Detection Is Better Than Cure: A Cloud Incidents Perspective

Vaibhav Ganatra
Anjaly Parayil
Supriyo Ghosh
ganatrav.research@gmail.com
aparayil@microsoft.com
supriyoghosh@microsoft.com
Microsoft
India

Yu Kang
Minghua Ma
Yu.Kang@microsoft.com
minghuama@microsoft.com
Microsoft
China

Chetan Bansal
Suman Nath
Jonathan Mace
chetanb@microsoft.com
sumann@microsoft.com
jonathanmace@microsoft.com
Microsoft
USA

ABSTRACT

Cloud providers use automated watchdogs or monitors to continuously observe service availability and to proactively report incidents when system performance degrades. Improper monitoring can lead to delays in the detection and mitigation of production incidents, which can be extremely expensive in terms of customer impacts and manual toil from engineering resources. Therefore, a systematic understanding of the pitfalls in current monitoring practices and how they can lead to production incidents is crucial for ensuring continuous reliability of cloud services.

In this work, we carefully study the production incidents from the past year at Microsoft to understand the monitoring gaps in a hyperscale cloud platform. We conduct an extensive empirical study to answer: (1) What are the major causes of failures in early detection of production incidents and what are the steps taken for mitigation, (2) What is the impact of failures in early detection, (3) How do we recommend best monitoring practices for different services, and (4) How can we leverage the insights from this study to enhance the reliability of the cloud services. This study provides a deeper understanding of existing monitoring gaps in cloud platforms, uncover interesting insights and provide guidance for best monitoring practices for ensuring continuous reliability.

CCS CONCEPTS

• **General and reference** → **Empirical studies**; **Reliability**.

KEYWORDS

Empirical Study, Reliability, Cloud Services

ACM Reference Format:
Vaibhav Ganatra, Anjaly Parayil, Supriyo Ghosh, Yu Kang, Minghua Ma, Chetan Bansal, Suman Nath, and Jonathan Mace. 2023. Detection Is Better Than Cure: A Cloud Incidents Perspective. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3611643.3613898

ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0327-0/23/12…$15.00
https://doi.org/10.1145/3611643.3613898

1 INTRODUCTION

At Microsoft, we operate at a massive scale with over thousands of internal and external cloud services that are deployed over 60 regions across the world and used by hundreds of millions of users. Continuous availability of such services is essential in ensuring customer satisfaction and business revenue. Despite significant continuous reliability effort, production incidents or failures occur, inevitably. Such incidents can adversely affect the customers and can be expensive in terms of engineering resources and manual effort required to mitigate their impact. Therefore, early detection and mitigation of such incidents, before their impact reaches customers, is vital. Therefore, service providers develop automated watchdogs to continuously monitor for service health degradation so as to proactively detect and mitigate incidents before their impact reaches customers.

Service owners often create scenario-specific monitors in an ad-hoc fashion, mostly by following fix items after an incident is resolved. While existing monitors can proactively detect majority of the incidents, there are gaps that lead to two major issues: (1) *lack of effective monitors*, which causes missing the detection of early symptoms of incidents (referred to as miss-detection) and (2) *abundance of not-so-useful monitors* that are created by many sub teams in ad-hoc ways, causing non-actionable or flood of alerts. We need a deeper understanding of the existing monitoring systems of hyperscale cloud platforms to identifying the right trade-off for jointly reducing miss-detection and flooding alerts together. Empirical studies have been an important tool to gain such deep understanding and for generating useful insights.

In recent years, several empirical studies have been conducted for understanding efficiency and reliability aspect of cloud services. Liu *et. al.* [32] analyzed incidents from Microsoft Azure for identifying common root causes caused by software bugs. In a similar direction, a recent work [18] holistically analyzed production incidents from Microsoft Teams service to identify common root cause and mitigation steps, and propose insights for improving incident management lifecycle. Several other empirical studies [19, 20, 28, 33] analyze specific types of open-source software bugs, which usually arises from small-scale cloud systems. However, progress remains slow in understanding the optimal detection mechanism of large-scale cloud services. Zhao *et. al.* [48] conducted an empirical study to analyze the alert storm or flooding alerts from a large commercial bank system, and provide an algorithm to detect alert storms, cluster similar alerts and finally recommending a small set of unique

alerts to reduce engineering efforts. However, they did not provide any insights on situations where the detection of an incident has been missed. In this paper, we holistically study the existing monitoring system of more than 300 services from Microsoft to understand the common causes of miss-detection of incidents and to generate insights for effective guidance of intelligent monitoring systems.

Specifically, we carefully study close to a thousand incidents with monitor-related repair items to generate key insights for "what to monitor" and "how to monitor", so as to improve existing cloud monitoring systems. Through extensive manual analysis and pseudo-labelling using state-of-the-art machine learning techniques, we identify the broader categories of reasons behind miss-detection of incidents. We observe that for more than 40% of the cases a relevant monitor was not present, whereas for 30% of the cases, existing monitors have either incorrect logic set up or missing appropriate signals or telemetry data. We then analyze the impact of these miss-detection on service performance and engineering efforts required to mitigate those incidents. We observe that a significant portion (27%) of these miss-detected incidents lead to outages or customer impacts.

To further investigate the underlying patterns that govern these miss-detection, we correlate these broader categories of causes with various service properties, including functionality, service-level indicators, number of dependencies and maturity of a service. We observe that service functionality strongly affects the distribution of miss-detection related monitoring problems. Moreover, we identify that the maturity of the service strongly determines the "what to monitor" aspect, while the number of external dependencies determine the "how to monitor" aspect. Finally, by performing multi-dimensional cross-correlation with associative rule mining, we generate concrete insights and monitor related suggestions to specific services based on their functionality and other properties.

In summary, we make the following key contributions:

(1) We present a large-scale empirical study of production incidents, with a particular focus on uncovering the monitoring gaps that exists in the cloud platforms. We adopt a two-step process while labelling the data, consisting of human annotation and pseudo-labelling using state-of-the-art ML models. We propose the use of instance selection methods, instead of random, for selecting data for manual annotation.
(2) We develop a taxonomy for major causes of miss-detection and analyze their impact on service performance.
(3) By correlating various service properties with major causes of miss-detection, we uncover useful patterns regarding monitoring gaps for specific types of services.
(4) By using a multi-dimensional cross-correlation analysis between multiple service properties and the gaps in monitoring, we highlighted various insights and suggestions that can be considered when designing efficient monitors for cloud services.

2 BACKGROUND

Production incidents inevitably occur in large-scale cloud services and often severely impact the customer experience. An incident lifecycle typically has the following four stages: (1) Detection: The first step is to generate an alert when an anomalous system behavior is observed. (2) Triaging: Once an incident is detected, it is routed to appropriate engineering team after an initial assessment. (3) Diagnosis: The incident diagnosis and root cause identification process requires iterative communication between on-call engineers (OCEs) inspecting the different aspects of the issue. (4) Mitigation: To recover the service health, several actions are taken by OCEs to mitigate the problem.

Early detection of anomalous behavior or symptoms can significantly reduce customer impact and accelerate the incident resolution process. Production incidents at Microsoft are either detected by automated system watchdogs that continuously monitor the service health and telemetry data, or reported by users (internal or external). Automated watchdogs or monitors are ubiquitous in cloud services and are widely used to automate the task of service health monitoring. Service owners or developers often use their expertise and domain knowledge to create monitors with specific alert logic and anomaly detection algorithms. Cloud services record information about their health in the form of run-time telemetry, which serve as signals to be analyzed for detecting anomalies. Besides the signal, monitors consist of multiple predicates which constitute the alerting logic. If the signal satisfies the condition for an alert, an incident is raised with an appropriate severity level. The severity of incidents can range from 0 to 4, where severity 0 incidents are the most critical and impactful.

To evaluate the state of existing monitors within Microsoft, we study incidents raised from more than 300 services over a period of one year in 2022. The performance of existing monitors could be evaluated in two-ways: (1) whether they generate a flood of false alarms or transient incidents; and (2) whether they are able to detect incidents in a timely and accurate manner.

