GRANO: Interactive Graph-based Root Cause Analysis for Cloud-Native Distributed Data Platform

Hanzhang Wang*, Phuong Nguyen*, Jun Li, Selcuk Kopru, Gene Zhang,
Sanjeev Katariya, Sami Ben-Romdhane
{hanzwang,phuonnguyen,junli5,skopru,genzhang,skatariya,sbenromdhane}@ebay.com
eBay Inc.

ABSTRACT

We demonstrate GRANO[1], an end-to-end anomaly detection and root cause analysis (or RCA for short) system for cloud-native distributed data platform by providing a holistic view of the system component topology, alarms and application events. GRANO provides: a *Detection Layer* to process large amount of time-series monitoring data to detect anomalies at logical and physical system components; an *Anomaly Graph Layer* with novel graph modeling and algorithms for leveraging system topology data and detection results to identify the root cause relevance at the system component level; and an *Application Layer* that automatically notifies on-call personnel and presents real-time and on-demand RCA support through an interactive graph interface. The system is deployed and evaluated using eBay's production data to help on-call personnel to shorten the identification of root cause from hours to minutes.

PVLDB Reference Format:
Hanzhang Wang, Phuong Nguyen, Jun Li, Selcuk Kopru, Gene Zhang, Sanjeev Katariya, Sami Ben-Romdhane. GRANO: Interactive Graph-based Root Cause Analysis for Cloud-Native Distributed Data Platform. *PVLDB*, 12(12): 1942-1945, 2019.
DOI: https://doi.org/10.14778/3352063.3352105

1. INTRODUCTION

Accurately identifying the root cause of an incident in a timely manner is critical to maintain high availability to business for most information technology companies. However, such a task in a cloud-native distributed data platform poses a number of challenges. **(1) The complexity of architecture**: To achieve availability and scalability, the system often consists of thousands of different interdependent *logical* (e.g., database's keyspaces, each keyspace consists of multiple shards, and each shard is replicated across multiple data centers) and *physical* components on which the logical components are actually deployed (e.g., racks, hosts, pods spanning across multiple data centers). NuData, a geo-distributed database developed at eBay, is deployed on thousands of hosts across data centers in eBay's internal Kubernetes-based cloud infrastructure. Thus, it is time consuming and troublesome to navigate through the system components to identify the root cause. **(2) The large number of the system metrics and events**: Each system component is monitored with hundreds of metrics of different categories, such as latency, throughput, resource, and error. eBay's NuData monitoring system captures 20 million metrics per scrape interval per data center. In addition, a large number of application events are produced to provide operational insights of the system. Therefore, it's challenging to develop and maintain effectively a large number of detection modules to digest the large amount of metrics and events. Moreover, **(3) System metrics and events of distributed data platform are very dynamic and unpredictable**, thus it is difficult to achieve high detection accuracy of anomalies and root causes. In fact, high sensitive rule-based alerting mechanisms and machine learning-based detection models that are often trained with insufficient data[2] can produce a large number of false positive alerts, which makes it challenging for on-call personnel to handle. The aforementioned challenges contribute to the long time gap from detection to response. During incidents, on-call personnel usually need to scan through hundreds of different metric/event dashboards and alarms to identify root cause. This process can take up to hours of intense effort, and thus significantly affects the availability to business.

In this paper, we present GRANO, an interactive RCA system for cloud-native distributed data platform. First, GRANO consists of a detection layer that supports anomaly detection for a large number of system components and metrics. We take a human/analyst-in-the-loop approach for anomaly detection, and we implement the detection algorithms that expose model parameters that are intuitive to tune and provide predictable model performance. Second, the detection results, in combination with other rule-based alerts and application events, are projected onto topology of distributed data platform as a unified *anomaly graph*. Third, we introduce a novel graph-based root cause relevance algorithm that leverages the inter-dependencies between system components, severity of anomaly, and metric importance to

*Equal contribution

[1] GRANO is the combination of **GRA**ph and **ANO**maly. It also means grain in Italian and Spanish, or small piece of something.

[2] Lack of training data is a common challenge for a lot of anomaly detection systems. Very often, only normal data, or data with highly imbalanced class distribution are available.

