Fighting the Fog of War: Automated Incident Detection for Cloud Systems

Liqun Li and Xu Zhang, *Microsoft Research;* Xin Zhao, *University of Chinese Academy of Sciences;* Hongyu Zhang, *The University of Newcastle;* Yu Kang, Pu Zhao, Bo Qiao, and Shilin He, *Microsoft Research;* Pochian Lee, Jeffrey Sun, Feng Gao, and Li Yang, *Microsoft Azure;* Qingwei Lin, *Microsoft Research;* Saravanakumar Rajmohan, *Microsoft 365;* Zhangwei Xu, *Microsoft Azure;* Dongmei Zhang, *Microsoft Research*

https://www.usenix.org/conference/atc21/presentation/li-liqun

This paper is included in the Proceedings of the 2021 USENIX Annual Technical Conference.

July 14–16, 2021

978-1-939133-23-6

Open access to the Proceedings of the 2021 USENIX Annual Technical Conference is sponsored by USENIX.

Fighting the Fog of War: Automated Incident Detection for Cloud Systems

Liqun Li[‡], Xu Zhang[‡], Xin Zhao[§*], Hongyu Zhang[⋆], Yu Kang[‡], Pu Zhao[‡], Bo Qiao[‡],
Shilin He[‡], Pochian Lee[◇], Jeffrey Sun[◇], Feng Gao[◇], Li Yang[◇], Qingwei Lin[‡†],
Saravanakumar Rajmohan[○], Zhangwei Xu[◇], and Dongmei Zhang[‡]
[‡]*Microsoft Research,* [◇]*Microsoft Azure,* [○]*Microsoft 365,*
[⋆]*The University of Newcastle,* [§]*University of Chinese Academy of Sciences*

Abstract

Incidents and outages dramatically degrade the availability of large-scale cloud computing systems such as AWS, Azure, and GCP. In current incident response practice, each team has only a partial view of the entire system, which makes the detection of incidents like fighting in the "fog of war". As a result, prolonged mitigation time and more financial loss are incurred. In this work, we propose an automatic incident detection system, namely Warden, as a part of the Incident Management (IcM) platform. Warden collects alerts from different services and detects the occurrence of incidents from a global perspective. For each detected potential incident, Warden notifies relevant on-call engineers so that they could properly prioritize their tasks and initiate cross-team collaboration. We implemented and deployed Warden in the IcM platform of Azure. Our evaluation results based on data collected in an 18-month period from 26 major services show that Warden is effective and outperforms the baseline methods. For the majority of successfully detected incidents ($\sim$ 68%), Warden is faster than human, and this is particularly the case for the incidents that take long time to detect manually.

1 Introduction

Reliability is a key quality attribute of large-scale cloud systems such as AWS, Azure, and Google Cloud Platform (GCP). Although tremendous effort has been devoted to improving reliability, cloud systems still suffer from incidents and outages [8, 10, 11]. Monitoring is widely used by cloud systems to check their runtime status. A monitoring system collects, processes, and aggregates quantitative data about a cloud system. When a system problem occurs, an alert is sent to an on-call engineer, who is expected to triage the problem and work toward its mitigation. A severe enough alert (or a group of aggregated alerts) is escalated as an *incident*.

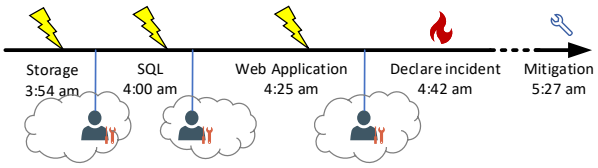

Figure 1: The timeline of handling a cross-service incident, where it took nearly 50 minutes to understand the situation and declare the incident.

Timely incident management is the key to reduce the system downtime. However, according to our experience, a reactive and ad-hoc incident detection is often employed in practice, which hinders effective incident management. We motivate our work using a real-world example. Fig. 1 shows the timeline of an incident caused by a flawed configuration change in the Storage service. The failed storage accounts affected several SQL databases, and the failure was further propagated to Web Application instances that were depending on the impaired databases. The failure triggered cascading alerts for Storage, SQL, and Web Application at 3:54 am, 4:00 am, and 4:25 am, respectively. After many rounds of discussions (which took nearly 50 minutes), the engineers finally realized that this was a cross-service issue, and an incident was declared. An experienced Incident Commander (IC) [4, 6] was then engaged to coordinate the mitigation process. Eventually, at 5:27 am, the incident was mitigated and all services were back to normal.

In this work, we focus on automatic incident detection for cloud systems, so that incidents could be declared and mitigated as early as possible. Incident declaration turns the mitigation process from chaotic to managed, especially for issues that require cross-team collaboration: the mitigation tasks are prioritized, teams are aligned, and customer impact is reduced. Our study is based on the Incident Management (IcM) platform of Microsoft Azure, one of the world-leading cloud computing platform. Though different companies [1–3]

[*]Work done during Xin Zhao's internship at Microsoft.
[†]Qingwei Lin is the corresponding author of this work.

implement slightly different incident response processes, we argue that they are essentially similar.

Incident detection is challenging. For a cloud system with extraordinary complex dependency among thousands of components, the on-call engineers, like in the fog of war, usually only have a partial view of the big picture. In the above-mentioned example, the engineers handling the storage alert could only see the affected storage accounts, but it was time-consuming to understand who was impacted. For engineers in the SQL and Web Application teams, it also took them a considerable amount of time to understand that they were affected by the underlying services. According to our interview with on-call engineers from the several service teams, such a diagnostic process could take as long as an hour. Also, there were a large number of concurrent alerts, making the situation even more challenging. Better communication processes/protocols can help, but will not solve the problem completely.

In this work, we propose Warden for effective and timely incident detection in IcM. Warden detects alerts that can potentially turn into incidents by employing a data-driven approach. A simple rule-based approach with human knowledge is insufficient in practice as it can be easily overwhelmed by a massive number of complex and ever-changing rules in a large-scale production cloud system. An example rule is "once a Storage alert was immediately followed by a SQL alert and they happened in the same datacenter, it is likely that they were related and constituted an incident".

We develop a machine learning model to detect the incidents, and point out incident-indicating alerts for the on-call engineers. We have evaluated Warden using data collected in an 18-month period from 26 major services on Azure. According to our experimental results, Warden is effective and outperforms the baseline methods. Warden is also faster than human for the majority of successfully detected cases ($\sim 68\%$).

Our major contributions are summarized as follows:

- We study the problem of incident detection in the incident management (IcM) platform of Microsoft Azure. We have identified the obstacles that result in prolonged incident declaration time.
- We propose Warden, a framework to automatically detect incidents based on alerts in IcM. Warden quickly detects incidents and assists in task prioritization and cross-team collaboration.
- We evaluate Warden with real-world data collected from 26 major services on Azure. The evaluation results confirm the effectiveness of the proposed approach. We have successfully deployed Warden in IcM.

2 Background and Problem Formulation

In this section, we briefly describe the basic concepts about alerts and incidents and then formulate the incident detection problem. To help explain the concepts and the problem, we perform an empirical study of 5 services (named Big5) of Azure, namely Compute, Storage, Networking, SQL DB, and Web Application. These 5 services are common to almost all cloud computing platforms.

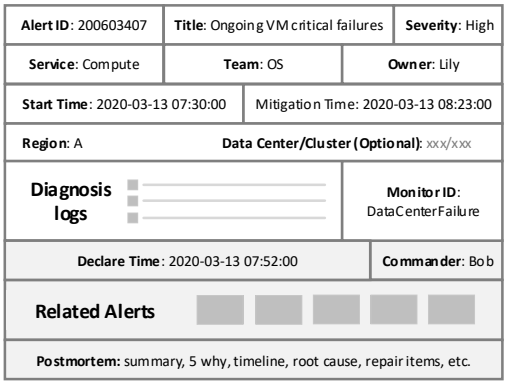

Figure 2: Main fields of an alert. Additional fields (grey colored) are appended when an alert is escalated into an incident. The escalated alert becomes the primary alert of the incident, while related alerts are manually linked.