We begin by evaluating the first aspect above. We say that a monitor has generated a false alarm if it has reported an incident that was ignored (i.e., not investigated) by OCEs. In Figure 1(a), we demonstrate the percentage of false positive incidents generated by various services. Although majority of the services have low proportions of incidents that were not investigated, there are a considerable number of services with a high proportion of uninvestigated incidents due to the fact that those services could own monitors that are continuously generating low-priority incidents. Therefore, we further evaluate the performance of existing monitors owned by a particular service. We assume that a monitor is non-actionable if less than 10% of its generated incidents are investigated by OCEs. In Figure 1(b), we show the percentage of non-actionable monitors owned by various services. We observe that while monitors from more than 60% of services indeed generate actionable incidents, there exists a few services for which more than half of their monitors need to be modified or depreciated. While flood of false positives increases the volume of incidents and sometimes manual efforts are spend on similar incidents in parallel, this challenge can be tackled to some extend with efficient similar incident linking algorithms [13, 48].

On the other hand, gaps in monitoring system can lead to miss-detection of anomalous behaviour and thus failed to raise an incident proactively. As those incidents need to be reported by humans, this could often lead to customer impact and severe effect. Furthermore, as human reported incidents might miss specific details, logs

and telemetry data regarding the incidents, it can lead to longer time-to-mitigate (TTM) and additional engineering efforts. Figure 1(c) delineates the proportion of incidents (across 306 services in year 2022) detected by monitors and other sources. While majority (96.7%) of the incidents were detected by monitors, we still observe several human (either by customers or internal engineers) reported incidents. Moreover, in some cases, although incidents are detected by monitors, they failed to detect the "early symptoms" proactively that lead to cascading effects on service performance. Therefore, we conducted a comprehensive empirical study on more than 950 incidents (arising from more than 300 services) to holistically understand the gaps in existing monitoring systems that lead to delay in detection or missing an incident entirely, and generate insights and suggestions for intelligent monitor management.

Problem Setting. In order to understand the monitoring gaps in large-scale cloud services, we consider production incidents that occurred at Microsoft over a period of 1 year (from Jan 1, 2022 - December 1, 2022). Every production incident is recorded a central database, along with information such as incident title, summary, severity level, discussion entries among OCEs, root cause and mitigation details, repair items assigned to developer teams. Postmortem reports are often created for high severity incidents (e.g., outages, live site incidents). The postmortem report includes additional details such as a set of "why" questions (e.g., why detection was delayed?) and their answers explaining the specifics and uniqueness of the incident. In addition, postmortem reports might have additional discussion on what could be improved in the detection and mitigation as a lessons learnt for future. These are identified as repair items. In this empirical study, we utilize all this information to understand the major underlying problems that lead to miss-detection of cloud incidents, in a timely and accurate manner.

3 METHODOLOGY

In this section, we explain the data curation and labelling methodologies used for conducting the empirical analysis.

3.1 Data Curation for Empirical Study

For our empirical study, we consider repair items that were created over a duration of 1 year (Jan 1, 2022 - Jan 1, 2023). To filter monitoring-related repair items, we make use of a keyword-based filtering method. We manually examined the 25 most frequently occurring unigrams in all the repair-item texts, and identified keywords, such as "alert", "SLI" (Service-Level Indicator), etc. that effectively filtered monitor related repair items. Additionally, we select repair items that (1) have associated postmortem reports, (2) are resolved, i.e., the repair items are fully implemented, and (3) have associated code changes that are fully implemented. These conditions ensure that we select items that involve repairs (i.e., they were not aborted) and can identify the exact changes made because of each repair item. These filters generate a total of 973 repair items for our study.

3.2 Data Labelling for Empirical Study

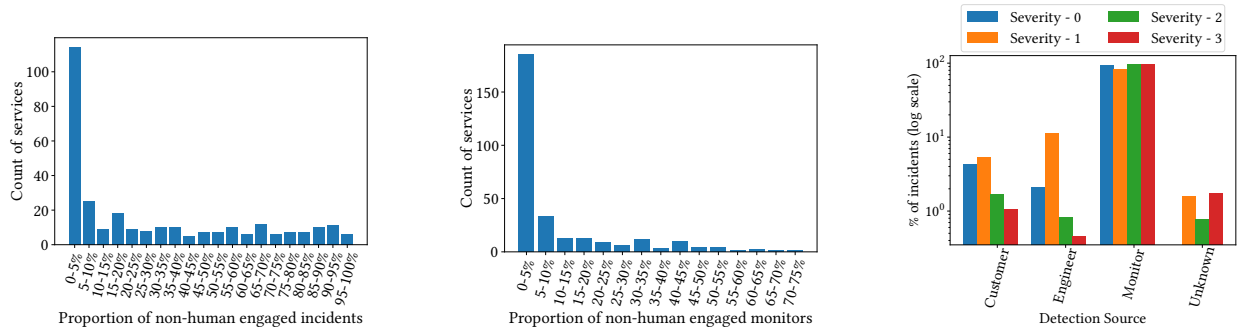
Given that we have multiple data sources - incident summary, postmortem report and repair items, manually consolidating all the information and manually annotating all 973 repair items requires significant human effort. Therefore, we adopt a two-step approach, where we sample a subset to be manually labelled, and then use the labelled subset to pseudo-label the remaining repair items (see Fig. 2 for summarized steps). This is a common approach adopted for limiting the need for labelled data [5]. Most studies typically use a random selection for the subset. However, using a random subset does not guarantee selection of representative and diverse samples in the subset. Selecting diverse samples is especially essential when sampling from imbalanced datasets to ensure that all classes have been included in the selected subset [15, 38]. However, the classes and class imbalance cannot be known apriori before manual annotation. Further, use of representative and diverse labelled samples enhances the performance of pseudo-labelling algorithms [7, 24, 44]. Therefore, we propose the use of the vote-k [24] instance selection algorithm to sample 200 representative and diverse samples from the set of all 973 repair items. We compare the diversity and representativeness of our selected sample against that of 200 randomly selected samples, using the DIV-I [16] and REPR scores [16, 50], respectively. The REPR score is calculated as the average cosine similarity with the 10 nearest instances from the entire subset (973), averaged for all instances in the selected subset. The higher the REPR score, the more representative the selected subset. The DIV-I score is calculated as the inverse of average of the minimum Euclidean distance for each point in the un-selected set from the selected subset. Higher values for DIV-I score indicates a diverse selection.

Method	REPR score	Div-I score
Random	0.2953	0.2925
vote-k selection	0.2994	0.3659

We observe that our selected samples are much more diverse than a random subset. Although the absolute differences in the scores are small, similar differences yielded significant (8-10%) improvements in pseudo-labelling [24]. This subset is then manually labelled using the strategy mentioned below.

3.2.1 Manual Labelling Strategy. We use the open coding approach [43] to manually label the data points. We first randomize the data and split it into 3 sets: the taxonomy, validation and label sets. The taxonomy set (30% of the total data) is used to initially identify classes based on what annotators perceive as the most appropriate labels and agree on the taxonomy. The validation set (20%) is used to make sure no new labels emerged, when we considered more data. Finally, the label set (remaining 50%) is labelled by each annotator independently and is used to evaluate if the annotators agree on the labels. This agreement is reported using the Cohen-Kappa score [14]. The final score observed during the open-coding of repair items was 0.95, which indicates a near-perfect agreement between the annotators. The disagreements in the labels were largely because of multiple changes being made in the repair items. They were settled by examining the major action taken in the repair items, for example, creation of new alerts may incorporate creation of signals.

3.2.2 Pseudo-Labelling using LLMs. The remaining 773 items were then pseudo-labelled using the manually labelled subset of 200 repair items. Since large language models (LLMs) such as GPT-3 and GPT-3.5 have been shown to be very effective few-shot learners



(a) (b) (c)

Figure 1: Number of services at various proportions of non-engagement (a) incidents; and (b) monitors. (c) Distribution of incidents detected by various sources.