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352105

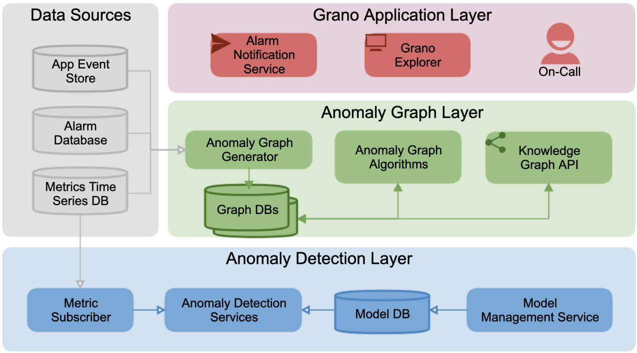


Figure 1: GRANO's system architecture

more accurately measure the relevance of each system component to the root cause and reduce the effect of false positive. Last but not least, the anomaly graph and relevance results are presented on an *interactive knowledge graph* interface to allow users to easily navigate, connect, and triage to identify the actual root cause.

Related Work. There is a large body of related work on the topic of anomaly detection on graphs [2], mostly focusing on the application-level graphs (e.g., social network, web, retail networks). To the best of our knowledge, there is no published anomaly detection work that use a graph-based approach for the distributed data platform. Furthermore, our end-to-end system leverages both logical and physical components to minimize the effect of false positives and identify the root cause more accurately. Other open-source anomaly detection systems [1] mainly focus on first-level detection. Root cause identification is also difficult to achieve using black-box machine learning models such as deep learning [4], due to the lack of the interpretability of the detection result. Cheng et al. [3] proposed a graph-based anomaly detection approach using a kernel matrix but without detection models as the first layer before pushing to the graph. Thalheim et al. [6] leverages call graph for anomaly detection. In GRANO, detection results are projected as new nodes on the anomaly graph of both physical and logical components for root cause identification.

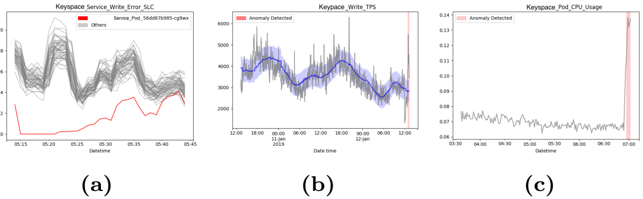
2. SYSTEM OVERVIEW

2.1 System Architecture

GRANO's system architecture is presented in Figure 1. GRANO consists of three main layers: anomaly detection layer, anomaly graph, and application layer.

The anomaly ***detection layer*** provides first level detection of anomalies for components and metrics of data platform. Detection models consist of machine learning-based and statistical anomaly detection models that are implemented as RESTful services. Metric subscribers periodically pull data from metric time series database and invokes detection models to detect anomalies. Detection events are created and dispatched to the alarm database. Model and feature database maintains model configurations and computed features used for online detection. Model management service provides a set of APIs and interfaces for managing model configurations, metric subscription, and easy onboarding of new detection models.

The anomaly ***graph layer*** provides the second-level aggregation of detection events and measures the *root cause relevance* (or RCR for short) of each component by leveraging system topology of distributed data platform. Anomaly graph generator constructs an anomaly graph that consists of all physical and logical system components and their interdependencies (i.e., represented as edges), alarms (including detection events produced by first level detection models and rule-based alerts), and application events. Each alarm or event is projected onto the anomaly graph as a node with an edge connecting it to the corresponding system component that triggered such alarm or event. The generated graph is stored in a graph database and is used by the anomaly graph algorithm to calculate the root cause relevance scores of the system components (to be described in Section 3). All the anomaly graph and relevance results can be retrieved via anomaly graph APIs.



(a) (b) (c)

Figure 2: Examples of detection use cases: (a) A service pod (red line) that behaves abnormally, compared to its peers (grey lines); (b) A keyspace's write TPS behaves abnormally from its historical pattern; (c) Spurious increase in CPU usage of a pod.

GRANO's ***application layer*** consists of an interactive frontend service, i.e., GRANO *Explorer*, that can be used by oncall personnel to interact with the anomaly graph to triage between system components and anomalies, and easily identify root cause using calculated RCR and knowledge graph. *Alarm Notification Service* sends out critical alerts through Slack with link to access detailed/enriched report of the alert and connect the alert to GRANO's Explorer tool for RCA.

