Online Detection of Utility Cloud Anomalies Using Metric Distributions

Chengwei Wang, Vanish Talwar*, Karsten Schwan, Parthasarathy Ranganathan*

Center for Experimental Research in Computer Systems
College of Computing, Georgia Institute of Technology, Atlanta, GA 30318, USA
{flinter, schwan}@cc.gatech.edu
* HP Labs, Palo Alto CA 94304
{vanish.talwar, partha.ranganathan}@hp.com

***Abstract*—The online detection of anomalies is a vital element of operations in data centers and in utility clouds like Amazon EC2. Given ever-increasing data center sizes coupled with the complexities of systems software, applications, and workload patterns, such anomaly detection must operate automatically, at runtime, and without the need for prior knowledge about normal or anomalous behaviors. Further, detection should function for different levels of abstraction like hardware and software, and for the multiple metrics used in cloud computing systems. This paper proposes EbAT – Entropy-based Anomaly Testing – offering novel methods that detect anomalies by analyzing for arbitrary metrics their distributions rather than individual metric thresholds. Entropy is used as a measurement that captures the degree of dispersal or concentration of such distributions, aggregating raw metric data across the cloud stack to form entropy time series. For scalability, such time series can then be combined hierarchically and across multiple cloud subsystems. Experimental results on utility cloud scenarios demonstrate the viability of the approach. EbAT outperforms threshold-based methods with on average 57.4% improvement in accuracy of anomaly detection and also does better by 59.3% on average in false alarm rate with a 'near-optimum' threshold-based method.**

I. Introduction

The online detection of anomalous system behavior [17] caused by operator errors [26], hardware/software failures [27], resource over-/under-provisioning [22], [21], and similar causes is a vital element of operations in large-scale data centers and utility clouds like Amazon EC2 [1]. Given the ever-increasing scale coupled with the increasing complexity of software, applications, and workload patterns, anomaly detection methods must operate automatically at runtime and without the need for prior knowledge about normal or anomalous behaviors. Further, they should be sufficiently general to apply to multiple levels of abstraction and subsystems and for the different metrics used in large-scale systems.

The detection methods [6], [5], [8] currently used in industry are often ad hoc or specific to certain applications, and they may require extensive tuning for sensitivity and/or to avoid high rates of false alarms. An issue with threshold-based methods, for instance, is that they detect anomalies after they occur instead of noticing their impending arrival. Further, potentially high false alarm rates can result from monitoring only individual rather than combinations of metrics. New methods [19], [12], [18], [15], [22], [10], [11], [14] developed in recent research can be unresponsive due to their use of complex statistical techniques and/or may suffer from a relative lack of scalability because they mine immense amounts of non-aggregated metric data. In addition, their analyses often require prior knowledge about application SLOs, service implementation, or request semantics.

This paper proposes the EbAT – Entropy-based Anomaly Testing – approach to anomaly detection. EbAT analyzes metric distributions rather than individual metric thresholds. We use entropy as a measurement to capture the degree of dispersal or concentration of such distributions. The resulting 'entropy distributions' aggregate raw metric data across the cloud stack to form 'entropy time series', and in addition, each hierarchy of a cloud (data center, container, rack, enclosure, node, socket, core) can generate higher level from lower level entropy time series. We then use online tools – spike detecting (visually or using time series analysis), signal processing or subspace method – to identify anomalies in entropy time series in general and at each level of the hierarchy. The current implementation employs wavelet analysis and visual spike detection.

EbAT constitutes a lightweight online approach to scalable monitoring, capable of raising alarms and zooming in on potential problem areas in clouds. Since the volume of monitoring data is exponentially reduced along the cloud hierarchy, the approach can easily scale as metric volume grows. When the metrics used are collected with black-box methods [12] like hypervisor-level monitoring, applications and systems can be monitored without client- or vendor-specific knowledge about their make, nature, or expected behavior. When applying detection to metrics collected with gray-box techniques [13], we exploit existing knowledge about certain hardware/software components or applications. In either case, there is no need for human involvement or intervention, thereby reducing operating costs, and there is minimum requirement of prior knowledge about hardware/software or of typical failure models. As a result, EbAT can detect anomalies that are not well understood (i.e., no prior models) or have not been experienced previously, and it can operate at any one and across multiple of the levels of abstraction existing in modern systems, ranging