[9], we use GPT-3.5 along with in-context learning [45] to pseudo-label the remaining items. Before pseudo-labelling the unlabelled data, it necessary to tune the input prompt to the model as well as get an estimate of the performance of the pseudo-labelling. To this end, we split the manually labelled set, where we use 120 items to label the remaining 80 items. After tuning the prompt (# in-context examples, label rules), we get a final accuracy of 72.9%. Although the model reported an accuracy of 72.9% on manually labelled data, a manual verification of the pseudo labelled instance is needed to use the data for the empirical study. Since manually verifying labels for 773 items requires significant manual effort and time, we use the vote-k algorithm [24] to further select the most representative and diverse samples and manually validate their labels. Finally, we combine the manually labelled data (200) and the manually validated pseudo-labelled data (400) for the empirical study. The use of the vote-k algorithm ensures that we have selected the most diverse and representative examples and have not missed any critical classes of monitoring. We have also verified that the distribution of the various dimensions that we consider in our study are similar for the selected (600) and overall (973) sets. 1 repair item from the manual annotation (200) and 20 repair items from the manual validation (400) are identified to not have enough information to be classified into any of the groups and are therefore dropped. The final study is conducted on 579 repair items.

4 RESULTS FROM EMPIRICAL STUDY

Through this empirical study, we aim to understand the notion of miss-detection in cloud services. We say that a production incident is miss-detected if no existing monitors have detected (and reported)

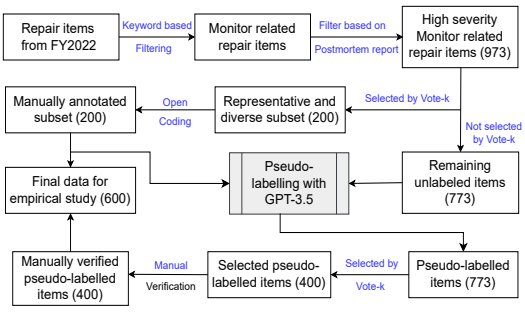


Figure 2: Overview of the data selection and labelling.

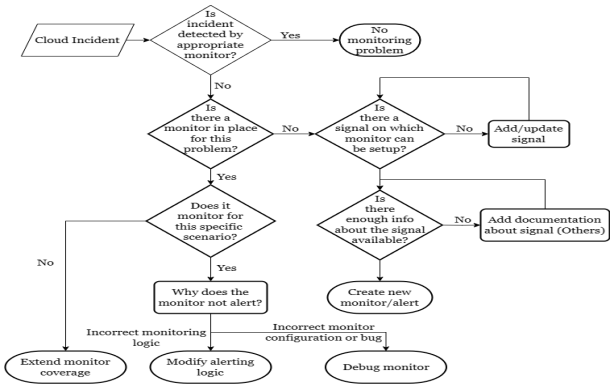


Figure 3: Overview of miss-detection in cloud service

its early symptoms; instead, the incident was reported by customers, by service owners or other monitors only after a significant impact. We answer the following research questions -

- What are the major causes of miss-detections?
- What are the impacts of miss-detections?
- How do the problems of miss-detection vary with various service properties?

4.1 Causes of Miss-detection

We identify six major categories of reasons behind the miss-detection of production incidents:

(1) **Missing/improper signal**: Key metrics/telemetry required for diagnosis or for creating new alerts are missing.
(2) **Missing monitor/alert**: Required metrics/telemetry were available, but there were no monitors/alerts for the incident's scope /environment/scenario.
(3) **Improper monitor coverage**: Existing monitors do not cover/capture the incident.
(4) **Incorrect alerting logic**: Existing alerting logic is incorrect (e.g., a threshold is too high).
(5) **Buggy monitor**: Existing monitors have configuration bugs even though their alerting logic is correct (e.g. they fail to use a new version of metrics for alerting).
(6) **Others**: Incorrect/inadequate documentation for alert enrichment and for technical support guides to assist on-call

engineers. This information about monitors/alerts is often needed when assessing the state of existing monitoring.

The mitigation actions from each category can help in enhanced monitoring of services. Fig. 3 summarizes the broad classes of miss-detection and their relationships. It must be noted that besides these classes, we also had a **Not relevant** class to accommodate for items that were not monitoring problems but were included in the set incorrectly because of the keyword based filtering. Fig. 4 shows the distribution of the broad classes that result in the miss-detection of incidents.

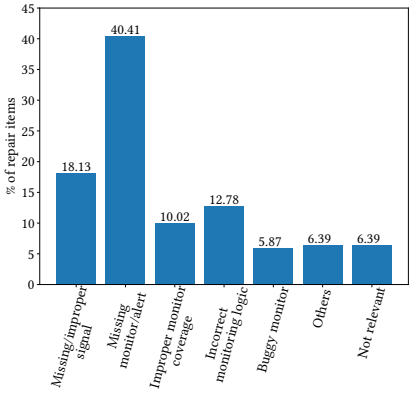


Figure 4: Major classes of miss detection

From Fig. 4, "Missing monitor/alert" is an overwhelming majority among all classes, accounting for more than 40% of all miss-detection problems. This indicates that for cloud services, the "what to monitor" question does not have a straightforward answer and despite the existing efforts, the answer is not clear. Further, the next major class is "Missing/improper signal", which hints the need to set up the signals on which new monitors are created. ~10% of existing monitors have improper coverage that need to be extended to detect incidents, whereas ~13% of the monitors require their alerting logic to be reviewed. ~6.5% of the data is not relevant, which corroborates with the accuracy of the keyword-based filtering method.

Takeaway 1

"What to monitor" in cloud services is unclear and leads to misses in incident detection.

In order to gain a greater insight into the broad classes, we further sub labelled each classes at a finer granularity.

Missing monitor/alert: Around 40% of repairs after miss detection aims to create new monitors and alerts. We observe the following sub classes within the new monitors created:

(1) **Add a new health check** (28.7%): where new monitors/alerts are added to check the health of services, e.g. availability of services and latency of requests, (2) **Add authentication check** (6.96%), where new monitors/alerts are added regarding certificates, authorizations and permissions, e.g. alerts if certificates are expired, (3) **Add integration check** (7.83%), which monitors the availability of dependencies, (4) **Add capacity check** (11.3%), which adds new alerts for utilization of resources and finite-sized components and (5) **Add scenario check** (45.22%), which adds new alerts/monitors on a very service-specific behaviour. These alerts have a very niche coverage.

Improper monitor coverage: Extending the coverage of existing monitor is performed in the following ways: (1) **Severity extension** (25.86%), where higher severity alerts are added for scenarios with existing lower severity extensions. This helps in correctly gauging the impact of the anomaly detected by the monitor. (2) **Breadth extension** (13.79%), where existing alerts are extended to similar scenarios e.g. extending alerts in an environment to others, (3) **Scenario extension** (53.45%), where new alerts need to be added within existing monitors for service-specific scope that was not being covered earlier, and (4) **Granularity extension** (6.9%), where the scope of the monitor is made more granular, e.g. splitting an HTTP request monitor in separate monitors for GET, POST, PUT and DELETE requests.

Incorrect monitoring logic: The alerting logic of existing monitors is modified in one of the following ways: (1) **Modify signal** (14.29%), where the signal from which the anomaly / incident is identified is modified, e.g. including HTTP status codes in metrics, (2) **Modify time window** (7.14%), where the signal lookback window considered in detecting incidents is modified, e.g. monitoring the number of 5xx HTTP requests in the last 15 mins, instead of 30 mins, (3) **Modify threshold** (18.57%), where the threshold of the signal is modified, e.g. alerting if the number of 5xx requests in the last 15 minutes are more than 5000, instead of 10000, (4) **Modify severity** (21.42%), where the severity of the alert is modified, (5) **Debug logic implementation** (21.43%), where the proposed logic is correct, but has a bug in implementation, e.g. using the correct version of the signal to alert on, and (6) **Extend alerting logic** (17.14%), where the existing logic missed out on capturing some aspects of the anomaly detection, and requires extensions.

