April 2023

NOISE REDUCTION IN METRIC, EVENT, LOG, AND TRACE (MELT) DATA USING DISTRIBUTED MACHINE LEARNING

Srinivasan Srinivasan

Linda Zhou

Manikandan Mari Vera Kumara

NOISE REDUCTION IN METRIC, EVENT, LOG, AND TRACE (MELT) DATA USING DISTRIBUTED MACHINE LEARNING

AUTHORS:
Srinivasan Arashanipalai
Linda Zhou
Manikandan Mari Vera Kumara

ABSTRACT

In the Observability domain, metric, event, log and trace (MELT) are basic data types generated by the infrastructure and applications. These datasets are not only ingested at high volume and high frequency but also related. Currently, available solutions are for individual data types, i.e., metric monitoring, log analytics, trace flow analysis, etc. These solutions do not provide a holistic view of the entire environment with MELT correlation. To address these types of challenges, techniques are presented herein that support a scalable, flexible, dynamic, and adaptive noise reduction system. While the system is running as expected, data is collected at a lower frequency. When the first sign of trouble appears, such a system may automatically increase collection frequency for change point detection, anomaly detection, log pattern detection, and causal inference. Aspects of the presented techniques employ a two-phase filtering mechanism comprising Edge Processors and Global Processors to intelligently apply machine learning techniques to scale up and down monitoring and root cause analysis capabilities.

DETAILED DESCRIPTION

To diagnose the cause of an anomaly related to business transactions or key performance indicators (KPIs), users need to drill down into the related logs, events, and traces. In typical cloud native applications, the volume of traces and logs that are associated with a business transaction represents a huge overhead. The problem is further aggravated by the fact that the logs are usually unstructured, low-level, noisy, and lack the information about changes to the states of resources. Consequently, users need to employ both traces and logs, as logs relate to intra-service behaviors while traces pertain to inter-service behaviors.

To identify and reduce noise, an integrated approach is used to decide the relationships and the dependencies between metrics and the related logs and traces. Such an approach should satisfy several design goals.

The first design goal encompasses scalability. In cloud native deployments it is quite common for thousands of requests related to metric, event, log, and trace (MELT) data types which need to be processed each second. The second design goal encompasses low overhead. The approach should identify the incoming data as either significant or noise without impacting the throughout. The third design goal encompasses configurability. The approach should allow users to specify quotas and those constraints should be used to decide whether to mark data as noisy or significant.

The fourth design goal encompasses flexibility. The approach should allow a user to configure the solution to fit their specific requirements. For example, in addition to an out-of-the-box (OOTB) configuration, users may tweak Relevance score thresholds to determine whether the data is noisy. A fifth design goal encompasses automated machine learning (ML). The ML models and algorithms should address different MELT data types without requiring manual intervention.

Before proceeding, let us clarify several important terms that are used in this proposal. Noise is defined as data that includes MELT data types which provide low functional value during a root cause analysis and impose a significant overhead.

A Relevance score is defined as a measure of its usefulness for diagnostic purposes. A low Relevance score points to noise while higher values relate to functional significance. MELT data that is associated with normal performance is typically associated with a low significance score while anomalous behaviors are associated with a high significance score. Typically, a significant percentage of data is associated with normal performance while anomalous behaviors are related with few data points. Computing a Relevance score is a multi-step approach and is influenced by the relationships between the MELT data types and within / across temporal time windows. A resource quota may be defined as a user-specified constraint which influences a “whether to mark data as noise” decision.

Figure 1, below, presents various of the functional components of a system according to the techniques presented herein and reflective of the above discussion (including, among other things, the described design goals).