2.1 Alerts and Incidents

Alerts represent system events that require attention, such as API timeouts, operation warnings, unexpected VM reboots, or network jitters. An alert consists mainly of fields shown in Fig. 2. Each alert has an impact starting time. Its mitigation time is filled when the problem is fixed. Alerts have different severity levels - low, medium, and high. Alerts are reported by monitors, which are continuously running programs that keep tracking the health status of a certain aspect of a service component. Each alert carries its monitor ID. Most alerts has its region information and some with more fine-grained location information such as datacenter or cluster.

Figure 3 shows the number of alerts per day for Big5 services in an one-year time frame. There are tens of thousands of concurrent alerts produced by a large number of components in the cloud system. Even for high severity alerts, the number could be hundreds or even several thousands. Figure 4 shows the number of active monitors per month. The number is gradually increasing as the platform scaling up and being improved.

In general, incidents are declared under severe situations. Often, problems taking a long time to solve or requiring cross-team collaboration are also declared as incidents. One incident usually triggers correlated alerts [21, 53]. An incident is thus

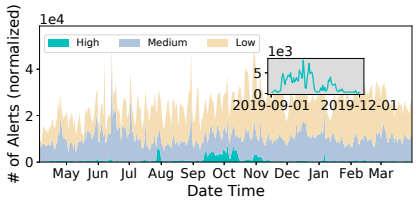

Figure 3: Number of alerts of different severity levels for the Big5 services.

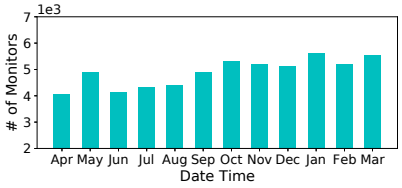

Figure 4: Number of monitors of the Big5 services actively reporting alerts per month.

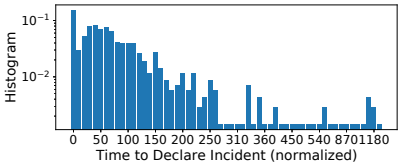

Figure 5: The histogram of time to declare incidents for the Big5 services.

escalated from one alert or a group of correlated alerts. Additional fields are added when alerts are escalated into incidents as shown in Fig. 2. The *primary* alert of an incident is usually the one closest to the root cause or with the earliest alerting time. The incident commander and the on-call engineers manually link related alerts with the primary alert based on expert knowledge. In the one-year-length period, there were several hundreds of incidents caused by the Big5 services, among which $\sim 67\%$ are incidents impacting more than one services. These incidents are typically the hard ones for detection and mitigation due to the partial view issue.

Fast incident detection is critical for ensuring timely incident management. The time to declare incidents takes around one-third of the whole mitigation time in our study. Figure 5 shows the histogram of time to declare incidents for the Big5 services. We normalized the time to declare incidents in Fig. 5 to protect company sensitive data. The reader could use the public incident reports [8, 10, 11] as a reference for the incident mitigation time. About one-third of the incidents are declared within minutes. Most of these incidents are single service issues that could be straightforwardly detected based on rules. Nearly half (47.6%) of the incidents take more than 30 NTUs (normalized time units) to declare, and a long tail of incidents require even hours. We manually analyzed these cases and found that most of these cases fall into the category where a small issue slowly turns into a big one. Our proposed approach is thus helpful by detecting incidents from a global perspective.

2.2 Problem Formulation

We now formally define the problem of incident detection. We denote the current wall clock as t. The input is the set of alerts coming from different services within the time window $\mathcal{W}_t$ whose duration is $(t-w,t]$, where w is the length of the window. One monitor can be triggered many times over its lifetime. The series of alerts reported by any monitor m is called an *alert signal*, denoted as s^m.

From the input time window, we want to estimate the probability of $P_t = P(I_t|\mathcal{A}_t)$, where $I_t \in \{0,1\}$ is the indicator of incident. $\mathcal{A}_t$ is the observed set of alert signals in the time window that ends at t. We then infer that an incident has occurred by examining whether P_t surpasses a threshold.

In addition, we need to extract a subset of alert signals which are indicating the incident from the time window, denoted as $\hat{\mathcal{A}}_t \subset \mathcal{A}_t$. System failures often lead to correlated alert signals. The physical meaning is that each alert signal $s^m \in \hat{\mathcal{A}}_t$ therein is one symptom caused by the underlying incident. We name $\hat{\mathcal{A}}_t$ as *incident-indicating* alert signals. According to our definition, we shall have $P_t = P(I_t|\mathcal{A}_t) = P(I_t|\hat{\mathcal{A}}_t)$.

In summary, we divide the incident detection problem into two steps. The first step is incident detection based on the observed alert signals from different services in a recent time window. The second step is to understand which alert signals are likely caused by the detected incident, i.e., identify the incident-indicating alert signals. When we detect the incident, we only notify the relevant on-call engineers. We describe the details of our solution in the next section.

3 The Proposed Approach

The workflow of Warden is shown in Fig. 6, which consists of the following major steps:

Alert signal selection Monitoring a very large-scale system is challenging due to the sheer number of components being watched. The raw alert data is not only large in volume but also contains noise. Therefore, we select only a subset of monitors which exhibit relatively strong association with incidents. Details about alert signal selection could be found in Sec. 3.1.

Incident Detection We adopt a data-driven paradigm for incident detection. We carefully extract a set of features from alerts in a recent time window based on domain knowledge in Sec. 3.2.1. Then, the feature vector is fed to a Balanced Random Forest (BRF) [37] model constructed offline. With this model, we could detect the occurrence of incidents. Section 3.2.2 explains how we train and apply the model.

Incident-indicating alerts identification We developed a novel approach to identify the incident-indicating alerts. We start by dividing the alert signals into groups. Then, we assign

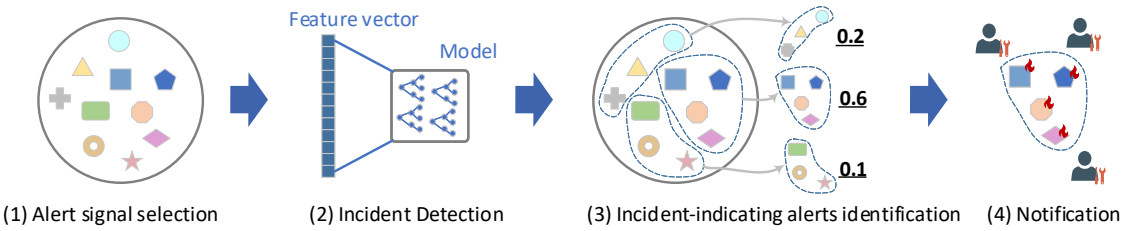

Figure 6: (1) Recent alert signals in a time window are selected and cached. (2) A feature vector is extracted and fed to a model for incident detection. (3) We extract a group of alert signals which most significantly indicate the detection result. (4) Relevant on-call engineers are notified for further investigation.

a score to each group based on an algorithm of our design called Group Shapely Value (GSV). The group of alert signals with the highest importance score is identified as the incident-indicating alerts. We explain the grouping and group-based model interpretation in Sec. 3.3. This approach is inspired by model interpretation [43], which is about understanding the relationship between inputs and outputs of a machine learning model. However, our goal is to extract a group of alerts indicating the detection result as shown in Fig.6 (3). The difference between incident-indicating alert identification and model interpretation is discussed in Sec. 5.