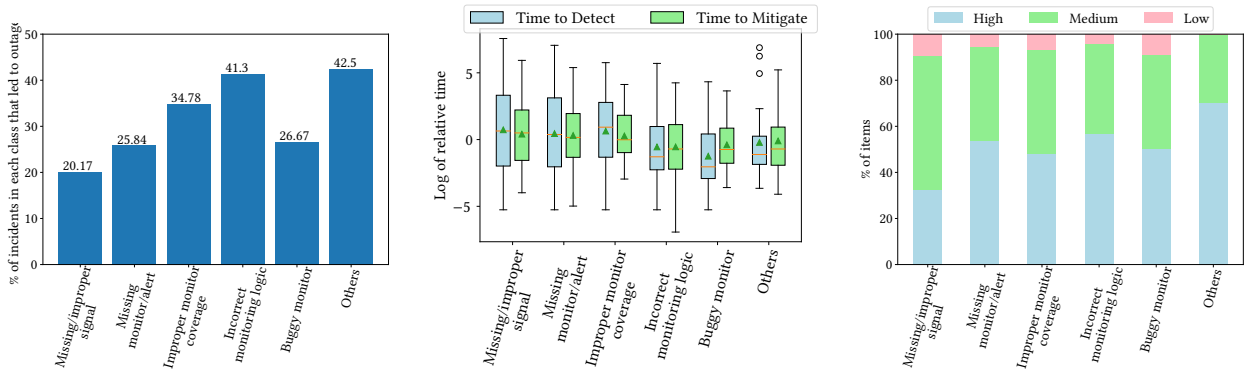
New signals are added either to be used for creating new monitors, or for capturing telemetry for faster diagnosis by the On-Call Engineer (OCE). Repairs made for "Buggy monitor" involve changes to pipeline and version modifications.

4.2 Impact of Miss-detection

In this section, we analyze the impact of the miss-detection in terms of their criticality (% of outages), human effort needed and urgency of fixing the repair items.

4.2.1 Distribution of Outages with Respect to the Major Causes of Miss-detection: A significant 27.25% of incidents associated with miss detection led to outages. Fig. 5(a) demonstrates the percentage of incidents in each monitoring class that lead to outages.

It reveals that incorrect monitoring logic and technical support guides around alerts are leading classes that surface in outages related to miss-detection. This gives us the crucial insight that only including various aspects of cloud services under monitor coverage is not sufficient to detect outage-causing incidents; the alerting logic needs to be thoroughly validated to detect potential outage-causing incidents early, before they magnify their impact and are seen as outages. Further, in many cases it was seen that for an outage, a corresponding lower severity incident was detected first, but due to the lack of TSGs and information on why the alert was triggered,



(a) (b) (c)

Figure 5: (a) Proportion of incidents from each miss-detection class that led to outages; (b) Time to Detect (TTD) and Time to Mitigate (TTM) for cloud incidents that were not detected properly. (Y-axis shows the normalized time with the median of time to detect or mitigate of all incidents as 1)**; (c) Major causes of miss-detection and corresponding priorities of the repair items created.**

the On-Call Engineer (OCE) failed to take actions to resolve the incident. Ultimately, the lower severity incident resulted in an outage. Therefore, maintaining TSGs and proper documentation about the signals and alerts are needed to prevent persistent lower-severity incidents from escalating to outages.

Takeaway 2

Validating the alerting logic and maintaining trouble shooting guides are necessary to detect and mitigate incidents early and prevent them from escalating to an outage.

4.2.2 How long does it take to detect and mitigate the incidents caused by each class? Customer and engineer reported incidents, on average take, ~ 10.7x and 9.96x times longer to be detected than monitor reported incidents. This average has been calculated after removing 10% outliers (5% on each side) in order to prevent skewing of results and to ensure robustness. Around 10% of the customer-reported incidents took more than 46x more time to be detected than the average time of monitor-reported incidents. This reinforces the importance of monitors for the timely detection of cloud incidents. Moreover, customer reported incidents took ~ 3.75x longer to be mitigated than monitor-detected incidents. Hence, early detection of incidents, before they cause customer impact, is vital.

Fig. 5(b) shows the variation of TTD and TTM across the mitigation steps suggested to avoid miss-detection of incidents. The figure reveals that the largest TTDs are seen for incidents where monitors/signals are missing. The presence of monitoring reduces the TTD (as can be seen for "Incorrect monitoring logic", "Debug monitor", etc.). The TTM is particularly high for cases where no signal and documentation is available. This is because it is difficult to diagnose an incident without the proper telemetry/indicators and documentation on the technical interpretation of these indicators. Similar to TTD, presence of signals, monitoring and documentation helps reduce the TTM (as can be seen for "Incorrect monitoring logic" and "Buggy monitor").

Takeaway 3

Presence of appropriate signals and monitoring helps in timely detection and mitigation of incidents. Faster mitigation also requires proper documentation such as trouble shooting guides.

4.2.3 Urgency of the Repair Items Identified from Miss-detected Incidents: Based on urgency and feasibility of implementation, repair items actions item may either be low, medium or high priority. The explicit definition and time-frames for priority varies for various service teams, however, short-term is the most urgent, followed by medium and long-terms.

Fig. 5(c) shows the distribution of repair priorities against various mitigation actions. More than 85% of all repair actions in each class are either high or medium priority (with high priority being more than 30% in all classes), which demonstrates the urgency of the monitoring problem. Action items demanding documentation and modification to alerting logic are the most critical ones (with the highest % of high priorities and lowest % of low priority repair items). This corroborates well with the fact that misses in detection due to these two classes frequently led to outages, and therefore, these aspects of monitors require immediate repair. More than 50% of repair items in the "Missing monitor/alert" class have a high priority which indicates that "what to monitor" is an major problem for the monitoring cloud services, that requires immediate attention. Repairs in the "Others" category are primarily high priority, probably because of the feasibility of adding documentation, whereas those in the "Missing signal" category have relatively more medium priority. This may be because of the fact that addition of new signals involve significant code changes, in terms of the metrics that it records and emits, and that may not be feasible immediately.

Takeaway 4

More than 85% of all repair items are either high or medium priority, based on their urgency and feasibility. Modifications to alert logic, despite being a complicated problem, has more than 90% repairs requiring urgent attention; more than 50%

of repair items in the "Missing monitor/alert" are also high priority.

4.3 Miss-detection and Service Properties

In the following sections, we analyze the distribution of the broad class of miss-detection with respect to the service properties. The general methodology followed here, is that for each service property, we group similar services together based on that property and consider the distribution of miss-detection classes in each group. Then, we check if the distribution for the service group is different from the overall distribution using chi-squared tests [37] at a 5% significance level (95% confidence interval). The service property affects the distribution of miss-detection classes if we reject the null hypothesis of the chi-squared test and the distribution of service group does not follow the overall distribution. For such groups, we draw insights about the correlation between service property and the miss-detection classes. For other groups, the variations in classes of miss-detection are not statistically significant to draw conclusions about them. We consider 5 different aspects of cloud services: service functionality, maturity of the service in the lifecycle, the number of service dependencies, whether or not the service , and the Service Level Indicators (SLIs) of the service. These service properties are available from the internal directory.

4.3.1 Functionality of Service. In order to correlate functionality of the service with the causes for miss-detection, we consider the description of the service functionality, and cluster services that have similar functionality together. This was done using Agglomerative Clustering [34], which hierarchically groups services with similar functionality together. This clustering was performed using BERT-based embeddings of the service description. Using the dendrogram, 13 clusters of service functionality were identified. In order to understand the common functionality of services within a cluster, we performed topic-modelling within each cluster using the algorithm specified in Zhang *et. al.* [47] to extract the underlying themes of each cluster. Further, we manually examine topic models and corresponding services to uncover intuitive meaning for each cluster. Fig. 6 shows the distributions of the broad monitoring classes against each cluster of service functionality.

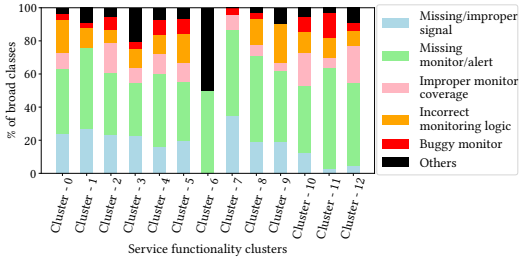


Figure 6: Major causes of miss-detection and corresponding priorities of the repair items created

Clusters 11 and 12 largely contain Revenue and Finance related services - Cluster 12 contains services for Financial Data Management and Payment Gateway Services, which involve financial analytics, whereas Cluster 11 consists of FinOps services, which provide functionalities such as Fraud Detection, ERP, etc. For services in both these clusters, there is a very low (close to 0) proportion of monitoring misses due to missing signals, which uncovers the fact the existing signals used in financial services are adequate for effective monitoring. However, Financial Data Management services face significant coverage issues with their monitors. At the same time, FinOps services have a much higher proportion of missing monitors (>60%) and bugs in the existing monitors.

