Towards Intelligent Incident Management: Why We Need It and How We Make It

Zhuangbin Chen†
The Chinese University of Hong Kong
Hong Kong, China

Yu Kang‡
Liqun Li
Xu Zhang
Microsoft Research, China

Hongyu Zhang
The University of Newcastle
NSW, Australia

Hui Xu
Yangfan Zhou
Fudan University
Shanghai, China

Li Yang
Jeffrey Sun
Microsoft Azure, USA

Zhangwei Xu
Yingnong Dang
Feng Gao
Microsoft Azure, USA

Pu Zhao
Bo Qiao
Qingwei Lin†
Microsoft Research, China

Dongmei Zhang
Microsoft Research, China

ABSTRACT

The management of cloud service incidents (unplanned interruptions or outages of a service/product) greatly affects customer satisfaction and business revenue. After years of efforts, cloud enterprises are able to solve most incidents automatically and timely. However, in practice, we still observe critical service incidents that occurred in an unexpected manner and orchestrated diagnosis workflow failed to mitigate them. In order to accelerate the understanding of unprecedented incidents and provide actionable recommendations, modern incident management system employs the strategy of AIOps (Artificial Intelligence for IT Operations). In this paper, to provide a broad view of industrial incident management and understand the modern incident management system, we conduct a comprehensive empirical study spanning over two years of incident management practices at Microsoft. Particularly, we identify two critical challenges (namely, incomplete service/resource dependencies and imprecise resource health assessment) and investigate the underlying reasons from the perspective of cloud system design and operations. We also present IcM BRAIN, our AIOps framework towards intelligent incident management, and show its practical benefits conveyed to the cloud services of Microsoft.

CCS CONCEPTS
- • Computer systems organization → Cloud computing; • Software and its engineering → Maintaining software.

KEYWORDS
Cloud Computing, Incident Management, AIOps

†Work done mainly during internship at Microsoft Research Asia.
‡Corresponding author (kay, qlin@microsoft.com).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ESEC/FSE ’20, November 8–13, 2020, Virtual Event, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7043-1/20/11... $15.00
https://doi.org/10.1145/3368089.3417055

1 INTRODUCTION

In recent years, IT enterprises started to deploy their applications and services on cloud platforms, such as Microsoft Azure, Amazon AWS, and Google Cloud Platform. Different from traditional shrink-wrapped software, online services serve hundreds of millions of customers around the world aiming at a 24x7 availability. Towards this end, cloud vendors have devoted tremendous efforts to service quality assurance [18, 27, 29, 38]. However, in reality, cloud services are still suffering from live-site service incidents, which can lead to huge economic loss, user dissatisfaction, and other unexpected consequences. For example, the cost of one hour’s service downtime for Amazon.com is estimated to be as high as $100 million [5].

Once a service incident occurs, service provider should immediately take actions to diagnose the problem and bring the service back to normal, which is called incident management. To ensure highly available cloud infrastructure, incident management should be efficient and effective. In practice, a typical procedure of incident management goes as follows: when a service incident is detected by engineers or machine-based monitors, an incident ticket documenting relevant information will be created in incident management system. Based on the ticket, the incident will be assigned to responsible service team to quickly restore the service. More details about incident management will be introduced in Section 2.2.1.

Ideally, if the entire procedure of incident management goes smoothly, the service can be quickly recovered. With years of efforts, Microsoft is now capable of alerting 97% incidents automatically and controlling more than 90% incidents by immediate mitigation. However, there are still severe and complex incidents that take a long time to handle. Our investigation reveals that the delay happens mainly in the following three scenarios. First, it is not rare that
critical incidents cannot be immediately detected as they are often induced by unexpected system/customer behaviors. Second, failure’s symptoms are sometimes hardly enough to directly pinpoint the responsible service team, as distinct problems could induce similar symptoms. Incidents are therefore reassigned for multiple times. Third, engineers usually need a long time to identify incident’s root cause and the corresponding impact scope, i.e., impacted services and customers. Motivated by these observations, we conduct an empirical study to understand the key challenges bringing about the delay in these steps as well as the fundamental reasons behind.

Microsoft runs world-wide cloud systems with thousands of services. Such a large scale makes it challenging to conduct incident management. Particularly, we have summarized two critical challenges: (1) building dependencies among massive services/resources and (2) assessing the health state of numerous cloud resources to fire reasonable alerts. In large cloud enterprises, the performance and reliability of any particular application may rely on multiple services and resources, spanning many hosts and network components [32]. The dependency issue refers to the incompleteness and vagueness of such relationships across the entire system. Consequently, the culprits of incident cannot be easily found. Even provided with the dependencies, we still need to assess the health condition of resources to locate root causes. In this paper, to help understand these issues, we carefully select some real-world counterintuitive incident cases to illustrate different types of pain points, and why heuristic solutions would fail. Meanwhile, we present four interesting lessons learned. For example, while root cause localization stands as the core of impactful incident mitigation, addressing all impacted services could be equally important; flood of alerts during impactful incidents is inevitable even careful aggregations and tuned thresholds have been applied. In addition, we quantitatively analyze the incidents collected from six core large-scale services at Microsoft and conduct a series of experiments to derive statistical support for our findings.