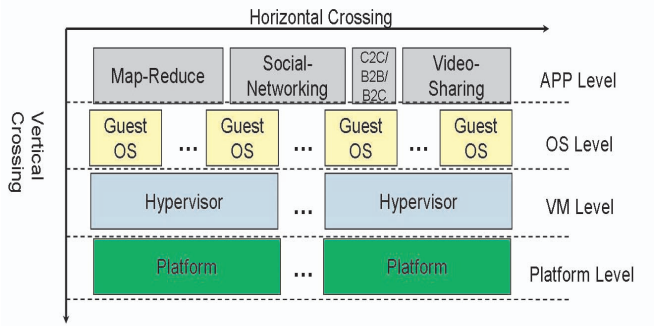

Fig. 1. The functional view of a utility cloud

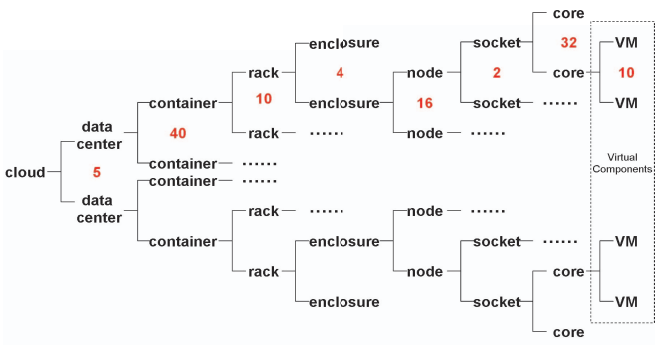

Fig. 2. The cloud hierarchy where red numbers are typical quantities of components (cores, sockets, racks, etc.) to be expected in a future cloud. These typical quantities would lead to 5*40*10*4*16*2*32 = 8'192'000 physical cores or 81'920'000 virtual machines

from the hypervisor, to OS kernels, to middleware, and to applications.

In summary and compared to prior work on cloud anomaly detection, EbAT's technical contributions include the following:

1) a novel metric distribution-based method for anomaly detection using entropy;
2) a hierarchical aggregation of entropy time series via multiple analytical methods, to attain cloud scalability;
3) an evaluation with two typical utility cloud scenarios, to demonstrate the viability and accuracy of the proposed techniques, and to compare EbAT's methods with the threshold-based techniques currently in wide-spread use.

Expanding on 3) above, results show that EbAT outperforms threshold-based methods with on average 57.4% in F_1 score and also does better by 59.3% on average in false alarm rate with a 'near-optimum' threshold-based method. The second case study uses a data intensive code to discuss interesting properties of EbAT and the manner in which its methods should be used.

The remainder of this paper is organized as follows. Section II describes the research challenges. Section III, Section IV, and Section V detail the EbAT approach. Experimental evaluation and discussion are described in Section VI and VII. Section VIII present conclusions and future work.

II. Problem Description

A. Utility Cloud Characteristics

A utility cloud's physical hierarchy is illustrated in Figure 2, where red numbers are typical quantities of components at each level. The hierarchical relationship between hardware components and virtual environment – data center, container, rack, enclosure, node, socket, core and virtual machines (VMs for short) – is typically configured statically at hardware levels. Considering virtual components, the simplified hierarchy used in our experiments describes VMs as direct children of nodes (hosts), but this can be generalized as shown in Figure 1:

1) $Exascale$: for up to 10M physical cores, there may be up to 10 virtual machines per node (or per core). Given a possibility of non-trivial number of sensors per node and additional sensors for each level of physical hierarchy, the total amount of metrics can reach exascale, 10^{18}.
2) $Dynamism$: utility clouds serve as a general computing facility. Heterogeneous applications include, but are not limited to, Map-Reduce, social networking, e-commerce solutions and multi-tier web applications, streaming applications and video sharing. While running simultaneously, these applications tend to have different workload/request patterns. Online management of virtual machines like live migration and power management make a utility cloud more dynamic than existing data center facilities. Further, utility clouds experience continuous churn of workloads with applications continuously entering and exiting the system.

B. State of the Art

1) Threshold-Based Approaches: Threshold-based methods are pervasively leveraged in industry monitoring products [3], [7], [6], [5], [8]. They firstly set up upper/lower bounds for each metric. Those threshold values come from predefined performance knowledge or constraints (e.g., SLOs) or from predictions based on long-term historical data analysis. They can be set statically or dynamically. Whenever any of the metric observation violates a threshold limit, an alarm of anomaly is triggered.

Providing a moderate volume of metrics to operation teams with highly trained expertise, threshold-based methods are widely used with advantages of simplicity and ease of visual presentation. They however do not meet utility cloud requirements due to their following intrinsic shortcomings:

1) *Incremental False Alarm Rate (FAR)*: consider a threshold based method monitoring n metrics: m_1, m_2 ... m_n for each mi , if the FAR is r_i, the overall FAR of this method is $\sum_{i=1}^{n} r_i$. Thus, this means that when monitoring 50 metrics with FAR 1/250 each (1 false alarm every 250 samples), there will be $50/250 = 1/5$, i.e., 1 false alarm every 5 samples! Thus, the false alarm rate in threshold-based technique grows fast with increase in the number of monitoring metrics.
2) *Detection after the Fact*: consider 100 Web Application Servers (WAS) running the same service deficiently coded with $memoryleaks$. The memory utilization

TABLE I
TYPICAL STATISTICAL APPROACHES AND THEIR FEATURES. H-CROSSING MEANS AGGREGATING METRICS AMONG PEER DISTRIBUTED COMPONENTS. V-CROSSING MEANS AGGREGATING METRICS CROSS-LAYER IN THE HW-SW IMPLEMENTATION STACK.

