Λ in order to balance the influence of past and future behaviour on the model.

We implemented a block-based MEWMA using sequential sliding windows so that it adapts with the data similar to our proposed technique. In Section VI, we investigate the detection accuracy of MEMWA relative to our method.

[2]http://statsmodels.sourceforge.net

[3]http://scikit-learn.org

IV. Global Anomaly Ranking

Cascading behaviour is a common occurrence in virtualized datacenters due to inherent spatial dependency between nodes [17], [26]. For example, abnormal resource usage in one node can easily propagate to other nodes. Moreover, considering the scale of virtualized datacenters, administrators or automated ticket management systems can easily be overwhelmed with multiple alarms since they have to relate alarms or problem tickeks within the same period based domain knowledge of the inter- and intra-dependencies in the datacenter to isolate root-causes. In addition, to mitigate service-level degradation, automated cloud management systems may take corrective actions such as adjusting resource capacity (via autoscaling [27]) or redistributing load across the datacenter (via migration [4]). However, such actions may fail to yield desired effect when there are too many alarms and no means to tell which alarm is more important.

Therefore, there is need for automated ranking of anomaly alarms at the cluster level in order to facilitate root-cause analysis and maximize impact of corrective actions since addressing an important anomaly may resolve other co-occurring alarms. In this section, we employ knowledge of datacenter topology in the form of landscape graphs [22] to rank alarms based on the causal relationship between nodes. The result is an associative mappings describing how one anomalous node relate with other co-occurring anomalies over a given period of time.

A *landscape* is a directed acyclic graph $G = (V, E)$ where V is the set of nodes representing a unique datacenter component (e.g. service, VM, network function, or physical server) and edges E represent dynamic dependencies among nodes. Using Fig. 5 nodes of a landscape are aggregated into three layers namely, `Service` (gray nodes), `Virtual` (green nodes) and `Physical` (yellow nodes), while an edge, $e \in E$, signifies either an inter- or intra-layer dependency. For example, in Fig. 5, a directed edge from F to J may indicate a VM or container running on a physical server. Also, a directed link from D to E typifies a web serving VM depending on a database server. Associated with each graph node $v \in V$ is an attribute indicating if the latest resource consumption pattern of v is anomalous or not Section III.

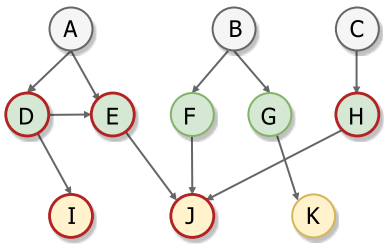


Fig. 5. Sample landscape graphs

The GAR module of Fig 3 receives anomaly alarms continuously from all NAD nodes and coalesce over a short interval for ranking purposes. Given a list of anomalous node V^*, we first extract an anomaly subgraph $g^* \in G$ for each $v^* \in V^*$ via Depth First Search (DFS) traversal of G and constructing a path containing only anomalous virtual and any physical nodes reachable from v^*. An anomaly subgraph is therefore a potential propagation path of anomalies from or to v^*. Let C^* be the set of unique nodes from all subgraphs, we define an influence count, $\mathcal{I}^{c^*}$ for each $c^* \in C^*$ as the number of times c^* appears in anomaly subgraphs other than its own. The set C^* is then ordered according to $\mathcal{I}^{c^*}$ so that nodes with high influence values impact or impacted by more anomalies compared to the rest.

We handle ties across layers by prioritizing nodes in the physical layer since most resource related issues in VMs can be eliminated by first resolving contentions in the PMs. Also, ties in the physical layer are resolved by physical nodes that are themselves classified anomalies or placed in any order otherwise. The output of this procedure is a ranked list of tuples (c^*, Φ) where Φ is the list of anomalous nodes that are potentially impacted by issues in node c^* or any other node dependent on it. This list can be displayed to an administrator to enhance root-cause analysis, remediation procedure or used by an automated cloud manager to optimize re-balancing operations.

Applying this algorithm on the graph in Fig. 5 (anomalous nodes in red) produces the associative mappings: $\{D \rightarrow E, I, J\}$, $\{E \rightarrow J\}$, $\{I \rightarrow \emptyset\}$, $\{J \rightarrow \emptyset\}$, and $\{H \rightarrow J\}$. This results in ranking order (J, I, E, D, H). Note that D and H can be in any order. This ranking suggests that eliminating the contention issues in node J may likely eliminate those of D, E and H respectively and likely resolve degradations in associated services A and C.

V. Evaluation

In this section we evaluate the efficacy of our framework for (a) forecasting multivariate resource metrics (b) detecting abnormal resource consumption behaviour and (c) relevance of detected anomalies to SLA violations at the service level. This is achieved using data obtained from an experimental testbed with the proposed framework deployed as well as public traces from a real production cluster. The rest of this section describes the experimental testbed, the public traces as well as performance metrics used in our evaluation.