Given incident management is data-driven by nature, the concept of AlOps was proposed to address the challenges of IT operations with AI techniques [10, 12, 23]. Particularly, we have seen its great potential in extracting patterns from recurrent incident symptoms to provide actionable recommendations. We present Icm BRAIN (BRAIN for short), our AlOps framework for incident management. First, we introduce different types of data utilized in the framework and the data preprocessing procedure. Then, we elaborate on the techniques for mitigating the aforementioned challenges. Finally, we share the application results to demonstrate the industrial benefits conveyed to the incident management of Microsoft.

To sum up, this work makes the following major contributions:

- We report the current state of incident management in a large-scale cloud production system. Particularly, we studied the points that cause the inefficient and error-prone management workflow for critical and complicated incidents.
- We are the first to conduct a comprehensive study (based on over two years of incident tickets) to provide a broad view of the key challenges of incident management and detail the associated problems. Meanwhile, we try to explain these challenges from the perspective of cloud system design and operations. In particular, we present representative real-world cases and statistical evidences for our findings.

We introduce our AlOps framework, Icm BRAIN, for incident management, which is an effort towards more failure-resilient cloud services. Our evaluation demonstrates a significant improvement in production systems.

The remainder of this paper is organized as follows. Section 2 provides some background of incident management. Section 3 discusses our study methodology and the identified characteristics of incidents. Section 4 presents the key challenges of incident management and the underlying reasons. Section 5 presents our AlOps framework for incident management. Section 6 discusses some related work. Finally, Section 7 concludes this work.

## 2 BACKGROUND

### 2.1 Incident

In cloud systems, an incident is any unplanned interruption or performance degradation of a service or product, which can lead to service shortages at all service levels, i.e., IaaS, PaaS, SaaS. For example, a bad HTTP request, security vulnerability, or customer-reported error could constitute an incident. In particular, each incident has a severity level, which is set according to its potential impact on customers. Every organization has a different classification criteria, but many follow such pattern: Low, Medium, High, and Critical. For example, a datacenter power failure may bring down dozens of services, which should be treated as a Critical incident. Typically, one service relies on many supporting services, such as SQL Database and Domain Name System (DNS), to function properly. Such dependency quickly increases the chances of incidents as any component along the dependency graph can be the source of failure.

### 2.2 Incident Management

In this subsection, we first elaborate on the typical procedure of incident management and then introduce the metrics that are widely-used to measure its performance.

#### 2.2.1 Incident Management Procedure

Incident management is a process of detecting a live service problem, creating an incident, determining the cause, restoring the service to full operation, and capturing knowledge to prevent the incident from happening again. Typically, there are three steps involved [5, 10]: incident reporting, incident triage, and incident mitigation, as shown in Fig. 1.

1) **Incident Reporting.** Incident reporting is to detect service violation or performance degradation and create a ticket to record relevant information. In cloud systems, engineers can manually submit incident ticket if abnormal system behaviors are detected or customer-reported failure messages are confirmed. Meanwhile, monitors can detect incidents by periodically monitoring the runtime information of service systems, such as software/system logs, performance counters, and process/machine/service-level events.
2) Incident Triage. Incident triage is to engage the responsible service team for problem investigation. However, due to cloud system's high complexity and dependencies, incidents are frequently assigned to wrong responsible teams, which significantly prolongs service downtime.

3) Incident Mitigation. Incident mitigation is the process of bringing problematic service back to normal, so it can continue to serve customers. In practice, some temporary workarounds (e.g., server rebooting and service redeployment) will be applied first to quickly mitigate the impact, as a short period of downtime could become an expensive drain on company revenue and user trust.

2.2.2 TTx Metrics of Incident Management. Incident management is critical for cloud vendors to pursue its ultimate goal: Service Level Agreement (SLA). Cloud SLA is a commitment of a cloud service provider to its customers, which guarantees a minimum level of system/service availability, reliability, and responsiveness. Similarly, objectives are also set for different phases of incident management, which are described by the TTx metrics, as shown in Fig. 1. The goal of improving incident management is to minimize these TTx, efficiently mitigate the incident impact, and reduce operation loads.

- **Time to Detect (TTD):** The time it takes to detect an incident from the start of its impact (SLA breached).
- **Time to Engage (TTE):** The time it takes to engage correct responsible service team from incident detection.
- **Time to Mitigate (TTM):** The time it takes to mitigate customer impact and re-establish SLA (SLA re-established).

3 ANALYZING INCIDENT TICKETS

3.1 Methodology

3.1.1 Raw Dataset. Microsoft provides thousands of cloud services and applications which run on a 24/7 basis. To support the requirements of services, cloud system incorporates sophisticated monitoring mechanisms, where each monitor is tailored for a specific type of service-affecting symptom. The entity over which to perform health evaluation, monitoring, and alerting is called resource, which can be physical resource like a computer device or logical resource like a virtual machine or role. Upon a violation on any predefined performance metric (e.g., availability and latency), the corresponding monitor will render an incident ticket with the timestamp, location, severity, involved services/teams, impacted resources/components, a title briefly describing the symptom, etc. Besides auto alerts, manual reporting (i.e., detected by customers or engineers) is another important source of incidents, in which an extra text snippet summarizing the cloud issue is included. Particularly, during problem investigation, discussion conducted by On-Call Engineers (OCEs) will be continuously added to the ticket.

We have studied incidents from all services over two years. Among them, we select six core services at Microsoft, namely, Datacenter Management (DCM), Network, Storage, Compute, Database, and Web Service (WS), which are known as the fundamental basis that thousands of other services rely on. In this paper, we report our findings obtained through the analysis of incident tickets reported by these services. Particularly, we exclude the incidents that are intentionally generated for testing purpose. For over two years of operations, these core services produce a large number of incidents and almost half of impactful incidents at Microsoft.