Cluster 10 largely contains services for Data Management & Analytics for Monitoring and ML. This includes data services that record and process the runtime telemetry of cloud services, thereby helping in monitoring, and data services for Machine Learning. While these services have a relatively lesser proportion of missing monitors, the majority of new monitors were added for health checks. However, their monitors suffer from more problems with improper coverage.

Cluster 9 includes Real-Time Communication (RTC) services which provide chat-bots, user chat and collaboration services. For these services, appropriate monitoring logic has a much higher importance as compared to other services, which indicates that configuring monitors with the correct logic is essential in RTC services for timely detection of cloud incidents. Specifically, configuring the correct threshold is required. In addition to this, adding new monitors for health checks is an overwhelming majority of the repairs done to create new monitors. Assessing the health of various components in a reliable manner is of paramount importance in RTC services. Notably, they have very few issues with buggy monitors or monitor coverage.

Cluster 7 consists of application platforms that deliver business apps and service handling their lifecycle operations. More than 80% of all problems in Cluster 7 are because of missing signals and monitors. This can be largely attributed to the complexity of these services. "What to monitor" is crucial for such complex services, because health checks are the majority among the monitors required. They do not face any issues with monitoring logic and documentation, which can probably be attributed to the lack of monitoring.

Cluster 6 includes services for ML Compute and Inference. These services only face issues about missing monitors and documentations, which indicates that the existing coverage and logic used for monitoring these services are adequate, however these services are not adequately monitored. The new monitors required are for capacity and integration tests. There are no repairs for health checks and authentication tests.

Cluster 3 (Network and IoT services) involves a lot of documentation issues whereas Cluster 1 services (mostly provide interfaces or act as a middle layer) do not face any issues with monitor coverage and very few issues with monitor bugs! Majority of the alerting logic repairs in both clusters involve modification of severity to gauge the impact of cloud incidents correctly.

In all clusters, missing monitors is the major reason for missing detection of cloud incidents.

Takeaway 5

The core functionality of the service strongly affects the distribution of monitoring problems that led to misses in

detecting cloud incidents. Missing monitors remains the primary cause, irrespective of the service functionality. However, what was missed in monitoring, is again determined by the service functionality

4.3.2 Maturity of Services. Based on its current stage, a service can be categorized into 4 stages - (1) In Development, (2) Preview (Private or Public), (3) Generally Available, or (4) Closing Down. Here, we analyze the broad classes of monitoring problems with respect to the maturity of a service in its life cycle (Fig. 7).

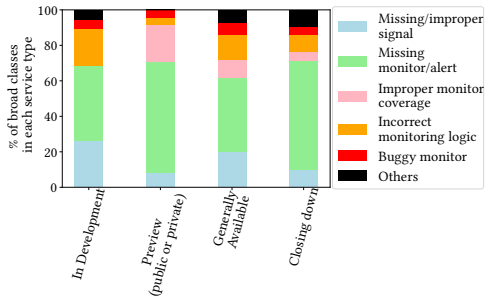


Figure 7: Major causes of miss-detection and corresponding service lifecycle stage

From the statistical tests, distributions for In Development Services and services in preview (private or public) were different from the overall distribution. We observe that In development services do not face any coverage problems in their monitors. This can be attributed to the fact that they are still under development and the coverage gets fine-tuned as the service is exposed to larger audiences. This is reinforced by the fact that services in preview have the largest proportion of coverage issues among all stages in the service lifecycle. The proportion of buggy monitors increases as the service matures. Although, missing monitors remains the major monitoring problem throughout the lifecycle, the nature of monitors added varies as the service progresses through various life-cycle stages. New monitors are added in developing services for scenario, health and authentication tests. There are no new monitors added for capacity or integration testing. Monitors that check the health of dependencies (integration testing) are added in services in preview, and as the service is made generally available, capacity testing is also included. New monitors added in closing services are majorly for authentication tests.

Takeaway 6

The current stage of the service in its life-cycle strongly determines the "what to monitor" aspect of cloud services.

4.3.3 Number of Dependencies of Service. Here, we uncover trends with respect to the number of dependencies of services. Firstly, we observe that the average number of repairs per service due to improper monitoring increases, as the number of dependencies increases (as shown in Fig 8(a)). This can be attributed to the fact that as the number of dependencies increases, the service has to keep track of the health of multiple components, which complicates the monitoring scenario. Fig 8(b) demonstrates how the distribution of broader classes of miss-detection varies with the number of dependencies. However, from statistical tests, it was observed that the distributions are different for services with 30-45 dependencies and >75 dependencies. For these services, we observe - Monitor coverage is unclear in services with 30-45 dependencies. For smaller services, the coverage is well-defined and for larger services, usually explicit rules are set about when to create new monitors, and when to extend existing ones. For services with 30-45 dependencies, it is not clear whether to create new monitors or extend existing ones, and therefore, suffer from higher proportion of coverage problems. This is corroborated by the fact that majority of the coverage issues in these services are scenario extensions, therefore, for service-specific scenarios, whether to create new monitors or extend existing ones is unclear.

The proportion of buggy monitors is small for services with lower number of dependencies, and increases as services become larger and incorporate more dependencies.

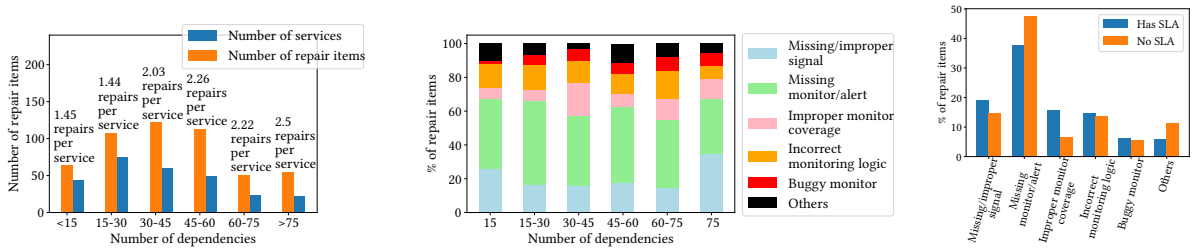
Improper monitoring logic remains a persistent problem irrespective of the number of dependencies. However, the nature of the problems varies significantly. For services with < 15 dependencies, debugging the existing logic implementation is the majority issue. For services with 15-30 dependencies, extending the alerting logic is the primary concern. For services with 30-45 and 45-60 dependencies, severity modifications assume primary importance whereas for services with 60-75 dependencies, configuring the correct threshold for alerting is a major problem. For services with >75 dependencies, the alerting logic problems are equally distributed among severity, signal, threshold and debugging logic implementation.

Takeaway 7

The dependencies of a service determine the "how to monitor" aspect of monitoring. Based on the number of dependencies of the service, various components of the monitoring logic assume central importance.

4.3.4 Whether or not the Service has SLA. Here, we compare and contrast the distribution of monitoring problems for services that have a Service-Level Agreement (SLA) vs services that dont́. Here, we adopt a different methodology for deriving insights. Instead of comparing the distribution of each class against the overall distribution, we compare the distributions of SLA vs no SLA services against each other. The statistical test reveals the distributions for both classes are different. However, it can be visually seen that the proportion of some classes ("Improper monitoring logic" and "Buggy monitor") are similar in both distributions, therefore, the difference in distributions comes only from a few classes.

Fig 8(c) delineates the distribution of broader classes of miss-detection for services with and without a SLA. The presence of SLA ensures that monitors and documentation are in place, i.e. the proportion of "Missing monitors" and "Others" is much lesser in services that have an SLA. However, an SLA only ensures the presence of monitors and not their proper configuration and coverage, as we observe significantly more repairs on extending coverage in services that have an SLA. Finally, services that have an SLA also have higher proportion of missing signal problems.