A. Experimental Testbed

Fig. 6 is a schematic of the testbed on which all experiments in this section are conducted. The testbed is composed of two Xen hypervisor based HP ProLiant physical machines (PM) and equipped with eight 2.66GHz Intel Xeon processors, 16GB of memory and 250GB disk per server.

1) Service Composition: To emulate realistic composition of services typical of virtualized environments, we deploy two real target service and 3 other tenant services and utilities for generating load on the testbed

- *RUBiS*[4]: a 2-tier eBay-like e-commerce application that provides selling, browsing and bidding functionalities.

[4]http://rubis.ow2.org/index.html

The service (green stack) is composed of two VMs hosting an Apache 2.0 web server (*VM3*) and MySQL 5.0 server (*VM4*) respectively.

- *RUBBoS*[5]: a Slashdot-like bulleting board web application running on two distributed VMs (red stack), an Apache 2.0 web server (*VM5*) and MySQL 5.0 server (*VM6*)
- *Load applications*: We dynamically deploy one of three application utilities in VMs of the blue stack (*VM1* and *VM2*) to generate background load on the testbed to mimic real environments where diverse applications are co-located: a) *Sudoku*, an application that solves a series of Sudoku puzzles in parallel[6], and b) *Stress-ng*, a load generator to stress system resources[7].

2) Workload Generation: To generate realistic workload for both RUBiS and RUBBoS (Fig. 6), we emulate virtual web users concurrently interacting with the applications using an open source HTTP load generator, `httpmon`[8], which supports both closed and open system model. Based on workload profiles, the tool dynamically generates realistic application behaviour by issuing HTTP `GET` or `POST` requests at varying intensities with exponential inter-arrival times.

3) Data Collection: We measure the utilization of major systems resources at regular interval (e.g. 1min) based on open-source Unix-based monitoring tools namely, `dstat`[9] and `psutil`[10].The NAD module (Fig. 3) is also deployed in each node for anomaly detection. We synchronize the clock of VMs relative with PM's to reduce dis-alignment effect.

Network utilization is derived from the traffic intensity in and out of each node. Let R and S be counters of accumulated traffic received and transferred for a given node. First we normalize by taking difference of consecutive measurements and dividing by the interval, e.g., normalized incoming traffic at time t is derived as $(R_t - R_{t-1})/(t-(t-1))$. Then for a full duplex link, the network utilization for the node is $max(R,S)/L$ where L is the link speed.

4) Model Adaptation: All experiments discussed later in Section VI are staged for 3 hours with sampling rate of 30 seconds making a total of 360 observations per experiment per node of the testbed. Adaptation is based on the interleaved train-test strategy using the most recent 1.5 hours measurements (180 samples) as baseline and the rolling forecast is applied to the remaining time, up to 1.5hrs. Recall that the training size is determined so that $k >> n^2p$ as mentioned in Section III-C and this holds for $k = 180$, $n = 4$ and $p_{max} = 10$.

5) Load Emulation: We emulate realistic contention scenarios typical of virtualized environment via active perturbation by dynamically deploying one or load tools in the load VMs to enact desired execution context. Though this approach is similar to the concept of fault injections in reliability studies where a target system is directly perturbed, instead we achieve this by indirectly inducing abnormal resource behaviour (and degradation) in target services through excessive resource consumption from other tenants. The Load Emulator module (Fig. 6) was used to automatically stage many different experiments but we present the following three contention scenarios in this section and discuss the results in Section VI.

a) *Compute Contention*: CPU resource contention was injected by automatically running a number of parallel *Sudoku* processes solving a random set of puzzles in *VM2* over a given period of time. The number and the duration for each solvers are poisson distributed with a given mean. This is repeated a specific number of runs based on exponential inter-run times with a specific average.

b) *Network Congestion*: oversubscribed network bandwidth is emulated by setting up a serial TCP communication between a single *iperf* server on *VM2* and a set of iperf clients on *VM1* with a random waiting time between each serial connection. The duration of each run is exponentially distributed with a specific average.

c) *Disk IO Saturation*: IO contention was injected by intermittently spawning the *stress-ng* utility on *VM2* to simulate a background backup operation thereby inducing excessive disk IO utilization. This involves a worker process writing about 50MB data to disk on average while another performs linear-search over 32-bit integers for a given period of time. The inter-run times is exponentially distributed with a certain mean value.