Technique	Function	Scalability	Online	Black-Box	H-Crossing	V-Crossing
SLIC [19]	problem determination	n	n	n	y	y
Nesting/Convolution [12]	performance debugging	n	n	y	y	n
Pinpoint [18]	problem determination	n	n	n	y	n
Magpie [15]	performance debugging	n	n	y	y	n
Pranaali [22]	SLA management	y	y	n	y	n
E2EProf[10]	performance management	n	n	y	y	n
SysProf [11]	performance management	n	n	y	y	n
Sherklock [14]	problem determination	n	n	y	y	n
EbAT	**anomaly detection**	**y**	**y**	**y**	**y**	**y**

metrics from all those WASes may stay below their thresholds for a period of time as memory use is slowly increasing. Thus, no anomaly is detected. When one of the WASes raises an alarm because it crosses the threshold, it is likely that all other 99 WASes raise alarms soon thereafter, thereby causing a revenue disaster.

3) *Poor Scalability*: it is obvious that as we approach exascale volumes in monitoring metrics, it is no longer efficient to monitor metrics individually.

The lesson we learn from threshold-based approaches is that detecting anomalies by individual metric value threshold may not work well in exascale, highly dynamic cloud environments. Needed are new detection gauges and novel ways of aggregating metrics.

2) Statistical Methods: There exist many promising methods for anomaly detection, typically based on statistical techniques. However, few of them can deal with the scale of future cloud computing systems and/or the need for online detection, because they use statistical algorithms with high computing overheads and/or onerously mine immense amounts of raw metric data without first aggregating it. In addition, they often require prior knowledge about application SLOs, service implementations, request semantics, or they solve specific problems at specific levels of abstraction (i.e., metric levels). A summary of features of well-known statistical methods appears in Table I along with a comparison with EbAT.

Specifically, Cohen et al. [19] developed an approach in the SLIC project that statistically clusters metrics with respect to SLOs to create system signatures. Chen et al. [18] proposed Pinpoint using clustering/correlation analysis for problem determination. Magpie [15] is a request extraction and workload modeling tool. Aguilera. et al's [12] nesting/convolution algorithms are black-box methods to find causal paths between service components. Arpaci-Dusseau et al. [13] develop gray-box techniques that use information about the internal states maintained in certain operating system components, e.g., to control file layout on top of FFS-like file systems [25]. Concerning data center management, Agarwala et al. [10], [11] propose profiling tools, E2EProf and SysProf, that can capture monitoring information at different levels of granularity. They however address different sets of VM behaviors, focusing on relationships among VMs rather than anomalies. Kumar et al. [22] proposed Pranaali, a state-space approach for SLA management of distributed software components. Bahl et al. [14] developed Sherklock using inference graph model to auto detect/localize internet services problems.

In contrast, our method (EbAT) aims to address the scalability needs of Utility Clouds, providing an online lightweight technique that can operate in a black-box manner across multiple horizontal and vertical metrics. We leverage entropy-based analysis technique that has been used in the past for network monitoring [23], [24], but we adapt it for data center monitoring operating at and across different levels of abstraction, including applications, middleware, operating systems, virtual machines, and hardware.

III. EBAT OVERVIEW

The EbAT approach is depicted in Figure 3 as operating in three steps: *metric collection*, *entropy time series construction*, and *entropy time series processing*.

The *metric collection* happens at all components on each physical level of the hierarchy. The types of data collected depends on the level. A leaf component collects raw metric data from its local sensors. A non-leaf component collects not only its local metric data but also entropy time series data from its child nodes.

In the *entropy time series construction* step, data is normalized and binned into intervals. Leaf nodes only generate monitoring events (m-events for short) from its local metrics. Non-leaf nodes generate m-events from local metrics and child nodes' entropy time series, respectively. Those m-events are then counted at runtime to calculate the entropy time series of current components. Details about m-event and entropy calculation appear in Section IV.

In the *entropy time series processing* step, entropy time series are analyzed by one or multiple methods from spike detection, signal processing or subspace method in order to find anomalous patterns which are indications of anomalies in monitored system. Details are discussed in Section V.

IV. ENTROPY TIME SERIES

A. Look-Back Window

As an online detection method, EbAT maintains a buffer of the last n samples' metrics observed. We call this buffer a *look-back window* which slides sample by sample during the monitoring process. The metrics in the look-back window at

