Identifying Changed or Sick Resources from Logs

Ashot N. Harutyunyan, Arnak V. Poghosyan, Naira M. Grigoryan, Nicholas Kushmerick, and Harutyun Beybutyan
Office of the CTO of Cloud Management
VMware
{aharutyunyan;apoghosyan;ngrigoryan;nicholask;hbeybutyan}@vmware.com

***Abstract*—The identification of important changes in a complex distributed system is a challenging data science problem. Solving this problem is critical for tools for managing modern cloud infrastructure stacks and other large complex distributed systems. In this paper, we investigate two specific approaches to using log data to solve this problem. The first approach is comparing a source's current and past behavior. Some solutions that perform anomaly detection on numeric data from the data center are inevitably relying on global change point detection concepts. On the other hand, while log data promises a significantly different perspectives and dimensions to accomplish a similar task, state-of-the-art of solutions lack a capability to automatically detect significant change points in the log stream of an event source through learning its behavioral patterns. Such change points indicate the most important times when the source's behavior significantly differs from the past. A second complementary approach to real-time change detection involves comparing a source's current behavior with the current behavior of its peers in a population of sources serving a common role in the data center. Employing the concept of event types of log messages introduced earlier, we propose algorithms for each of these approaches that apply classical statistical and machine learning techniques to data capturing the distribution of those constructs. We demonstrate experimental results from our prototype algorithms.**

Keywords—Automated log management, anomaly detection, change point detection, "sick" log source, machine learning.

I. Introduction

For proactive management of data centers, infrastructure and application administrators are interested in any unexpected changes in system behavior. The unexpected patterns can be categorized as *anomalies* (extreme events of a random process that has still the same overall characteristics as in the past) and *changes* (alterations in characteristics/distribution of the random process itself). To adequately identify the anomalies of the process, one needs to characterize its typical behavior.

Cloud management solutions, in particular, vRealize Operations (vR Ops) [1] and vRealize Log Insight (LI) [2], employ data science and machine learning solutions to reliably manage today's Software Defined Data Centers (SDDC) of high complexity. These products detect both context-independent atypical patterns of data center object indicators (see Dynamic Thresholds analytics [3]) and context-dependent anomalous events and their transactions [4].

The analytics developed in [3] applies such a global change point detection internally to appropriately form the relevant anomaly reports (alarms/alerts) of numeric indicators (time series metrics) of infrastructure objects. Compared to those time series data of the IT environment any analytics learns from, the relevant log data contains significantly different perspectives to perform a change analysis. However, for log management solutions it is not an easy task to provide an automatic capability to identify changes in characteristics of log streams. Instead, they enable only a manual functionality to query or inspect log messages to find patterns the users are interested in. Particularly, LI performs such an inspection of a log source using a feature called Event Trends.

Our work proposes an enhancement analytics to empower log management solutions with relevant automation of change point detection with setting two types of problems. The first problem type concerns introducing a "log plus metric" data processing framework for change detection from historic perspectives of an event source. Moreover, since modern cloud applications are served through populations of IT resources with similar roles and behaviors, an additional run-time perspective arises to compare an event source to its peers in the population as a second type of problem. Such a change for a log source indicates its *sickness* condition within the group it belongs to for immediate remediation management.

For identifying changes of an event source, we apply a version of the classical CUSUM, an off-line change point detection algorithm (explained in Sections 2 and 3), to its meta-data and point out a suitable alternative for the streaming log scenario. For diagnosing *sick resources*, first we propose an unsupervised machine learning approach to identifying similarity groups of event sources for applications topology discovery. Then we apply population properties to detect outlying or sick event sources subject to healing.

The various changes that either of these methods detect may have many different causes, such as hitting a new software bug, hardware failure, software upgrade, configuration changes, change in workload, etc. But all of these changes relate to the many aspects of the data center administration (security, troubleshooting, performance monitoring, capacity planning, provisioning and configuration, compliance auditing, policy enforcement, etc.), and so are worthy of identification and further classification for intelligent incident management. For example, a virtual machine (VM) or an application that is being attacked probably will change its log content and structure (i.e. with a large number of "failed login" messages). These are examples of a specific kind of changed or sick sources of logs.