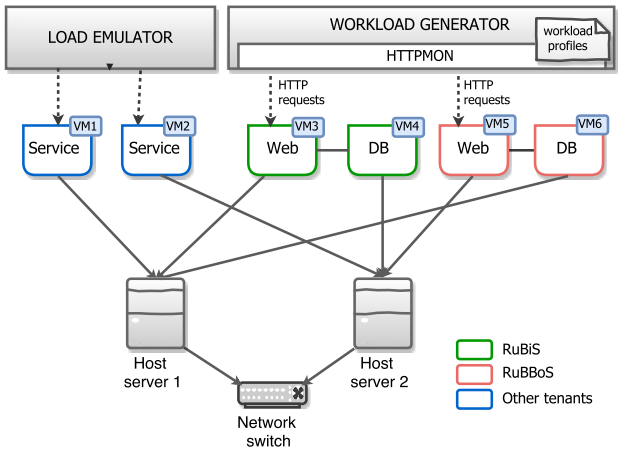
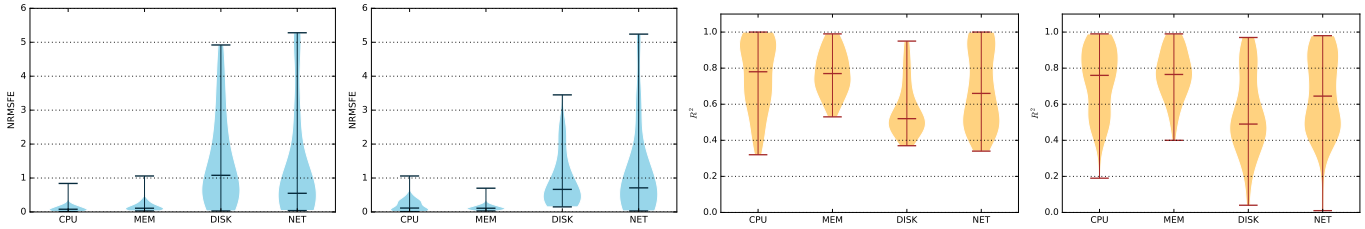


Fig. 6. Experimental Testbed

B. Real world Datacenter Traces

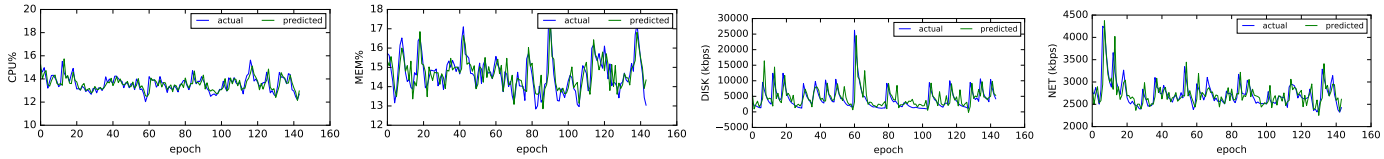
To gauge forecast accuracy and applicability of proposed method for handling scale-out workloads, we evaluate on resource consumption traces of over 100 virtual machines from the Materna dataset [28]. The traces were collected from the distributed Materna's datacenter hosting a variety of highly

[5]http://jmob.ow2.org/rubbos.html
[6]http://norvig.com/sudoku.html
[7]http://kernel.ubuntu.com/ cking/stress-ng/
[8]https://github.com/cloud-control/httpmon
[9]https://github.com/dagwieers/dstat
[10]https://pythonhosted.org/psutil/



(a) *NRMSFE* (low order lags) (b) *NRMSFE* (high order lags) (c) R^2 (low order lags) (d) R^2 (high order lags)

Fig. 7. Forecast accuracy across resource metrics obtained from resource traces of over 100 VM from the Materna datasets. Low order lags: average number of lags selected $p^* \leq 4$. High order lags: average number of lags selected $p^* > 4$. The median of each bar is indicated with the horizontal line in the middle.



(a) CPU Utilization (b) MEM Utilization (c) DISK Throughput (d) NET Throughput

Fig. 8. Plots showing evolution of actual observations and predictions for each system resource utilization

critical business applications (e.g. SAP, government, IT, etc.) over a 1-month period. Each VM is associated with a trace file containing mainly a column each for utilization readings of CPU, Memory, Disk IO, Network resources, sampled at 5-minute interval.

C. Evaluation Metrics

The following are the different performance metrics used in our evaluation.

1) Forecast accuracy: to quantify and compare the forecast accuracy of our technique across different attributes and datasets, we use two metrics namely; R^2 and normalized root mean squared forecast errors (*NRMSFE*). The R^2 explains the amount of variability in each attribute captured by the multivariate model;

$$R^2 = 1 - \left(\frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \right) \tag{7}$$

while *NRMSFE* describes the standard deviation (spread) of forecast residuals normalized by the mean and is defined per individual resource metric;

$$NRSMFE = \frac{\sqrt{\frac{1}{T}\sum_i (y_i - \hat{y}_i)^2}}{\bar{y}} \tag{8}$$

Variables y_i, $\hat{y}_i$, $\bar{y}$ are the observed value, prediction and mean value of associated resource metric. We also report means of these metrics aggregated across metrics of the multivariate model.

2) Detection accuracy: though the method is unsupervised in nature, we define a custom technique for describing the ability of correctly identifying anomalous resource behaviour. Since the load emulator in Fig. 6 tracks period where contentions were induced in the testbed, we extract indices of the known fault periods. A *HIT* occurs when the index of a system alarm falls within the known fault period (True Positive) while a *MISS* occurs if a system alarm index falls outside of the known fault period (False Positive).