Notification We integrated our system into the IcM of Azure. A notification is pushed to all involved alerts when a potential incident is detected. When the corresponding on-call engineers are aware of the detected emerging issue, they could better understand the big picture and form a collaboration group to declare and resolve the incident. We introduce the practical usage scenario in Sec. 3.4.

3.1 Alert Signal Selection

Feeding alerts from all monitors could overwhelm the detection model. Figure 3 and Figure 4 show the number of active monitors and produced alerts for the Big5 services, respectively. There are even more monitors and alters considering other services. In contrast, the number of incidents is relatively small.

The purpose of alert signal selection is to identify a subset of monitors which emit alerts exhibiting a relatively high correlation with incidents. For this purpose, we calculate a score for each monitor. The higher the score, the more likely the alert signal from this monitor will indicate that the cloud is experiencing an incident. If we observe that the alerts from a certain monitor co-occur with incidents frequently, we should assign a high score to that monitor. With this intuition, we adopt the Weighted Mutual Information (WMI) [26]. WMI is a measure of the mutual dependence between two variables, i.e., quantifying the "amount of information" obtained about one random variable through observing the other random variable [50].

Formally, denote I as the set of all incident. For each monitor m, we could calculate the WMI between the alert signal s^m and all incidents, i.e., $I(s^m; I)$. One monitor is usually only effective in detecting a specific problem. However, there are many different types of incidents. Therefore, calculating the WMI of one monitor against all incidents is not appropriate in characterizing its effectiveness, especially given one incident type may have far more cases than another type. Yet, a minority incident type may be equally, if not more, important than a majority type.

We divide all incidents into different subtypes based on their owning team that handled this incident. We notice that, in a production organization, each team usually focuses on specific functionalities [17, 18, 30]. Therefore, we assume that all incidents handled by the same team are similar, and therefore, constitute a subtype of incidents. We use $i \subset I$ to denote the a subtype of incidents. Then, the WMI between s^m and i could be calculated in Eq. (1).

$$I(s^m; i) = \sum_{s^m} \sum_{i} w(s^m, i) \cdot P(s^m, i) \cdot \log \left(\frac{P(s^m, i)}{P(s^m) P(i)} \right) \quad (1)$$

Both s^m and i are series of events with $s^m, i \in \{0, 1\}$. Eq. (1) essentially elaborates all value combinations of the two variables and examines their inherent dependence. We divide historic data into windows in order to calculate Eq. (1). For instance, to derive the joint probability $P(s^m = 1, i = 1)$, we simply count the number of windows containing both events. This value is then divided by the total number of windows. Other joint and marginal probabilities could be derived in a similarly way. The coefficients $w(s^m, i)$ in Eq. (1) are used to assign different weights to different value combinations.

Once we have the WMI for every pair of monitor and incident subtype, we can calculate the sum of score for m by going through all incident subtypes. We use this score to rank the monitors and pick the top 150, which are most effective for incident detection. We will evaluate the number of selected monitors in Sec. 4.6.2. The alerts not reported by the selected monitors are ignored by our system.

3.2 Incident Detection

We formulate the incident detection problem as a binary classification problem. The input consists of alert signals in a time window. A classifier is trained offline based on the historic labelled samples. Then, we apply the trained model for online incident detection.

3.2.1 Feature Extraction

We identify the following features from alerts generated by selected monitors in a time window:

Alert signals: The first set of features characterize the alert signals. Each alert signal is a series of alerts generated by a certain monitor within the time window.

- Alert count. In the event of a malfunction, the alert signal is typically stronger than usual. As a result, we count the total number of alerts and the high-severe alerts for each monitor, respectively. Multiple services may start to report alerts, and we, therefore, count the number of alerts from each service. Besides, we count the total number of alerts, the total number of monitors reporting high-severe alerts, and the total number of services, respectively.

- Alert burst. We adopt a time window length of 3 hours[1]. Our window size is large enough to accommodate related alerts. However, under certain conditions, alerts erupt only in a small time and space. For instance, a partial power failure may hit only a few racks in a datacenter. To capture this feature, we calculate the max number of monitors reporting alerts and the max number of high-severity alerts in any datacenter, respectively. Likewise, in the temporal dimension, we calculate the two features for all child sliding windows of 10/30/60 minutes in length, respectively.

Engineer activities: Engineers leave their footprints when they are working on alerts in IcM. The second set of features characterize the intensiveness of engineer activities. Engineers could post their diagnosis logs on the alert page. They could also launch a bridge meeting for online discussion. The bridge meetings are hosted by IcM and attached to the alerts. If people in one service want to reach out to another service, they could use the notification tool in IcM which automatically routes them to the on-call team in the other service. Though private communications (via personal phone calls or IM apps) still exist, most traffic today is through IcM.

- Diagnosis log count. The engineers tend to leave diagnosis logs on the alert web pages as well as in the bridge meetings. We count the number of discussion posts for each alert and the number of concurrent bridge meetings in IcM, respectively.

- Notification count. Notifications are sent to initiate cross-team collaborations. We thus count the number of notifications within the time window.

[1] According to our empirical study, over 80% alerts are triggered within 3 hours after the impact start time.

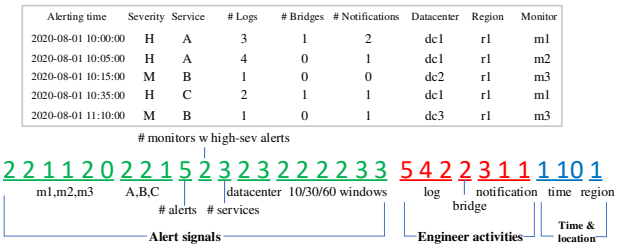

Figure 7: An example of feature extraction with 5 alerts in the time window generated by 3 monitors from 3 services. One feature vector is constructed for each time window.

In addition to the aforementioned features, we also include features about time (day and hour) and location (Region ID). We illustrate the process of feature extraction using a dummy example of 5 alerts in Fig. 7. We show a few example columns of the derived feature vector.

3.2.2 Model Construction

We construct data samples using alert data with a sliding window to training an incident detection model. The window size is set to 3 hours as discussed in the previous section. The window is sliding with a stride length of 5 minutes, which is small enough in order not to miss any significant changes. The label of the sliding windows overlapping with incidents is positive; otherwise, negative. The labeling methodology is illustrated in Fig. 8. The two vertical bars represent the impact starting time and the mitigation time of the incident, respectively. The example sliding window (i.e., the shadowed rectangle) in Fig. 8 is thus labeled as positive. The 5-minute stride length is only used for offline model construction. For online usage, Warden runs once every minute. More details could be found in Sec. 3.4.

Not all alert signals in the window are part of the incident. In Fig. 8, only the alerts marked by the dashed curve are symptoms of the incident. This group of alert signals is $\hat{\mathcal{A}}_t$ according to our definition in Sec. 2.2. In our evaluation, we use the links between alerts to obtain the relationship between an incident and its related alerts. The links were manually labelled by on-call engineers or the incident commander during the incident mitigation process. We note that the related alerts are only available for historic data. For online detection, this information is unknown (or partially unknown).