In Section 2 we mention about our initial motivation for this research and reference the related work. Section 3 outlines the relevant algorithms towards solutions of the first change point detection problem, while Section 4 describes our machine learning approach to identifying sick log resources. Section 5 concludes the paper with notes on a future work.

II. Motivation and Related Work

Event Types are native constructs of LI [2] that perform dimensionality reduction of the original log space. Using machine learning, LI clusters raw log messages into abstract event types based on similarity of message tokens. The screenshot in Fig 1 shows how the product's interactive analytics represents a pie chart of distinct event types for a time range corresponding to a raw log data portion. The pie chart representation of event types for a time window can be converted to a probability distribution – relative frequencies of event types (number of each event type over the counts of log messages), see Fig. 2. Currently LI supports comparison of event types for two queried time windows, enabling the user to look for message differences between those time ranges. This functionality is provided with the Event Trends feature (Fig. 3) of the product. The Event Trends view in Fig. 3 highlights event-types that occur at different rates in two periods of time with 768 types increasing.

The motivation for our work started with the following question. Can we automate the Event Trends feature in order to automatically detect the most "interesting" time ranges of a given source to compare rather than having to manually select those time ranges? Here the "interestingness" is defined by a significant difference in their event type distributions compared to the source's historical baseline distribution.

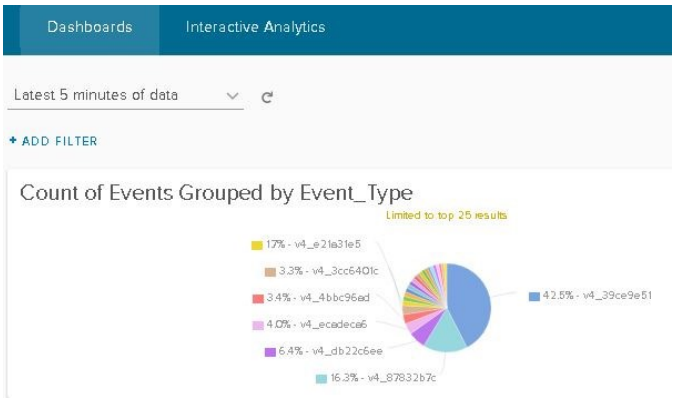

Fig. 1. Pie chart view of event types as clusters of log messages.

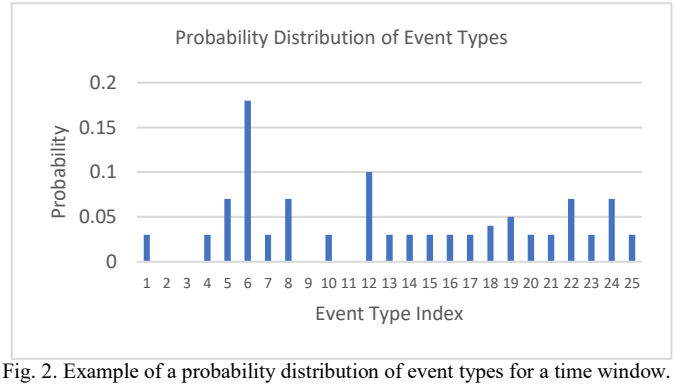

Fig. 2. Example of a probability distribution of event types for a time window.

In addition to the being useful in offering "intelligent" suggestions for the time-range in the Event Type view, such a capability would have other uses. For example, a user could automatically get *alerted on any atypical variation/change* (point) occurring over time in the event types, that could potentially tell him/her about a problem or other issue worthy of attention. Actually, we are interested in quantification of variations/ changes occurring along the log stream and learning *historical typicality* of differences between event type distributions over time to be able to identify atypical behaviors in those constructs.