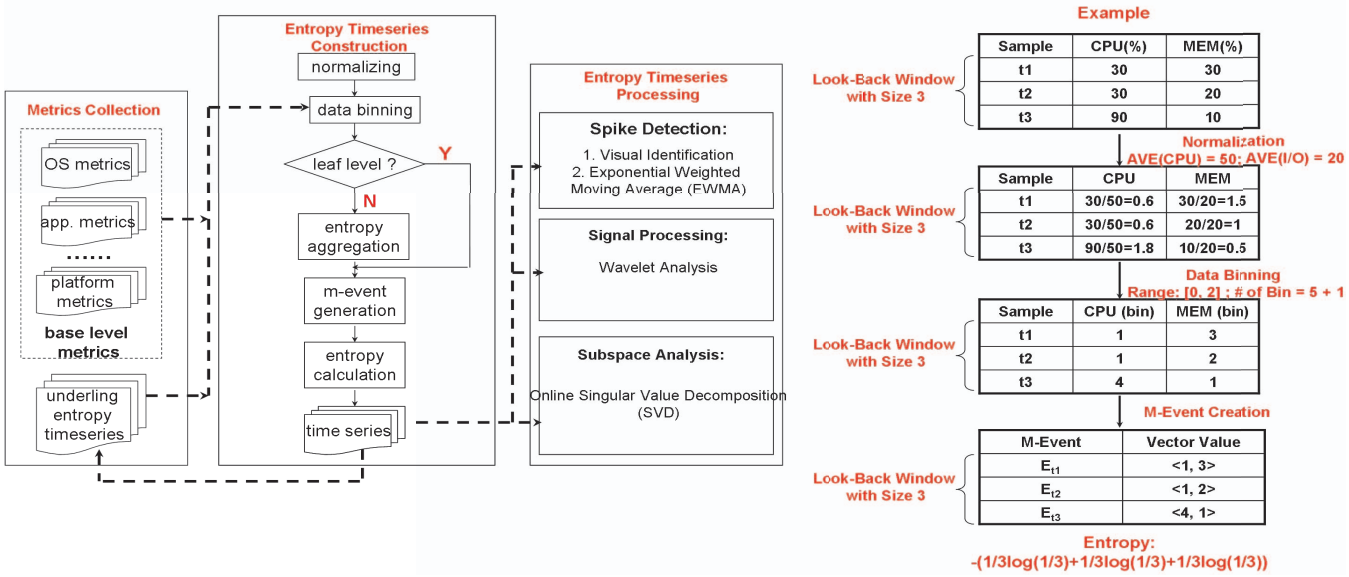

Fig. 3. (a) The EbAT framework, (b) Illustration of the EbAT method

each time instance serve as inputs for pre-processing, m-event creation, and entropy calculation. The look-back window is used for multiple reasons. First, at exascale, it is impractical to maintain all history data. Second, shifts of work patterns may render old history data misleading or even useless. Third, the look-back window can be implemented in high speed RAM, which can further increase detection performance.

B. Pre-Processing Raw Metrics

The monitoring data within each look-back window is first pre-processed and transformed into a form that can be readily used by EbAT. This pre-processing involves two steps explained below (also shown in Figure 3). Note that at each monitoring sample, multiple types of metric can be collected simultaneously , e.g. in our experiment, we collect CPU utilization, memory utilization and VBD read/write in each monitoring sample. In addition, the entropy values of child components can be collected simultaneously by non-leaf components. The pre-processing steps below are thus done for samples collected for every type of metric and/or child.

Step 1: Normalization In this step, EbAT transforms a sample value to a normalized value by dividing the sample value by the mean of all values of the same type in the current look-back window.

Step 2: Data binning Once normalization is complete, each of the normalized sample values are hashed to a bin of size m+1. This happens as follows. We predefine a value range [0,r], and split it into m equal-sized bins indexed from 0 to *m-1*. We define another bin indexed m which captures values larger than r. Both m and r are pre-determined statistically but are configurable parameters to the method. Each of the normalized values from Step 1 are put into a specific bin - if the value is greater than r, then it is placed in the bin with index m, else it is put in a bin with index, the floor of $sample_{value}$/(r/m). Thus, now each of the sample values are associated with a bin index number. This number is recorded and used to create m-events described in next subsection.

Figure 3(b) shows an example illustration of the above steps for a look-back window of size 3 with CPU and memory utilization metrics.

TABLE II
DEFINITIONS OF B_{tj} AND k

	B_{tj}	k
global	bin index of j^{th} child entropy	total# of children
local	bin index of j^{th} local metric	total# of local metrics

C. M-Event Creation

Once the sample data is pre-processed and transformed to a series of bin index numbers for every metric type and/or child, an m-event is generated that includes the transformed values from multiple metric types and/or child into a single vector for each sample instance. More specifically, an m-event of a component at sample t, E_t is formulated as the following vector description:

$$E_t =< B_{t1}, B_{t2}...., B_{tj}, ...B_{tk} > \quad (1)$$

where B_{tj} and k are defined as in Table II,

The only restriction is that an entropy value and a local metric value should not be in the same vector of an m-event. The m-event for the example illustrated in previous subsection is shown in Figure 3(b).

According to the above definitions, at each component except for leaves, there will be two types of m-events:

1) *global m-events* aggregating entropies of its subtree, and
2) *local m-events* recording local metric transformation values, i.e. bin index numbers.

Within a specific component, the template of m-events is fixed, i.e. the types and the order of raw metrics in the m-event(local or global) vector are fixed. Therefore two m-events, E_a and E_b have the same vector value if they are created on the same component and $\forall$ j $\in$ [1, k],B_{aj} = B_{bj}

From above descriptions, an m-event is actually the representation of a monitoring sample. Therefore, on each component, there will be n m-events in the look-back window with size n. The next step is to extract the statistical characteristics of m-events in this look-back window.

D. Entropy Calculation and Aggregation

Entropy [29] is a widely used measurement that captures the degree of dispersal or concentration of random variable distributions. For a discrete random variable *X* with possible values $\{x_1$,x_2 ..., $x_n\}$, its entropy is:

$$H(X) = -\sum_{i=1}^{n} P(x_i) log P(x_i) \quad (2)$$

where $P(x_i)$ is the probability mass function of outcome x_i. $-logP(x_i)$ is called $surprisal$ or *self-information* of x_i.