(a) (b) (c)

Figure 8: (a) Count of the repair items and number of service in each #-dependencies bucket; (b) Distribution of broad classes against #-dependencies buckets; (c) Distribution of broad classes against proportion of services having SLA.

Takeaway 8

Existence of an SLA on a cloud service performance only ensures the presence of monitors and documentation. The configuration and coverage of monitors in services with SLA still requires significant repairs.

4.3.5 SLIs of the Service. Here, we examine the correlation between various SLIs (Service-Level Indicators) that are associated with cloud services and the causes of miss detection. Each cloud service has multiple objectives (SLOs), each covering one or more of the following aspects - Availability, Latency, Success Rate, Capacity and Interruption Rate. In order to evaluate the service performance for these objectives, each service has multiple metrics (SLIs) that is continuously monitored. Using the same clustering method as Sec. 4.3.1, we cluster services based on the metrics that they monitor. Fig. 9 shows the distribution of broad classes of miss detection against the SLI clusters. We make the following observations - Cluster-3 in Fig.9 largely consists of services where majority of the metrics monitor the health of dependencies. This indicates that the service deals with multiple dependencies, and therefore alerting logic is a significantly larger problem. Cluster-5 in Fig.9 has services where majority of the metrics track system-level information. These services do not face any problems with alerting logic or documentation, but face significantly more buggy monitors. Since these services collect system-level metrics, any change in the system configuration translates to an update in the monitor configuration too. Cluster-7 in Fig.9 has services that largely monitor on infrastructure-related metrics face a significantly higher proportion of documentation and alert enrichment issues.

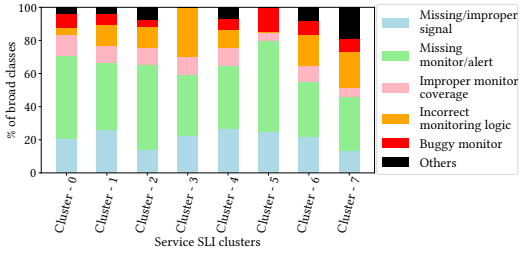


Figure 9: Distribution of broad classes against clusters of service SLIs

4.4 Multi-dimensional Correlation Analysis

In this section, we summarize the insights by correlating different service property analysis from Section 4 and discuss potential opportunities for ensuring service reliability. In short, we answer the following questions: (1) What are the immediate recommendations to avoid miss-detection of production incidents across different services and (2) how can we leverage the insights from this study towards an intelligent framework to ensure the reliability and availability of cloud services.

4.4.1 What are the Immediate Recommendations for cloud Services to Avoid Miss-detection of Production Incidents? Empirical study in Section 4 shows significant correlation between the major causes of miss-detection and services properties. To correlate multiple service properties with causes of miss-detection, we use the F-P growth algorithm for associative rule mining [1, 23] and derive associative rules. Lift of the associative rule is proportional to the likeliness of the all items occurring together in the rule. An increasing value of lift is relatively associated with the increase of other measures such as confidence and support and value of more than 1 reflects the strength of association rule [8]. Here we use "lift" to evaluate rules and provide only those rules with relatively higher values of lift.

The insights based on the association between different service properties and miss-detection classes are as follows:

(1) Network and IoT services (Cluster 3) that depend on 45-60 services should improve documentation issues. These services are often Generally available and monitors a lot of infrastructure metrics (lift = 20). Network and IoT services with 45-60 service dependencies are advised to ensure the presence of proper monitoring and alerts (lift = 14.8).
(2) Generally available services that depend on 60-75 services should ensure proper monitoring logic (lift = 4.7).
(3) Services that monitor Cluster 2 SLIs (includes metrics for pipeline failures and capacity thresholds) should ensure proper monitoring coverage. This is often evident for orchestration services that manage infrastructure (lift = 11.7).
(4) Data management services (cluster 10) that depend on 30-45 services frequently face issues with missing monitors are advised to ensure the existence of appropriate alerts (lift = 13).
(5) Generally available services such as Web App, user-facing services (Cluster 5) and data management, analytic services

(Cluster 10) are advised to ensure proper monitor configuration (lift =2.1 and 2).
(6) Services with dependencies between 30-60 exhibit relatively higher monitor bugs (lift =1.2).

In addition to the correlation between service properties and the broad causes of miss-detection, Section 4 also indicates a variation in "what to monitor" and "how to monitor" problems with respect to the lifecycle stage, functionality, and associated dependencies of the services. Therefore, we use the sub-labels for missing monitor/alert, improper monitor coverage, and incorrect monitoring logic and derive additional insights using associate rule mining. The associative rules and the corresponding lift values are as follows: What to detect defines service specific guidelines for monitor creation:

- Section 4 indicates correlation between the classes of monitors created and corresponding service functionality and lifecycle stage. For instance, higher association between cluster 12 (data management and payment gateways) and health checks.
- Monitors created for services from cluster 11 (Fraud detection, ERP) with 45-60 dependencies show higher association with authentication testing (Lift = 29).

Based on Section 4, "how to detect" (changes in alerting logic and coverage extensions) is closely related to dependency range, service functionality, and maturity of the service. We summarize insights on how to detect incidents for different services:

(1) Services from cluster 2 (orchestration services that manages infrastructure) with more than 75 service dependencies shows higher associated with scenario extension (lift = 25).
(2) Services that are in preview and depend on 30-45 services and services from cluster 2 (orchestration services that manages infrastructure) with 30-45 service dependencies exhibit association with severity extension (lift = 17 and 12).
(3) Extension of monitor coverage in generally available cluster 0 services (includes tools for software development and IT operations) with 30-45 is often associated with granularity extension (lift = 17).
(4) Extension of monitor coverage in generally available cluster 2 services (orchestration services that manages infrastructure) with 30-45 is often associated with breadth extension (lift = 12.7).
(5) Modification of alert logic in cluster 11 (FinOps) services with 45-60 service dependency is often associated with modification of the signal used (lift =60).
(6) Modifying time window is the alert logic modification performed for Generally available cluster 12 (Financial Data Management & Payment Gateways) services with 60-75 service dependencies (lift=12).
(7) Alert logic is extended for cluster 4 services (include data Governance service responsible for metadata management, data discovery, data classification, policy authoring, and enforcement) with 45-60 service dependencies (lift=60).

4.4.2 How Can We Improve the General State of Monitoring? Apart from the correlation between service properties and causes of miss-detection, results from Section 4.4.1 hints at inter-correlation between service properties and suggests the prospects for proactively suggesting monitors. Assuming that we have the information about services at a finer granularity, can we recommend monitors accounting for the various service properties? We illustrate this with an example from our data.

Here, we consider two historical incidents both having a severity level of 2. In the first incident, a service in a certain region could not connect to the database because the database lost its ability to open new SQL connections. This happened because of an SQL outage in the same region. However, the service was not monitoring for database connection failures and could not detect the incident. A repair item was identified to monitor for failed database connections. After sometime the second incident occured (with a similar issue where a certain job got stuck because of enqueuing a partition took a long time) from another service, sharing more than 40% of the first service's dependencies, and having a significant number of common monitors. It faced SQL timeouts, however, the incident was not detected by monitoring, but was rather manually observed. There is room for improvement in the monitoring, where the second incident could have been detected (and mitigated) early, by using the knowledge of the first incident and the common dependencies & monitors between the two services. An intelligent monitoring framework, that can capture these trends, and proactively suggest monitors (it would have suggested to monitor for database failures for the second service after the occurrence of the first incident) would go a long way in optimal detection of cloud incidents.

4.5 Threats to Validity

One potential threat to the validity of this study is the fact that 60% of the data into consideration was used for analysis. However, the use of instance selection framework for representative and diverse samples address the concerns over the generalizability of the findings. Further, we also analyze the distribution of service properties for selected data to ensure the trend remains same with respect to the overall data. The results, statistics and insights reported in this study should be interpreted within the context of the specific data and conditions analyzed. Our insights may not be generalized across all the services in Microsoft that was not considered in our study and may not represent the monitoring behaviour of other public cloud services. Furthermore, it is important to consider the possibility that the taxonomy developed in this study may evolve and new classes may arise over time. The construct threats to validity mainly lie in the used model version and parameters of the large language model. To reduce the threat from the parameters, we adopted the default parameters provided by the mature GPT-3.5.