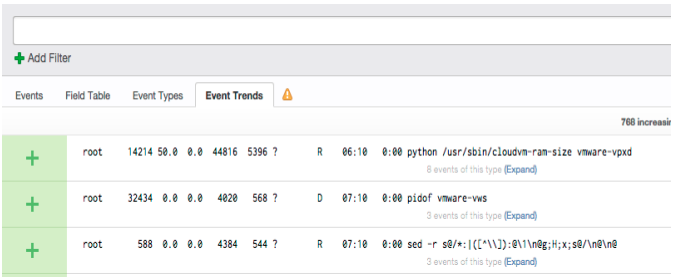

Fig. 3. Event Trends: 768 event types are increasing.

To perform such an analysis first we need to monitor and quantify the event trends over time. We take relative frequencies of event types observed in two log portions as probability distributions P and Q (Fig. 2) for which information radius or Jensen-Shannon divergence (a symmetric version of Kullback-Leibler (KL) divergence [5]):

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

where $M = \frac{P+Q}{2}$ and $D(P,Q)$ is the KL divergence between P and Q. $JSD(P,Q)$ satisfies the property $0 \leq JSD(P,Q) \leq 1$.

This divergence computed over time with a sliding window measures the "differences" between event type distributions in neighboring intervals (Fig. 3).

Different algorithms are known in literature for off-line [6] and online change detection (see [7] and references therein) in time series data for various purposes. We are going to apply a modified version of the classical change point detection algorithm called CUSUM [6] which is based on analysis of cumulative sums of "departure" of the data from its average over time.

Related prior work includes [8] where KL divergence was proposed to measure the "distance" between two log portions applied to anomaly detection. However, the approach in [8] is not built on learning "normal" behavioural patterns of event sources for full automation of the anomaly detection.

Moreover, the problem of clustering of event sources and related similarity-based management in real-time is a novel formulation. Our proposal on diagnostics of an event source based on comparing its behavioural patterns to other peers within the same population parallels with an important prior art [9] for identification of sick VMs using trade-offs between their workload and latency metrics and with [10] for an app-aware analytics.

Detection of specific change patterns related to security of applications can be found in analyses [11,12].

III. Change Detectiona: Algorithms and Experiments

The "difference" data of event type distributions described in Section 2 can be used in several different ways:

1. *change point detection* algorithms [6,7] that capture unusual behaviors in the quantified event trends data (or divergences in event types);

2. *outlier detection* algorithms (like [13] based on extreme value theory and maximum entropy principle) to detect spikes/abrupt changes in the quantified event trends data;
3. *periodicity analysis* (with Dynamic Thresholding [3]) to exclude the cases of spikes that might have cyclical nature (e.g. nightly backups).

Ideally, we are also interested to find the approximate time when the change *started* to develop (problem start time). This is an indicative time for the user to investigate the reasons of change further, comparing the event types around it and consecutive time ranges after to make conclusions using the existing Event Trends capabilities.

Below we focus on change point detection approach 1. There are different methods to detect change points in time series data. We experimented with a version of CUSUM [14] that provides a procedure to identifying the change start time as a data fitting optimization problem.

For scenarios of 2 and 3, control charting and DT analysis are solutions, respectively. One of our experimental set-ups beyond the scope of this paper deals with the approach 3.

A. Experiments based on CUSUM

To conduct a controlled experiment, we created an instance of a particular application, and then manually introduced specific known changes. Namely, we chose Log Insight as our proof-of-concept application, and we made two changes: change the workload by increasing the number of queries launched against the system, and changing the logs directly by modifying the log level from INFO to DEBUG. These two artificial changes do not capture all nuances of real-world change detection. But these preliminary experiments allowed us to evaluate our algorithms in a systematic and controlled manner.