3.1.2 Study Approaches. After collecting incident tickets, we perform the following investigations to derive insights:

1) Incident ticket analysis. We calculate the distribution of incidents along multiple dimensions (e.g., severity and root cause) to obtain a clear view of their characteristics. Moreover, we manually study the impactful incidents as well as their postmortem reports to understand the issues of incident management that constitute the unique challenges of troubleshooting in cloud systems.

2) Field studies. Besides statistical analysis, we discuss with OCEs to collect first-hand information regarding the pain points of incident handling and empirically verify our hypotheses. In this process, we acquire much valuable feedback and suggestions, e.g., the selection of representative incident examples (Section 4.1).

3) Validation. To support our findings, we design dedicated experiments to obtain statistical evidences from the collected incidents. Moreover, to validate the effectiveness of our AIOps framework in production systems, we perform a non-parametric hypothesis testing on incidents with and without BRAIN support.

3.2 Characteristics of Incident Tickets

3.2.1 Incident Severity. Table 1 shows the distribution of incident severity among six services. We can see that in all services, the Low and Medium incidents together take up more than 90% of the total. Particularly, in Network and Storage, the numbers of these two types of incidents are similar; while in others, Medium incidents outnumber Low incidents by a substantial margin. The number of High incidents drops significantly, whose proportion ranges from 1.21% (Network) to 5.48% (DCM). Finally, incidents of Critical type account for a very small portion, i.e., < 0.5%. However, such incidents constitute a great threat to the SLA of cloud vendors and thus should be addressed promptly and carefully.

3.2.2 Incident Fixing Time. We calculate incident fixing time and denote it as Time to Fix (TTF). Formally, it is defined as the time from the start of an incident to its final mitigation, i.e., $TTF = TTD + TTE + TTM$. In particular, to ignore infrequent peaks, we report 90th percentile, which is chosen empirically. The results are shown in Table 2. Due to privacy concern, we conceal the absolute results by dividing them by the smallest figure obtained in each experiment. A counter-intuitive observation is that the TTF of incidents with a lower severity (i.e., Medium and Low) is usually larger than that of incidents with a High severity. We find it is because low-severity incidents are usually trivial issues. Engineers will not address them immediately as they often could be mitigated by automatic routines. Meanwhile, except in Network, Critical incidents are always the most time-consuming incidents to mitigate. One reason is that the respective root causes (such as wrong configurations, software bugs, faulty devices, etc.) of Critical incidents will be fixed soon after postmortem analysis and most of them will never re-occur. Every new Critical incident is likely to carry a brand-new failure, so OCEs need a long time for root cause identification and mitigation. Moreover, there exist hierarchical dependencies among these services. DCM is in charge of infrastructure maintenance and thus is a service at the lowest layer. On top of DCM is Network and then Storage, which are also fundamental services. Compute belongs to the next tier, followed by Database and WS, which have complicated dependencies on low-layer services.
Therefore, high-layer services (i.e., Compute, Database, and WS) may have hierarchical root causes. The increased problem search space leads to a longer TTF.

### 3.2.3 Root Causes

To have an in-depth analysis of the incidents, we need to understand the reasons of their occurrence, which also helps characterize the failure patterns of cloud systems. Thus, we manually inspect the Critical incidents along with their postmortem reports and summarize their root causes into different categories. The results are shown in Table 3. We group incident root causes into six categories, which are Network Issue, Human Error, Deployment Issue, External Issue, Capacity Issue, and Others. Furthermore, we summarize them into 16 subcategories. From Table 3, we can see Network Issue with hardware failure and Human Error with code defect are the two dominant root causes, accounting for 22.95% and 19.23%, respectively. Meanwhile, Human Error with configuration issue and Capacity Issue are another two important causes.

### 4 UNDERSTAND INCIDENT MANAGEMENT

In the management of high-impact incidents, we have observed an inefficient workflow of the system. Particularly, we have seen cases where a small-scale issue in one service yielded more severe impact across multiple services before its official declaration. A natural problem then arises: why is the issue not detected at the first place? In incident triage phase, we have noticed that incidents often require a long runtime to find the correct responsible team, especially for incidents with high-level severity. Similar pain points can be seen in incident mitigation phase. The connection between issue, cause, and impact can sometimes take a long time to establish.

In this section, we first summarize the key challenges that lead to the aforementioned pain points of incident management. Then, we investigate the reasons behind these challenges from the perspective of cloud system design and operations. Particularly, we design a series of validation experiments to derive statistical evidences from raw dataset. The results are in relative value due to company policy. Moreover, to facilitate a better understanding of the challenges, we provide some interesting real-world incident examples, which are suggested by on-call engineers during field studies. Similarly, we hide sensitive information for privacy protection.

#### 4.1 Key Challenges of Incident Management

We identify two fundamental challenges of incident management and the associated pain points, which are general across different cloud vendors because of the high resemblance in the design principles of cloud systems.

1. **Service/Resource Dependency Discovery**

Dependency is the relationship that a cloud application relies on multiple services/APIs and physical/logical resources to function properly. The dependencies can be either static or dynamic, which play an important role in the troubleshooting of distributed systems. However, thus far only the few proven approach to discover these dependencies, especially fine-grained ones, is by gathering human expert knowledge cross different service teams. More often than not, the dependency requires the confirmation of two related teams. This approach is not only inefficient, but also unscalable due to numerous services and resources in large enterprises. In incident management, the absence of a complete and real-time dependency graph will mainly bring about the following two critical problems.