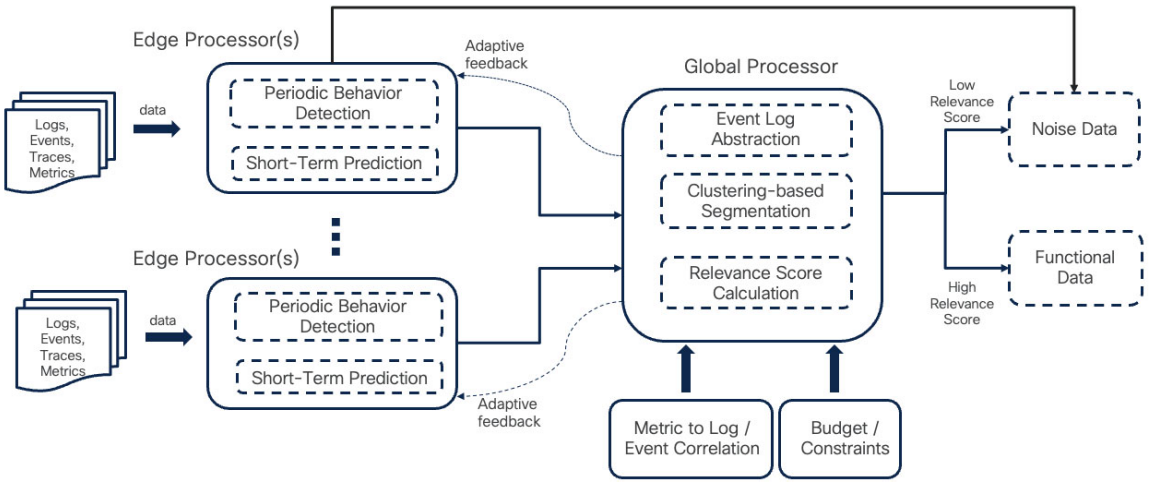


Figure 1: Functional Components

Identifying noise is completed in two phases, Edge Processors and Global Processors, to evaluate if not incoming data should be classified as noise. Such a distributed filtering mechanism is employed to avoid the potential bottleneck, which may arise with a single centralized filter e.g., when a substantial number of log requests arrive simultaneously.

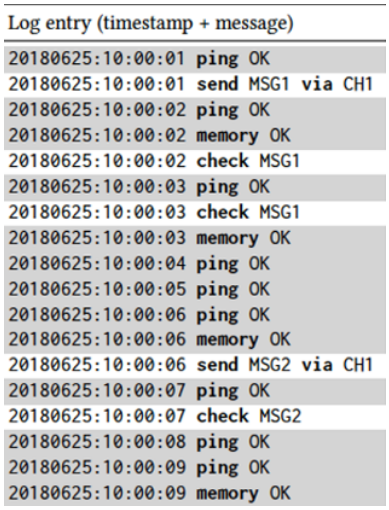
The Edge Processors are responsible for quickly classifying requests as noise. The Global Processors consider the relationships across the MELT entities and state changes to compute a Relevance score. Based on the Relevance score, a request may be tagged as either noise or functional data.

It is important to note that based on user requirements, the functional components in the Edge Processors may be included in the Global Processor (e.g., if the volume of data is not significant).

The Edge Processors introduced above are responsible for quickly classifying a request as noise using periodicity analysis. Periodicity analysis may detect low significance logs by identifying log templates (i.e., a heartbeat message such as “Now listening on: https://localhost:xxxx" or "20230125:10:00:03 ping OK"). These messages may be characterized by a fixed periodicity.

The first step in the above-described process is log abstraction detection, which is used to identify log templates. Such a log parsing process extracts the pattern of recurring

logs by automatically separating the constant parts and the variable parts (i.e., tokens) of a raw log message and applying regular expressions to each token resulting in the log templates. An example of such an approach is illustrated in Figure 2, below, where low significance log messages are presented in grey and high significance log messages are presented in white.