For experimental demonstrations on a change point detection using event type constructs of LI, we performed a monitoring of a LI log stream using another LI instance. When we significantly increase the number of query requests to the LI being monitored, we start observing new event types as well as alteration in distribution of those. A similar phenomenon occurs when we modify the logging level of LI from INFO to DEBUG. We made this set-up modification (on 11/15/2016 at 19:35 in Fig. 4 and 5) in LI while collecting its log messages for a ~3.5-hour duration. Fig. 4 and 5 display the counts of events and their types over time for the above-mentioned period of observations. Those counts as well as the rate of arriving messages were multiplied after the debug mode was switched on.

Fig. 6 represents the JSD meta-data quantification for event trends computed for each 20-minute window sliding it by 2 minutes. Instead of 20 minutes can be another parameter tuned to the application dynamism. JSD=0 implies that the distributions are exactly the same, and JSD=1 means that distributions are totally different and "no event types in common between the two distributions".

So, for a time stamp i, we derive the distributions of event types within $\Delta t = 20$ minutes sliding-window with shift equal to 2 minutes and calculate JSD (D_i) between those two consecutive distributions. Then the goal is to run a change point analysis on the time series data $\{D_{i=1}^{N}\}$ (Fig. 6).

The procedure to perform the change-point analysis iteratively uses a combination of cumulative sum charts and bootstrapping to detect the changes. Known from statistical quality control, CUSUM algorithms [6] are widely used in change point detection in time series data. These algorithms alarm about a change based on "large deviations" in cumulative sums or variances from the baseline (like mean or median) of data.

The analysis begins with construction of the CUSUM chart for the data in Fig. 6 using the classical mean-based approach. From Fig. 6 we see that the debug logging introduces a change in the initial behaviour of the data and want to detect that change.

Outline of the CUSUM-Mean [14]. The input is a data set of length N with values $D_i, i = \overline{1,N}$. The algorithm computes the mean of data $\overline{D}$ and its cumulative sums S_i. The index m corresponding to the maximum of those cumulative sums is a candidate for change point but it should be validated by a bootstrapping (random reordering of the data) to measure the confidence C in change. Then to detect the change start time the data is divided into two parts starting from m and the corresponding mean square error (MSE) is calculated against the averages of left and right portions of data, while decreasing m (a fitting data procedure). The index resulting the minimum MSE is picked up as the start time of the change.

Detected change time. CUSUM-Mean's output as the change index was 68 with confidence level equal to 0.98 and there is no process found (start time is the same index=68). So, from the mean-based CUSUM perspectives this is an abrupt change.

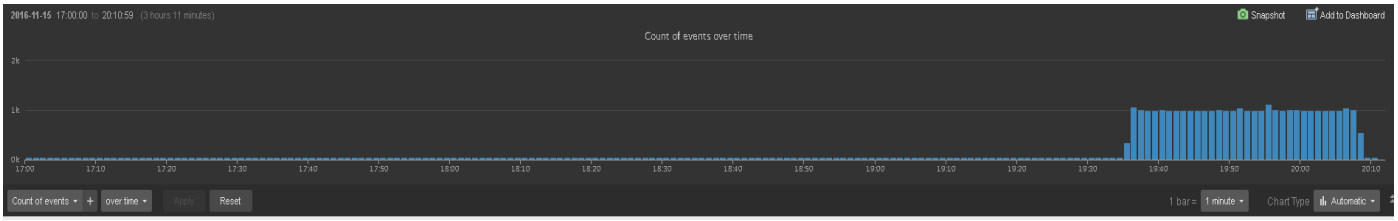

Fig. 4. Events Count over time, before and after the change from INFO to DEBUG logging.