2.2 Detection Models

We identify the most common anomaly detection use cases in distributed data platform (Figure 2). *First*, it is to identify outliers from a group of targets sharing **similar** behavior (Figure 2a). For this use case, we use density-based clustering algorithm to detect anomalies, as it does not require to specify the number of clusters and also exposes intuitive parameters to control the desirable density level of normal clusters. *Second*, it is to identify if a target behaves differently from what it **normally** does (i.e., its temporal pattern) on a given metric (Figure 2b). For this use case, we implement a detection model based on additive decomposition forecasting [5] to detect anomalies. In particular, a forecasting service is run in batched mode to produce forecasts of the target metric, and the online detection model compares the real-time metric data with the forecasts to detect anomalies. The forecasting model also exposes intuitive parameters to tune its flexibility to change points in data, and its preservation of the temporal pattern in data. *Third*, it is to identify if a metric behaves differently from its **recent** behavior (Figure 2c). Exponential smoothing-based detection model is implemented to detect anomalies in the third use case. The model exposes data smoothing factor to let analysts to easily tune how quick the model reacts to the systematic changes in the metric time series.

In order to easily scale out and onboard detection to a large number of targets and metrics, we use a *model inheritance* approach for model implementation and deployment.

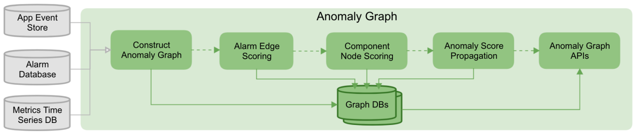


Figure 3: Graph-based root cause relevance procedure.

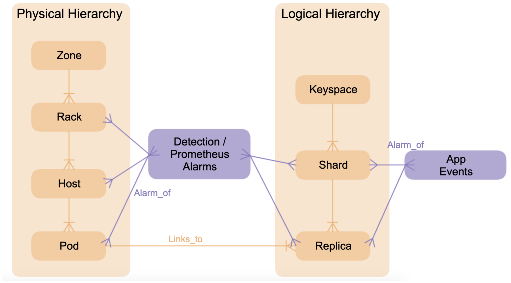


Figure 4: Example of distributed data infrastructure topology.

Specifically, we abstract the implementation of detection models, targets, and metric types. An actual instance of a model is onboarded simply by extending the base detection model with specific configurations of a target and a metric type.

3. GRAPH-BASED ROOT CAUSE RELEVANCE

In this section, we present in more details how our second level graph-based RCA approach could help to leverage the results from the first-level detection and provide better insights to system component's root cause relevance. The basic concept used by GRANO is to use graph modeling and propagation algorithm to measure the importance of detection events and minimize the effect of false positive alarms. The alarm's importance is then used to measure the root cause relevance of system components. Final results are presented on an interactive knowledge graph interface for easy identification of root cause and causality.

In the following, we explain in detail the step-by-step process (Figure 3) of measuring root cause relevance of system component using graph-based representation of system topology and alarm events.

Step 1 - Graph Construction: Our approach takes input as the detection alarms from the first level anomaly detection models, as well as rule-based alarms produced by real-time monitoring system[3] and application events. Given a time range, the set of alarms and events retrieved, denoted as $\mathbf{A}$, are then projected on the topology graph of the distributed data platform to form a unified anomaly graph denoted as $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V} = \mathbf{C} \cup \mathbf{A}$ with $\mathbf{C}$ being the set of system components, and each edge in $\mathbf{E}$ represents the interdependency between components (e.g., a container is connected to a host that the container is provisioned on) or the relationship between a component and an alarm.

Figure 4 shows an example of a common distributed data platform topology that we will use throughout this paper and demo scenarios. The graph's vertices consist of logical components, such as keyspaces, shards, and replicas, and physical components, such as zone, rack, host, and pod. Each keyspace corresponds to a database schema, and a keyspace can be splitted into multiple shards, each shard is replicated to multiple replicas for high availability. A replica is deployed as a physical pod on the distributed infrastructure, and multiple pods are located on the same host. Zone represents a data center that consists of multiple racks, each rack houses multiple hosts.

Each alarm event is associated with a *criticality*, denoted as ρ_a, $a \in \mathbf{A}$ [4]. An alarm "*edge*" $e(a, c)$ is created between an alarm a and its corresponding component c with a weight that represents alarm *severity*, denoted as $\sigma_{e(a,c)}$ (to be described).