Log entry (timestamp + message)
20180625:10:00:01 ping OK
20180625:10:00:01 send MSG1 via CH1
20180625:10:00:02 ping OK
20180625:10:00:02 memory OK
20180625:10:00:02 check MSG1
20180625:10:00:03 ping OK
20180625:10:00:03 check MSG1
20180625:10:00:03 memory OK
20180625:10:00:04 ping OK
20180625:10:00:05 ping OK
20180625:10:00:06 ping OK
20180625:10:00:06 memory OK
20180625:10:00:06 send MSG2 via CH1
20180625:10:00:07 ping OK
20180625:10:00:07 check MSG2
20180625:10:00:08 ping OK
20180625:10:00:09 ping OK
20180625:10:00:09 memory OK

Figure 2: Illustrative Log Messages

One periodicity checking algorithm, according to the techniques presented herein and reflective of the above discussion, is presented in Figure 3, below.

Algorithm 1: Periodicity Checking

1. Derive the log templates using log abstraction
2. Compute the mean of the original data time series for each log template
3. Compute the difference between Original Data and Mean for all the observations
4. Square the output of (2) step
5. Compute the SUM of squared difference between Original Data and Mean for all the observations
6. Compute the difference between Lag 1 series and Mean for (n-k) observations
7. Compute the product between the output of (2) and (5)
8. Compute the SUM of output of step (6)
9. ACF of Lag 1 = Output(6) / Output(4)
10. Apply Durbin-Watson statistic to test for autocorrelation
11. If Durbin-Watson statistic value is near 0, mark logs as “low significance”

Figure 3: Periodicity Checking Algorithm

The proposed approach to detect periodic behavior is based on autocorrelation or serial correlation. This type of correlation is used to understand how the time series observations depend on with values of the same series of a previous time window. The past observation in the series is referred to as lags. We first compute the correlation coefficient between X and Y time series, then extend it to compute the correlation between the same time series.

$$\text{Corr}\,(X,Y) = \frac{\text{Cov}\,(X,Y)}{\sigma_X \sigma_Y} = \frac{\frac{1}{n-1}\sum_{i=1}^{n}\left((x_i - \bar{x})(y_i - \bar{y})\right)}{\sigma_X \sigma_Y}$$

The correlation of the same series (Lag 0) is computed as

$$\text{Corr}\,(X,X) = \frac{\text{Cov}\,(X,X)}{\sigma_X \sigma_X} = \frac{\frac{1}{n-1}\sum_{i=1}^{n}\left((x_i - \bar{x})(x_i - \bar{x})\right)}{\sigma_X \sigma_X}$$

$$\text{Corr}\,(X,X) = \frac{\text{Cov}\,(X,X)}{\sigma_X \sigma_X} = \frac{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}{\sigma_X \sigma_X} = \frac{{\sigma_X}^2}{{\sigma_X}^2} = 1$$

Let $L1 = \text{Lag } 1$ of X time series. The length of L1 series will be $n-1$
Since the summation is applied for both the series, the summation will have $n-1$ terms.

$$\text{Corr}\,(X,L1) = \frac{\text{Cov}\,(X,L1)}{\sigma_X \sigma_{L1}} = \frac{\frac{1}{n-1}\sum_{i=1}^{n-1}\left((x_{i+1} - \bar{x})(L1_i - \bar{x})\right)}{\sigma_X \sigma_X} = \text{ACF (Lag 1)}$$

We computed the ACF of Lag 1, and we can same extend the same formula to generalize the lag terms.

$$\text{ACF}(L1) = \frac{\frac{1}{n-1}\sum_{i=1}^{n-1}\left((x_{i+1} - \bar{x})(L1_i - \bar{x})\right)}{\sigma_X \sigma_X}$$

Now we derive $\text{ACF}(L_K)$ as below

$$\text{ACF}\,(L_K) = \frac{\frac{1}{n-1}\sum_{i=1}^{n-k}\left((x_{i+k} - \bar{x})(Lk_i - \bar{x})\right)}{\sigma_X \sigma_X}$$

Based on the data behavior detected in a prior time window, a near term forecasting algorithm predicts if the data in the succeeding time window(s) are likely to be of high or low value. The intuition is that that the data value should be higher if the performance properties of the data in the current time window deviates from its behavior in a prior time window(s). If the forecasted value is low, then those requests have a higher likelihood of being low significance data. The forecasted values may be dynamically updated over each time window.

