Journal of Universal Computer Science, vol. 25, no. 8 (2019), 925-945
submitted: 26/12/18, accepted: 6/6/19, appeared: 28/8/19 © J.UCS

On Machine Learning Approaches for Automated Log Management

Ashot N. Harutyunyan
(Office of CTO of Cloud Management, VMware, Yerevan, Armenia
aharutyunyan@vmware.com)

Arnak V. Poghosyan
(Office of CTO of Cloud Management, VMware, Yerevan, Armenia
apoghosyan@vmware.com)

Naira M. Grigoryan
(Office of CTO of Cloud Management, VMware, Yerevan, Armenia
ngrigoryan@vmware.com)

Narek A. Hovhannisyan
(Wavefront by VMware, Yerevan, Armenia
nhovhannisya@vmware.com)

Nicholas Kushmerick
(Office of CTO of Cloud Management, VMware, Seattle, USA
nicholask@vmware.com)

Abstract. We address several problems in intelligent log management of distributed cloud computing applications and their machine learning solutions. Those problems concern various tasks on characterizing data center states from logs, as well as from related or other quantitative metrics (time series), such as anomaly and change detection, identification of baseline models, impact quantification of abnormalities, and classification of incidents. These are highly required jobs to be performed by today's enterprise-grade cloud management solutions. We describe several approaches and algorithms that are validated to be effective in an automated log analytics combined with analytics from time series perspectives. The paper introduces novel concepts, approaches, and algorithms for feasible log-plus-metric-based management of data center applications in the context of integration of relevant technology products in the market.

Keywords: Cloud computing, distributed systems, automated log management, time series, anomaly detection, change detection, forecasting, state characterization, baseline model, sampling with confidence control, binomial distribution, clustering, machine learning.
Categories: C.3, C.4, E.0.

1 Introduction

Cloud management solutions provide an effective control over data centers only through continuous and granular monitoring of time series indicators and logs of those complex environments. Diagnosing IT systems from their logs is subject to many studies. The papers [He et al 2016], [Lin et al 2016], [Jia et al 2017], [Ambre et al

2015], [Hu et al 2017] (see also references therein) are targeting automated log analysis from different perspectives. Anomaly and change detection are major topics in those studies (see also [Harutyunyan et al 2018], [Harutyunyan et al 2014], [Brown and Kushmerick, 2015]). We share our experiences while dealing with other relevant problems in automated log management. In particular, we discuss methods for learning baseline models of log streams, classifying data center incidents using log data, quantifying impact of issues based on log content, etc. We describe our approaches to these problems using machine learning techniques.

Leveraging cloud management products to effectively control performance of IT applications and infrastructures inevitably leads to the issue of automatically identifying baseline structures (typical behavioral patterns) of measured data sets including log sources. Those structures can be utilized for a variety of purposes, from anomaly and change detection to characterization of the application or infrastructure state in large. For instance, high/low stress levels, sickness, overprovisioning, security threats, etc. Particularly, VMware vRealize Operations Manager [vR Ops, 2019] performs such an analysis for any time series metric of an IT object through its basic dynamic thresholding analytics [Marvasti et al 2014] and forecasting of capacity indicators, while building a similar capability for log analytics is challenging - the very high volume of log data makes machine learning extremely expensive. To overcome the learning complexity, we propose a random sampling technique based on the binomial distribution. Our method allows for controlling the confidence of the learned model by tuning the sampling rate.

With growing interest in the industry in application-aware cloud management and analytics, the log intelligence becomes especially important. The above-mentioned cloud management solutions are enterprise platforms to empower the modern Software-Defined Data Center (SDDC) management with automated machine learning capabilities, self-tuning and optimization, to further evolve into AI-enabled autonomous solutions in cloud computing. vRealize Log Insight (LI) [Log Insight, 2019] supports two important machine learning features of 1) Event Types as similarity clusters of raw log data and the 2) Event Trends allowing to compare two selected time windows by their differences of corresponding event types. While vR Ops performs pattern detection for any metric data from data center objects and derives expected ranges of processes based on a complex time series analysis, accounting for change, trends, and periodicity, LI is lacking a similar capability to automatically identify the main behavioral patterns of the log source. It makes troubleshooting and pattern detection in log data mostly a query-based task with intensive user efforts to find problem root causes or track the application state in general.

In this regard, to have a deeper characterization of the application, identification of its *baseline model* or behavioral fingerprint from logs history is of exceptional importance. Intelligent proactive management of data centers and applications from logs perspective with building an accurate expert baseline requires a huge knowledgebase and extensive efforts. At the same time, it cannot be easily generalizable because of many factors coming from conditions of the IT ecosystem. Hence, machine identification of application baselines can be a powerful addition to any log analytics. Evidently, with such a fundamental structure then, the real-time anomaly detection and many other core tasks will be easily automated. By comparing the current log stream against its historically typical model, we'll be able to effectively

describe the state of the application in real-time and efficiently identify issues and incidents. Those can be new software bugs, sickness conditions, hardware failures, software upgrades, configuration changes, changes in workload. They all are related to various aspects of data center management (troubleshooting, performance monitoring, capacity planning, provisioning and configuration, compliance auditing, policy enforcement, etc.) This means that those structures should be enough informative to reveal the whole complexity of the log stream with sophisticated relationships between events. Any method dealing with extracting and continuously updating baseline structures along the log stream requires an expensive unsupervised learning plan. Although alternative approaches applying meta-data analysis or quantification of log information bypass such a complexity (discussed in Section 2), however they address only a specific problem without structural characterization of the log source in general.

In this paper, we focus on learning the baseline model of log sources in terms of the distribution of log event types generated by LI. Moreover, our algorithms identify the expected *normal discrepancy* from such a baseline that the log source exhibits. We demonstrate the proposed approach by applying our prototype algorithms to data measured from controlled experiments.

Although the task of learning baselines is central to the paper, we focus also on the other analytical tasks mentioned above. Sections 4-6 are devoted to each of those frameworks tackling the automated log management from various perspectives with different levels of complexity and sophistication. In Section 2, we motivate our research and discuss the related work. Section 3 describes our methods to identifying baselines of log sources and also demonstrateing their application to log data sets. In Section 7, we conclude the paper, mention about future works and larger experimental plans.

2 Motivation and Related Work

As we mentioned in the introduction, one approach to overcome the complex machine learning tasks for log data is to extract different meta-data properties from those sets and proceed with numeric data (time series) analysis (e.g. [Marvasti et al 2014]) or build event correlation models (e.g. [Harutyunyan et al 2014]). In particular, in our earlier work [Harutyunyan et al 2018], we applied quantification of log data with information theory [Cover and Thomas (1991)]. The quantified/extracted time series metric representing stream's Jensen-Shannon divergence over time was analyzed for change detection purposes. Although this kind of metric plus log analytics framework empowers the log intelligence with highly effective toolset of low complexity, but it remains an indirect method for behavioral analysis (without revealing the complete characteristics of the log source and hiding much of the content in logs).