3) Relevancy: this describes how relevant resource misbehaviour alarms at the system level explains service-level SLA violations reported in absolute counts of matches.

4) Leadtime: this tells how early performance degradation are detected at the system level in terms of the latency between the first resource misbehaviour alarm at the system level and the first SLA violation alarm. Administrators can take advantage of this quantity to mitigate or minimize impact of degradations before it manifests to end-users.

VI. Experimental Results

In this section, we show the efficacy of the proposed framework by performing a series of experiments on the testbed introduced previously in Section V-A. We follow evaluation methodologies described in Section V to address the following research questions:

A. ***RQ1***: *How well does the online method predict multidimensional resource behaviour of nodes?*

The objective is to quantify the forecast accuracy of proposed framework on traces from the real world as well as empirical traces obtained from running realistic workloads on experimental testbed shown in Fig 6.

Firstly, we randomly selected traces of over 100 VMs from the Materna datasets. Then we apply our technique on one week worth of measurements (2016 observations) using the first three days (864 observations) as the baseline and performing rolling forecasts (one-step shifts) for the next day (288 samples). Fig. 7 presents vertical kernel density plots showing the distribution of *NRMSFE* and R^2 metrics for individual resource metrics. Furthermore, we delineate the results into two categories based on the average optimal p^* (lags) selected by the model for rolling predictions namely, *low* order lags–average lags of 4 or below, *high* order lags–average

TABLE I
FORECAST ACCURACY OF PROPOSED FRAMEWORK ON EMPIRICAL DATA AGGREGATED ACROSS HOSTS AND TARGET SERVICES

	CPU		MEM		DISK		NET		MEAN	
	R^2	**NRMSFE**	**R^2**	**NRMSFE**	**R^2**	**NRMSFE**	**R^2**	**NRMSFE**	**R^2**	**NRMSFE**
Hosts	0.98	0.06	0.94	0.00	0.76	0.21	1.00	0.03	0.92	0.07
RUBiS	0.97	0.09	0.62	0.00	0.37	2.50	0.97	0.05	0.72	0.65
RUBBoS	0.77	0.04	0.70	0.00	0.34	0.46	0.93	0.01	0.62	0.13

lags above 4. Recall that p^* is the amount of historical values required to correctly predict resource behaviour of a node in the next time step automatically determined the model. About 47% of VMs traces have a low lag values while the remaining 63% have high lags. Selected order ranges between 2 and 10 with a modal value of 2 overall and 7 for the high order category. As shown in Fig. 7, CPU and Memory utilization are the easiest metric to forecast with average *NRMSFE* and R^2 of (0.15, 0.74) and (0.14, 0.77) respectively in both cases. Though the *NRMSFE* is quite similar for both low and high order models, the R^2 behaviour is slightly different especially for Disk IO utilization of the high order group with low median correlation of around 50%. Some traces in the high order group also have correlation as low as 0% for Network utilization. Fig. 8 is a shows correlation between predictions and actual observations of a sample VM trace.

Secondly, we present the prediction accuracy of the technique when deployed on our testbed in Table I aggregated for all hosts and VMs of target services. Results show that the technique offer high correlation and low variability in forecast errors across the datasets. The relatively high *NRMSFE* for *RUBiS*'s Disk resource can be explained by its disk-bound workload. The mean R^2 across host and VM nodes is 0.92 and 0.67 and average *NRMSFE* of 0.07 and 0.39 respectively. *The main finding is that while our technique provide high forecast accuracy, it is harder predicting IO behaviour in virtualized environments than compute and memory in both the public and empirical traces studied.*

B. *RQ2: To what extent can the technique identify abnormal multidimensional resource behaviour in the nodes?*

We examine the anomaly detection accuracy of the method by injecting the three contention scenarios described in Section V-A5 while running active workloads in the target services. Table II showcases the accuracy obtained from each scenario. Note that results are only aggregated across VMs of target services. The blanks represents where no *HIT* or *MISS* was observed and are not considered in the aggregation. What is noteworthy is not that the technique achieves an average accuracy of about 88% across the datasets but how excessive resource consumption in one of the load VMs propagate across the testbed.