Important considerations in an Edge Processor are low additional overhead and scalability. To address those requirements, a distributed architecture and the inclusion of only filters that are related to periodicity checks are suggested.

A significant percentage of requests typically relate to the normal performance of an entity. A key intuition is that the information content in such requests is low while the information associated with resource state changes is high.

The Global process functions are described in the following sections. Data in a normal period are typically used to train ML model. However, after the ML models are trained such requests are of low Relevance.

Figure 4, below, presents elements of a high-level view of a Kubernetes (K8) lifecycle from a log perspective.

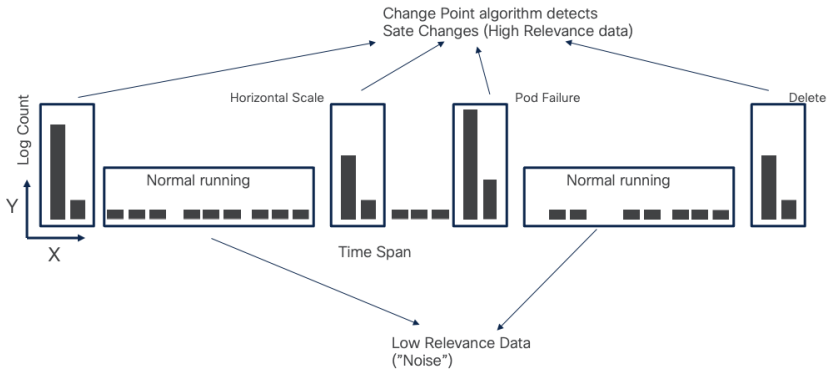


Figure 4: k8 Illustrative Lifecycle with Transient Actions and State Transitions

Logs and events represent the behavior (i.e., context) of a system while metrics show the performance status of a system. The metrics may be related to resources (e.g., infrastructure) or to an application (e.g., traces and spans).

Changes in a resource state may be identified using change point (CP) detection algorithms and lifecycle templates. The CPs may be considered when computing a Relevance score.

The techniques presented herein support the discovery of the relationships between the entity types. A clustering algorithm may be used to identify affinity groups based on their significance (i.e., a Relevance score). The Relevance score determines if a request is tagged as noise or functional data.

As may be seen from the above, logs are generated at various points in time and metrics are collected at potentially different points in time. Collecting monitoring data points usually happens at fixed intervals, such as every minute or every five minutes. In contrast, observation of system operations behavior through logs happens at non-fixed intervals, such as the occurrences of a logs within one second and then then missing for the next few seconds / minutes.

If a dataset is compliant with the OpenTelemetry (OTel) framework, then linking the traces to logs may be accomplished using the Trace ID which is included on the related logs. However, associating traces to other infrastructure logs (e.g., K8 logs) is more involved as a Trace ID is not included in them.

To associate Traces and infrastructure logs, it is necessary to extract a set of metrics that show the occurrences of different logs. Although the styles of logging may be different, almost all types of logs contain time-stamped information whether they are application logs, database logs, or operation logs as shown in Figure 2. Furthermore, a log message represents information about an event, including logs that indicate preparation or waiting periods.

Resource metrics contain a combination of performance metrics (e.g., memory or CPU usage) and state-based metrics which indicate a transition in state (as depicted in Figure 5, below).