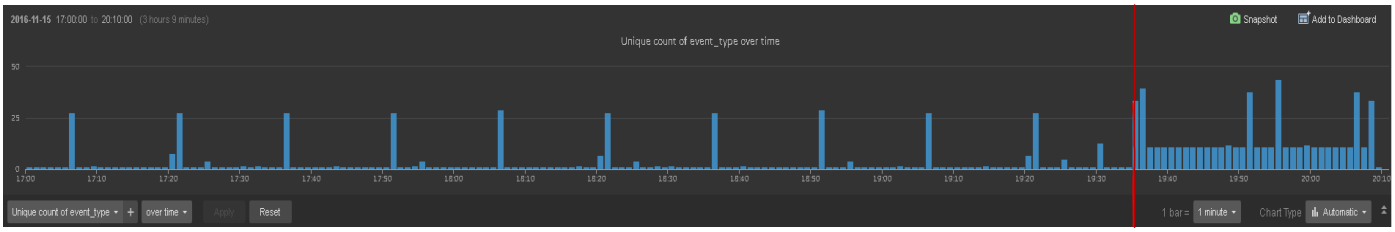

Fig. 5. Events Types Count, before and after the change from INFO to DEBUG logging.

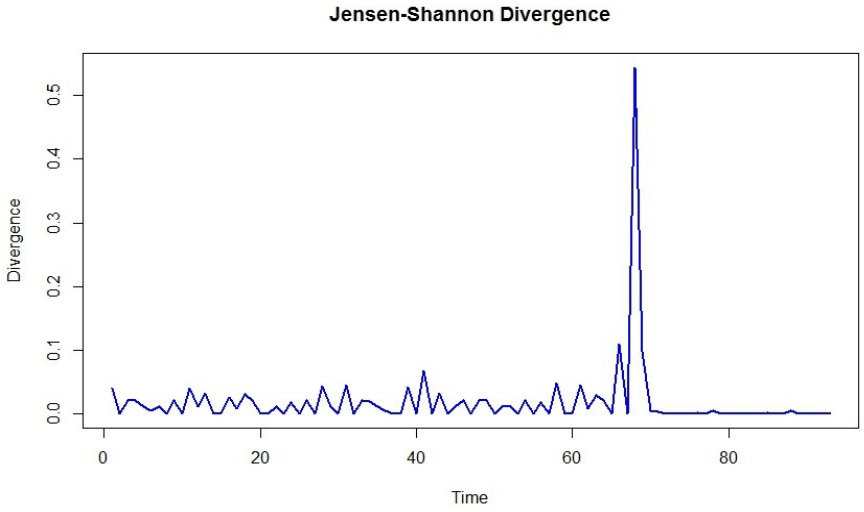

Fig. 6. Jensen-Shannon divergence (JSD) between two distributions A and B [A is distribution of event types before the change from INFO to DEBUG, B is event-type distribution after the change from DEBUG to INFO]. The graph shows that there is a large change in the distribution of event types after Time=65 (corresponds to the red line in Fig. 4/5).

B. Content-Based Change Inspection

Our algorithm then calculates the rate of change for each event type after detected change point. In other words, it produces a ranked list of event types "responsible" for the mismatch between pre- and post-change log patterns.

In particular, we calculated the "change rate" (simple difference between relative frequencies) for each event type when comparing two distributions of those at different time stamps: pre-change (66th index in Fig. 6) histogram with the changed one (68th index). Then LI's Event Trends is executed to compare the change interval [Nov 15 2016 19:16:00, Nov 15 2016 19:36:00] with previous 20 minutes to see that 81 event types are increasing, 1 decreasing and top increasing event types relate to the "debug" messages.

C. Streaming Algorithm for Change Detection

Although the meta-data (JSD) conversion of the log stream mitigates the complexity to perform a retrospective analysis of the entire log history for change point detection, it is still preferable to have an incremental/streaming algorithm reacting to run-time changes instead of the batch processing of CUSUM-type algorithms.

A simplistic streaming alternative solution is implemented by incrementally computing the histogram distribution of JSD data itself. We build the empirical distribution of JSD for an event source and update it as events arrive. For this we just need to keep and recalculate the counters of JSD values falling into selected number of histogram ranges from [0,1]. This distribution over time contains all patterns of consecutive changes in the stream. Then any JSD variation in neighbouring sliding windows matching the ϵ-tail of this distribution indicates an atypicality or anomaly in the data.