Step 2 - Alarm Edge Scoring: The idea is to evaluate the alarm's importance to a connected system component. Since alarms may appear to different components with different severity and criticality, we calculate and assign a score for each edge between an alarm and a component. For each component, we treat the set of alarms it has during a given time interval as a "*bag-of-alarms*". Each alarm is measured by: i) Alarm's Severity: Based on how severe an alarm is for a particular system component during the given time interval; and ii) Alarm's Component (or Inverse) Frequency: Based on how frequent the alarm is across all components during the given time interval.

Let us consider an alarm edge $e(a, c)$ between alarm a and system component c. During a given time interval, c may trigger alarm a multiple times (e.g., due to the dynamic change of component metric) with different severity levels, denoted as $x_0, x_1, ..., x_t$, with $x_i (0 \leq i \leq t)$ being the severity at time i. To measure the severity of $e(a, c)$ (i.e., $\sigma_{e(a,c)}$) we aggregate different alarm severity levels during the time interval using exponential smoothing formula:

$$\sigma^0_{e(a,c)} = x_0$$
$$\sigma^t_{e(a,c)} = \alpha x_t + (1 - \alpha)\sigma^{t-1}_{e(a,c)} (t > 0)$$

Where α is the smoothing factor and the severity score calculated in the last iteration is then used as the final severity score of the edge $e(a, c)$.

The alarm edge score for the edge $e(a, c)$ between a and c, denoted as $\mathbf{s}_{e(a,c)}$, is then calculated as follows:

$$\mathbf{s}_{e(a,c)} = \sigma_{e(a,c)} \log(\frac{|\mathbf{C}|}{|\mathbb{C}_a|})$$

Where $\mathbb{C}_a$ is the set of system components that have triggered alarm a. The score $\mathbf{s}_{e(a,c)}$ is normalized to $(0, 1]$.

Step 3 - Component Node Scoring: After the previous step, all alarm edges are assigned with a score which reflects the importance of an alarm to a component. In this step, we calculate the aggregated confidence score on the components. The confidence score, denoted as $\mathbf{cs}$, is calculated using the criticality of the alarms and the edges' score that connected to a component c:

$$\mathbf{cs}_c = \sum_{a \in \mathbb{A}_c} \rho_a \mathbf{s}_{e(a,c)}$$

Where $\mathbb{A}_c$ is the set of all alarms that are connected to the component c.

Step 4 - Score Propagation: This step is designed to leverage system topology to detect the actual root cause. We design a customized algorithm to propagate confidence scores on the nodes. The algorithm propagates the confidence score on every node that is connected by at least

[3] We use rule-based alarms generated by Prometheus monitoring system.

[4] Alarm criticality is based on domain knowledge

one alarm to all its connected components. Then the connected components continues the propagation using its assigned score until reaching the end of the graph (Zone or Keyspace node). For the receiving node c that connects to other components $\mathbb{V}_c$ (including both alarms and system components), the propagated score $\mathbf{p}_c$ is calculated as follow:

$$\mathbf{p}_c = \begin{cases} \beta \mathbf{cs}_c & \text{if } |\mathbb{V}_c| = 1 \\ \gamma \frac{\sum_{\hat{c} \in \mathbb{V}_c} \mathbf{cs}_{\hat{c}}}{|\mathbb{V}_c|} & \text{if } |\mathbb{V}_c| > 1 \end{cases}$$

where β and γ are algorithm parameters that we set to 0.9 and $\log(|\mathbb{V}_c| + 1)$. All propagated scores will be added with the initial score to calculated the final RCR score. This score represents the overall relevance of this system node being root cause.

Step 5 - Result Presentation: We define the customized graph searches and scenarios to explore the graph. We return the results in a sorted and aggregated way. Therefore, the users are able to browse over the root cause, the alarm types, alarm frequency, and topology for the suspected incidents.

4. DEMONSTRATION DESCRIPTION

To demonstrate the functionality of GRANO, and to make it easy for analyst to inference the anomalies and identify the root cause, we have built the complete system in Figure 1 and a web-based front-end interface. We describe our demonstration scenarios in the following.