Once we have the extracted features and the labels, we train a model and apply it for online incident detection. In our

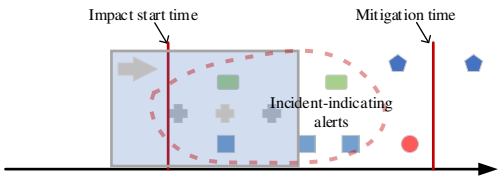

Figure 8: The sliding window is labelled as positive if any incident impact start time falls into it. Those time windows with no overlap with incident impact time ranges (from impact start to mitigation) are labelled as negative.

context, the number of incidents is much smaller than that of alerts. We therefore choose the BRF [37] classifier because of its robustness to noise and ability to handle imbalanced data. Warden is a generic framework, and any model could be plugged in for the classification task. More details about the model selection are presented in Sec. 4.6.1.

The output of the model is the probability $P_t \in [0.0, 1.0]$. We use a confidence threshold as discussed in our problem formulation in Sec. 2.2. If the confidence is higher than the threshold, we believe a potential incident is happening.

3.3 Identifying the Incident-indicating Alerts

We need to inform the right people when a potential incident is detected, and otherwise, it is non-actionable. In other words, we need to find out which alerts in the window are related to the detected incident, as illustrated in Fig. 8. After we identify these incident-indicating alerts, we notify their responsible on-call engineers.

It is a non-trivial task given many concurrent alert signals in the window. We propose a novel approach inspired by two intuitions: first, the target alert signals must be a group of correlated alert signals due to the same reason; second, the target alert signals should contribute significantly to the detection result. Such a group of correlated alerts is indicative to the detected incident.

Our approach has two steps. We first group alert signals that are likely related to each other into signal groups. Then, we interpret the incident detection model on a group basis.

3.3.1 Alert signal grouping

Correlated alert signals could be identified based on statistics on their co-occurrence history. We put two alert signals into the same group under one of the two conditions: (1) they co-occur frequently in history; (2) they fire from the same cluster.

An alert signal is a series of events. We thus first characterize the correlation between a pair of alert signals as we did for monitor selection in Sec. 3.1. We use a threshold to tell if one is strongly correlated with another. In addition, we conduct statistics on related alerts for historic incidents. For each incident, people manually link related alerts. If two alert signals are frequently linked in history (more than 3 times), we believe that they are correlated if they present in the same time window.

Then, if two alert signals fire in the same cluster, we believe that they are correlated. In a large-scale cloud computing platform like Azure, a cluster is usually dedicated for a certain functionality (e.g., for storage, compute, or database). If two alert signals emerge from the same cluster in a short time frame, they are inherently related in a high probability.

We apply the aforementioned rules to group alert signals in the current time window. After grouping, the alert signals are divided into corresponding groups, as illustrated in Fig. 6.

3.3.2 Group-based Model Interpretation

We now discuss how to obtain the incident-indicating alerts. The basic idea is to rank the signal groups according to their contribution to the prediction result of the detection model. Then, we pick the top group of alert signals.

For this purpose, we develop a novel algorithm called Group Shapely Value (GSV). GSV is heavily inspired by the Shapely Value method [47] which is a model-agnostic interpreter.

Algorithm 1 Group Shapely Value (GSV)

Require:
The classification model, M;
A testing window, $\mathcal{W}_t$;
The alert signal groups set associated with $\mathcal{W}_t$, $\mathcal{G}_t$;
The target group, $g \in \mathcal{G}_t$;
Background training data, B
Sampling rounds, n

Ensure:
The Shapely Value c for target group g;

$\varphi = 0$
for 1 to n **do**
  Randomly select a subset $\mathcal{G}', \mathcal{G}' \subseteq \mathcal{G}_t$
  $\mathcal{G}_{other} = \mathcal{G}_t \setminus (g \cup \mathcal{G}')$
  Randomly select a sample $x_b \in B$
  $b_1 = x_t(\mathcal{G}_{other}) \vee x_t(g) \vee x_b(\mathcal{G}')$
  $b_2 = x_t(\mathcal{G}_{other}) \vee x_b(g) \vee x_b(\mathcal{G}')$
  $\varphi = \varphi + M(b_1) - M(b_2)$
end for
$c = \frac{\varphi}{n}$

The original Shapely Value method only estimates the importance for each feature value. In contract, GSV is able to interpret the model for each alert signal group.

The intuition behind GSV is that we assemble combinations of alert signal groups to evaluate the contribution from each individual group. We leverage the Monte Carlo approximation method [47] to overcome the combination explosion

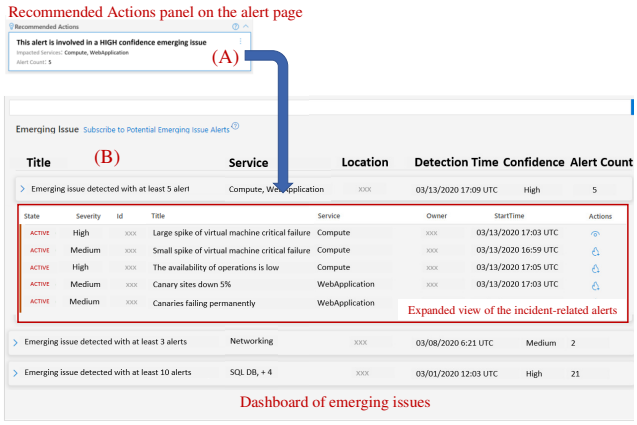

Figure 9: Web-based UI of Warden in IcM. (A) shows the "Recommended Actions" panel and (B) shows the "Emerging Issues" dashboard.

problem. The details of GSV can be referred in Algorithm 1. The input of the algorithm are the model M and the set of alert signal groups in the time window denoted as $\mathcal{G}_t$ obtained in the previous grouping step. GSV requires a set of background data B, which is randomly sampled from the training data. n is a hyper-parameter which is typically set to several hundred.

In Algorithm 1, we construct two synthetic samples b_1 and b_2 in each iteration, as shown in Line 6 and Line 7, respectively. Then, we calculate their difference in prediction probabilities by the model M. The operator $x(\cdot)$ retains only the feature values from a subset of alert signal groups of the testing sample x. Intuitively, the difference between b_1 and b_2 indicates the significance of the current value of g (i.e., $x_t(g)$) given a background sample $x_b \in B$. The iteration continues for n rounds and the accumulated φ is normalized to c. We use c as the score for the group g. We pick the group with the highest score as the incident-indicating alert signals.

3.4 Using Warden in Practice

To facilitate using Warden in practice, we develop a Web-based UI as shown in Fig. 9. The UI consists of two parts: (A) a "Recommended Action" panel located on each alert page, and (B) a dashboard of "Emerging Issues" for all detected incidents. Each detected incident is with a list of incident-indicating alerts.

Warden runs periodically at a fixed interval (in our practice, Warden runs once every minute, and the main task takes around 20 seconds). When Warden detects a potential incident, a notification is pushed to the "Recommended Action" panels of all identified incident-indicating alerts. Our notification is basically saying: *this alert is of high priority and it may be part of an emerging incident. Please check other alerts are likely triggered by the same issue as well.* When the engineer sees the notification pops up in the "Recommended Action" panel, she or he could click to jump to the dashboard for more details. The list of related alerts on the dashboard helps the engineers to understand the big picture, prioritize their work, and initiate cross-team collaboration. We will describe real-world cases in Sec. 4.7.

4 Experimentation

In our evaluation, we aim to answer the following research questions:

- RQ1: How effective is Warden in detecting incidents?
- RQ2: How fast can Warden detect incidents compared with human?
- RQ3: How accurate can Warden extract incident-indicating alert signals?
- RQ4: What is the impact of key system settings on incident detection performance?