In [Harutyunyan et al 2018], we intensively utilized distributions of event types generated by LI in change point detection for a single source, sickness detection of a source within population of similar sources, as well as for an application topology discovery using hierarchical clustering. Earlier works by [Ambre et al 2015] and [Hu et al 2017] analyze specific change patterns related to security of applications. Below, we are going to utilize LI's event types further to identify the sought baseline models for log sources.

Importance of universal baseline models for log analytics was first realized in [Brown and Kushmerick, 2015], where authors applied information divergence measures to detect anomalies subject to a known/assumed baseline distribution of event types. The work was largely motivational for us to address the problem of automatic discovery of baseline distributions of log events.

For a short overview on Event Types (ET) by LI, let us mention that they are the main machine learning constructs of the product that represent abstract clusters of raw log events into similarity groups. With such a similarity grouping the product performs a dramatic data reduction, mapping thousands or millions of log messages into a manageable number of groups/types. Fig. 1 illustrates log data of a source for a 10-minute period as a bar chart of events of distinct types (in different colors). It highlights those distinct groups in a fractional view within each bar of the chart. Those fractions/rates in each bar (10 seconds) of the chart can be converted into relative frequencies or probability distributions of ETs within the window. Then if we want to compare two log portions in terms of their content, we can apply information measures (or other similarity distances like *cosine*) to estimate their "difference".

In particular, taking relative frequencies of ETs observed in two log portions as probability distributions

$$P = (p_1, p_2, \dots, p_n) \text{ and } Q = (q_1, q_2, \dots, q_n)$$

of n different ETs, we applied Jensen-Shannon divergence varying between 0 and 1:

$$JSD(P, Q) = \frac{1}{2} D(P, M) + \frac{1}{2} D(Q, M),$$

where $M = \frac{P+Q}{2}$ and $D(P, Q)$ is the Kullback-Leibler divergence [Cover and Thomas (1991)] between P and Q:

$$D(P, Q) = \sum_{i=1}^{n} p_i \log \frac{p_i}{q_i}.$$

Respectively, the cosine similarity is based on the angle between two vectors:

$$\cos(\theta) = \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} {p_i}^2} \sqrt{\sum_{i=1}^{n} {q_i}^2}}.$$

For more details regarding ETs, their probabilistic representation and application of information measures to anomaly, change, and sickness detection, we refer the reader to the same papers [Harutyunyan et al 2018], [Brown and Kushmerick, 2015].

Based on the above review, the proposed machine learning identification of baseline structures for log sources is a novel formulation.

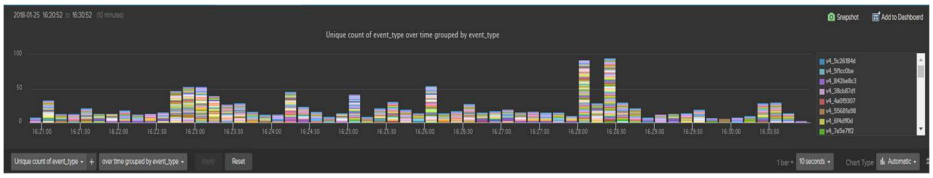

Figure 1: Bar chart representation of fractions of distinct event types by LI for each 10 seconds within a 10-minute window.

3 Identifying Baselines with ML

Our algorithms are based on random sampling employing the binomial probability distribution. This allows us to tune the confidence of our ML algorithms. Below, we give a brief information about the binomial distributions.

3.1 Binomial Distribution and Sampling

The binomial distribution with parameters n and p is the discrete probability distribution of the number of successes in a sequence of n independent experiments. The probability of number of success (Yes) k versus $n-k$ failures (No) in n trials is given by the formula:

$$Prob(k\ success\ in\ n) = C_n^k p^k (1-p)^{n-k},$$

where p is the probability of success in a single trial.

Let us assume that the log source stays at its normal operational state most (99%) of the time. Instead of 99% can be another prior probability. This means that if we randomly sample some log portions during the progress of the stream, we'll get mostly normal (i.e., the success outcome) behavioral patterns of the log source (let us say in terms of its ETs).

How many randomly sampled event type distributions are "enough" to verify if 99% of those distributions describe the normal mode of the log stream? The answer of the question can be given with the help of binomial distributions. Namely, if normal samples occur with probability 0.99 versus 0.01, then applying the binomial distribution (see the online calculator http://stattrek.com/online-calculator/binomial.aspx), we can measure how many sampled probability distributions of ETs are "enough" to identify the "normal" (success) ones. Fig. 2 shows the calculator in action.

Probability of success on a single trial	0.99
Number of trials	5
Number of success	4
Binomial Probability: P(X = 4)	0.0480298005
Cumulative Probability: P(X < 4)	0.0009801496
Cumulative Probability: P(X <= 4)	0.0490099501
Cumulative Probability: P(X > 4)	0.9509900499
Cumulative Probability: P(X >= 4)	0.9990198504

Figure 2: Results of the binomial distribution calculation for 5 trials with 4 "success".

The table in Fig. 2 illustrates that from 5 distributions at least 4 are the normal patterns with confidence (cumulative probability) equal to 0.999. The number of necessary samples will evidently grow if we assume lesser probability of success, while guaranteeing the same level of confidence in our experiment/trial. In general, for a large number of samples collected, say, 10,000, we need to calculate the relevant quantiles of the binomial distribution https://keisan.casio.com/exec/system/1180573200. So, for the cumulative distribution equal to 0.999, with number of trials equal to 10,000 and

the probability of success to be 0.99, we expect at least 9,868 sampled distributions representing the normal state of the log source which should be identified.

3.2 Algorithms and Experiments

As we mentioned in Section 1 and 2, when performing anomaly detection and other important tasks for a log source using LI, we face the problem of having a baseline model for the source as a typical characteristic of its historically normal behavior. More specifically, if the streams' current distribution of ETs is largely deviating (as a matter of a distance measure) from the baseline, we can automatically raise an alert to the system administrator. We have already shown in [Harutyunyan et al 2018] that in tasks such as anomaly, change, and sickness detection, the ETs are invaluable "signatures" or "fingerprints" of log sources to rely on.

In this subsection, we describe our ML algorithms implemented in Python for identification of that baseline structure using LI's ETs with random sampling. We describe two methods to perform such a learning task. The first method applies random sampling of log messages with confidence control of the inference. The second algorithm indicates the most generic and sophisticated solution to the problem although with much higher complexity.

Method I (with random sampling). How to identify the baseline distribution with the sampled 5 distributions in the example in subsection 3.1? The next question is then how to identify those 4 dominant (in terms of characterizing the state of the source) distributions out of 5?