**Imprecise Impact Estimation.** When an incident occurs, OCEs need to estimate the impact scope of the failure and understand how the failure is propagated across the system. Such information is essential as a high-layer service (e.g., Database) needs to know which low-layer services (e.g., Storage) depend on and are impacted by the failure. Precise impact estimation for incident plays an important role to automatically identify the affected customers, which is the main criteria to decide incident’s severity. Workarounds or solutions can then be delivered to the customers suffering from the failure promptly and proactively.

### Table 1: Distribution of incident severity

<table>
<thead>
<tr>
<th>Root Cause</th>
<th>Dist.</th>
<th>Root Cause</th>
<th>Dist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCM</td>
<td></td>
<td>Network</td>
<td></td>
</tr>
<tr>
<td>Critical</td>
<td>0.00%</td>
<td>0.01%</td>
<td>0.00%</td>
</tr>
<tr>
<td>High</td>
<td>5.48%</td>
<td>1.21%</td>
<td>2.57%</td>
</tr>
<tr>
<td>Medium</td>
<td>86.65%</td>
<td>46.90%</td>
<td>43.32%</td>
</tr>
<tr>
<td>Low</td>
<td>7.86%</td>
<td>51.88%</td>
<td>54.10%</td>
</tr>
</tbody>
</table>

### Table 2: Distribution of relative incident fixing time

<table>
<thead>
<tr>
<th>Root Cause</th>
<th>Dist.</th>
<th>Root Cause</th>
<th>Dist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCM</td>
<td></td>
<td>Network</td>
<td></td>
</tr>
<tr>
<td>Critical</td>
<td>38.33x</td>
<td>8.46x</td>
<td>10.06x</td>
</tr>
<tr>
<td>High</td>
<td>19.25x</td>
<td>3.18x</td>
<td>2.52x</td>
</tr>
<tr>
<td>Medium</td>
<td>1x</td>
<td>9.8x</td>
<td>7.09x</td>
</tr>
<tr>
<td>Low</td>
<td>3.01x</td>
<td>5.49x</td>
<td>1.09x</td>
</tr>
</tbody>
</table>

### Table 3: Distribution of incident root causes

<table>
<thead>
<tr>
<th>Root Cause</th>
<th>Dist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network (Hardware)</td>
<td>22.95%</td>
</tr>
<tr>
<td>Network (Connectivity)</td>
<td>2.24%</td>
</tr>
<tr>
<td>Network (Config.)</td>
<td>0.89%</td>
</tr>
<tr>
<td>Network (Other)</td>
<td>4.47%</td>
</tr>
<tr>
<td>Deployment (Upgrade)</td>
<td>5.22%</td>
</tr>
<tr>
<td>Deployment (Config.)</td>
<td>3.87%</td>
</tr>
<tr>
<td>Deployment (Other)</td>
<td>1.19%</td>
</tr>
<tr>
<td>Capacity Issue</td>
<td>6.56%</td>
</tr>
</tbody>
</table>

| Dist. | Others | 10.88% |

---

ESEC/FSE ’20, November 8–13, 2020, Virtual Event, USA
Zhuangbin Chen et al.

---

1490
resources. Clear service hierarchical dependencies can dramatically facilitate this process.

To understand how the estimation of incident’s impact is delayed, we carefully study the postmortem report of impactful incidents. Particularly, we define Time to Broadcast (TTB) as the time it takes to broadcast a failure to all of the impacted services, and compare it with other incident management phases, i.e., TTD, TTE, and TTM. Table 4 presents the results, where, again, the absolute values are concealed. We can see TTB has comparable values with TTM in almost all services and they are the two dominant TTx in incident management workflow. Particularly, DCM, serving as a fundamental support to many services, owns the largest TTB. This is because serious failures happened to it often have widespread impact.

Table 4: Distribution of TTx from postmortem reports

<table>
<thead>
<tr>
<th></th>
<th>DCM</th>
<th>Network</th>
<th>Storage</th>
<th>Compute</th>
<th>Database</th>
<th>WS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTD</td>
<td>15.17x</td>
<td>5.24x</td>
<td>1x</td>
<td>13.63x</td>
<td>18.47x</td>
<td>5.72x</td>
</tr>
<tr>
<td>TTE</td>
<td>15.91x</td>
<td>11.23x</td>
<td>9.82x</td>
<td>7.27x</td>
<td>13.35x</td>
<td>1.57x</td>
</tr>
<tr>
<td>TTM</td>
<td>10.95x</td>
<td>12.13x</td>
<td>14.71x</td>
<td>16.21x</td>
<td>15.51x</td>
<td>14.84x</td>
</tr>
<tr>
<td>TTB</td>
<td>17.91x</td>
<td>13.59x</td>
<td>11.06x</td>
<td>12.59x</td>
<td>13.09x</td>
<td>8.69x</td>
</tr>
</tbody>
</table>

Figure 4: No. of incidents incurring redundant effort and average no. of involved service teams (with different bases)

being handled by different teams, it may not be immediately obvious that there exists a caused-by relation among some of them. This may lead to not only delay in mitigation, but also redundant engineering efforts as different teams are addressing the same problem. Fig. 3 presents one such incident. A fine-grained dependency graph can dramatically improve the situation as we can correlate incidents by, for example, comparing their origins and tracing their impact. Another circumstance where redundant effort often happens is dealing with historically repeated incidents. In this case, incident correlation can also help by providing similar solutions.