For the compute scenario, the contention for compute resources on *Host 2* from *VM2* led to abnormal consumption in two potential victim VMs, the DB server (*VM4*) of *RUBiS* service and the Web server (*VM5*) of *RUBBoS*. Hence, there are more alarms from the *RUBiS* service as well as the Host 2 than for *RUBBoS* and *Host 1* respectively. Also, observe how the network congestion induced by heavy communication between *VM1* (client) and *VM2* (server) triggered many more alarms in the hosts than in the service VMs. The 4 alarms from *RUBBoS* is due to colocating the network server (*VM2*) with its Web server (*VM5*) on *Host 2* thereby limiting the bandwidth between the Web and DB servers. This issue does not seem to affect *RUBiS* perhaps due its web server residing on Host 1 or randomization of contention events leaves some bandwidth for communicating with the DB server. The high number of alarms for *RUBiS* under the Disk IO contention scenario are inadvertently triggered by IO noise from *VM2* running together with *VM4* (*RUBiS*' DB server). *The main finding is that our technique is able to detect the majority (88%) of abnormal resource usage incidents in the testbed under a variety of resource contention scenarios.*

TABLE II
DETECTION ACCURACY OF PROPOSED FRAMEWORK ON EMPIRICAL DATA AGGREGATED ACROSS HOSTS AND TARGET SERVICES

	Compute Contention			Disk IO Saturation			Network Congestion		
	#HIT	**#MISS**	**ACC%**	**#HIT**	**#MISS**	**ACC%**	**#HIT**	**#MISS**	**ACC%**
Host 1	12	00	100	-	-	-	26	02	93
Host 2	30	25	55	04	00	100	26	03	90
RUBiS	12	00	100	26	03	90	-	-	-
RUBBoS	07	00	100	01	01	50	04	00	100
Average			89%			80%			94%

C. *RQ3: How does the proposed approach compare to a baseline technique in terms of detection accuracy?*

We apply the baseline multivariate EWMA control chart on the same dataset obtained for *RQ1* in order to determine if it offers better accuracy than our proposed method. The results are presented in Table III as aggregated across hosts and service VMs. The approach produced too many false alarms out side of the contention periods caused by arbitrary change in workload levels, cases where our technique correctly fail to raise alarms. This is more pronounced for *RUBiS* where it generated significantly more false alarms than true positives. In comparison to our technique, Fig. 9 indicates that for compute issues both the baseline and proposed technique achieve comparable performance, while the accuracy of the baseline is very low under the Disk IO and Network contention scenarios. *The main finding is that the accuracy of our approach is at least a factor of 2 greater than a baseline method on the same dataset.*

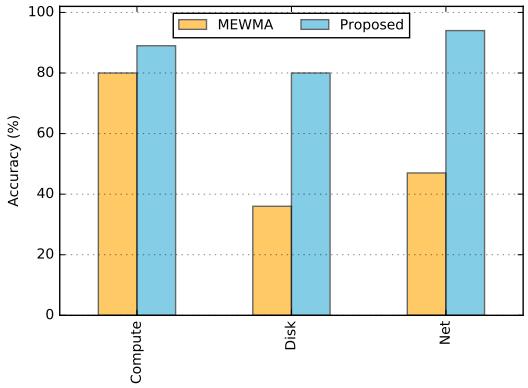


Fig. 9. Comparing average detection accuracy of proposed method and the baseline.

TABLE III
DETECTION ACCURACY OF MEWMA CONTROL CHART ON EMPIRICAL DATA AGGREGATED ACROSS HOSTS AND TARGET SERVICES

	Compute Contention			Disk IO Saturation			Network Congestion		
	#HIT	**#MISS**	**ACC%**	**#HIT**	**#MISS**	**ACC%**	**#HIT**	**#MISS**	**ACC%**
Host 1	08	01	89	03	04	43	10	08	56
Host 2	13	02	87	02	03	40	11	04	73
RUBiS	13	11	54	07	12	37	00	10	00
RUBBoS	10	01	91	01	03	25	07	05	58
Average			80%			36%			47%

D. *RQ4: How relevant are detected system anomalies to SLA violations at the service-level and how early can impending service degradation be discovered?*

The goal here is to quantify the feasibility of our approach to explain service-level degradations (*relevancy*) as well as how early this can be achieved (*leadtime*). Detection of SLA violations at the service-level is based on a user-defined threshold. Table IV is a summary of these metrics for the three contention scenarios from the same set of experiments in *RQ1*. The proposed approach was able to explain 46% of reported SLA violations across the two target services under Compute contention, 100% for Disk IO saturation and 40% for the Network congestion. The high relevancy for IO saturation effects impact end-users more than contention incidents involving other resources. However, it seems impact of abnormal IO events often happen abruptly considering its zero leadtime unlike abnormal network events and to a degree, compute contention. In general, the mean relevancy and leadtime achieved by our technique is 62% and 6 time-steps respectively. For an environment where the frequency of sampling and learning is 1mins, a leadtime of 6 suggests that system administrators can be notified of impending SLA violations before it causes harm.