Our solutions below indicate how to choose the baseline event type distribution and the related *normal discrepancy radius* of the stream that quantifies the tolerable "distances" of the observed event type distributions from this baseline as still within the expected behavior:

1. compute cosine similarity distances between all pairs of event type distributions (histograms) derived for each of sampled log portion;
2. compute average cosine similarity distance (ACSD) for each sample histogram from the rest;
3. rank sampled histograms in decreasing order of their ACSDs and pick up the top 4 (tries to identify the most similar subset of 4 distributions.);
4. pick up the histogram with minimum ACSD as the baseline (centroid) distribution;
5. if there are several histograms with min ACSD, compute Shannon entropy of those and pick up the one with maximum entropy value as a baseline distribution.

In an alternative implementation, the step 4 can be replaced with the maximum entropy principle applied to the top distributions directly to identify the most unbiased baseline distribution.

Shannon entropy [Cover and Thomas (1991)] measures the uncertainty in a random variable defined by

$$H(P, a) = -\sum p_i \log_a p_i \leq \log_a n$$

and its binary version's plot is depicted in Fig. 3.

For a demonstration purposes, we performed a small experiment on an Apache server (consisting of web, email, and ftp services), a similar experiment described in [Harutyunyan et al 2018] for sickness detection task within a population of peers. We

emulated a stress or security attack (using ApacheBench test tool) on the web host with a high-rate service requests for a 5-minute duration, after observing it in a "normal" operational workload for half an hour. Then, we sampled 5 different five log portions of 5-minute length that captured the stressed window (Sample 2) as well. For each of log portions (Samples 1-5 shortened to S1-S5), we computed the probability distributions of observed 25 ETs within, which are shown in Table 1.

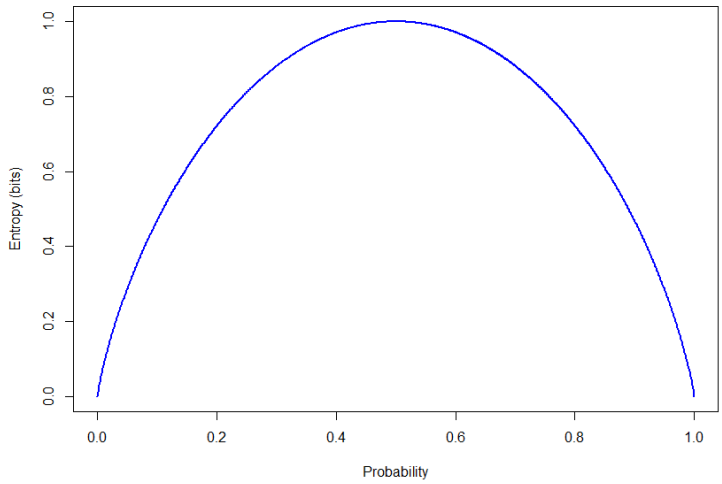

Figure 3: Shannon entropy function for binary distribution and log base $a = 2$ *with the maximum uncertainty at probability = 0.5.*

By ranking (step 4) samples/distributions in decreasing order of the distance measure, we pick up the following four having highest average similarity (this is the dominant similarity set representing the normal workload mode of the host):

$ACSD(S1) = 0.97, ACSD(S3) = 0.96, ACSD(S4) = 0.97, ACSD(S5) = 0.97.$

The chosen four distributions are the baseline "candidates". With this ranking, the anomalous Sample 2 indicated in red in Table I dropped from the "candidates" list.

Since there are three samples with the same score, we are going to identify the one with maximum uncertainty as the "safest" unbiased model for the baseline distribution. The entropies (in *nats*, $a = 10$) of sample distributions are:

$$H(S1) = 2.83,$$
$$H(S2) = 2.78,$$
$$H(S3) = 2.85,$$
$$H(S4) = 2.80,$$
$$H(S5) = 2.84.$$

Which is the best "candidate" for a baseline? Applying the maximum entropy principle, it is the distribution that has the highest information uncertainty. Sample 3

(green column in Table I) has the maximum entropy distribution (Fig. 4) and is chosen to be the baseline for the log source.

	Probabilities				
LI's Event Types	**S1**	**S2**	**S3**	**S4**	**S5**
v4_18ca9254	0.03	**0.02**	**0.03**	0.03	0.03
v4_1a6ac047	0	**0**	**0**	0	0
v4_28077ade	0	**0**	**0**	0	0
v4_2f6e41a2	0.03	**0.02**	**0.03**	0.03	0.03
v4_36c81ef6	0.05	**0.04**	**0.07**	0.05	0.06
v4_393c8071	0.2	**0.17**	**0.18**	0.21	0.19
v4_59cd0174	0.03	**0.02**	**0.03**	0.03	0.03
v4_681f6046	0.05	**0.04**	**0.07**	0.05	0.06
v4_69475cc1	0	**0.08**	**0**	0	0
v4_6dd466a5	0.03	**0.02**	**0.03**	0.03	0.03
v4_71183f87	0	**0**	**0**	0	0
v4_802bd0d4	0.11	**0.1**	**0.1**	0.12	0.11
v4_87e0ca23	0.03	**0.02**	**0.03**	0.03	0.03
v4_88de5e12	0.03	**0.02**	**0.03**	0.03	0.03
v4_8ebbb638	0.03	**0.02**	**0.03**	0.03	0.03
v4_94680e71	0.03	**0.02**	**0.03**	0.03	0.03
v4_9d3e7bdd	0.03	**0.02**	**0.03**	0.03	0.03
v4_9fd2eafd	0.04	**0.11**	**0.04**	0.03	0.04
v4_a7f56e13	0.06	**0.05**	**0.05**	0.06	0.06
v4_a8a71825	0.03	**0.02**	**0.03**	0.03	0.03
v4_b610f232	0.03	**0.02**	**0.03**	0.03	0.03
v4_b9100c8f	0.05	**0.04**	**0.07**	0.05	0.06
v4_bafd4270	0.03	**0.02**	**0.03**	0.03	0.03
v4_bfebb8d	0.05	**0.04**	**0.07**	0.05	0.06
v4_f0533255	0.03	**0.02**	**0.03**	0.03	0.03

Table I: Five samples of log event types taken from a host for a 5-minute time range each.

The final step is to derive the "normal discrepancy radius" of the baseline/source. This is the variance in ACSD that we observe in the top similarity subset of 4.