We deem the random variable as an observation E in the look-back window with size n samples. The outcomes of E are v m-event vector values $\{e_1$,e_2 ..., $e_v\}$ where v != n when there are m-events with the same value in the n samples. For each of these v values, we keep a count of the number of occurrence of that e_i in the n samples. This is designated as n_i and represents the number of m-events with vector value e_i. We then calculate $H(E)$ by the Formula 3. By calculating $H(E)$ for the look back window, we get the global and local entropy time series describing metric distributions for that look back window. Since, the look back window slides sample by sample as mentioned in Section IVA, $H(E)$ gets calculated at every monitoring sample for the corresponding look back window.

$$H(E) = -\sum_{i=1}^{v} \frac{n_i}{n} log \frac{n_i}{n} \quad (3)$$

In the case of local entropy calculation, the m-events represent the pre-processing results of the raw metrics. In the case of global entropy calculation, the m-events represent the pre-processing results of the child entropy time series. While Equation 3 is one way to calculate global entropy using child entropy m-events, one can imagine other alternative aggregations too. We have in particular considered the following alternative formula (Equation 4) in our implementation which considers the combination of sum and product of the individual child entropies, and we show the evaluation comparison for the two approaches in Section VI.

$$A = \sum_{i=1}^{c} H_i + \prod_{i=1}^{c} H_i \quad (4)$$

Above, c is the number of children nodes, say, VMs on the same host (as in our experiment), and H_i represents the local entropy of the child i.

E. Qualitative Scalability Discussion

It is easy to conclude, from above sections, that the following lightweight-natures of EbAT contribute to the scalability: (1) maintaining constant in-memory historical data for processing and minimizing communication cost by aggregation (2) extremely low computation/space complexity on entropy calculation. We are in progress of large scale experiments to prove the scalability of EbAT.

V. Entropy Time Series Processing

Entropy Time Series Analysis: After gathering the local/global entropy time series as described in the previous section, we can use data analysis methods to identify normal entropies and anomalous ones. The detection will reveal anomalous distribution changes in monitoring metrics. Appropriate techniques include 1) spike detection, 2) signal processing, and 3) subspace methods. The EbAT software framework permits one to choose any or multiple of those methods online, thereby enabling users to deal with individual methods' limitations and tune detection performance. The current implementation uses visual spike detection, as shown in Figure 5(a) for visually obvious spikes and wavelet analysis to identify abnormal patterns when visual detection is infeasible, as shown in Figure 5(b).

VI. Evaluation with Distributed Online Service

Online services are an important class of applications in utility clouds. Using the widely used RUBiS benchmark deployed as a set of virtual machines, EbAT is evaluated for its' viability and in terms of effectiveness compared to the threshold-based methods in wide use. The goal is to detect synthetic anomalies injected into the RUBiS services. Effectiveness is evaluated using precision, recall and F_1 score as metrics which will be discussed in Section VI-D. Results show that EbAT outperforms threshold-based methods with on average 18.9% increase in F_1 score and we also do better by 50% on average in false alarm rate with the 'near-optimum' threshold-based method I.

A. Experiment Setup

RUBiS [16] is a distributed online service benchmark implementing the core functionality of an auction site. When used in the cloud context, its services are deployed in virtual machines mapped to the cloud's machine resources, as depicted in Figure 4.

The testbed uses 5 virtual machines (VM1 to VM5) on Xen platform hosted on two Dell PowerEdge 1950 servers (Host1 and Host2). VM1, VM2, and VM3 are created on Host1. The frontend server processing or redirecting service requests runs in VM1. The application server handling the application logic runs in VM2. The database backend server is deployed on VM3. The deployment is typical in its use of multiple VMs and the consolidation of such VMs onto a smaller number of hosts.

A request load generator and an anomaly injector are running on two virtual machines, VM4 and VM5, on another

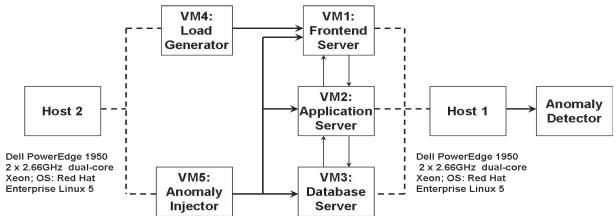

Fig. 4. The experiment setup for RUBiS

TABLE III
ANOMALIES INJECTED

Anomaly	Typical Source Type	# of Anomalies
Apache Server Inaccessbile	Operation Error	10
Mysql Server Inaccessible	Operation Error	16
Tomcat Server Inaccessible	Operation Error	10
Apache Server CPU Exhaustion	Resource Provision	3
Mysql Server CPU Exhaustion	Resource Provision	8
Tomcat Server CPU Exhaustion	Resource Provision	3

server, Host2. The generator creates 10 hours' worth of service request load for Host1 where the auction site resides. The load emulates concurrent clients, sessions, and human activities. During the experiment, the anomaly injector injects 50 anomalies into the RUBiS online service in Host1. Those 50 anomalies, as shown in Table III, come from major sources of failures or performance issues in online services [26], [22], [21], [27]. We inject them into the testbed using a uniform distribution. The virtual machine metrics and the host metrics are collected using Xentop and analyzed in an anomaly detector. Anomaly detection applies threshold-based and EbAT's methods to online observations of the CPU utilizations of virtual machines and hosts (i.e., using black-box monitoring).