For each contention scenario, we applied the ranking algorithm described in Section IV to rank unique anomaly alarms after each injection period. Fig. 10 shows dependency between the SLA violations in the victim service and system alarms in the most influential anomalous node as well alarms in the victim service VMs. The alignment between each pair of plots forms the basis for deriving the relevancy and leadtimes while

TABLE IV
AVERAGE RELEVANCY AND LEAD-TIMES

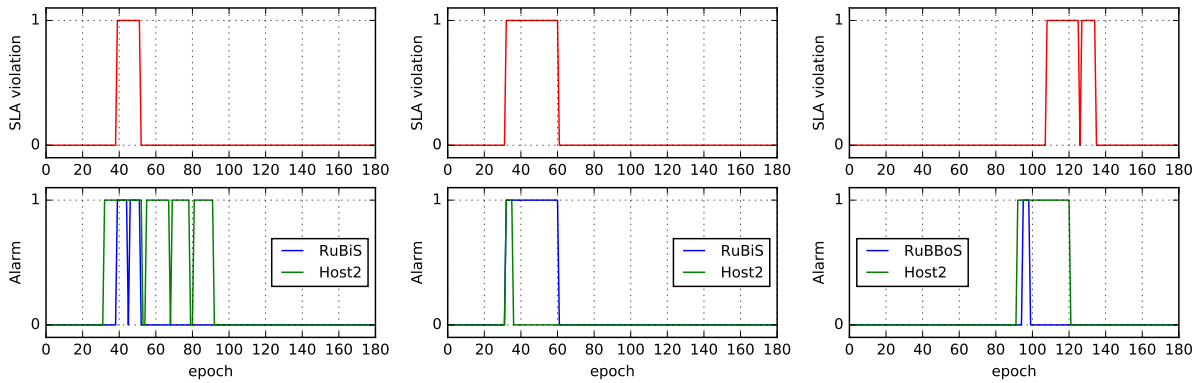
	Compute	IO	Network	Mean
Relevancy	46%	100%	40%	62%
Leadtime	03	00	15	06

the overlap between the two signals in the alarm plots is used to determine whether nodes are ranked correctly. For example, the SLA violations of *RUBiS* under the Disk IO contention scenario can be explained by IO noise from *Host 2* due to excessive disk activity from *VM2* thereby starving *RUBiS*' DB server hosted on the same host. *The main finding is that abnormal resource behaviour alarms triggered by our system is able to explain SLA violations 62% of the time. It is also able to trigger a system alarm 6 time steps ahead giving administrators some time to address impending service degradation.*

E. *RQ5: Is the performance of the proposed approach sensitive to varying sizes of the sliding windows?*

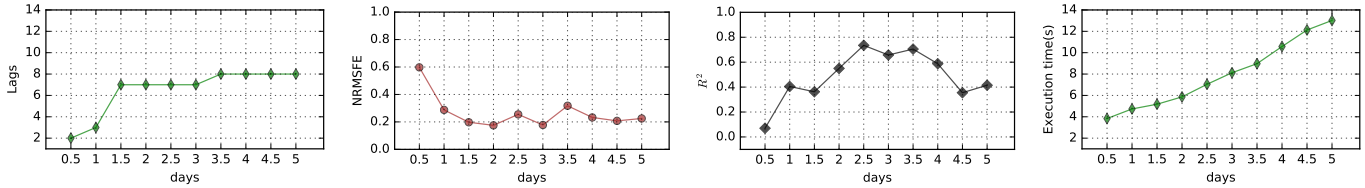
Though the hyper-parameter k–size of the sliding-windows–can be determined based on parameters n, and p_{max} of the VAR model, it is still objective to determine how varying k impact the optimal lag order $p*$, forecast accuracy (*NRMSFE* and R^2), detection accuracy as well as the runtime requirement (execution time). For a sample trace in the Materna datasets, Fig 11 demonstrates the impact of increasing k on the model obtained by incrementally predicting the next 12 hours while increasing the size of the sliding windows by 12-hour data starting from the past 0.5 to the past 5 days. The average selected order increased sharply from 2 given a half-day look-back window to about 7 on a 1.5-day look-back window and remains static until the baseline window grows to 3.5 days where it increases again to 8. The variability in forecast error also seem to reduce roughly with larger training window which is similar to the amount of variability explained in the data. The runtime requirement plot shows the total execution time (in seconds) required to complete 144 (12 hours) interleaved train-test cycle with increasing training size. The execution time ranges between 26ms for the smallest training size to about 90ms for the largest window, with an average of 55ms. This suggests a linear growth rate with increasing training window size.

We did the same for the empirical datasets using data from *Host 2* under compute contention, by varying the training size between the past 15mins (30 samples) and the past 90mins (180 samples) to incrementally detect anomalies in measurements over the next 90mins. We did not observe any significant impact on detection accuracy and combined execution time for prediction and detection. The mean detection accuracy remains 55% while the runtime range between 17s (94ms per cycle) to 18s (100ms per cycle) for the 15mins and 90mins training sets respectively. *The main finding is that experiments suggest that the runtime requirement grows linearly with size of training window. In addition, while forecast performance,*



(a) Compute Contention (b) Disk IO Saturation (c) Network Contention

Fig. 10. Plots showing dependency between SLA violations and system alarms of ranked nodes (1 for alarm and 0 otherwise).