To understand the situation of redundant efforts at Microsoft, we calculate the number of incidents that are redundantly handled by more than one service teams and the average number of teams involved for such incidents. Particularly, we make use of the links between incidents to identify the incidents of interest. These links are marked by OCEs during incident investigation and the caused-by relations can be deduced from them. Specifically, for each incident, we first find its responsible team and the incident that triggers it (if any), called parent incident. Then, for incidents with the same parent, the associated teams will be considered as addressing a same root cause, i.e., the parent incident. Since the links are incomplete, this selection criteria is quite conservative, yet we still notice a serious situation of redundant effort across different services. In Fig. 4, we can see Database has both the least number of redundant effort-inducing incidents and the average number of teams. However, Network and Storage generate nearly five times more such incidents and involve more service teams.

Lesson Learned 2. Merging separate maintenance work from dependent service teams is important for a quick incident mitigation and effort saving.

2) Resource Health Assessment. In cloud systems, it is an art to design monitoring mechanisms that are able to cover different types of failures for numerous resources and APIs. Particularly, it includes signal (time-series telemetry data from resources) capturing and anomaly detection. On one hand, too sensitive alerts would cause flooding alarms; on the other hand, too tolerant alerts would cause...
Incident ID
Resolved
Critical
High Storage resource utilization detected
Service: Storage
Datacenter: DC #4
# of impacted requests: ~100,000
# of impacted accounts: ~10,000

Summary
Monitor has detected an anomalous high storage resource utilization, impacting multiple properties in [Region].

Diagnosis
The root cause found was the misconfiguration on the request throttling threshold for a specific group of storage nodes. The storage nodes were therefore overload-
ed. In addition, an OS upgrade by management service was underway at the same time, which further increased the number of transactions and pushed the CPU uti-
lization to an overwhelming level. As many services rely on virtual storage service, 12 other services got impacted, triggering a large number of alerts although they were already aggregated. For example, a web app service reported high failure rate, long response time of some APIs, etc.; a business app service reported SLA drop for different job types, high ratio of unhealthy nodes, etc. Besides, the cloud storage service reported even more types of incidents, such as customer low availability, high impacted counts on VM, different API requests in low availability, high error rate of different container types, etc.

Figure 5: Incident example 3

Incident ID
Resolved
Critical
Disk firmware update disabled disk cache
Service: Storage
Datacenter: DC #4
# of impacted requests: ~100,000
# of impacted accounts: ~10,000

Summary
Writing to a big data storage platform experienced high failure counts.

Diagnosis
Firmware upgrade to a game drive service inadvertently disabled write cache. At the beginning, there was no direct impact on the service because the number of machines getting into bad state was small and the system was built to tolerate such instances. However, as more and more machines were getting upgraded, the overall latency of the service stack was slowly accumulating and at some point got tipped. It took quite some time to detect the incident which unfortunately deteriorated into a critical issue.

Figure 6: Incident example 4

Flooding Alarms. Normally, the alerting threshold of monitors is set to be static and conservative, which will inevitably produce a large number of non-critical alarms. Moreover, due to cloud application’s multi-tiered structure, services in each tier will generate alerts for their failed components. Such chain effect will trigger a flood of homologous incidents. To alleviate this situation, cloud systems adopt aggregation policies to merge duplicated alerts in appropriate resource levels. Each respective level may contain failures happened to finer level of resources. For example, let us assume that a datacenter has the following toy resource hierarchy: datacenter > row > rack > node (there are more logical layers in real-world systems). When failure happens, instead of generating incidents for each separate node, incidents should be created in datacenter level to merge the failures of impacted rows, in row level to merge failed racks, and so forth. Such fine-grained aggregation rules should be carefully kept as merging all failures only in the highest possible level would mislead the diagnosis. We have seen cases where similar failures of hardware devices in a datacenter were coincidentally caused by distinct reasons (power failure and firmware bug). In general, to profile service failures more comprehensively, cloud systems apply aggregation to the following monitoring aspects:

- **Resource/API:** Cloud entities associated with the failures.
- **Failure type:** Error type, error code, etc.

Gray Failures. Different from fail-stop failures, the manifestations of gray failures are fairly subtle and thus defy quick and definitive detection, which is quite common for large-scale services. Huang et al. [22] conducted a systematic study on gray failures. For example, if a system’s request handling module is stuck but its heartbeat module is not, an error-handling module dependent on heartbeats will perceive the system as healthy while a client seeking service will regard it as failed. We have also observed such gray failures incident in cloud systems (Fig. 6 and 7). In both cases, the error rate reported by monitors is in a reasonable level, so the issues are mistakenly tolerated by the monitoring systems.

Design the following experiments to study the problem of incident false detection. For flooding alarms, although falsely detected incidents will be marked in our system, we also consider incidents which never get handled and mitigate automatically as flooding cases. Another type of flooding alarms is the incidents generated due to the chain effect of cloud failures. We adopt redundant effort-inducing incidents (Fig. 4) as such cases, as they stem from identical issues. Particularly, duplicated incidents found by different criteria are removed. Regarding gray failure, our system does not explicitly mark them because: (1) trivial mistakes can be safely ignored as they have no impact on services; (2) serious failures will

Lesson Learned 3. Monitoring cloud system in different resource levels is proven to be effective for improving the coverage of failure detection in early stage. However, this may induce flooding alarms even signal aggregations have been applied. More sophisticated signal aggregation strategies are in high demand.