The *normal discrepancy radius* (R) then can be defined as the range of ACSD's in the "dominant" similarity set. In our experimental example, it is the following difference:

$$R = 0.97 - 0.96 = 0.01.$$

Then, computing the ACSDs for each distribution from the rest, we get:

$$ACSD(S1) = 0.97, ACSD(S2) = 0.89,$$
$$ACSD(S3) = 0.96, ACSD(S4) = 0.97, ACSD(S5) = 0.97.$$

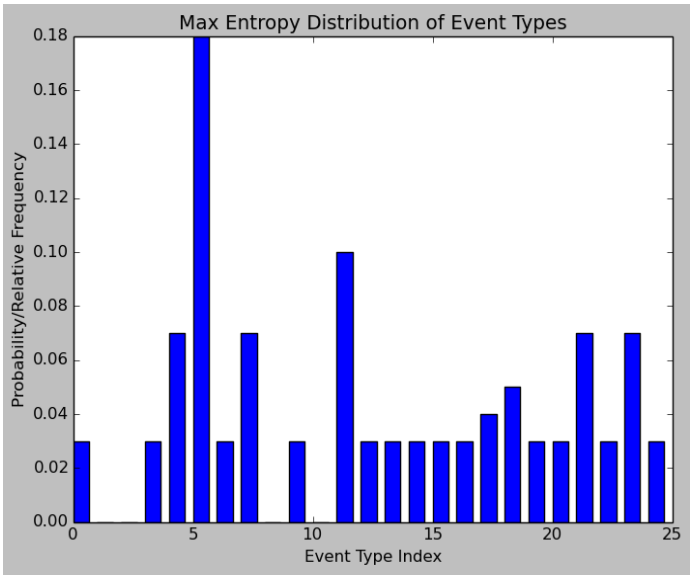

Figure 4: Maximum entropy distribution as a learned baseline.

Method II (clustering of event type distributions without random sampling). In an alternative, highly complex implementation, without randomized sampling, with continuously measured and stored probability distributions/histograms of ETs that contain all normal and abnormal patterns in the log stream, our algorithm performs the following steps:

1. calculate local outlier factors of all histograms using LOF algorithm [Breunig et al 2000];
2. pick up the "centroid" event type distribution as the histogram having the smallest LOF;
3. if those are several, the one that has the highest entropy;
4. compute the average and standard deviation of distances of all histograms from the centroid;
5. define the "normal discrepancy radius" of the log source from its baseline with 3-sigma rule, assuming that distances are distributed normally.

LOF is based on a concept of a local density (or similarity distance in our case), locality defined by k nearest (similar) neighbors. Those neighbors are employed to estimate the density. Based on this evaluation, outliers are detected as those distributions that have substantially lower density than their neighbors. The local density is estimated by the distance at which a point can be "reached" from its neighbors [Breunig et al 2000]. In a simpler implementation, the centroid can be the histogram having the minimum average distance from the rest of the histograms. The normalcy

radius can be also linked to Chebyshev's inequality without making the assumption of normality.

Fig. 5 pictorially supports the main ideas of Methods I and II.

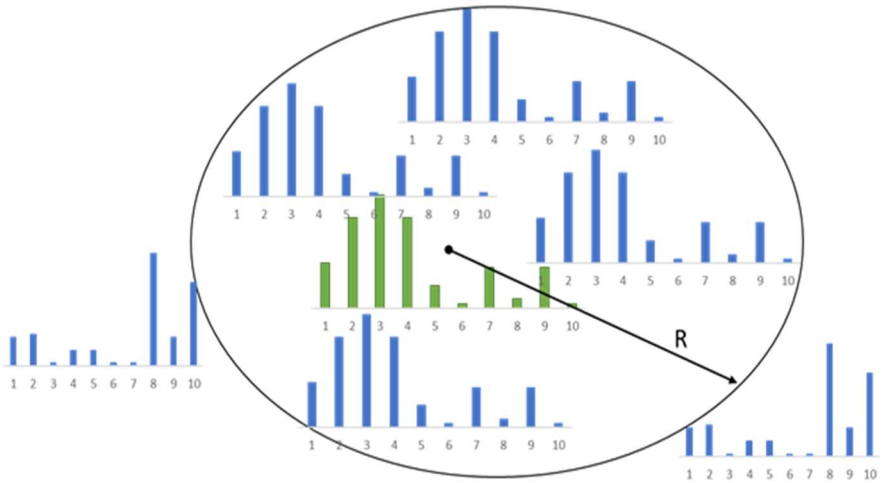

Figure 5: Illustrating Baseline as a Centroid Distribution of the cluster and its Normal Discrepancy Radius.

Abnormality degree of baseline violation. We can measure the abnormality degree or criticality of alerts raised using the baseline. It is defined by how far is the run-time ET distribution from the centroid (excluding the discrepancy radius). So, if the normal operations are within 0.1-radius, a run-time distribution having distance=0.4 exceeds the tolerance by 0.3. Then we can use a scale to map those over-tolerance distances into a score range from "lowest" to "highest" (continuous or discrete). Alternatively, measure and communicate to the user how many times or by which percent the tolerance radius is exceeded.

It can happen that the event source operates in different modes and have different baseline characteristics accordingly (for example, the corresponding IT resource has high and low utilization modes). In such cases, k-means clustering can be applied to derive relevant clusters and their centroids, then apply the above-mentioned normality assumption or Chebyshev inequality to extract the normal discrepancy radius for each of the clusters. This means that for the corresponding anomaly detection we identify in which mode the system operates and apply the relevant centroid baseline.

We conducted another controlled experiment as in [Harutyunyan et al 2018], again choosing vR LI as our proof-of-concept application. We monitored LI's logs in INFO and DEBUG modes using another LI instance. This is a way to observe the application in two different stress modes (high and low). In those logging modes, our algorithm implemented in Python observes different event type distributions. Representative distributions are depicted in Figs 6 and 7, respectively.

The graphs illustrate different number and rates of ETs. Therefore, having detected possible workload modes of an application with complex clustering methods, then the sampling technique of Method I can be applied to derive the baseline for each mode individually.

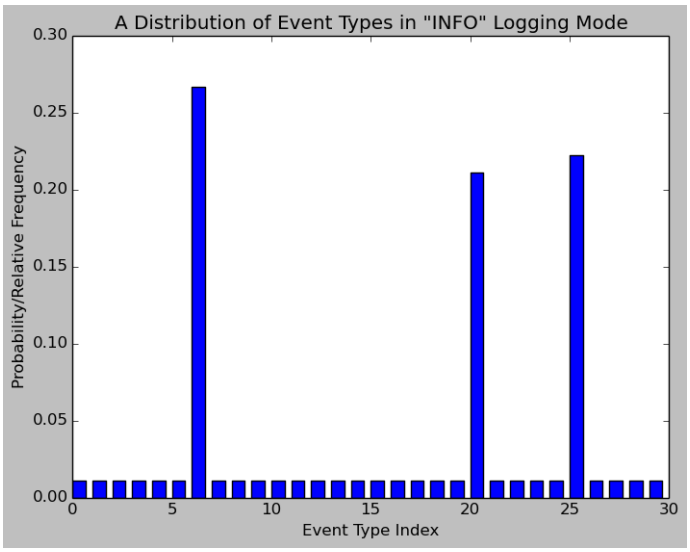

Figure 6: Representative distribution in INFO logging mode of LI.