4.1 Dataset

To evaluate Warden, we collect alert data (∼240G) from the IcM of Azure within a 18-month-length period, starting from Oct.2018. We carefully selected 26 major services of Azure, including the Big5. Hundreds of people in dozens of teams are behind each service. The total number of incidents reported by the 26 services accounts for ∼ 72% of all incidents in Azure. The dataset includes over 10 million alerts. We use the data in the last two month for testing and the previous months for training. Our training and testing data contain around 82% and 18% of all incidents, respectively. We organize our dataset into sliding windows and conduct data labeling as discussed in Sec. 3.2.2. The positive-to-negative ratio is around 1:25. More details about our dataset could be found in Table 1.

	Training	Testing
Low-severity alert	31.4%	7.8%
Med-severity alert	51.6%	6.3%
High-severity alert	2.6%	0.3%
Incident	82.2%	17.8%

Table 1: Details about the dataset for experimentation.

4.2 Evaluation Metrics

Warden runs on a fixed interval. Once a potential incident is inferred, Warden notifies related on-call engineers. The metric we care most about is the accuracy of incident detection and incident-indicating alert identification. For incident detection, we calculate the precision, recall, and F1-score. Precision measures the ratio of the sliding windows identified by Warden are truly containing incidents. Recall measures

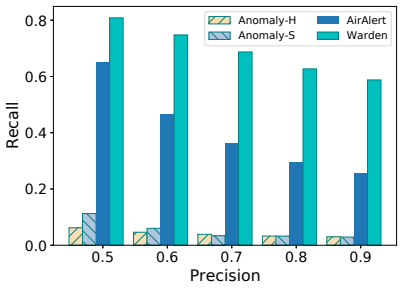

Figure 10: The recall of Warden and three baselines recorded with precision from 0.5 to 0.9.

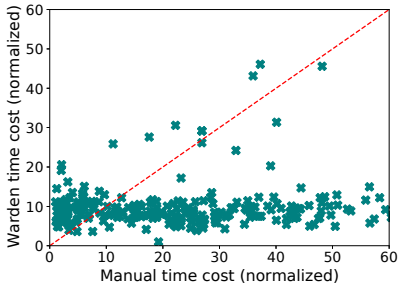

Figure 11: The time taken by manual incident declaration versus that by Warden.

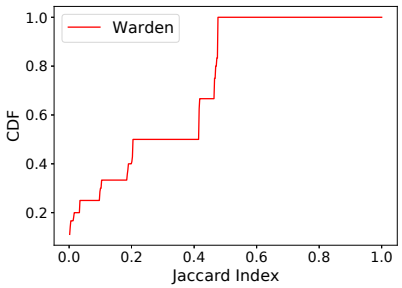

Figure 12: The CDF of Jaccard index between the owning services of the identified alerts and the actual impacted services.

the ratio of positively labeled windows recognized by Warden. F1-score is the harmonic mean of precision and recall. We also calculated AUC-PR in model selection, which is the area under the precision-recall curve. For incident-indicating alerts identification, we use Jaccard Index [33] to measure the difference between the identified group of alerts versus the manually linked alerts.

4.3 RQ1: Incident Detection Effectiveness

To our best knowledge, Warden is the only deployed system for incident detection based on monitor reported alerts in large-scale cloud systems. To better evaluate the effectiveness of incident detection, we compare Warden with the following baselines:

Anomaly detection Anomaly detection is a widely used approach to discover system faults. An incident is detected if the actual number of alerts is significantly higher than the prediction. We implement our baselines based on Prophet [49] from Facebook, which is widely adopted for time-series forecasting. The first baseline, namely **anomaly-H**, is evaluated on high-severity incidents; the second, namely **anomaly-S**, is evaluated on incidents reported by our selected monitors.

AirAlert AirAlert [23] uses a XGBoost [20] model to predict outages with only the alert count features from each monitor. We implemented AirAlert with input from our selected monitors.

The comparison results of Warden with the three baselines are presented in Fig. 10. There is a trade-off between precision and recall for a certain model. As discussed in Sec. 3.2.2, we use a confidence threshold to tune our model. In Fig. 10, we tune the models to change their precision to different values ($0.5 \sim 0.9$) and record the corresponding recall values. The x-axis of Fig. 10 is the precision and the bar charts show the corresponding recall values. The results show that Warden is effective and outperforms the baseline methods. The anomaly detection based approaches achieve very low recall. The key reason is that an incident does not necessarily cause a spike, especially given the large volume of alerts. Only those incidents with extremely large impact (usually referred to as outages) could lead to high peaks. The anomaly-S achieves slightly higher recall than anomaly-H, which demonstrates the effectiveness of our monitor selection. AirAlert achieves better performance than the anomaly detection based approaches. As discussed in Sec. 3.2.1, we identified more features in addition to alert count. With the full set of features, Warden significantly outperforms AirAlert. This evaluation result justifies the effectiveness of feature engineering.

False alarms could be annoying to deal with in real-world scenario, as they can cause incorrect task prioritization or unnecessary communication. To avoid spamming the engineering team with false alarms, we ensure a precision of above 90%, which corresponds to a recall of $\sim 58\%$. The achieved F1-score is 0.71.

4.4 RQ2: Fast Incident Detection

We compare the detection time using Warden and the manual incident declaration time for incidents in our test dataset. For all successfully detected incidents, Warden is able to perform detection faster than human in $\sim 68\%$ of cases. The time saving for an incident is measured between the notification time reported by Warden and the incident declaration time reported by the on-call engineers. The median time saving for these cases is 21.8 NTUs, which accounts for $\sim 15\%$ of the whole time-to-mitigate for these incidents. The readers could use the public incident reports [8, 10, 11] as a reference for incident mitigation time.

Figure 11 compares the time taken by manual incident declaration and that by Warden. Each cross in the figure represents one case in our testing dataset. It is clear Warden beats human for majority of cases. Moreover, we can see that Warden can particularly help reduce the incident detection time for those cases performed poorly by human. In practice, for those cases when Warden falls behind human, we simply

ignore them without sending notifications. To make the figure more compact, we only show data points within 60 NTUs.

4.5 RQ3: Accuracy of Extracting Incident-indicating Alert Signals

It is tricky to evaluate how accurately can Warden extract the incident-indicating alert signals from the current time window. Our ground truth is based on manually linked alerts in incidents. As discussed in Sec. 2.1, the incident commander and on-call engineers manually link related alerts to the primary alert of an incident. Since this is done manually, it is easy to miss some related alerts. According to our empirical study, usually, only partial alerts are linked during a failure event. Therefore, it is infeasible to rely on the manual links for evaluation. However, we discovered that the impacted services are more reliable. When an incident occurs, more than one alert usually gets triggered for each impacted service. People usually selectively pick one or a few alerts and add to the primary alert, which are used to track the health status of that service. As a result, we use only the group of impacted services, instead of the linked alert signals, for evaluation.

We extract the incident-indicating alerts from the current time window as described in Sec. 3.3. For evaluation, we compare the owning services of the identified alert signals with the actual impacted services. Figure 12 shows the CDF of the Jaccard indices for the cases recalled by Warden in our testing dataset. First of all, we can see that Warden achieves a perfect alignment with the ground truth for 53% cases. For 78% cases, the Jaccard index is above or equal to 50%. Moreover, the curve in Fig. 12 is constantly above 0, which means that Warden can find at least one impacted service when incidents have been detected. In general, Warden can accurately find the impacted services inside the current time window.

4.6 RQ4: The Impact of Key System Settings

So far we have evaluated our system with a fixed set of settings. We now address the question of how different system settings affect the performance. Specifically, we examine three sub-questions: (1) What is the impact of different classification models on incident detection? (2) how many monitors should we select? and (3) how much data and how often is required to train our model?