(a) Selected order (b) Residual variability (c) Explained variability (d) Runtime requirement

Fig. 11. Sensitivity of the model to increasing size of training windows.

and the selected model order seem to influenced by varying training sizes, detection accuracy is not so influenced.

VII. Related Work

A few extensive surveys [6], [7] have been dedicated to reviewing existing research contributions on performance anomaly detection in distributed and computing systems in general.

Beyond approaches [8], [9] relying on thresholds that are sensitive to variations in distribution and workloads, or rule-based systems [19], researchers have exploited statistical properties of individual resource metrics for anomaly detection such as metric density [10], [29], metric distribution [30] and relative entropy [11], and forecast residuals [31], [32]. Correlations between metrics have also been assessed for uncovering relational anomalies such as in [13], [14] based on relationship between service workloads and resource consumption and [12], [15] considering correlation between application-level QoS and resource utilization as the basis for anomaly detection.

Blackbox techniques relying on machine learning have also been investigated for multivariate anomaly detection in [33] using Gaussian Mixture Models and [16] via Local Outlier Factor, methods which generally ignore inherent temporal properties of metrics data. Huang et al [34] has also proposed using Recurrent Neural Networks (RNN) but the approach require extensive training period and retraining to determine optimal network structure and parameters.

Recently, experimental works such as [35], [36] have investigated and evaluated the pros and cons of a variety of techniques for forecasting resource consumption pattern in datacenters. Nonetheless, forecasting is done along individual metric dimensions thereby ignoring contemporaneous associations among metrics.

VIII. Conclusion

Considering the ever increasing scale and complexity of cloud datacenters as well as the diversification of service workloads, cloud services face the growing risk of performance degradation due to systems anomalies such as abnormal resource usage propagated from tenants to tenants in complex ways. A key challenge facing cloud providers is how to detect system anomalies effectively. This paper proposed and evaluated a prediction-based framework for automated, distributed, and unsupervised detection of anomalies in multi-dimensional resource behaviour of nodes in virtualized environments as well as a graph-theoretic technique for ranking anomalous nodes. Anomalies are detected by uncovering abnormal changes in the temporal resource pattern via a block-based Vector Autoregressive (VAR) model. The proposed framework is evaluated using resource traces of over 100 virtual machines obtained from a production cluster as well as traces obtained from an experimental testbed under realistic service composition. The technique achieve average normalized root mean squared forecast error and R^2 of (0.92, 0.07) across hosts servers and (0.70, 0.39) across virtual machines. Also, the average detection rate is 88% while explaining 62% of SLA violations with an average lead-time of 6 time-points when the testbed is actively perturbed under three contention scenarios. Future work will focus on extended evaluation of the

anomaly ranking, investigate end-to-end root-cause analysis and self-healing approaches to reduce the impact of systems anomalies on end-user performance.

ACKNOWLEDGMENT

This work is supported by the Swedish Research Council (VR) under contract C0590801 for the Cloud Control project, the Swedish Strategic Research Program eSSENCE, and the EU Seventh Framework Programme under grant agreement 610711 (CACTOS). Special thanks to the Grid Workloads Archive for making the Materna traces publicly available.

REFERENCES

[1] Compuware/Research In Action, "The Hidden Costs of Managing Applications in the Cloud," http://research-in-action.wks-international.de/images/pdf/05%20RIA%20WP%20Cost%20Of%20Cloud%20.pdf, 2012, accessed: 2017-04-18.

[2] Tammy Everts (Radware), "The Real Cost of Slow Time vs Downtime," http://www.slideshare.net/Radware/radware-cmg2014-tammyevertsslowtimevsdowntime, 2014, accessed: 2017-04-18.

[3] F. Meng, M. Wegman, J. Xu, X. Zhang, P. Chen, and G. Chafle, "IT Troubleshooting with Drift Analysis in the DevOps Era," *IBM Journal of Research and Development*, vol. 61, no. 1, pp. 6–62, 2017.

[4] M. Sedaghat, F. Hernández-Rodriguez, and E. Elmroth, "Autonomic Resource Allocation for Cloud Data Centers: A Peer to Peer Approach," in *International Conference on Cloud and Autonomic Computing (ICCAC)*. IEEE, 2014, pp. 131–140.

[5] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments," in *3rd International Conference on Cloud Computing (CLOUD)*. IEEE, 2010, pp. 51–58.

[6] C. Wang, S. P. Kavulya, J. Tan, L. Hu, M. Kutare, M. Kasick, K. Schwan, P. Narasimhan, and R. Gandhi, "Performance Troubleshooting in Data Centers: An Annotated Bibliography?" *ACM SIGOPS Operating Systems Review*, vol. 47, no. 3, pp. 50–62, 2013.

[7] 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

[8] 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.

[9] M. Peiris, J. H. Hill, J. Thelin, S. Bykov, G. Kliot, and C. Konig, "PAD: Performance Anomaly Detection in Multi-server Distributed Systems," in *7th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 2014, pp. 769–776.