5 RELATED WORK

Incident Management. Incident management has been an active area of research recently in software engineering community. There are several practical research challenges across the entire lifecycle of incident management including automated triaging [5, 11], safe deployment [31], diagnosis or root causing [6, 35] and mitigation [2, 27]. Recent efforts [41, 42] have focused on extracting structural

knowledge from incident reports for faster diagnosis. Ahmed *et. al.* [2] proposed to use state-of-the-art LLM models such as GPT-3.5 for automatically generating recommendations for root cause and mitigation plans when an incident occurs. Our work is complementary to these thread of research as the insights generated from this study will motivate future research on improving detection and further diagnosis of cloud incidents.

Incident Detection. Prior works on incident detection focus mainly on two directions. First, there have been several studies on analysing and detecting specific types of failures. Gunawi *et. al.* [21] characterize the symptoms and root-causes of fail-slow hardware failures by carefully studying 101 failure reports. In a similar direction, several study [33, 36] propose methods (e.g., IASO, OmegaGen and Panorama) for effective detection of fail-slow faults. Huang *et. al.* [25] propose techniques to detect grey (partial) failures that are difficult to detect due to differential observability, while [4, 39] provide techniques to quickly detect network failures by studying relevant incidents. The second thread of research, and more related to our work, focuses on developing end-to-end monitoring frameworks [29, 35, 40]. Li *et. al.* [30] developed a comprehensive data-driven framework for incident detection. [22, 26] propose monitoring tools for detecting network-related incidents. Roumani *et. al.* [40] propose to use machine learning and time-series based methods for forecasting future cloud incidents. Despite these efforts, miss-detection of cloud incidents poses a significant challenge in cloud reliability. We present a holistic study to uncover gaps in existing monitoring systems and provide insights on how the existing detection methods can be improved.

Empirical Studies of Incidents. There has been a significant amount of prior work has focused on empirical analysis of production incidents and outages. These can be categorized in two threads. First, several studies focused on understanding a specific type of production issues such as scalability bugs [28], crash recovery bugs [17], exception handling [10], upgrade failures [46], task scheduling failures [12] and network partitioning failure [3]. The second thread of research is on holistic understanding of a particular type of service [32, 49]. Liu et al. [32] analyzed 112 production incidents from Microsoft Azure to analyze the type of software bugs. Ghosh *et. al.* [18] analyzed production incidents from Microsoft Teams service to identify common root cause and mitigation steps. Contrary to these efforts, we empirically studied detection problems across several services within Microsoft to understand monitoring gaps in large-scale cloud services. Further, by leveraging a multi-dimensional correlation analysis between service properties and monitoring gaps, we generate customized insights for improving existing monitoring systems

6 CONCLUSION

In this work, we conducted a large-scale study of production incidents at Microsoft from last year to understand the opportunities to further strengthen cloud monitoring systems. For efficient categorization of miss-detection problems, we proposed the use of instance selection and make use of in-context inference for the data curation. The study provides a comprehensive taxonomy of major causes of miss-detection and the corresponding mitigation. We also provide deeper insights into their correlation with various service properties. Finally, we derived association rules to uncover insights for service teams, which will provide guidelines for future research in intelligent cloud monitoring system.

REFERENCES

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*. 207–216. https://dl.acm.org/doi/pdf/10.1145/170035.170072

[2] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models. In *45th International Conference on Software Engineering*.

[3] Ahmed Alquraan, Hatem Takruri, Mohammed Alfatafta, and Samer Al-Kiswany. 2018. An Analysis of {Network-Partitioning} Failures in Cloud Systems. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 51–68. https://www.usenix.org/system/files/osdi18-alquraan.pdf

[4] Behnaz Arzani, Selim Ciraci, Boon Thau Loo, Assaf Schuster, and Geoff Outhred. 2016. Taking the blame game out of data centers operations with netpoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 440–453. https://dl.acm.org/doi/abs/10.1145/2934872.2934884

[5] Amar Prakash Azad, Supriyo Ghosh, Ajay Gupta, Harshit Kumar, Prateeti Mohapatra, Lena Eckstein, Leonard Posner, and Robert Kern. 2022. Picking Pearl From Seabed: Extracting Artefacts from Noisy Issue Triaging Collaborative Conversations for Hybrid Cloud Services. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 12440–12446. https://ojs.aaai.org/index.php/AAAI/article/view/21510

[6] Chetan Bansal, Sundararajan Renganathan, Ashima Asudani, Olivier Midy, and Mathru Janakiraman. 2020. DeCaf: Diagnosing and Triaging Performance Issues in Large-Scale Cloud Services. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. https://dl.acm.org/doi/abs/10.1145/3377813.3381353

[7] Cephas A. S. Barreto, Arthur Costa Gorgônio, João C. Xavier Junior, and Anne Magály de Paula Canuto. 2022. Applying Efficient Selection Techniques of Unlabeled Instances for Wrapper-Based Semi-Supervised Methods. *IEEE Access* 10 (2022), 43535–43551. https://doi.org/10.1109/ACCESS.2022.3169498

[8] Sergey Brin, Rajeev Motwani, Jeffrey D Ullman, and Shalom Tsur. 1997. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*. 255–264. https://dl.acm.org/doi/abs/10.1145/253260.253325

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]

[10] Haicheng Chen, Wensheng Dou, Yanyan Jiang, and Feng Qin. 2019. Understanding exception-related bugs in large-scale cloud systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 339–351. https://doi.org/10.1109/ASE.2019.00040

[11] J. Chen, X. He, Q. Lin, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang. 2019. Continuous Incident Triage for Large-Scale Online Service Systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 364–375. https://doi.org/10.1109/ASE.2019.00042

[12] Xin Chen, Charng-Da Lu, and Karthik Pattabiraman. 2014. Failure analysis of jobs in compute clouds: A google cluster case study. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*. IEEE, 167–177. https://doi.org/10.1109/ISSRE.2014.34

[13] Yujun Chen, Xian Yang, Hang Dong, Xiaoting He, Hongyu Zhang, Qingwei Lin, Junjie Chen, Pu Zhao, Yu Kang, Feng Gao, et al. 2020. Identifying linked incidents in large-scale online service systems. In *Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 304–314. https://doi.org/10.1145/3368089.3409768

[14] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46. https://doi.org/10.1177/001316446002002

[15] Aida de Haro-García and Nicolás García-Pedrajas. 2011. A scalable method for instance selection for class-imbalance datasets. In *2011 11th International Conference on Intelligent Systems Design and Applications*. 1383–1390. https://doi.org/10.1109/ISDA.2011.6121853

[16] Liat Ein Dor, Alon Halfon, Ariel Gera, Eyal Shnarch, Lena Dankin, Leshem Choshen, Marina Danilevsky, Ranit Aharonov, Yoav Katz, and Noam Slonim. 2020. Active learning for BERT: An empirical study. In *Proceedings of the 2020*

Conference on Empirical Methods in Natural Language Processing (EMNLP). 7949–7962. https://doi.org/10.18653/v1/2020.emnlp-main.638

[17] Yu Gao, Wensheng Dou, Feng Qin, Chushu Gao, Dong Wang, Jun Wei, Ruirui Huang, Li Zhou, and Yongming Wu. 2018. An empirical study on crash recovery bugs in large-scale distributed systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 539–550. https://doi.org/10.1145/3236024.3236030

[18] Supriyo Ghosh, Manish Shetty, Chetan Bansal, and Suman Nath. 2022. How to fight production incidents? an empirical study on a large-scale cloud service. In *Proceedings of the 13th Symposium on Cloud Computing*. 126–141. https://doi.org/10.1145/3542929.3563482

[19] Haryadi S Gunawi, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tiratat Patana-anake, Thanh Do, Jeffry Adityatama, Kurnia J Eliazar, Agung Laksono, Jeffrey F Lukman, Vincentius Martin, et al. 2014. What bugs live in the cloud? a study of 3000+ issues in cloud systems. In *Proceedings of the ACM symposium on cloud computing*. 1–14. https://doi.org/10.1145/2670979.2670986