Fig. 7 depicts the corresponding histogram distribution of JSD data in our experiments.

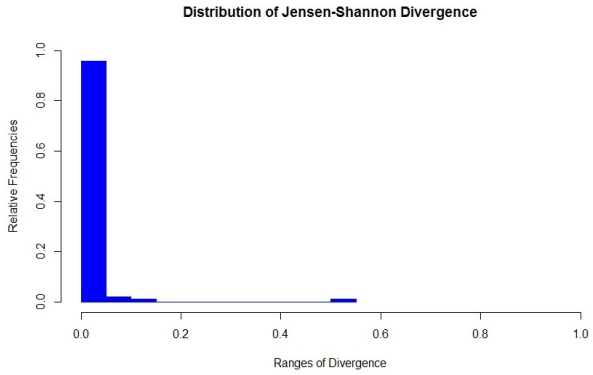

Fig. 7. Histogram of JSD data in Fig. 6.

The tail analysis of the histogram shows that the index 68 with value 0.54 is an outlier distance and should be reported as change in the original log stream. We get the same result, if assuming the JSD is normally distributed and any observations falling outside

$$[mean - 3std, mean + 3std]$$

interval of the metric is anomalous.

IV. Run-Time Detection of Sick Event Sources

Turning to the second kind of change point problems, note that automatic ways of detecting atypical/anomalous event sources based on either their historical behavioral analysis or run-time status is not a trivial task. It is a common practice that an anomaly status of a log source is alerted based on user-defined alert conditions containing an expert knowledge. However, it is very difficult to incorporate expert knowledge that covers every kind of change. Fortunately, our experiments demonstrate that event type distributions serve as generic indicators of change. We believe we can build a generic change detector to identify "sick" sources without needing complex expert knowledge. As we saw in the last sections, event types and their characteristics are such indicators. Therefore, they can be applied in solving a wider class of problems in managing the distributed cloud ecosystems. For instance, while individual learning of behavioral patterns for each event source in a big cloud is a complex analytical task, for the cloud-native applications it is much more effective to realize a segmentation-based management approach. This means that first we need to identify similar event sources of logs that might serve a common role in the application infrastructure (hence, can be treated and managed as population/crowd of objects). Then populations are subject to a management based on their properties. This means that any event source changing its behavior against the rest of the population can be identified run-time. This leads to application of machine learning both in clustering of event sources and detecting dissimilar log resources. In those tasks, the event types are invaluable "signatures" or "fingerprints" of log sources to rely on again.

Based on event type distributions of log sources and cosine similarity measure of two vectors, our prototype implementations apply

1. hierarchical clustering [15] to identify populations of similar sources;
2. Local Outlier Factor (LOF) [16] analysis for each population to identify its *sick* members.

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 objects 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 [16].

In addition to the identifying anomalous sources, this meta-data could be exposed to end-users for other purposes, such as allowing users to filter their logs to find "all sources like this one". In this way, users will be able to more easily define populations of objects to apply the relevant analytics. In this context, a simplistic version of the clustering instead of item 1 can be applied, making a cluster of event sources with the following property – minimum pairwise similarity within the cluster is higher than a bar.

Our experiments with log data from large environements show very high similarities (close to 1) between ESXi hosts in terms of their event type distributions and, hence, can be efficiently clustered using hierarchical clustering [16] to manage population of ESXi hosts accordingly.

For a demonstration purposes, we performed a small experiment on five Apache servers (consisting of web, email, and ftp services), with similar behavioral patterns of event types. We emulated a DDOS attack (using ApacheBench test tool) on the web host of one of servers with high-rate service requests compared to the rest of servers during a 5-minute period. This load "shifted" the corresponding server from the population according to the LOF analysis. Moreover, even with the simple statistical whisker's method applied to the "average distance from the rest of server group" data computed for each server it was possible to identify the sick host.