B. Baseline Methods – Threshold-Based Detection

We choose a threshold-based method as the baseline. This baseline method compares observed CPU utilization with a lower bound and higher bound threshold. Whenever the utilization goes below the lower bound threshold, or above the higher bound threshold, a violation to the thresholds is detected, and the baseline method will raise an alarm. Consecutive violations are aggregated into a single alarm. For our evaluations, we consider two separate values of the thresholds. The first one is a near-optimum threshold value set by an 'oracle'-based method, and the other is a statically set threshold value that is not optimum but representative of how current state of the art real-world deployments use.

To obtain the near optimum threshold value, we feed the complete historical knowledge of the host CPU utilization to the baseline method. This historical trace matches exactly what the host will experience during the online RUBiS experimentation when the baseline method will be operating. From the historical CPU utilization trace, we calculate the histogram and from that, the lowest and highest 1% of the values are identified as representing outliers outside an acceptable operating range. The corresponding 1%boundary values are then chosen as the lower and upper bound thresholds. This is illustrated in Figure 6(b) for Host1. Specifically, 3.58% is chosen as the lower bound and 29.5% as the upper bound. 2% is chosen as the total value for outliers because the time taken by anomalies injected occupies 2% of the total experiment period. 2% is then appropriate in the sense that it can catch as many anomalies as possible without sacrificing accuracy, making the chosen threshold values near-optimum/ideal.

Figure 6(a) shows the results when applying these near-optimum thresholds to the host CPU utilization data for detecting anomalies. Red circles are violations when applying the baseline threshold method Consecutive red circles are deemed as a single anomaly because it may hold for some period of time. A successful alarm and the according actual anomaly injection may not happen at the same time because there is a latency between anomaly injection and its effects on the metrics being collected and analyzed.

For the non-optimum statically set threshold values, we choose 90% and 5% as the upper and lower bounds for the thresholds based on representative values used in state of the art deployments. We employ a similar exercise on the Host1 CPU utilization data using these threshold values to detect anomalies. The overall results are summarized in Table V which would be explained in Section VI-D.

TABLE IV
PARAMETERS FOR ENTROPY CALCULATION AND AGGREGATION

	look-back window size	range	# of bins
local entropy calculation	20	5	6
entropy I aggregation	100	5	5

C. EbAT Method Implementation

We apply the EbAT method implementation at the VM level and host level. As mentioned in Section II, we deem VM1 to VM3 as the direct children of Host1. We thereby first calculate the local entropy time series in each VM and then aggregate them to global entropy time series for Host1. Two aggregations for global entropy described in Section IV-D are evaluated and they are named "Entropy I" and "Entropy II", respectively (corresponding to Equations 3 & 4). The parameters chosen for entropy calculation and aggregation are presented in Table IV.

D. Evaluation Results

We use four measures [28], [9], [2], in statistics to evaluate the effectiveness of anomaly detection:

$$\textit{Precision} = \frac{\textit{\# of successful detections}}{\textit{\# of total alarms}} \tag{5}$$

$$\textit{Recall} = \frac{\textit{\# of successful detections}}{\textit{\# of total anomalies}} \tag{6}$$

$$\textit{Accuracy}(F_1) = \frac{2 * precision * recall}{precision + recall} \tag{7}$$

$$\textit{False Alarm Rate (FAR)} = \frac{\textit{\# of false alarms}}{\textit{\# of total alarms}} = 1 - Precision \tag{8}$$

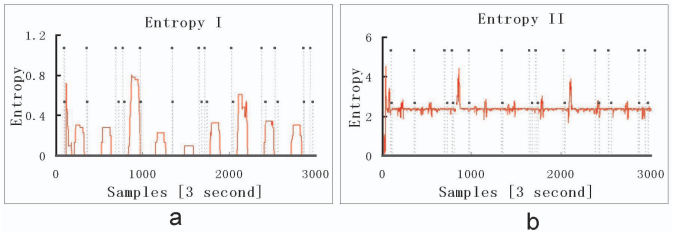

Fig. 5. Fragment (3000 samples out of 12000 samples) of Entropy I (a) and Entropy II (b) traces: The spotted vertical lines represent anomaly occurrences. Red lines represent entropy traces. Spikes in those traces indicate an alarm.

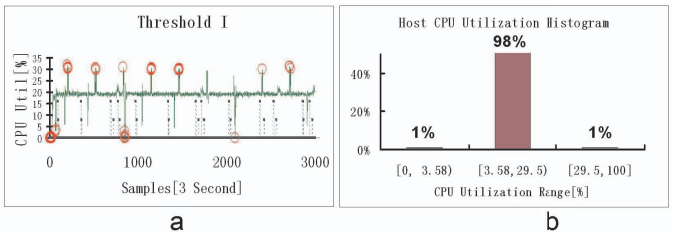

Fig. 6. (a) A Fragment (3000 samples out of 12000 samples) of Threshold I Trace, and (b) Histogram of CPU Utilization. The spotted vertical lines represent anomaly occurrences. The red circles represent CPU utilizations violating thresholds.