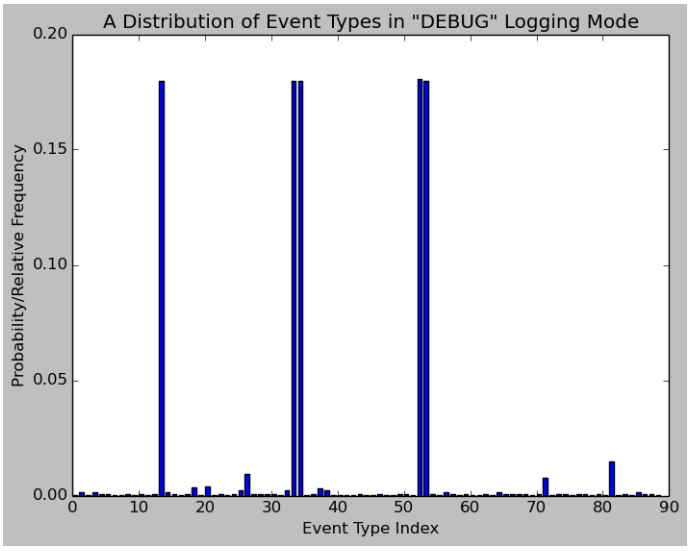

Figure 7: Representative distribution in DEBUG logging mode of LI.

To validate how effective is the learned baseline we had to focus on specific data center management tasks it can be leveraged for, rather than benchmarking it against either an existing expert-built model or a model obtained with similar unsupervised way of deduction. Both of those are hard to provide – in one case because of various and complex environment-specific factors and enormous human efforts, and of unavailable relevant prior art known to us in the other case. According to our experimental evaluations, the algorithms introduced here were enough indicative to capture the specific change and outlying behaviors in the log sources of interest.

4 Quantifying Log Content into Metric Indicators

As motivated in Section 2, one of our approaches to implement analytics for log data and alleviate the hard problem of learning from voluminous log events is through quantification of characteristics of the log source.

Uni-Variate case. First, we apply a "property extractor" for the log stream to quantify its parameters over time, such as *volume, velocity* (rate of growth) *and acceleration, variety* (number of different ETs), etc., which can be out-of-the-box meta data indicators. Those parameters are stored as time series data. That allows to leverage analytical modules designed for metric data [Marvasti et al 2014]. It means we can learn historical patterns of those properties and forecast their expected behavior. Out-of-normalcy states of properties are subject to *property-digression alerts.* Within a less-complex data analytics scenario, those property-digression alerts can be formed based on Extreme Value Theory (see [Poghosyan et al 2016]) not counting for periodic patterns in the time series.

The next natural step is to correlate those alerts with other violations (out-of-expected patterns) or events observed in the environment in a temporal "closeness". That helps in a faster identification of the root causes of system's deterioration from its historical profile. Furthermore, keeping track of historical co-occurrences of property digression alerts with other anomalies in the system is an alternative way to measure their correlations to recommend the user. The list of recommended high co-occurrence anomalies currently active or expectedly appearing makes the user planning of the best remediation strategies of the source much easier, since it accounts for long-term correlation patterns of those alerts. Some of the co-occurrence patterns might indicate bottleneck issues in the system permanently impacting its performance.

Entropy is an indicative property defined on log ET's. Let we measure the entropy of distribution of ET's of a source at every time stamp t. Aggregating the entropy property in a time series data allows us analyzing it with the above-mentioned time series analytics. Thus, we can observe *uncertainty increase/decrease* patterns which might indicate different issues in the system. An automatic workload placement decision can be made for a VM to reside under a host with a "stable" entropy.

Fig. 8 demonstrates a computed property of entropy (not normalized) with a spike for a host and a short period of time.

Multi-variate case. Property indicators can be aggregated into a multi-variate representation to analyze the event source behavior using unsupervised learning. In other words, the values of k different properties at time stamp t

$$P_1(t), P_2(t), \dots, P_L(t)$$

make a point in *L*-dimensional space (let those points be measured for the past monitoring time stamps over several weeks/days/hours). It is an interesting perspective to evaluate the source performance in terms of the trade-off of those properties by analyzing the vector of values. In particular, the stand-alone properties may stay within their historical typicality but be in a trade-off conflict with each other. Therefore, using clustering algorithms (such as Local Outlier Factor algorithm [Breunig et al 2000], DBSCAN [Ester et al 1996], etc.) we detect anomalous trade-off states of the source at run-time as vector values deviating from the dominating cluster.

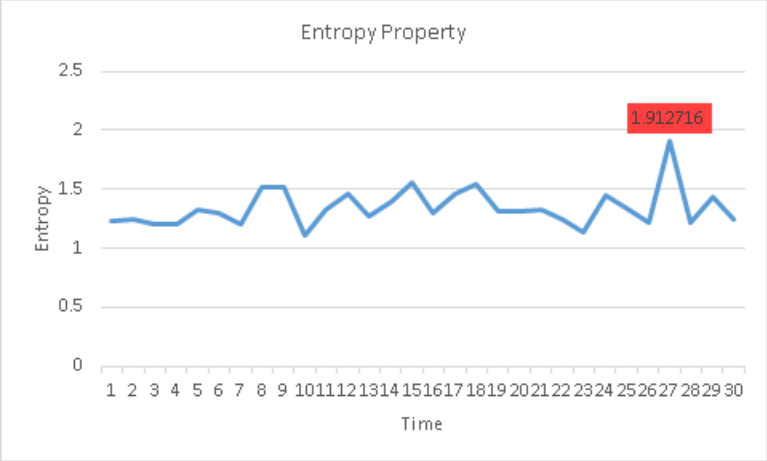

Figure 8: Time series representing entropy property for an event source/host.

Based on the clustering results the enhanced analytics generates alerts that point out trade-off breaches between the meta-data indicators.

2D and 3D meta-data analysis are the most reasonable variants of the generic idea to consider. Fig. 9 demonstrates a volume vs variety analysis of a log source with an outlying (volume, variety) point far away from the historically dominant cluster – although the volume of logs has decreased, the variety (number of different ET's) has significantly increased.

This ML approach can be applied to store all historical anomaly patterns in multi-variate property analysis with appropriate tags/labels on their context which can then be used to correlate newly observed outlying patterns with for root causing purposes (see also Section 6).

Quantification of properties of log sources and pattern detection on those within a "metrics-logs-events" integrated management substantially increases holistic visibility into the data center and its characterization accuracy. That also enables more proactive control over those systems and faster troubleshooting. Use case scenarios for application of our approaches are many. We are not going to focus on evaluation results for a specific problem to solve using the above-mentioned principles and ML algorithms but note that the example on a breakage of the trade-off between Volume vs Variety was indicative to detect invisible problems otherwise. At the same time, the classical ML algorithms (LOF/DBSCAN) for outlier detection, as well as our

proprietary engine [Marvasti et al 2014] for analyzing any time series metric/property are highly effective in identifying atypical behaviors subject to parameter tuning and thus largely automating data center operations.