For simplicity, we show the distributions of event types (during the observation window) for only one "normal" host (Fig. 8) from the population of five (the rest exhibited very similar distributions while being in their normal operational workload modes) and the detected sick one (Fig. 9).

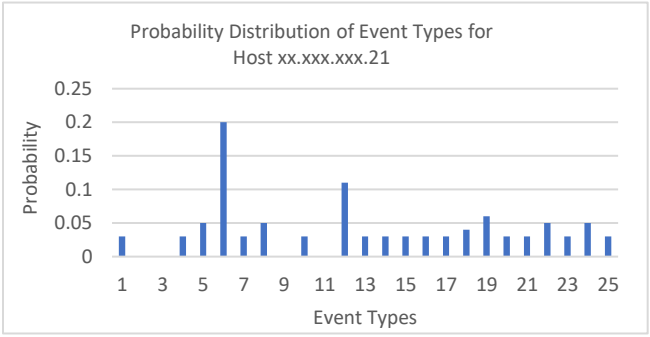

Fig. 8. Event type distribution of one of the "normal" host members of Apache servers population.

Details of the experiment are as follows. All the pair-wise similarities between the hosts using cosine similarity measure were above 0.95 (a very high similarity) before the simulation. Because of the attack on the host xx.xxx.xxx.22 (Fig. 9) with a 5-minute duration, the latter already exhibits a similarity drop from the rest of the hosts. The naïve whisker's anomaly detection method from non-parametric statistics was useful for our purposes to observe the "sickness" of the attacked host. This method defines anomalous data values as those that stay outside the interval

$$[Q_{25} - 1.5IQR, Q_{75} + 1.5IQR],$$

where Q_{25} and Q_{75} are the first and third quartiles of the data, and $IQR = Q_{75} - Q_{25}$ is its interquartile range. The average cosine distance for the loaded host from the rest was detected as outlying. Since the average cosine similarity distances (ACSD) for the hosts were

ACSD(host21)=0.97,

ACSD(host22)=0.89,

ACSD(host 23)=0.96,

ACSD(host 24)=0.97,

ACSD(host 25)=0.97,

the ACSD score of 0.89 for the over-requested host is outside the above-mentioned interval. Therefore, the attacked host with ACSD=0.89 *is dissimilar from the rest of the crowd and is identified to be "sick"*.

Note that in this case we deal with an unexpected situation which is not about observing specific log messages and their types containing a contextual information like "error", etc., but about a statistically atypical distribution of events as an important change indicator the system incurs in terms of a security attack.

Although distributions in Fig. 8 and 9 may look similar, their dissimilarity is sufficient for the algorithms to declare about the sickness of the stressed host compared to the rest of population. In particular, event type #9 was absent at normal hosts, while at the sick host it has 8%-presence, or, event type #18 that increased from about 4% (at normal hosts) to 11% (at sick host), etc.

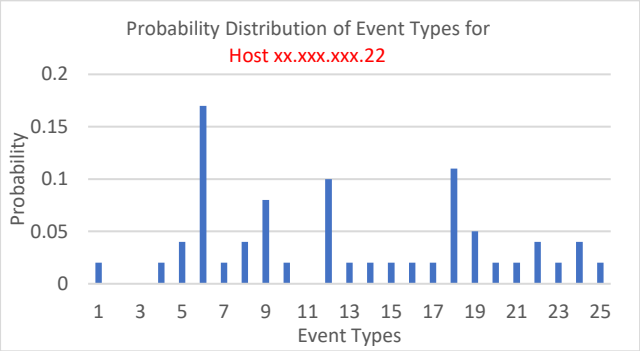

Fig. 9. Event type distribution of the "sick" host in population of Apache servers.

Apparently, the naïve outlier detection approach mentioned above is only for demonstration purposes. In reality, we need to rely on the most sophisticated outlier detection methods of machine learning like LOF analysis [16] to maximize reliability of capturing the patterns we are looking for. However, in both cases, we deal with algorithms of polynomial complexity by the size of the population which is not a very large number in applications. Therefore, these run-time decisions making algorithms are scalable with the population dynamics, but might require storing large vectors of event type distributions depending on the nature of event sources (very high volume of events can be clustered by LI into a high number of event types in one case, and into a small number of event types in other case).