[20] Haryadi S Gunawi, Mingzhe Hao, Riza O Suminto, Agung Laksono, Anang D Satria, Jeffry Adityatama, and Kurnia J Eliazar. 2016. Why does the cloud stop computing? lessons from hundreds of service outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. 1–16. https://doi.org/10.1145/2987550.2987583

[21] Haryadi S Gunawi, Riza O Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, et al. 2018. Fail-slow at scale: Evidence of hardware performance faults in large production systems. *ACM Transactions on Storage (TOS)* 14, 3 (2018), 1–26. https://doi.org/10.1145/3242086

[22] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. 2015. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 139–152. https://doi.org/10.1145/2785956.2787496

[23] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. *ACM sigmod record* 29, 2 (2000), 1–12. https://doi.org/10.1145/335191.335372

[24] SU Hongjin, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, et al. 2023. Selective Annotation Makes Language Models Better Few-Shot Learners. In *The Eleventh International Conference on Learning Representations*. https://doi.org/10.48550/arXiv.2209.01975

[25] Peng Huang, Chuanxiong Guo, Lidong Zhou, Jacob R Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. 2017. Gray failure: The achilles' heel of cloud-scale systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. 150–155. https://doi.org/10.1145/3102980.3103005

[26] Masoume Jabbarifar, Alireza Shameli-Sendi, and Bettina Kemme. 2019. A scalable network-aware framework for cloud monitoring orchestration. *Journal of Network and Computer Applications* 133 (2019), 1–14. https://doi.org/10.1016/j.jnca.2019.02.006

[27] Jiajun Jiang, Weihai Lu, Junjie Chen, Qingwei Lin, Pu Zhao, Yu Kang, Hongyu Zhang, Yingfei Xiong, Feng Gao, Zhangwei Xu, et al. 2020. How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1410–1420. https://doi.org/10.1145/3368089.3417054

[28] Tanakorn Leesatapornwongsa, Jeffrey F Lukman, Shan Lu, and Haryadi S Gunawi. 2016. TaxDC: A taxonomy of non-deterministic concurrency bugs in datacenter distributed systems. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. 517–530. https://doi.org/10.1145/2872362.2872374

[29] Liqun Li, Xu Zhang, Xin Zhao, Hongyu Zhang, Yu Kang, Pu Zhao, Bo Qiao, Shilin He, Pochian Lee, Jeffrey Sun, et al. 2021. Fighting the fog of war: Automated incident detection for cloud systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 131–146.

[30] Yichen Li, Xu Zhang, Shilin He, Zhuangbin Chen, Yu Kang, Jinyang Liu, Liqun Li, Yingnong Dang, Feng Gao, Zhangwei Xu, et al. 2022. An Intelligent Framework for Timely, Accurate, and Comprehensive Cloud Incident Detection. *ACM SIGOPS Operating Systems Review* 56, 1 (2022), 1–7. https://doi.org/10.1145/3544497.3544499

[31] Ze Li, Qian Cheng, Ken Hsieh, Yingnong Dang, Peng Huang, Pankaj Singh, Xinsheng Yang, Qingwei Lin, Youjiang Wu, Sebastien Levy, et al. 2020. Gandalf: An Intelligent,{End-To-End} Analytics Service for Safe Deployment in {Large-Scale} Cloud Infrastructure. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 389–402.

[32] Haopeng Liu, Shan Lu, Madan Musuvathi, and Suman Nath. 2019. What bugs cause production cloud incidents?. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. 155–162. https://doi.org/10.1145/3317550.3321438

[33] Chang Lou, Peng Huang, and Scott Smith. 2020. Understanding, detecting and localizing partial failures in large system software. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 559–574. https://www.usenix.org/conference/nsdi20/presentation/lou

[34] Alena Lukasová. 1979. Hierarchical agglomerative clustering procedure. *Pattern Recognition* 11, 5-6 (1979), 365–381.

[35] Vinod Nair, Ameya Raul, Shwetabh Khanduja, Vikas Bahirwani, Qihong Shao, Sundararajan Sellamanickam, Sathiya Keerthi, Steve Herbert, and Sudheer Dhulipalla. 2015. Learning a hierarchical monitoring system for detecting and diagnosing service issues. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2029–2038. https://doi.org/10.1145/2783258.2788624

[36] Biswaranjan Panda, Deepthi Srinivasan, Huan Ke, Karan Gupta, Vinayak Khot, and Haryadi S Gunawi. 2019. IASO: A Fail-Slow Detection and Mitigation Framework for Distributed Storage Services.. In *USENIX Annual Technical Conference*. 47–62.

[37] Karl Pearson. 1900. X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50, 302 (1900), 157–175. https://doi.org/10.1080/14786440009463897

[38] Javier Pérez-Rodríguez, Aida de Haro-García, and Nicolás García-Pedrajas. 2011. Instance Selection for Class Imbalanced Problems by Means of Selecting Instances More than Once. In *Advances in Artificial Intelligence*, Jose A. Lozano, José A. Gámez, and José A. Moreno (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 104–113. https://doi.org/10.1007/978-3-642-25274-7_11

[39] Rahul Potharaju and Navendu Jain. 2013. When the network crumbles: An empirical study of cloud network failures and their impact on services. In *Proceedings of the 4th annual Symposium on Cloud Computing*. 1–17. https://doi.org/10.1145/2523616.2523638

[40] Yaman Roumani and Joseph K Nwankpa. 2019. An empirical study on predicting cloud incidents. *International journal of information management* 47 (2019), 131–139. https://doi.org/10.1016/j.ijinfomgt.2019.01.014

[41] Amrita Saha and Steven CH Hoi. 2022. Mining root cause knowledge from cloud service incident investigations for AIOps. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*. 197–206. https://doi.org/10.1145/3510457.3513030

[42] Manish Shetty, Chetan Bansal, Sumit Kumar, Nikitha Rao, Nachiappan Nagappan, and Thomas Zimmermann. 2021. Neural knowledge extraction from cloud service incidents. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 218–227. https://doi.org/10.1109/ICSE-SEIP52600.2021.00031

[43] Anselm Strauss and Juliet M Corbin. 1997. *Grounded theory in practice*. Sage.

[44] Junshu Wang, Nan Jiang, Guoming Zhang, Bin Hu, and Yang Li. 2015. Automatic framework for semi-supervised hyperspectral image classification using self-training with data editing. In *2015 7th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*. 1–4. https://doi.org/10.1109/WHISPERS.2015.8075485

[45] Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, and Tengyu Ma. 2023. Larger language models do in-context learning differently. arXiv:2303.03846 [cs.CL]

[46] Yongle Zhang, Junwen Yang, Zhuqi Jin, Utsav Sethi, Kirk Rodrigues, Shan Lu, and Ding Yuan. 2021. Understanding and detecting software upgrade failures in distributed systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 116–131. https://doi.org/g/10.1145/3477132.3483577

[47] Zihan Zhang, Meng Fang, Ling Chen, and Mohammad-Reza Namazi-Rad. 2022. Is neural topic modelling better than clustering? An empirical study on clustering with contextual embeddings for topics. *arXiv preprint arXiv:2204.09874* (2022). https://doi.org/10.48550/arXiv.2204.09874

[48] Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu, Yuanzong Zhang, Zikai Chen, Xiangzhong Zheng, Xiaohui Nie, Gang Wang, et al. 2020. Understanding and handling alert storm for online service systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*. 162–171. https://doi.org/10.1145/3377813.3381363

[49] Feng Zhu, Lijie Xu, Gang Ma, Shuping Ji, Jie Wang, Gang Wang, Hongyi Zhang, Kun Wan, Mingming Wang, Xingchao Zhang, et al. 2022. An Empirical Study on Quality Issues of eBay's Big Data SQL Analytics Platform. (2022). https://doi.org/10.1145/3510457.3513034

[50] Jingbo Zhu, Huizhen Wang, Tianshun Yao, and Benjamin K Tsou. 2008. Active Learning with Sampling by Uncertainty and Density for Word Sense Disambiguation and Text Classification. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*. Coling 2008 Organizing Committee, Manchester, UK, 1137–1144. https://aclanthology.org/C08-1143

Received 2023-05-18; accepted 2023-07-31