4.6.1 Classification Model Selection

Warden uses a classification model for incident detection. We evaluated a few popular candidates, namely Penalized Linear Regression (PLR) [23], SVM [24], Balanced Random Forest (BRF) [37], and LightGBM [36]. To account for data imbalance, we assign different class weights to positive/negative samples for PLR, SVM, and LightGBM; for BRF, we adjust the sampling rate. These settings are all tuned with grid search using validation data. The results are shown in Table 2.

Model	F1	Recall	Precision	AUC-PR
PLR	0.505	0.361	**0.840**	0.64
SVM	0.604	0.655	0.561	-
LightGBM	0.680	**0.658**	0.704	0.71
BRF	**0.704**	**0.658**	0.757	**0.73**

Table 2: The comparison results of different models.

We mainly use the AUC-PR for comparison. Since AUC-PR does not apply for SVM, we also record the F1-score and the corresponding precision and recall. BRF achieves the highest AUC-PR of 0.73 compared with PLR and LightGBM. For BRF and SVM, we could see BRF achieves higher precision and recall. In conclusion, BRF outperforms the other candidates, and therefore we adopt BRF in our system.

4.6.2 Alert signal selection

In Sec. 3.1, we have derived a score for each monitor and ranked all monitors accordingly. One remaining question is how many monitors should be selected. Intuitively, selecting more monitors will improve the coverage by detecting more incidents. However, more monitors also increase the complexity of the model as well as the data volume. Therefore, we try to find the minimum set of monitors while achieving satisfactory precision and recall.

We measure the recall at different precision values (0.6, 0.7, and 0.8) varying the number of monitors. The results are presented in Fig. 13, where the x-axis denotes the number of monitors and the y-axis shows the recall. The three curves are the recall values with different precision values, respectively. The number of monitors is ranging from 25 to 300.

We can see that the recall rises significantly for all three curves, when the number of monitors increases from 25 to 150, which is in line with our expectation. This observation indicates that a moderate fraction of incidents could be detected with a relatively small number of monitors. Then, increasing the number (when exceeds 150) would not increase the coverage as the newly-added monitors become less indicative of incidents. On the other hand, adding more monitors will increase the number of alerts to process. As a result, we set the number of selected monitors to 150 in our system.

4.6.3 Training data and frequency

We also evaluate our model with different training data lengths. Specifically, we keep the later two months in our dataset for testing and vary the length of training data from 1 to 16 months. For each round, we record the F1 score and the result is shown in Fig. 14.

The x-axis shows the length of training data from 1 to 16 months, and the y-axis is the F1 score. We can see that F1 rises

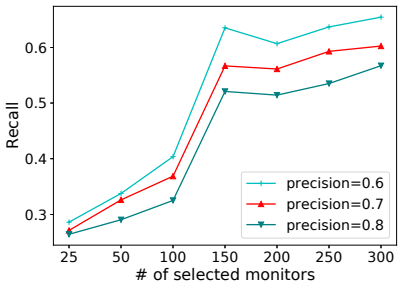

Figure 13: The recall at different precision values with different number of selected monitors.

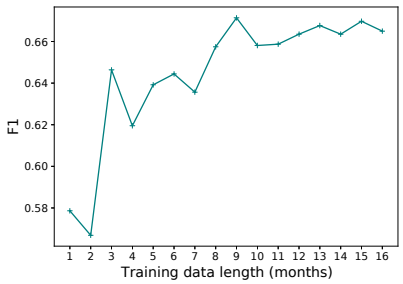

Figure 14: The F1 value with training data length from 1 to 16 months.

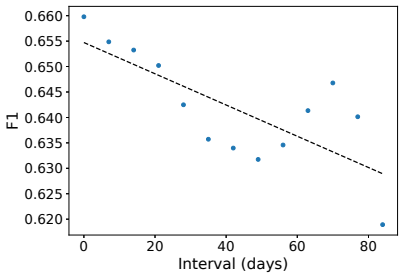

Figure 15: The F1-score with different training and testing data interleaving intervals.

with more training data. However, the gain becomes minor with the training data length exceeding 9 months. Therefore, for production usage, we collect data from the last 12 months for training, which is sufficient to reach optimal performance.

Another question is how often we need to retrain the model. We use 9 months of data for training and 2 months for testing. We interleave the training and testing data by m days, where m ranges from 0 to 80. We record the F1 scores with different interleaving intervals. Fig. 15 shows the result where the x-axis is the interleaving days and the y-axis is the mean F1 score. The dashed line is the regression fit which clearly shows a decreasing trend.

We can see a decline in performance as the data evolves (e.g., newly added monitors as shown in Fig. 4). In our system, we retrain our model on a weekly basis to capture incremental data changes. Our training is conducted on a node with 48 cores and 512 G memory. The training process with a 12-month dataset takes 16 cores and $\sim$ 8G memory, and costs $\sim$ 1.2 hours.

4.7 Case Studies

Warden has been deployed online in IcM of Azure for around 3 months till May 2020. In this section, we describe two real-world cases.

Case 1: power failure incident The first case is an incident caused by a datacenter power failure. Counterintuitively, not all incidents caused by power issues are easy to detect. Few power issues lead to real incidents due to resource redundancy. In this case, the region of the datacenter was hit by an ice storm, resulting in a large-scale power failure. The affected racks lost power when their supporting UPS units drained. It took tens of minutes before running out of power supply.

Multiple services got affected gradually during this incident. Warden sent out an initial notification at receiving alerts from the Compute service. The alerts were about “Compute Manager QoS degradation". Our notification immediately attracted the attention of the on-call engineer. When the engineer was investigating the problem, Warden gradually identified more than 10 alert signals (including “Virtual machines unexpected reboot", “Web Application probe alert", “TOR dead event", “Too many partition load failures", “API Unexpected Failures", etc.) in following time intervals. Not only the inference confidence of the model was raised substantially, but also more on-call engineers from multiple teams were engaged. They were on the same page, and a bridge meeting was set up for collaborative problem-solving. Without Warden, the engineers in different service teams would have to run individual diagnoses and discuss back-and-forth for a prolonged time until they realized that they were impacted by a power failure.

Case 2: networking incident The second case is an incident caused by TOR (top-of-rack router) down. When the TOR device is down, a VM will lose the connection to its storage, resulting in an unexpected reboot. Meanwhile, the issue of VM unexpected reboot could also be caused by storage problems such as disk failures. The current practice is that on-call engineers run diagnostic tools to find out the root cause [51]. This diagnosis is time-consuming and requires cross-team collaboration.

In this case, a VIP customer’s service was impacted and timely root cause analysis (RCA) was required. When the TOR down alert was received, Warden notified the Networking on-call engineer immediately. Several minutes later, the alert from the Compute service arrived, and Warden correlated the two alerts together. When the RCA request came to the Compute team, it was immediately concluded the incident was caused by the networking issue. Timely RCA helped increase our customer’s confidence.

5 Discussions

Why can incidents be detected? Incidents can be caused by one-off issues like code bugs or flawed configurations. Once these issues get fixed, they might never happen again. However, the detection is not based on the root causes, but on symptoms (the triggered alerts) which indeed exhibits

certain repetitive patterns. In addition, some root cause issues indeed happen again and again, like hardware failures. For example, we examined the incidents caused by cluster-level partial power failures in history. The consequential impact vary slightly from one case to another. Therefore, it is also non-trivial to detect based on manually crafted rules. In conclusion, a data-driven approach is appropriate for incident detection.

Generalizability of the proposed approach Incident detection is a common problem for all cloud platforms. Different companies implement different incident response processes. For example, Azure has a 4-step incident management process as discussed in [17]. The steps are detection, triage, mitigation, and resolution. GCP employs a very similar 4-step process as described in [1], with more detailed child processes in each step. NIST proposed a high level of abstraction for incident response with similar steps in [5]. Warden relies on the monitoring system which is a key building block to ensure service reliability. Therefore, our proposed approach could be extended to all these incident response platforms.