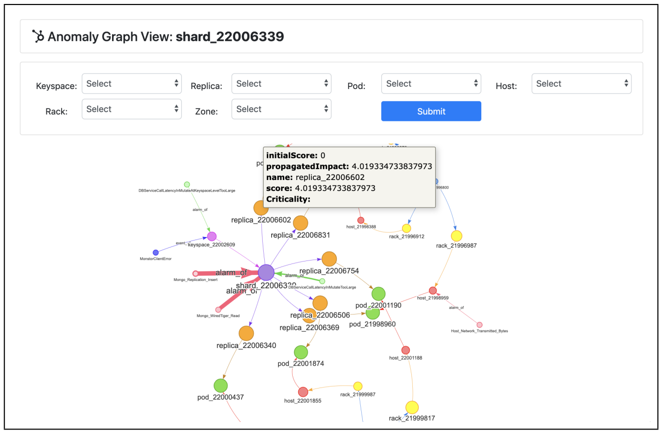
4.1 Manage Detection Model Life-cycle

For the first demo scenario, we will let attendees experience the process of onboarding new detection models on GRANO. Users will use GRANO's model management service interface and follow step-by-step process to onboard new detection model by selecting system metric and keyspace for the new model, and then review the model definition that is automatically generated by GRANO before submitting the final model definition. Once the new model is submitted, we will demonstrate how users can easily manage the life-cycle of the new model on GRANO by controlling model subscription and its parameters and observe the results via GRANO's detection event timeline. In particular, users will be able to update model definition, such as intuitive model parameters, detection interval, and subscription configuration, on-the-fly and observe how new update influences the detection results in real-time. From that, users can further experience how to interactively tune the model to achieve desirable performance via GRANO's model management service.

4.2 RCA with Knowledge Graph

In the second scenario, we will demonstrate how to use our Anomaly Graph to identify root cause using a real desensitized data. Starting from a critical alert, attendees will be able to use GRANO's enrichment tool to quickly understand more about the alert and affected keyspace.

Then, attendees will use the EXPLORER to retrieve *real-time* RCR scores of all system components (including both logical and physical components). In addition to RCR scores, the Explorer also presents other component scores, represented as "Initial" and "Propagated" (which are calculated from step **3** and step **4** in Section 3). On-call people can see the system prediction about the initial root cause relevance of each component and the propagated impact it receives from its connected components. The filter button can show the results only for the connected components (directly or indirectly though other different type of components) of any selected platform entity. The user can also select a specific time range to generate anomaly detection results and refresh.



Anomaly Graph View: **shard_22006339**

Keyspace: Select
Replica: Select
Pod: Select
Host: Select
Rack: Select
Zone: Select

initialScore: 0
propagatedImpact: 4.019334733837973
name: replica_22006602
score: 4.019334733837973
Criticality:

Figure 5: GRANO's Graph Explorer.

By clicking on a component to visualize a dynamic graph which retrieves all connected system components and alarms (rule-based and model detection) and app events of them. Figure 5 presented an anomaly knowledge graph example for a data shard component where different types of nodes filled by different colors and the size of a node is based on its RCR score (for system component nodes) or criticality (for alarm/event nodes). For the edges between the alarms and system components, the more alarm severity the thicker. Users can see the detailed scoring and related information of a node by moving the cursor over any node.

From the component's graph view, users can also *interactively* traverse the graph to search the root cause by select one or multiple targeted entities on the top and updated knowledge graph view.

5. CONCLUSIONS

We have presented an overview and demonstration descriptions of GRANO, the anomaly detection framework for cloud-native distributed data platform. The primary results demonstrate the usefulness of using GRANO to quickly detect anomalies and identify the root cause of system incidents. In the future, we would like to focus more on bridging the gap between component-level and system-wide anomalies, and integrating GRANO with automated remediation engine.

6. REFERENCES

[1] Anomaly detection projects. https://github.com/rob-med/awesome-TS-anomaly-detection. Accessed: 2019-03-15.

[2] L. Akoglu, H. Tong, and D. Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688, 2015.

[3] H. Cheng, P.-N. Tan, C. Potter, and S. Klooster. Detection and characterization of anomalies in multivariate time series. In *Proceedings of the 2009 SDM*, pages 413–424. SIAM, 2009.

[4] M. Du et al. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of CCS 2017*, pages 1285–1298. ACM, 2017.

[5] R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[6] J. Thalheim, Rodrigues, et al. Sieve: actionable insights from monitored metrics in distributed systems. In *Proceedings of the 18th Middleware Conference*, pages 14–27. ACM, 2017.