Gray Failures. Different from fail-stop failures, the manifestations of gray failures are fairly subtle and thus defy quick and definitive detection, which is quite common for large-scale services. Huang et al. [22] conducted a systematic study on gray failures. For example, if a system’s request handling module is stuck but its heartbeat module is not, an error-handling module dependent on heartbeats will perceive the system as healthy while a client seeking service will regard it as failed. We have also observed such gray failures incident in cloud systems (Fig. 6 and 7). In both cases, the error rate reported by monitors is in a reasonable level, so the issues are mistakenly tolerated by the monitoring systems.

Lesson Learned 3. Monitoring cloud system in different resource levels is proven to be effective for improving the coverage of failure detection in early stage. However, this may induce flooding alarms even signal aggregations have been applied. More sophisticated signal aggregation strategies are in high demand.

- **Impact type:** Availability, performance, task error rate, etc.
- **Customer:** Users experiencing the failures.
- **Location:** Regions where service failures happen.

The goal of alert aggregation is to provide engineers and operators a clear and integrated view of service health status. However, there are still entities from identical aspects that cannot be merged (e.g., multiple APIs impacted, multiple failure types). Thus, the number of incidents is still overwhelming incident management system. Fig. 5 presents a case showing a failure could trigger a large number of incidents even aggregations have been applied.
eventually be found when they manifest themselves, but there is no need to mark them as technically they are all failed detection. To tackle this problem more reasonably, we make use of incident’s severity. Specifically, during the lifetime of an incident, if its severity level ever gets upgraded, it will be considered as a gray failure because it is not correctly identified regarding how serious it is at the beginning. Again, the designed rules for discovering incident false detection are quite conservative, but we can still see it is an ubiquitous problem in cloud services. As shown in Fig. 8, compared to DCM and WS, other services have a much more serious situation of flooding alarms. Regarding gray failure, other services encounter much more cases than WS does, especially Network. This aligns with the results shown in Table 3, demonstrating the complexity of Network-related failures.

**Lesson Learned 4.** Large-scale cloud systems are prone to gray failures. Setting insensitive alarms to avoid flooding alerts may cause missing of critical issues.

### 4.2 Understand the Key Challenges

In this section, we detail the fundamental reasons behind the aforementioned key challenges. Specifically, we believe the dependency issues are essentially brought by system modular design and the virtualization of physical infrastructure; the difficulty of resource health assessment is twofold: (1) system’s fault-tolerant property and (2) a series of monitor design and distribution problems.

**Software System Modularity.** In cloud systems, applications follow a microservice-based architecture that decomposes the application logic into several interacting component services. These components are often independently developed in a cloud hosting environment, making cloud systems much more complicated than conventional ones. The complexity of dependency graph would grow exponentially with the number of services. One important reason is that there is no general way to identify from which source an API is called. The capacity planner and load balancer used in the architecture further complicate the API calling, as shown in Fig. 9. Therefore, we need to start from the source side to collect respective dependency sub-graphs and integrate them into a global one. In this manner, it might seem that the graph can be easily constructed if service designer can generate rules to specify its dependencies. However, typical challenges include diversity of different services, fast system evolution, and rule unavailability of legacy systems, which are also mentioned by Bahli et al. [1].

**Physical Infrastructure Virtualization.** Virtualization allows abstraction of physical infrastructure, which however makes it difficult to identify the dependency graph from an incident to the problematic physical component(s). There are mainly two reasons. First, the dependencies are dynamically constructed due to system reconfiguration and/or resource migration. For example, if one VM is temporarily stopped on its host machine, it can be moved to a new host without disrupting its users. Therefore, services can be dynamically deployed in different VMs and the dependencies between VMs and physical machines are also dynamic, as shown in Fig. 9. Second, logically related resources can be highly geographically-distributed. For example, many modern cloud applications use clusters of virtual machines to implement load-balancing and ensure resilience for critical tasks. The physical nodes that host the VMs in a same cluster could be in different datacenters or regions, which increase the difficulty of problem localization, as illustrated in Fig. 9.

**Fault Tolerance.** Fault tolerance is critical for cloud platforms to provide highly stable availability and business continuity of mission-critical systems or applications. In cloud computing, availability zone is one of the best practices of fault tolerance, which protects service availability from datacenter failures by replicating applications and data. The resiliency is ensured by its physical separation in terms of power, data, networking, etc. However, in some cases, fault tolerance hinders the assessment of resource health by hiding problems in the early stage. These small issues have the potential to incur fatal consequences if not handled seriously and timely. Fig. 6 presents such a potential threat, demonstrating the need of more sophisticated fault-tolerant mechanisms.

**Monitor Design and Distribution.** Monitor design and monitor distribution are two important factors affecting the performance of assessing resource health. Specifically, monitor design refers to what signals should be monitored and the corresponding alerting logic; monitor distribution describes what resources should be monitored to pursue an accurate and timely incident detection.