Trade-off in detection accuracy and delay Warden aims to detect the incidents as early as possible. However, there is a trade-off between the accuracy and the delay. We can wait for long enough until the alerts accumulate so that we can accurately detect incidents. On the other hand, we can make detection with only early weak signals, which will save time but also lower our confidence. So far, the primary audience of our system are the on-call engineers. Therefore, we conducted user studies to understand their practical concern. We conclude a precision above 90% is satisfactory for real usage, and we then tune our system accordingly.

Difference between incident-indicating alert identification and model interpretation Identifying the incident-indicating alerts is not solely a model interpretation problem [43]. Algorithms, such as SHAP or LIME [40, 41, 45], rank the input features based on their contribution to the prediction result, without considering the inherent relationship between the features. In our system, the features are constructed from the alert signals and we want to understand the contribution of each alert signal, instead of individual features.

5.1 Threats to Validity

Subject system: We have limited our evaluations to 26 major services on Azure, which produce the majority of incidents. The services we pick are representative of large-scale cloud computing platforms. For example, the Compute, Networking, and Storage are the three most fundamental services. We also include SQL DB, Web Application, Backup, Data Pipeline, etc., which are common services provided by all platforms such as AWS, Azure, and GCP. The incidents in our dataset are caused by a variety of common reasons. The top 5 deterministic reasons are code bug, network hardware issue, configuration, code defect, and design flaw. In our future plan, we will extend our evaluations with more services.

Label quality: We rely on manually linked alerts in our evaluation in Sec. 4.5. In practice, usually, only part of related alerts are linked correctly. The major problem is the missing links. Instead of manually linking all related alerts, the on-call engineers often only selectively link one alert for one service to track the cause-effect relationship. As a result, we have to evaluate the correctness of our extracted alerts on a service basis. Sometimes, people may even fail to involve a truly impacted service or add wrong links between irrelevant alerts. We incorporated experts from IcM to verify our label data and made a number of corrections. The label quality problem is thus greatly mitigated.

6 Related Work

Fault Detection and Localization: A significant amount of work [14, 28, 29, 31, 32, 34, 42, 44, 48, 51] exists on failure detection in cloud environments. Researchers [28,31,44] have advocated the scenario where minor failures from low-level services may cause large-scale impact to dependent services, which actually motivates our work of using alerts from multiple services to detect incidents. Some work [13,25,35,42,51] leverages the service dependency graph or API invocation chain to localize the root cause. Other work [14, 29, 32] proposes to use agents in distributed systems for failure detection and localization. In this work, we lay our foundation on top of the existing failure monitoring infrastructure built by individual services. We do not assume that we know the hierarchical dependency among components, which we believe is challenging for a large-scale dynamically changing cloud computing platform.

Time-Series Anomaly Detection: We notice that there is a body of work on anomaly detection and root cause localization for multi-dimensional time-series data [16, 38, 52]. There are plenty of commercial anomaly detection frameworks [7, 9, 12] offered by companies. In contrast, our work is based on alert data produced by cloud monitors. The alert data (described in Sec. 2.1) is very different from time-series metric data or console logs, and is commonly seen in cloud systems. Besides, our approach can not only detect the occurrence of cross-service incidents but also identify the incident-indicating alerts. We propose a novel algorithm called GSV for extracting the incident-indicating alerts. Finally, we target incidents that require cross-team collaboration (i.e., cross-service incidents). Therefore, we believe our work is novel and significantly different from the related work on time-series anomaly detection.

Incident Management: Haryadi et al. [27] analyze hundreds of officially published outages. Recently, there are also increasing interests in incident triage [17, 18, 39, 46, 56]. Jun-

jie et al. [19] use deep learning to find low severity incidents that represent self-healing or transient issues. Yujun et al. [22] study the problem of grouping different incidents with intrinsic relationships. Incident detection or prediction has gained increasing interest in recent years. A few recent work leverages customer feedback [55], tweets [15], or alerts with rich textual information [54] to detect real-time issues for online systems. They focus on extracting effective indicators from textual information. However, in our system, the alerts carry only machine generated texts as shown in Fig. 9 which is different from natural languages. Our previous work AirAlert [23] uses monitor generated alerts to predict several specific types of outages. In this work, Warden is designed for generic incident detection. We compare the performance of Warden with AirAlert in Sec. 4.3.

7 Conclusion and Future Work

In this work, we propose Warden, a framework to automatically detect incidents. We train an inference model based on historic failure patterns with input from a set of carefully selected monitors. Upon detecting potential incidents, Warden extracts a set of related alert signals and notifies relevant on-call engineers. This information assists the on-call engineers to prioritize their tasks and initiate cross-team collaboration. We have evaluated Warden using data collected from 26 major services on Azure. The evaluation results confirm the effectiveness of Warden. Furthermore, Warden has been successfully deployed in production.

We notice that not all incidents are covered by the monitoring system. Therefore, in our future work, we plan to exploit more signals such as the customer submitted support cases and social streams for incident detection.

Acknowledgement

We thank Kiran-Kumar Muniswamy-Reddy, our shepherd, and the anonymous reviewers for their tremendous feedback and comments. We also thank our partners Januelle Roldan, Sai Ramani, Navendu Jain, Irfan Mohiuddin, Andrew Edwards Douglas Phillips, Rakesh Namineni, David Wilson, Paul Lorimer, Victor Rühle, Jim Kleewein, Perry Clarke, Alex Dong, Wesley Miao, Andrew Zhou, Robert Gu, Yining Wei, Yingnong Dang, and Murali Chintalapati in the production teams for their valuable feedback on this work and using our system. Hongyu Zhang is supported by Australian Research Council (ARC) Discovery Project DP200102940.

References

[1] Data incident response process. https://services.google.com/fh/files/misc/data_incident_response_2018.pdf, 2018. [Online; accessed 10-August-2020].

[2] Site reliability engineering documentation. https://docs.microsoft.com/en-us/azure/site-reliability-engineering/, 2018. [Online; accessed 10-August-2020].

[3] Atlassian Incident Handbook. https://www.atlassian.com/incident-management/handbook/incident-response, 2020. [Online; accessed 10-August-2020].

[4] Bringing order to chaos: The role of the incident commander. https://www.atlassian.com/incident-management/incident-response/incident-commander, 2020. [Online; accessed 10-August-2020].

[5] Computer Security Incident Handling Guide. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf, 2020. [Online; accessed 10-August-2020].

[6] Managing Incidents. https://landing.google.com/sre/sre-book/chapters/managing-incidents/, 2020. [Online; accessed 10-August-2020].

[7] Amazon CloudWatch. https://aws.amazon.com/cloudwatch/, 2021. [Online; accessed 28-Apr-2021].

[8] AWS Post-Event Summaries. https://aws.amazon.com/cn/premiumsupport/technology/pes/, 2021. [Online; accessed 28-Apr-2021].

[9] Azure Cognitive Services. https://azure.microsoft.com/en-us/services/cognitive-services/, 2021. [Online; accessed 28-Apr-2021].

[10] Azure status history. https://status.azure.com/en-us/status/history/, 2021. [Online; accessed 28-Apr-2021].

[11] Google Cloud Status Dashboard. https://status.cloud.google.com/summary, 2021. [Online; accessed 28-Apr-2021].

[12] Splunk: Machine Learning and Anomaly Detection. https://www.splunk.com/, 2021. [Online; accessed 28-Apr-2021].

[13] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang (Harry) Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 007: Democratically finding the cause of packet drops. In *NSDI*, 2018.