Precision is used to measure the exactness of the detection. Recall measures the completeness. Neither precision nor recall alone can judge the goodness of an anomaly detection method. Therefore we further use Accuracy, or F_1 score which is the harmonic mean of precision and recall, to compare the performances of EbAT and threshold-based methods. FAR is really 1-$Precision$.

A summary of results appears in Table V, and detailed traces of entropy, raw metrics, anomalies injected and alarms are depicted in Figure 5 and 6(a). Overall, compared with the threshold-based detection methods, the use of EbAT results in on average 57.4% improvement in F_1 score. We also do better by 59.3% on average in false alarm rate with the 'near-optimum' threshold-based method I.

EbAT methods outperform threshold-based methods in accuracy and almost all precision and recall measurements. The only exception is that of precision with Threshold II, however Threshold II only detects 16 anomalies out of total 50, thus missing detection of many anomalies reflected through poor recall. Entropy II has worse FAR competed to Entropy I because the sum/multiply of entropies accumulate false alarms in each VM's entropy time series. The comparison between Entropy I and Threshold I, for example, indicates that EbAT's metric distribution-based detection aggregating metrics across multiple vertical levels has advantages over solely looking at host level, even when compared to an oracle detector.

VII. Discussion: Using Hadoop Applications

Hadoop [4] is an implementation of the MapReduce framework [20] supporting distributed computing on large data sets. This section uses Hadoop to explore and illustrate select issues in using EbAT to monitor complex, large-scale cloud applications.

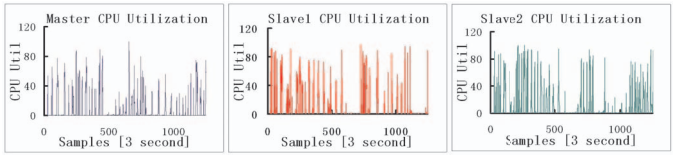

Fig. 7. CPU utilizations of master VM, slave1 VM and slave2 VM

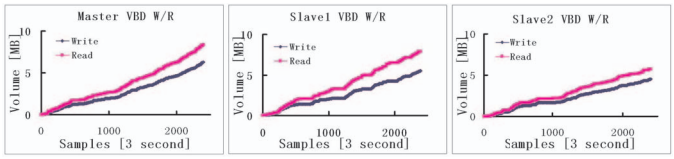

Fig. 8. VBD-write and VBD-read in master, slave1 and slave2 VMs

We deploy a Hadoop platform on three virtual machines named $master$, $slave1$ and $slave2$. Those VMs are hosted in one physical server with the same hardware configurations as in the RUBiS experiment. Master acts as the master node in MapReduce and also operates as a worker (i.e., running a worker process). Slave1 and slave2 are worker nodes. We run a distributed word counter application on the platform for 2 hours with 6 anomalies caused by application level task failures. EbAT observes CPU utilization and the number of VBD-writes and VBD-reads (virtual block device writes and reads), and calculates their entropy time series.

We first calculate entropies based on the CPU utilizations of the three VMs and aggregate those entropies as the host entropy. The parameters, look-back window size, range and bin number are presented in Table IV. Figure 7 shows each VM's CPU utilization trace. CPU utilizations are high when the associated VMs have tasks to run, but approach 0 in idle state. As a result, the entropy of the host, which is the aggregation of the three VM's entropies, exhibits high variability due to the high variations in raw metric values as seen in Figure 10(b). This demonstrates that it is not suitable to apply EbAT to metrics whose 'normal' behavior is highly variable. Such variations are simply transferred to the entropy trace, thereby affecting detection accuracy.

A more appropriate way of using EbAT with Hadoop applications is one in which its methods are applied to a derived metric. Specifically and as shown in Figure 8, although the VBD-write and VBD-read metrics keep increasing (will affect EbAT anomaly detection as mentioned above), they are highly correlated, as intuitively obvious from the fact that data intensive codes implemented with Hadoop attain high performance by best using the aggregate disk bandwidth of the underlying machines on which they run. Thus, when calculating their correlation value traces as depicted in Figure 9, these correlations are stable most of the time, but have sharp decreases when anomalies occur. By applying EbAT calculations on correlation traces instead of raw metric traces, with the same parameters as those used for CPU utilization, we then attain the detection results presented in Figure 10(a). Here, six spikes in the whole host entropy time series reflect the six anomalies during the experiment period.

TABLE V
EXPERIMENT RESULTS

Methods	Description	# of Alarms	# of Successful Detections	Recall	Precision	Accuracy(F_1)	FAR
Entropy I	Global Entropy (using Eq. 3 - Child M-Events)	45	43	0.86	0.96	0.91	0.04
Entropy II	Global Entropy (using Eq. 4 - E1+E2+E3+E1*E2*E3)	56	45	0.90	0.80	0.85	0.20
Threshold I	Using oracle-based method (Near-optimum thresholds)	46	33	0.66	0.72	0.69	0.28
Threshold II	Static Thresholds: $> 90\%$ or $< 5\%$	18	16	0.32	0.89	0.47	0.11