When designing monitors, we need to identify what metrics and events that are most representative of resource health status. More often than not, a set of metrics collectively can constitute a stronger performance predictor as they provide more complete and comprehensive information. This is a typical feature engineering problem which relies on IT practitioners’ domain knowledge. Another issue of monitor design is the alerting rules determining when to raise incidents. One widely-adopted rule is setting thresholds on the time-series signals of resource and check whether any of them is violated. However, this kind of rules is too simple which causes a
As shown by our previous study, critical cloud incidents often occur in an unexpected manner and thus dedicated approaches could fail. Nevertheless, we notice the root causes of critical incidents share many similar features. This is where we see AI/ML techniques can help by extracting patterns from recurrent incident symptoms and providing actionable recommendations.

We present BRAIN, an AIOps framework aiming at improving the entire pipeline of incident management at Microsoft. As shown in Fig. 10, BRAIN consists of three modules: Data and Feature, Data Preprocessing, and Techniques.

5.1 Design Principles

Based on our experience and empirical analysis, we first describe the design principles of BRAIN, i.e., how BRAIN addresses the identified key challenges (Section 4.1).

Regarding dependency discovery, attempts have been made to track the run-time dependencies of applications by standardizing the middleware infrastructure [2, 4, 9]. However, as applications come from a wide variety of vendors, it is impractical that all vendors will instrument their services in a common fashion [1]. Log analysis [3, 30, 35] would be a non-intrusive way to construct dependencies across different servers, processes, and third-party services. However, this solution cannot meet the real-time needs of extremely large-scale distributed systems due to data explosion, log’s heterogeneity, dynamic changes of dependencies, etc. On the other hand, we notice that before the occurrence of a critical cloud incident, many related incidents would have happened in a short period of time. In this process, individual service teams are alerting and mitigating incidents separately. Being able to provide OCEs with related incidents can dramatically save redundant engineering effort and facilitate root cause localization. Therefore, instead of tracking fine-grained service dependencies, BRAIN resorts to incident correlation to pursue reliable cloud services.

The accuracy of resource health assessment is crucial to cloud systems. However, it cannot be achieved by pursuing the perfection and completeness of purely rule-based monitoring system. As in Fig. 7, the essential reason of such failure is the absence of per-service monitors. Given system’s dynamicity and the intransparency between different application tiers, it is extremely hard to formulate the problem of monitor design and distribution mathematically. In contrast, BRAIN develops a series of incident detection algorithms on top of various system signals, e.g., service health data, infrastructure signals. Moreover, the resource hierarchy relationship is used to understand topologies, resiliency models, and dependencies among the entire cloud system.

5.2 Data and Features

Two sources of data are utilized in BRAIN, namely, raw incident tickets and various system input signals.

1) Raw incident tickets. In BRAIN, we utilize all incident tickets that have been reported to the incident management system at Microsoft. These incidents come from different service teams and therefore can provide us with a global view of service health state across the cloud system.

2) System input signals. BRAIN runs 24x7 non-stop analyzing the signals and patterns to detect anomalies in the systems. Particularly, the input to BRAIN includes the following categories:

- **Near Real Time (NRT) health signals.** The health signals are collected from each cloud resource, individual services’ monitors, and system deployed active monitors.
- **NRT metrics.** Service availability, performance metrics, request volume, etc.
- **System topology.** The hierarchy information of different resources across the entire cloud system.
- **Infrastructure signals.** Low level infrastructure sensors sensing datacenter traffic volume, temperature, power consumption, local weather, etc.
- **Customer input.** Customer Service & Support reports, which consist of many categorical attributes such as product version, the problematic product feature, product configuration, client OS, service package, etc. [28, 39].
- **Historical data.** Change history, metric history, etc.

5.3 Data Preprocessing

Incident management system at Microsoft is a hub system which involves thousands of service teams. Particularly, different teams may have their own platforms of monitoring and processing service failures. The tools for incident diagnosis may also vary. Consequently, incident tickets in the system are rendered by different monitoring platforms with different diagnosis tools, and thus contain various types of data. Poor signal-to-noise ratio and data inconsistency are therefore inevitable. Moreover, incident management system is
To tackle this problem, we develop a Bayesian network inference approach for incident detection.

During this process, knowledge is accumulated with the number of discussions. BRAIN tries to automate the triage of incidents with fewer discussions such that problem investigation can be triggered earlier. Specifically, we design a GRU-based model to effectively utilize incremental discussions by considering their temporal relationship. Three types of data are fed into our model: 1) incident title and summary, 2) incident raw discussions, and 3) environment information, e.g., incident type, monitor and device reporting the incident, etc. The global entity dictionaries can be used to ensure the correctness and consistency of these information. However, since discussions are conducted by engineers, it tends to introduce noise. We propose an attention-based mask strategy to bypass the noise. In this way, different weights can be automatically assigned to different discussion information and noise can be masked out by assigning it trivial weights. Due to low frequency, special terms (e.g., API and component names) cannot be properly encoded by traditional text encoding methods. We adopt a CNN-based neural-language model to perform domain-specific text encoding. Particularly, our model has achieved a notable accuracy of 0.64–0.73, which outperforms the state-of-the-art bug triage approach by a significant margin of 12.2%–35.5%.