[14] Behnaz Arzani, Selim Ciraci, Boon Thau Loo, Assaf Schuster, and Geoff Outhred. Taking the blame game out of data centers operations with netpoirot. In *SIGCOMM*, 2016.

[15] Eriq Augustine, Cailin Cushing, Alex Dekhtyar, Kevin McEntee, Kimberly Paterson, and Matt Tognetti. Outage detection via real-time social stream analysis: Leveraging the power of online complaints. In *WWW*, 2012.

[16] Ranjita Bhagwan, Rahul Kumar, Ramachandran Ramjee, George Varghese, Surjyakanta Mohapatra, Hemanth Manoharan, and Piyush Shah. Adtributor: Revenue debugging in advertising systems. In *Symposium on Networked Systems Design and Implemenentation (NSDI)*, 2014.

[17] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. An empirical investigation of incident triage for online service systems. In *ICSE-SEIP*, 2019.

[18] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. Continuous incident triage for large-scale online service systems. In *ASE*, pages 364–375, 11 2019.

[19] Junjie Chen, Shu Zhang, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Yu Kang, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. How incidental are the incidents? characterizing and prioritizing incidents for large-scale online service systems. In *ASE*, 2020.

[20] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *KDD*, 2016.

[21] Yujun Chen, Xian Yang, Hang Dong, Xiaoting He, Hongyu Zhang, Qingwei Lin, Junjie Chen, Pu Zhao, Yu Kang, Feng Gao, Zhangwei Xu, and Dongmei Zhang. Identifying linked incidents in large-scale online service systems. In *ESEC/FSE*, 2020.

[22] Yujun Chen, Xian Yang, Hang Dong, Xiaoting He, Hongyu Zhang, Qingwei Lin, Junjie Chen, Pu Zhao, Yu Kang, Feng Gao, Zhangwei Xu, and Dongmei Zhang. Identifying linked incidents in large-scale online service systems. In *ESEC/FSE*, 2020.

[23] Yujun Chen, Xian Yang, Qingwei Lin, Dongmei Zhang, Hang Dong, Yong Xu, Hao Li, Yu Kang, Hongyu Zhang, Feng Gao, Zhangwei Xu, and Yingnong Dang. Outage prediction and diagnosis for cloud service systems. *WWW*, pages 2659–2665, 2019.

[24] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[25] Yu Gan, Yanqi Zhang, Kelvin Hu, Dailun Cheng, Yuan He, Meghna Pancholi, and Christina Delimitrou. Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices. In *ASPLOS*, 2019.

[26] Silviu Guiaşu. *Information theory with new applications*. McGraw-Hill Companies, 1977.

[27] Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama, and Kurnia J. Eliazar. Why does the cloud stop computing? Lessons from hundreds of service outages. *SoCC*, 2016.

[28] Haryadi S. Gunawi, Riza O. Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, Deepthi Srinivasan, Biswaranjan Panda, Andrew Baptist, Gary Grider, Parks M. Fields, Kevin Harms, Robert B. Ross, Andree Jacobson, Robert Ricci, Kirk Webb, Peter Alvaro, H. Birali Runesha, Mingzhe Hao, and Huaicheng Li. Fail-slow at scale: Evidence of hardware performance faults in large production systems. In *FAST*, volume 14, 2018.

[29] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi Wei Lin, and Varugis Kurien. Pingmesh: A Large-scale system for data center network latency measurement and analysis. *SIGCOMM*, pages 139–152, 2015.

[30] Hao Hu, Hongyu Zhang, Jifeng Xuan, and Weigang Sun. Effective bug triage based on historical bug-fix information. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pages 122–132, 2014.

[31] Peng Huang, Chuanxiong Guo, Lidong Zhou, Jacob R. Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. Gray Failure: The Achilles' Heel of Cloud-Scale Systems. *HotOS*, Part F1293:150–155, 2017.

[32] Peng Huang, Jacob R Lorch, Lidong Zhou, Chuanxiong Guo, and Yingnong Dang. Capturing and Enhancing In Situ System Observability for Failure Detection Capturing and Enhancing In Situ System Observabil. *OSDI*, 2018.

[33] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579, 1901.

[34] Nader Jafari Rad and Sayyed Heidar Jafari. Passive Realtime Datacenter Fault Detection and Localization. In *NSDI*, 2017.

[35] Hiranya Jayathilaka, Chandra Krintz, and Rich Wolski. Performance monitoring and root cause analysis for cloud-hosted web applications. In *Proceedings of the 26th International Conference on World Wide Web*, pages 469–478, 2017.

[36] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NIPS*. 2017.

[37] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 2017.

[38] Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, and Dongmei Zhang. idice: problem identification for emerging issues. In *ICSE*, pages 214–224, 2016.

[39] Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. Experience report on applying software analytics in incident management of online service. *Automated Software Engineering*, 2017.

[40] Scott M Lundberg, Gabriel G Erion, and Su-In Lee. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*, 2018.

[41] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774, 2017.

[42] Meng Ma, Jingmin Xu, Yuan Wang, Pengfei Chen, Zonghua Zhang, and Ping Wang. Automap: Diagnose your microservice-based web applications automatically. In *WWW*, 2020.

[43] Christoph Molnar. *Interpretable Machine Learning*. 2019. https://christophm.github.io/interpretable-ml-book/.

[44] Biswaranjan Panda, Deepthi Srinivasan, Karan Gupta, Vinayak Khot, Huan Ke, and Haryadi S Gunawi. IASO: A Fail-Slow Detection and Mitigation Framework for Distributed Storage Services. *ATC*, 2019.

[45] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *SIGKDD*, 2016.

[46] Sara Silva, Rúben Pereira, and Ricardo Ribeiro. Machine learning in incident categorization automation. In *CISTI*. IEEE, 2018.

[47] Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, 41(3):647–665, 2014.

[48] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, Dong Xiang, and Implementation Nsdi. NetBouncer: Active Device and Link Failure Localization in Data Center Networks. In *NSDI*, pages 1–14, 2019.

[49] Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 2018.

[50] Cover TM and Thomas JA. Elements of information theory. *Wiley Series in Telecommunications*, 1991.

[51] Erci Xu, Yikang Xu, Jiesheng Wu, Mai Zheng, Feng Qin, and Alibaba Group. Lessons and Actions: What We Learned from 10K SSD-Related Storage System Failures. *ATC*, 2019.

[52] Hang Zhao, Yujing Wang, Juanyong Duan, Congrui Huang, Defu Cao, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. Multivariate time-series anomaly detection via graph attention network. In *ICDM*, 2020.

[53] Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu, Yuanzong Zhang, Zikai Chen, Xiangzhong Zheng, Xiaohui Nie, Gang Wang, Yong Wu, Fang Zhou, Wenchi Zhang, Kaixin Sui, and Dan Pei. Understanding and handling alert storm for online service systems. ICSE-SEIP '20, 2020.

[54] Nengwen Zhao, Junjie Chen, Zhou Wang, Xiao Peng, Gang Wang, Yong Wu, Fang Zhou, Zhen Feng, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. Real-time incident prediction for online service systems. In *ESEC/FSE*, 2020.

[55] Wujie Zheng, Haochuan Lu, Yangfan Zhou, Jianming Liang, Haibing Zheng, and Yuetang Deng. ifeedback: Exploiting user feedback for real-time issue detection in large-scale online service systems. In *ASE*, pages 352–363, 11 2019.

[56] Wubai Zhou, Wei Xue, Ramesh Baral, Qing Wang, Chunqiu Zeng, Tao Li, Jian Xu, Zheng Liu, Larisa Shwartz, and Genady Ya. Grabarnik. Star: A system for ticket analysis and resolution. In *SIGKDD*, 2017.