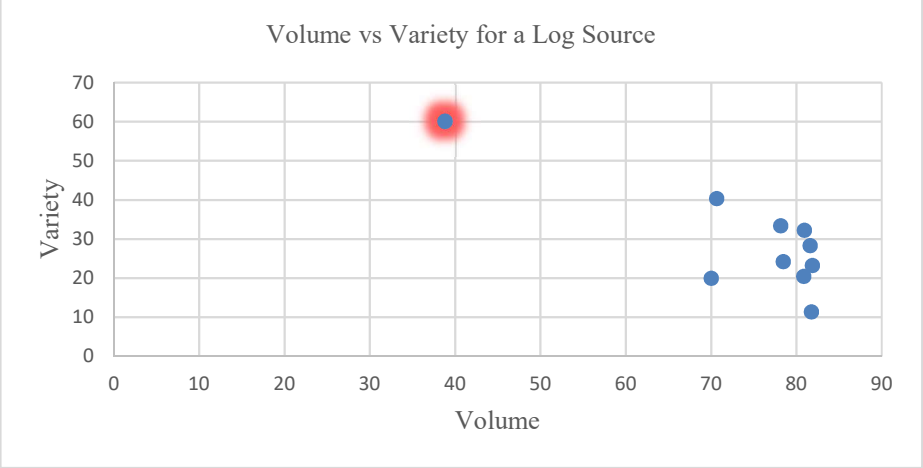

Figure 9: Multi-variate analysis of a log source by events volume vs variety values.

5 Quantifying Problem Impacts using Log Structures

Cloud management solutions notify the users/admins of the infrastructures and applications on performance and capacity anomalies as well as compliance/configuration issues in the system. Design and appropriate management of alerts that react to unwanted patterns in diverse systems is a hard process. Cloud ops/log management platforms can have multitude of alerts with implications of noise, alert fatigue, etc. Prioritization of alerts for optimal troubleshooting of the infrastructure is a need for the users. It is not an easy task to prioritize those alerts for troubleshooting purposes, as well as estimate their indicative power in terms of real implications for the system or environment. Moreover, the same alert can have different indicative value in different environments. It is especially hard to measure the impact of alerts which are triggered based on behaviors of different time series metrics of the application/infrastructure and relevant IT objects. Therefore, having a more objective way of measuring alerts impact on the system might greatly reduce the noise they produce, optimize their indicative power, thus maximize their effectiveness. The simplistic algorithms below demonstrate an effective way of identifying criticality of an alert condition measured by change magnitude in the log space.

Algorithmic notes. We quantify Alert's impact by the change it introduces into the system. For this kind of approach, the log data is of pivotal importance. In particular, the ETs of log messages remain important for such a task. Our method is generic in nature and addresses the following three problems:

1. *Categorization of alerts per global and local impact* generation ability;

2. Estimation of run-time *impact of an alert or its rank* (categorized per global and local classes): is measured by the change it provokes in the system that can be interpreted also as abnormality degree of the system;
3. *Historic impact factor of an alert*: as the average change level (per item 1) observed for the same alert in the past.

To perform such an analysis, we quantify the alert impact (real importance for infrastructure or admin) based on the change it actually reflects. Steps of the algorithm to perform such a quantification for change are:

1. Take a snapshot of log messages for a pre-alert and post-alert time intervals;
2. Compute relative frequencies or probability distributions of corresponding ETs (or a specific field values) for those two intervals;
3. Apply distance measures between two probability distributions that shows how distinct are log patterns for pre- and post-alert periods and assign to the alert as a run-time impact factor.
4. Alerts showing "low" distance between pre- and post-alert periods are categorized as having no global impact on the system, and subject to inspection for a local impact;
5. Alerts showing "high" distance (higher than during normal operations) between pre- and post-alert periods are categorized as having global impact on the system.
6. Rank alerts having global impact according to their impact factors in item 3.
7. Rank alerts according to their historical impact factors as an average (including weighting with time importance) of all run-time impacts observed in the past.

The above-mentioned steps can be particularized with additional filters for the snapshots of log messages (like event sources, e.g., hosts, app-names, etc.).

Inspecting local impacts. As mentioned above, alerts having no global impact are further checked for their local impact. In this case, attributes of an alert definition are used to filter those fields (and their values) of log messages that are responsible for the alert occurrence. Then the above-mentioned algorithm is applied to those field values instead of ETs in items 1-3. The rest of the steps 6 and 7 (omitting 4 and 5) in the algorithm are analogously applied to rank locally impacting alerts run-time and historic-based.

Below, we bring two examples of alerts with high local and global impacts, respectively.

Example 1. *Alert defined as a condition on metric data and having a global impact.* For a simulated DDOS attack scenario on a server, an alert is triggered on the number of messages logged from the system using time series analytics. We want to measure how this anomaly impacted the environment (for the time range correlated with the alert's trigger time) in overall from the corresponding log space perspectives. Fig. 10 and Fig. 11 depict those ET distributions measured right before and immediately after the alert was triggered, respectively.

Before the attack on the server, the "difference" between event type distributions was negligible, but as soon as the alert on "unusually high volume of log messages" popped up, we observe a significant change in the system in terms of Jensen-Shannon

Distance=0.307. The user might conclude that this alert implies a global impact on the system.

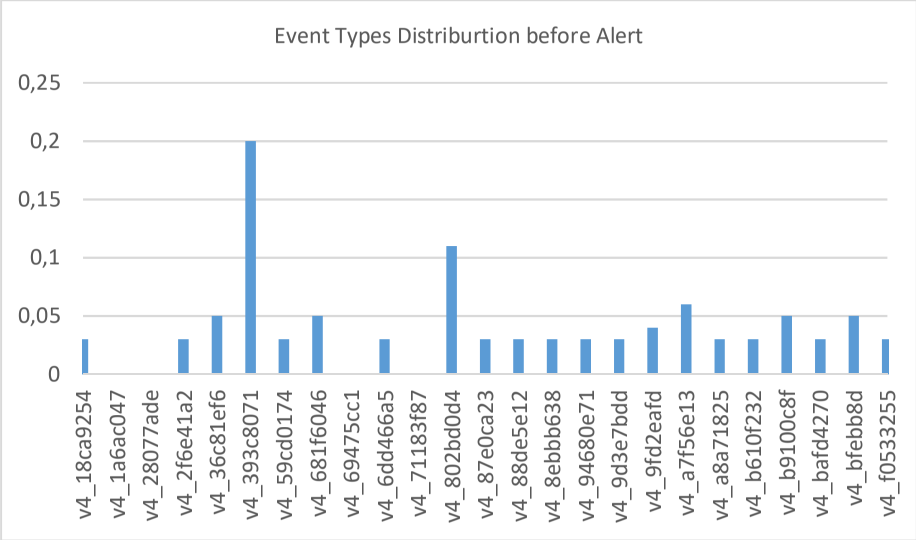

Figure 10: Distribution of event types before alert.