V. Conclusion and Future Work

We introduced a new framework for change point analysis for log event sources with two complementary problem formulations. The first relates to an individual historical learning of typical characteristics of a log source, while the second links to a real-time comparison of event sources with each other within a population of peers in distributed cloud environments. Our solution approaches to those problems and implemented prototypes demonstrated a reliable and effective identification of change points in log streams and sick log sources.

We have built a large experimental set-up for our algorithms to consume the streamed log data of a large environment and perform near-real-time detection of ESXi hosts experiencing change. Our plan includes validating observed changes with the incident data weekly being reported by the operations team. The live never-ending stream of all ESXi events with their event-type and hostname fields make a big volume (approximately ~14 billion events per day for ~2500 ESXi hosts).

*Classifying Change Incident Data***.** Already occurred changes (with corresponding training data) that have been investigated can be tagged with corresponding change categories observed by the admin (such as 'restart', 'upgrade', 'disk full', 'overprovisioning', etc). Having such a labelled data will enable automatic classification of a newly detected change into one of known classes applying supervised machine learning algorithms.

References

[1] VMware vRealize Operations Manager, http://www.vmware.com/products/vrealize-operations.html, last accessed 07/12/2018.

[2] VMware vRealize Log Insight, https://www.vmware.com/products/vrealize-log-insight, last accessed 07/12/2018.

[3] M.A. Marvasti, A.V. Poghosyan, A.N. Harutyunyan, and N.M. Grigoryan, An enterprise dynamic thresholding system, *USENIX International Conference on Autonomic Computing*, June 18-20, Philadelphia, US, pp. 129-135, 2014.

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

[5] T. Cover and J. Thomas, *Elements of Information Theory*, Wiley, 1991.

[6] E.S. Page, Continuous inspection schemes, *Biometrika*, vol. 41, issue 1-2, pp. 100–115, June 1954.

[7] A.B. Downey, A novel changepoint detection algorithm, arXiv:0812.1237, 2008.

[8] D. Brown and N. Kushmerick, 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, *VMware technical paper,* 2015.

[9] V. Gupta, P. Padala, A. Holler, and A. Desai, Automated management of "sick" virtual machines for cloud applications, *VMware technical paper,* 2014.

[10] A.N. Harutyunyan, N.M. Grigoryan, A.V. Poghosyan, M. Avagyan, and A. Dangizyan, Application-aware analytics based on object populations, *VMware technical paper,* 2016.

[11] A. Ambre and N. Shekokar, Insider threat detection using log analysis and event correlation, *Procedia Computer Science*, vol. 45, pp. 436-445, Elsevier, 2015.

[12] Q. Hu, B. Tang, D. Lin, Anomalous user activity detection in enterprise multi-source logs, *IEEE International Conference on Data Mining Workshops (ICDMW)*, Nov. 18-21, New Orleans, LA, USA, 2017.

[13] A.V. Poghosyan, A.N. Harutyunyan, and N.M. Grigoryan, Managing cloud infrastructures by a multi-layer data analytics, *IEEE International Conference on Autonomic Computing*, July 19-22, Wurzburg, Germany, pp. 351-356, 2016.

[14] W.A. Taylor, Change Point Analysis: http://www.variation.com/cpa/tech/changepoint.html, last accessed 07/12/2018.

[15] L. Rokach and O. Maimon, *Data Mining and Knowledge Discovery Handbook,* Springer, Boston, MA, 2005.

[16] M.M. Breunig, H.-P. Kriegel, R.T. Ng, and J. Sander, LOF: Identifying density-based local outliers, Proc. *ACM SIGMOD International Conference on Management of Data*, pp. 93–104, 2000.