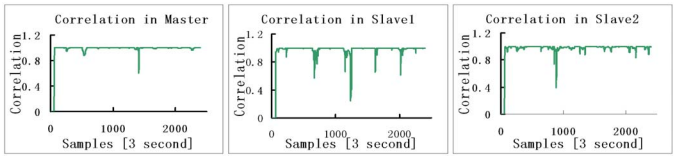

Fig. 9. Correlations of VBD-write and VBD-read in master, slave1 and slave2 VMs

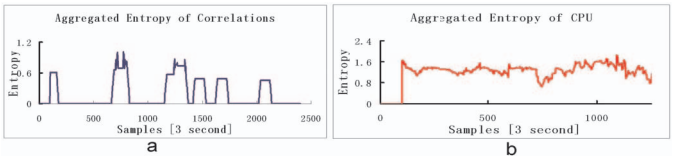

Fig. 10. (a) Aggregated correlation entropies, and (b) CPU utilization entropy of the physical server hosting master, slave1 and slave2 VMs

The above being a preliminarily empirical study on Hadoop use case, we are continuing to explore more sophisticated clustering techniques to find correlated metrics and then track multiple entropies using EbAT on those correlation values.

VIII. CONCLUSIONS AND FUTURE WORK

EbAT is an automated online detection framework for anomaly identification and tracking in data center systems. It does not require human intervention or use predefined anomaly models/rules. To deal with the complexity and scale of monitoring, EbAT uses efficient m-events to aggregate different levels of metrics in clouds, leverages entropy-based metric distributions, time series diagnosis methods to detect anomalies at runtime.

Future work concerning EbAT includes (1) Zoom in detection to focus on possible areas of causes, (2) extending and evaluating the methods for cross-stack (multiple) metrics, (3) evaluating scalability with large volumes of data and numbers of machines, and (4) continued evaluation with representative cloud workloads such as Hadoop.

IX. ACKNOWLEDGEMENTS

We are grateful to Lakshminarayan Choudur, Vinay Deolalikar, Kivanc Ozonat, Rob Schreiber, Ram Swaminathan, and Niraj Tolia for providing insightful comments and inputs to the idea of the paper. We also thank the anonymous reviewers for their constructive suggestions to improve the quality of the paper. We thank Matthew Wolf, Ada Gavrilovska and Greg Eisenhauer for helping initiate the project and guidance. We thank Mukil Kesavan, Adit Ranadive and Renuka Apte for creating the benchmark VMs and constructing the experiment testbeds. We thank Mahendra Kutare for developing the monitoring substrate.

REFERENCES

[1] Amazon ec2 website. `http://aws.amazon.com/ec2/`.
[2] f_1 score in wikipedia. `http://en.wikipedia.org/wiki/F-score`.
[3] Ganglia website. `http://ganglia.info/`.
[4] Hadoop website. `http://hadoop.apache.org/`.
[5] Hp systems insight manager. `http://www.hp.com/go/sim`.
[6] Ibm tivoli. `http://www.ibm.com/tivoli`.
[7] Nagios website. `http://www.nagios.org/`.
[8] Nimsoft website. `http://www.nimsoft.com`.
[9] precision and recall in wikipedia. `http://en.wikipedia.org/wiki/Precisionandrecall`.
[10] S. Agarwala and et. al. E2eprof: Automated end-to-end performance management for enterprise systems. In *DSN*, 2007.
[11] S. Agarwala and K. Schwan. Sysprof: Online distributed behavior diagnosis through fine-grain system monitoring. In *ICDCS*, 2006.
[12] M. K. Aguilera and et. al. Performance debugging for distributed systems of black boxes. In *SOSP '03*, 2003.
[13] A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau. Information and control in gray-box systems. In *SOSP*, pages 43–56, 2001.
[14] P. Bahl and et. al. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *SIGCOMM*, 2007.
[15] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using magpie for request extraction and workload modelling. In *OSDI'04*, 2004.
[16] E. Cecchet, J. Marguerite, and W. Zwaenepoel. Performance and scalability of ejb applications. In *OOPSLA*, 2002.
[17] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. In *ACM Computing Surveys*, 2009.
[18] M. Y. Chen and et. al. Pinpoint: Problem determination in large, dynamic internet services. In *DSN*, 2002.
[19] I. Cohen and et. al. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *OSDI*, 2004.
[20] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.
[21] V. Kumar and et. al. imanage: policy-driven self-management for enterprise-scale systems. In *Middleware*, 2007.
[22] V. Kumar, K. Schwan, S. Iyer, Y. Chen, and A. Sahai. A state-space approach to sla based management. In *NOMS*, 2008.
[23] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *SIGCOMM*, 2005.
[24] A. Lakhina and et. al. Diagnosing network-wide traffic anomalies. In *SIGCOMM*, 2004.
[25] J. A. Nugent, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Controlling your place in the file system with gray-box techniques. In *USENIX Annual Technical Conference, General Track*, pages 311–323, 2003.
[26] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do internet services fail, and what can be done about it? In *USITS*, 2003.
[27] S. Pertet and P. Narasimhan. Causes of failure in web applications. Technical report, CMU-PDL-05-109, December, 2005.
[28] C. J. v. Rijsbergen. *Information Retrieval*.
[29] C. E. Shannon. A mathematical theory of communication. *Mobile Computing and Communications Review*, 5(1):3–55, 2001.