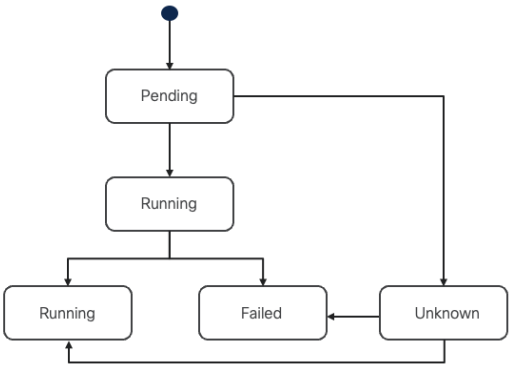


Figure 5: Illustrative State Transitions k8 Pod Lifecycle

Identifying state changes (such as, for example, a pod failure) is important as the related logs are of high value in a root cause analysis (RCA), and thus have a high Relevance score, while the lack of state changes correspond to a lower Relevance score. The state changes relate to change points.

The techniques presented herein detect change points using a Bayesian model. The model is agnostic regarding the type of state change and may be applied to different resource lifecycles. Rather than retrospective segmentation, the focus is on causal predictive filtering, generating an accurate distribution of the next unseen datum in the sequence given only data already observed. The logs and traces that are associated with the change points are assigned a higher Relevance score.

Figure 6, below, presents elements of an approach overview according to the techniques presented herein and reflective of the above discussion.

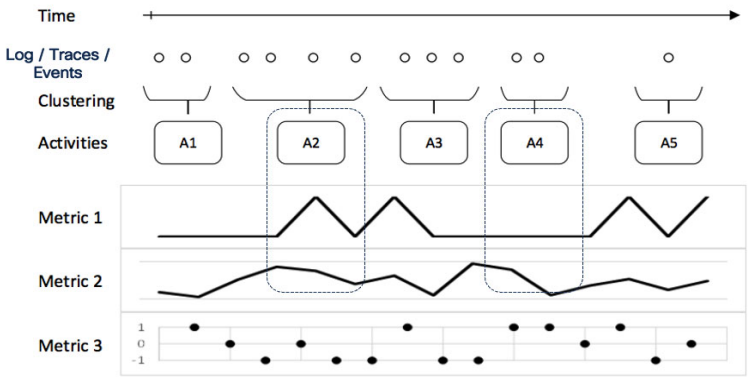


Figure 6: Mapping & Metrics & Logs Approach Overview

Under the approach that was described and illustrated above:

- Logs refer to individual log lines. Typically, each log line includes a timestamp and a description.
- Clustering is a process that clusters fine-grained correlated logs to a set of coarse-grained activities. This process may be referred to herein as log abstraction or log clustering.
- Log groups refer to sets of logs that together are responsible for making a change in a system or indicating a status of a system or application. For example, the launching of a K8 pod instance.
- Metrics represent status and utilization of system resources within a time window. For example, latency, average CPU or Memory utilization.

As described and illustrated in the above narrative, one of the elements of the techniques presented herein is a Relevance score. The key steps in computing a Relevance score are depicted in the flowchart that is presented in Figure 7, below.

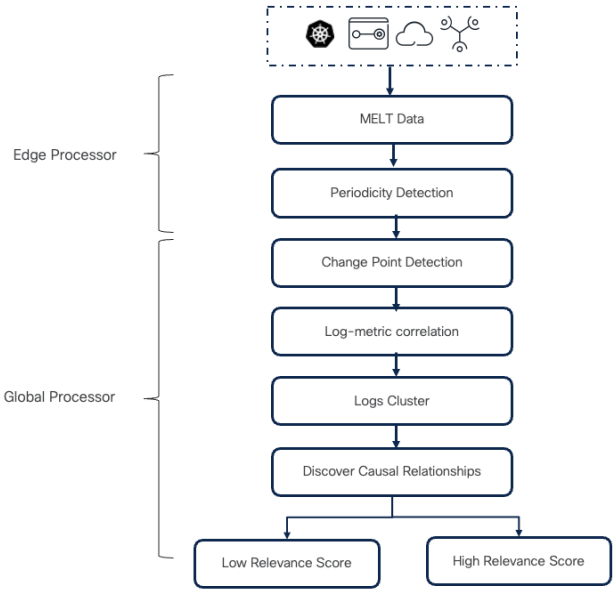


Figure 7: Relevance Score Computation

In the flowchart that was presented in Figure 7, above, the step encompassing the identification of logs using a log abstraction was described previously in connection with the above discussion of Edge Processors.