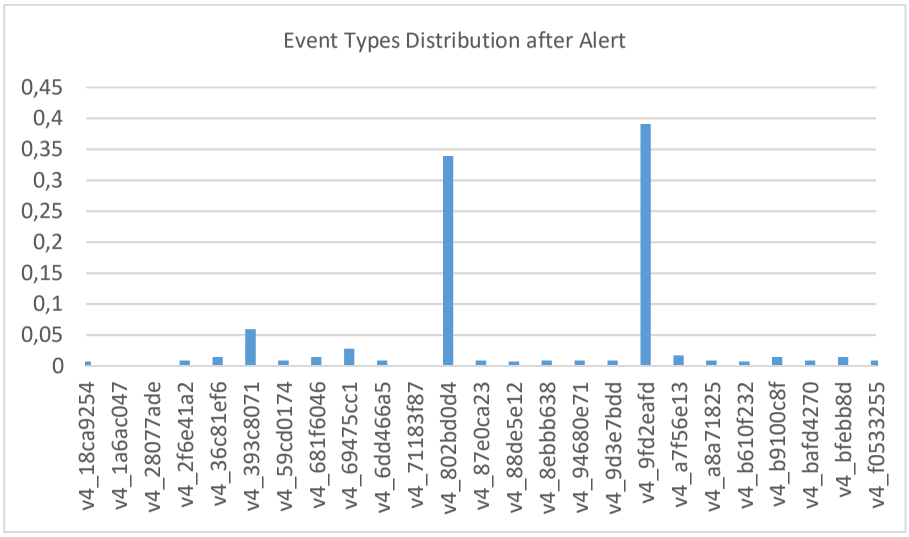

Figure 11: Distribution of event types when alert is sent.

Example 2. *Security alert in LI impacting locally (field-level granularity)*. Assume we are interested in an event that indicates a security issue at the web service (one of services at the VCI – virtual computing instance: physical server, VM, container), and configure the corresponding alert based on httpd status codes count (20x, 30x, 401, 404) to be high. Our algorithm takes probes of the corresponding field data immediately before the alert's occurrence and after it and computes relative frequencies of the status codes interpreted as probabilities (Table II). The change between pre- and post-patterns of the log field computed by Jensen-Shannon Distance is 0.316, which is a big shift compared to two consecutive normal operations patterns (where the same distance was very small/negligible).

At normal operations		At Alert time (after 5 min of normal operations)	
Status code	Probability	Status code	Probability
20x	0.80	20x	0.22
30x	0.05	30x	0.03
401	0.05	401	0.55
404	0.10	404	0.20

Table II: Httpd status codes distribution before and after the alert.

6 Classifying Problems based on Log Structures

For a rapid troubleshooting of performance incidents in data centers, it is crucially important to rightly characterize the nature of the problem. Managing IT systems from measured time series perspectives, it is hard to identify the nature of an alert that indicates an out-of-normal operations/states (or out of hard or dynamic thresholds) of one or several processes. Two aberrations in the same flow (say application response time, which is a Key Performance Indicator (KPI)) at different time instances can indicate totally distinct problems in nature. To effectively characterize an incident, we need to fully utilize the knowledge and experience of admins from the already occurred problems. This is not realistic for increasingly huge and complex infrastructures and applications requiring highly automated recommendations and solutions.

Integrating the system's log data into the metric management plate empowers us with such an automation solution. Namely, statistically characterizing the system's log content of historic incidents and using them as training data, we can then classify a real-time incident into one of problem types (which is tagged with an indicative description hinting the remediation solution). In such a way, we perform an automated "management with memory". This is a typical supervised learning or multiple hypothesis testing task.

The method below trains a ML model on ET distributions (can be regarded as an artificial labeling problem) for the data center application in order to categorize incidents identified in the metric space.

Algorithmic notes. Our solution relies on integration of two management platforms (metrics and logs). ETs greatly help in implementing the above-mentioned classification solution.

Assume we already have a history of incidents detected by an aberrative/anomaly behavior of a relevant KPI (Fig. 12). Let the corresponding log data feature be extracted for each of those incidents (for a selected time window) and attached to its descriptions. Then, for every occurring incident, we need to perform an appropriate classification of the related log sample into the existing classes of problems using a closeness distance. For instance, if ET distribution is the log characteristic, then cosine similarity distance between two vectors can be chosen.

This will be the most generic solution with storing all extracted log feature data, the set of probability distributions of ETs for each incident, so we have the representation in Fig. 13. It pictorially demonstrates the main concept behind the algorithm:

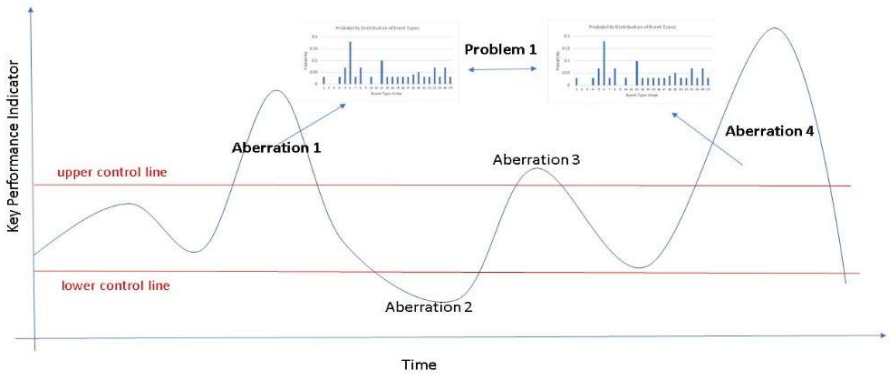

Figure 12: Different aberrations in the KPI (time series). For each aberration/ incident, the corresponding log data structure is extracted and stored. Aberration 4 is similar to Aberration 1 and they might indicate similar problems of the same type.

1. Over time, we have observed different incidents/aberrations in a KPI of an IT service (for instance, Aberrations 1 to 3 as our historic patterns);
2. Each aberration is represented by relevant log data feature (distribution of ETs) for an actual time window;
3. When facing Aberration 4, we take its log data feature and compare it against the representative features of the past incidents and observe that it is a similar or identical problem with Aberration 1;
4. So, if the latter was caused by a low network bandwidth issue (a tag), then it turns out that for Aberration 4 we have that issue again and thus remediation becomes much more efficient.

A simplified alternative solution does not store the event type distributions for all incidents, but only an "average" representative – a task to identify a centroid distribution. Then based on those representative distributions of problems, we can perform a multiple hypothesis testing of a newly observed incident or aberration's pattern with maximum likelihood principle: voting for the Problem type which is maximally close to the observed distribution.

Effectiveness of this method was validated for the same security-related problem of Section 5 with correct association of new atypical pattern in the logs with historically occurred DDOS attack.