[10] 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.

[11] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan, "Statistical Techniques for Online Anomaly Detection in Data Centers," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. IEEE, 2011, pp. 385–392.

[12] P. Xiong, C. Pu, X. Zhu, and R. Griffith, "vPerfGuard: an Automated Model-driven Framework for Application Performance Diagnosis in Consolidated Cloud Environments," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. ACM, 2013, pp. 271–282.

[13] J. P. Magalhaes and L. M. Silva, "Detection of Performance Anomalies in Web-based Applications," in *9th IEEE International Symposium on Network Computing and Applications (NCA)*. IEEE, 2010, pp. 60–67.

[14] 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.

[15] T. Matsuki and N. Matsuoka, "A Resource Contention Analysis Framework for Diagnosis of Application Performance Anomalies in Consolidated Cloud Environments," in *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*. ACM, 2016, pp. 173–184.

[16] T. Huang, Y. Zhu, Y. Wu, S. Bressan, and G. Dobbie, "Anomaly Detection and Identification Scheme for VM Live Migration in Cloud Infrastructure," *Future Generation Computer Systems*, vol. 56, pp. 736–745, 2016.

[17] J. Lin, Q. Zhang, H. Bannazadeh, and A. Leon-Garcia, "Automated Anomaly Detection and Root Cause Analysis in Virtualized Cloud Infrastructures," in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. IEEE, 2016, pp. 550–556.

[18] R. Birke, L. Y. Chen, and E. Smirni, "Usage Patterns in Multi-tenant Data Centers: A Temporal Perspective," in *Proceedings of the 9th International Conference on Autonomic Computing*. ACM, 2012, pp. 161–166.

[19] R. Ahad, E. Chan, and A. Santos, "Toward Autonomic Cloud: Automatic Anomaly Detection and Resolution," in *International Conference on Cloud and Autonomic Computing (ICCAC)*. IEEE, 2015, pp. 200–203.

[20] R. S. Tsay, *Multivariate Time Series Analysis: with R and financial applications*. John Wiley & Sons, 2013.

[21] P. J. Rousseeuw and K. V. Driessen, "A Fast Algorithm for the Minimum Covariance Determinant Estimator," *Technometrics*, vol. 41, no. 3, pp. 212–223, 1999.

[22] T. Metsch, O. Ibidunmoye, V. Bayon-Molino, J. Butler, F. Hernández-Rodriguez, and E. Elmroth, "Apex Lake: A Framework for Enabling Smart Orchestration," in *Proceedings of the Industrial Track of the 16th International Middleware Conference*. ACM, 2015, p. 1.

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

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

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

[26] S. Barbhuiya, Z. Papazachos, P. Kilpatrick, and D. S. Nikolopoulos, "LS-ADT: Lightweight and Scalable Anomaly Detection for Cloud Datacentres," in *International Conference on Cloud Computing and Services Science*. Springer, 2015, pp. 135–152.

[27] S. Farokhi, E. B. Lakew, C. Klein, I. Brandic, and E. Elmroth, "Coordinating CPU and Memory Elasticity Controllers to Meet Service Response Time Constraints," in *International Conference on Cloud and Autonomic Computing (ICCAC)*. IEEE, 2015, pp. 69–80.

[28] Andreas Kohne, "GWA-T13-materna-trace," http://gwa.ewi.tudelft.nl/datasets/gwa-t-13-materna, accessed: 2017-05-05.

[29] S. N. U. H. Shirazi, S. Simpson, A. Gouglidis, A. U. Mauthe, and D. Hutchison, "Anomaly Detection in the Cloud using Data Density," in *IEEE International Conference on Cloud Computing*, 2016.

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

[31] J. Tan, P. Dube, X. Meng, and L. Zhang, "Exploiting Resource Usage Patterns for Better Utilization Prediction," in *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*. IEEE, 2011, pp. 14–19.

[32] L. Yang, C. Liu, J. M. Schopf, and I. Foster, "Anomaly Detection and Diagnosis in Grid Environments," in *Proceedings of the ACM/IEEE Conference on Supercomputing*. IEEE, 2007, pp. 1–9.

[33] G. M. Abdelrahman and M. M. Nasr, "Detection of Performance Anomalies in Cloud Services: A Correlation Analysis Approach," *International Journal of Mechanical Engineering and Information Technology*, vol. 4, no. 9, pp. 1773–1781, 2016.

[34] S. Huang, C. Fung, K. Wang, P. Pei, Z. Luan, and D. Qian, "*Using Recurrent Neural Networks Toward Black-box System Anomaly Prediction*," in *IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*. IEEE, 2016, pp. 1–10.

[35] 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.

[36] J. Xue, F. Yan, R. Birke, L. Y. Chen, T. Scherer, and E. Smirni, "PRACTICE: Robust Prediction of Data Center Time Series," in *11th International Conference on Network and Service Management (CNSM)*. IEEE, 2015, pp. 126–134.