Under the step encompassing the mapping of an log to metrics, (e.g., a trace latency) may be collected with, for example, a granularity of one minute. In contrast, log may be logged with a frequency of seconds or several minutes. We interpolate the occurrence strength of log clusters that occurred within each one-minute long time window to the respective minute.

Typically, a similar cohort of log Groups together cause a tangible impact on the status of resources. Thus, to find the impact of logs on related it is necessary to group related logs. A correlation coefficient may be employed to identify the strength of relationships across all of the related entities (i.e., logs, events, and traces). Log Groups with high correlations may be combined to an log group (as depicted in Figure 9, below, which presents an example of a JBoss log with the clustering of related logs to log groups).

Figure 9: Exemplary Clustering of Related Logs to Log Groups

A coefficient may be represented by the symbol r. In the instant context there is one dataset $x_1, \ldots, x_n$ containing n values and another dataset $y_1, \ldots, y_n$ containing n values. Accordingly, a value for r may be obtained as follows:

$$r = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{\sqrt{[n\sum x_i^2 - (\sum x_i)^2][n\sum y_i^2 - (\sum y_i)^2]}}$$

In the case of calculating the correlation strength between two log types, in the above equation n is the number of monitoring observations, x_i denotes the interpolated occurrence strength of type x at time i, and y_i denotes the interpolated occurrence strength of type y at time i.

A value of r that is close to one (1) indicates a strong positive correlation between the variables, which, in the instant case, indicates that event types are high highly correlated and may be part of the same event group cluster.

The above proposed log abstraction technique using extracted interpolated occurrence strength is a novel approach in the domain of log abstraction. The techniques presented herein take into account the fixed time interval (which is enforced by metric availability) and interpolation occurrences of log event types.

This approach has the advantage of being less dependent upon the quality of the text of the logs, which is particularly relevant as the quality of the message content may vary.

Finally, under the step encompassing log group and metric correlation, in typical enterprise cloud native deployments there are an excessive number of metrics. Further many metrics are of low significance & leads to cognitive overload. Changes in a log's activities data are typically precursors to a change with respect to related metric. For example, it may be assumed that as services initiate new sessions recorded in logs leads to changes in associated resource consumption metrics. Such observations may be used to detect the normal (and changes in the) behavior of the system. Consequently, it is important to identify the few critical metrics that are of high Relevance. To achieve this, we discover the causal relationship between clustered log and changes in metrics may be discovered. Based on this, a hypothesis posits that the status of metrics may be predicted from changes in activities in the logs. The dependent variables are the metrics while the independent variables are the activities that have been abstracted from logs. Those metrics that have a high causality relationship are tagged as having a high significance i.e. high Relevance score.

Figure 9, below, presents elements of a flowchart that captures the checking of the relevancy of a monitoring metric.