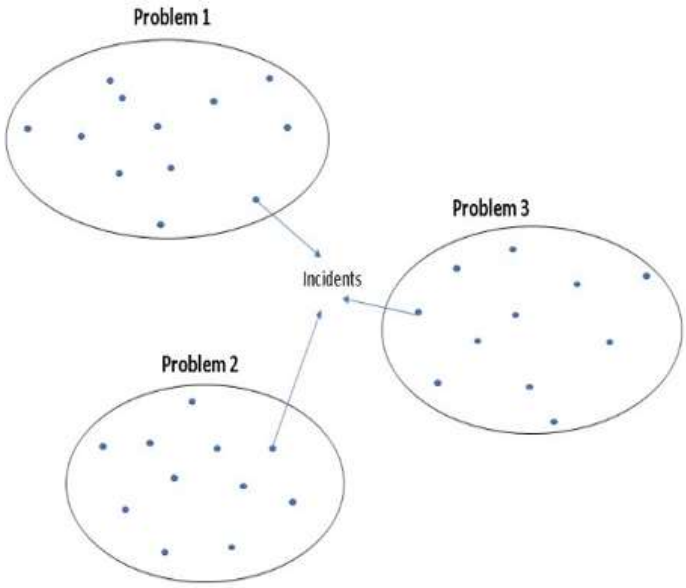

Figure 13: Problem types as classes of incidents.

7 Conclusion and Future Work

The current contribution focuses on specific problems in automated and intelligent log management that can be addressed with practical machine learning solutions continuously delivered at the customer data center. Those solutions can be core analytics functions of larger industrial products on an integrated setting. Our methods (subject to pending patents, see, in particular, [Harutyunyan et al 2019a], [Harutyunyan et al 2019b]) deal with identifying baseline distributions of log streams, controlling properties of log streams through quantified time series data, measuring impact of data center problems through change in the log structure, as well as classifying data center incidents for faster characterization of their nature. Those solutions are building blocks of a much more comprehensive data center management platform leveraging both logs and metrics and tending to self-drive the IT operations.

We introduced the concept of *baseline models* for log sources employing event types as clustered log messages and described algorithms to efficiently identify those structures using statistical sampling and machine learning. Baseline models are comprehensive for various analytical tasks in log management from real-time anomaly and change detection to other pattern detection problems. With controlled experiments, we demonstrated how the baselines are learned.

We plan to run experiments on huge data sets. In particular, within a large experimental setup we are going to learn baselines of ESXi hosts in various environments by applying our sampling concepts with confidence control and employ the learned structures in real-time anomaly detection. Then we will be able to validate observed anomalies with the incident data regularly being reported by operations teams. The live never-ending stream of ESXi events in large cloud infrastructures make a big volume. This means that even we observe a large number of different event types and hence deal with storage of large-size histograms, with the sampling techniques based on binomial distributions, the number of those histograms will be moderate. Therefore, feasibility of our implementations is not under risk because of algorithmic complexity.

References

[Ambre et al 2015] Ambre, A. and Shekokar, N.: Insider threat detection using log analysis and event correlation; Procedia Computer Science 45 (2015), 436-445, Elsevier.

[Breunig et al 2000] Breunig, M.M., Kriegel, H.-P., Ng, R.T., and Sander, J.: LOF: Identifying density-based local outliers; Proc. ACM SIGMOD Int. Conference on Management of Data, pp. 93–104, May 15-18, Dallas (2000), TX, USA.

[Brown and Kushmerick, 2015] Brown, D. and Kushmerick, N.: Anomaly detection using log summary divergence. A method for computing the similarity of two log message queries and its application in anomaly detection in distributed environments; Technical paper, VMware (2015).

[Cover and Thomas (1991)] Cover, T. and Thomas J.: Elements of Information Theory. Wiley, (1991).

[Ester et al 1996] Ester, M., Kriegel, H.-P, Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise; Proc. Second Int. Conference on Knowledge Discovery and Data Mining (KDD-96), pp. 226–231, August 2–4, Portland (1996), Oregon.

[Harutyunyan et al 2014] Harutyunyan, A.N., Poghosyan, A.V., Grigoryan, N.M., and Marvasti, M.A.: Abnormality analysis of streamed log data; Proc. IEEE Network Operations and Management Symposium (NOMS), 7p., May 5-9, Krakow (2014), Poland.

[Harutyunyan et al 2018] Harutyunyan, A.N., Poghosyan, A.V., Grigoryan, N.M., Kushmerick, N., and Beybutyan, H.: Identifying changed or sick resources from logs; Proc. Int. Conference on Autonomic Computing (ICAC), September 3-7, Trento (2018), Italy.

[Harutyunyan et al 2019a] Harutyunyan, A.N., Poghosyan A., Grigoryan, N.M., and Movsisyan, V.: Methods and systems to analyze event sources with extracted properties, detect anomalies, and generate recommendations to correct anomalies. Pending US patent US20190026459A1.

[Harutyunyan et al 2019b] Harutyunyan, A.N., Movsisyan, V., Poghosyan A., and Grigoryan, N.M.: Methods and systems to prioritize alerts with quantification of alert impacts. Pending US patent US20180341566A1.

[He et al 2016] He, S., Zhu, J., He, P., and Lyu, M.R.: Experience report: System log analysis for anomaly detection; Proc. IEEE 27th Int. Symposium on Software Reliability Engineering, October 23-27, Ottawa (2016), Canada.

[Hu et al 2017] Hu, Q., Tang, B., and Lin, D.: Anomalous user activity detection in enterprise multi-source logs; Proc. IEEE Int. Conference on Data Mining Workshops (ICDMW), Nov. 18-21, New Orleans (2017), LA, USA.

[Jia et al 2017] Jia, T., Chen, P., Yang, L., Li, Y., Meng, F., and Xu, J.: An approach for anomaly diagnosis based on hybrid graph model with logs for distributed services; Proc. IEEE International Conference on Web Services (ICWS), Hilton Village, Honolulu (2017), HI, USA, June 25-30.

[Lin et al 2016] Lin, Q., Zhang, H., Lou, J., Zhang, Y., and Chen, X.: Log clustering-based problem identification for online service systems; Proc. 38th Int. Conference on Software Engineering Companion (ICSE), Austin (2016), TX, USA, May 14-22.

[Log Insight, 2019] VMware vRealize Log Insight: https://www.vmware.com/products/vrealize-log-insight.

[Marvasti et al 2014] Marvasti, M.A., Poghosyan, A.V., Harutyunyan, A.N., and Grigoryan, N.M.: An enterprise dynamic thresholding system; Proc. USENIX Int. Conference on Autonomic Computing (ICAC), pp. 129-135, June 18-20, Philadelphia (2014), PA, USA.

[Poghosyan et al 2016] Poghosyan, A.V., Harutyunyan, A.N., and Grigoryan, N.M.: Managing cloud infrastructures by a multi-layer data analytics; Proc. IEEE Int. Conference on Autonomic Computing (ICAC), pp. 351-356, July 18-22, Wurzburg (2016), Germany.

[vR Ops, 2019] VMware vRealize Operations Manager: http://www.vmware.com/products/vrealize-operations.html.