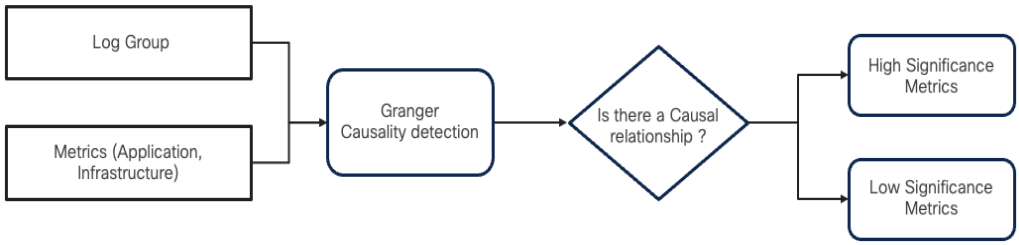


Figure 10: Illustrative Metric Relevancy Detection

Flexibility is an important design goal. According to aspects of the techniques presented herein, a user may define a logging quota – Such a quota is a threshold that is checked by the Edge Processors thereby complying with the logging quota. At runtime, the logging system decides "whether to log" such that the logging overhead is constrained under the quota while the logging effectiveness is maximized. The quota or logging overhead is defined as logging bandwidth, which is the maximum volume of logs that are to be allowed to be output during a specified time interval (such as, for example, one kilobyte (KB) per second). There are two reasons for choosing logging bandwidth as the quota. Typically, input and output (I/O) bandwidth is the most concerning overhead in practice. Further, in general most logging overhead such as disk storage, network I/O and CPU consumption are directly or indirectly affected by I/O bandwidth.

Aspects of the techniques presented herein employ adaptive feedback. For example, based on current logging performance and a user-defined quota, the Edge Processors may increase or decrease a log identification rate.

Use of the techniques presented herein offers a number of advantages. Several of those advantages will be briefly described below.

A first advantage reflects the reality that given the high volume and high frequency of metric, log, and trace information that is collected from different agents, it is very costly to store every bit of information in fast storage. In production, the uptime of any system is at a minimum 99.9% (i.e., three nines). Most of the time, 99.999% (i.e., five nines) is the standard under service-level agreements (SLAs). Therefore, most of the time, collected

metric, log, and trace data are never used for observability purposes. The techniques presented herein maintain the fidelity of the information in cold or archive storage (according to data management policies) while enabling DevOps to use it to detect anomalies and perform RCA.

Under a second advantage, with metric and log information both ML and user-defined rules may be used to increase or decrease the data that is stored in the fast tier storage. The list price for one cloud-based fast storage solution is $0.021 per gigabyte (GB) while the price for a cloud-based archiving solution is $0.004 per GB. For example, one large mobile operator ingests 600 terabytes (TB) of log content each day, yielding a storage cost for cloud-based fast storage of $12,600 per day while the storage cost for a cloud-based archiving solution would be $2,400 per day (in other words, a savings of 425%). Not only can operational efficiency be increased by actively moving data to slow storage, but it is also possible pass the savings along to customers and further introduce competitive pricing. For instant access the techniques presented herein employ ML models to actively categorize data as being important or not important. If the data is tagged as important, the ML models may detect anomalies, build log context, and create a trace knowledge graph. All of the ingested data may be stored for audit trail purposes in the slow tier of storage. For example, using the above example 600TB of ingested log content would be on cold storage and approximately 0.1% would be on the fast tier (costing only $12.60 per day) providing a 425% savings.

A third advantage encompasses a very important aspect of the techniques presented herein – the noise reduction system is adaptive. When the first sign of trouble arises, the system may automatically increase data storing on the fast tier storage for metric change point detection, anomaly detection, log pattern detection, log context detection, and trace/span graph anomaly detection. In this way, when DevOps tries to look for the root cause of system problems, all of the relevant information will be at their fingertips. By significantly reducing the amount of irrelevant data that is stored for anomaly detection and RCA, a dashboard according to the techniques presented herein will run faster while the user can reduce their time to recovery.

In summary, techniques have been presented herein that support a scalable, flexible, dynamic, and adaptive noise reduction system that may facilitate addressing various cloud

native application problems. When the first sign of trouble appears, such a system may automatically increase data storage on the fast tier storage for metric change point detection, anomaly detection, log pattern detection, log context detection, and trace or span graph anomaly detection such that MELT data can be used to reduce noise in log content processing. Aspects of the presented techniques employ a two-phase filtering mechanism comprising Edge Processors and Global Processors (to evaluate if an incoming request should be classified as noise or functional data that is useful for problem diagnosis), leverage automated ML facilities, and employ a Relevance score (that may be defined as a measure of the functional relevance of data (i.e., its usefulness for diagnostic purposes)).