3) Incident Correlation. Incident correlation tries to alleviate the situation of redundant efforts and assist the impact estimation of failures. We propose two algorithms: event-based and resource-guided methods. In event-based method, due to the high resemblance between incident title and log, we use an automatic log parsing method to extract templates from the repaired incident titles. Based on word-level similarity, templates are grouped to form incident events, representing different types of service issues. The relationship among incident events are deduced from incidents’ historical links, which are marked by OCEs during incident investigation. Such links are used for model training and evaluation. During evaluation, incidents will be connected if their representing events are ever linked before. Although this is an effective way of using OCEs’ domain knowledge, in some cases, log parsing methods cannot meet our needs. It is because in titles, special terms with small frequency are often erased and the extracted events are indistinguishable in terms of identifying which resource is unhealthy. Thus, we develop a resource-guided method to perform incident correlation in a fine-grained manner. Specifically, two incidents are considered as correlated if they are tagged with identical or related resources (with appropriate location and time constraints). In particular, there are two ways for identifying related resources: (1) leveraging existing hierarchy information of resources at Microsoft, and (2) mining their spatial and temporal co-occurrences in incidents. Combining these two methods, we are able to achieve >0.89 precision, recall, and F1 score for incident correlation.

5.4 Techniques of BRAIN

1) Incident Detection. Incident detection is to identify service issues based on various system signals. It pursues an early detection of gray failures and recognizes important issues from trivial ones. In cloud systems, time series and event sequence are two major types of telemetry data, where anomalies often manifest themselves as having a large magnitude of upward/downward changes. Besides traditional martingale methodologies, BRAIN also exploits sophisticated characteristics of the signals. Particularly, signals are classified as temporal or spatial, which are tackled by a LSTM model and a Random Forest model, respectively. To enhance the interaction among different signals, BRAIN calculates a series of statistical features for a set of data points in a rolling window, e.g., mean, variance. In BRAIN, significant progress (e.g., ~0.7 F1 score) has been made when detecting certain types of cloud failures, e.g., unplanned VM reboot, node failure, API throttling.

2) Incident Auto-triage. In incident triage phase, OCEs continuously hold discussions until the correct service team is found. During this process, knowledge is accumulated with the number of discussions. BRAIN tries to automate the triage of incidents with fewer discussions such that problem investigation can be triggered earlier. Specifically, we design a GRU-based model to effectively utilize incremental discussions by considering their temporal relationship. Three types of data are fed into our model: 1) incident title and summary, 2) incident raw discussions, and 3) environment information, e.g., incident type, monitor and device reporting the incident, etc. The global entity dictionaries can be used to ensure the correctness and consistency of these information. However, since discussions are conducted by engineers, it tends to introduce noise. We propose an attention-based mask strategy to bypass the noise. In this way, different weights can be automatically assigned to different discussion information and noise can be masked out by assigning it trivial weights. Due to low frequency, special terms (e.g., API and component names) cannot be properly encoded by traditional text encoding methods. We adopt a CNN-based neural-language model to perform domain-specific text encoding. Particularly, our model has achieved a notable accuracy of 0.64–0.73, which outperforms the state-of-the-art bug triage approach by a significant margin of 12.2%–35.5%.
and it clearly shows impactful incidents engaged with BRAIN have shorter TTD, TTE, TTM, TTB (Section 4.1), and TTF (Section 3.2.2).

To account for the sample-to-sample variation, we performed a non-parametric hypothesis test (Mann-Whitney-Wilcoxon test). The null hypothesis is that the reduction TTx seen here is due to the randomness in the data. The p-value measures how probable the null hypothesis is, given the observed trend in the sample. If the p-value is low (i.e., <0.05, at the 95% significance level), we would claim that the null hypothesis is improbable and reject it in favor of the alternative hypothesis - the observed reduction of TTx is indeed related to BRAIN. The key results are summarized in Table 5. Supported by this testing, we conclude that BRAIN’s associations with shorter TTx are statistically significant. Therefore, BRAIN manifests itself as an effective facilitator for TTx reduction.

### 6 RELATED WORK

#### 6.1 Reliability and Resilience of Cloud Services

There are many methods focusing on improving the reliability and resilience of cloud systems. In terms of failure prediction in cloud systems, Xu et al. [34] formulated the disk failure prediction problem (a major source of service incidents) as a ranking problem and adopted FastTree algorithm to do prediction. Log analysis is also an important means of failure detection [11, 13, 15, 17–19]. Particularly, He et al. [18] proposed a cascading clustering algorithm to identify the impactful problems by correlating the clusters of log event sequences with system KPIs. Zhang et al. [36] addressed the problem of general bug management in software systems. Hu et al. [21] automated this process by constructing a developer-component-bug network, which models the relationship among developers, source code components, and the associated bugs.

Some approaches have been proposed to address the service dependency issues for cloud services. For example, Bahl et al. [1] presented Leslie Graph, an abstraction describing the complex dependencies between network, host, and application components in networked systems. Particularly, they systematically discussed the challenges and difficulties of discovering service dependencies. Chen et al. [7] introduced Orion, a system searches dependencies using packet headers and timing information in network traffic.

#### 6.2 Analysis of Incident Management

Recently, cloud service incidents and their management are gaining more and more popularity. For example, Zhou et al. [40] performed an empirical study on the quality issues of a big data computing platform. They analyzed 210 real service quality issues and investigated their common symptoms, causes, and mitigation solutions. Their findings show that 21.0% of major issues encountered by customers are caused by hardware faults, 36.2% are caused by system side defects, and 37.2% are due to customer side faults. Huang et al. [22] studied the gray failures in production cloud-scale systems. They found this type of failure is hardly noticed by system’s failure detectors even when applications are afflicted by them. Chen et al. [5] studied the incident triage problem on 20 large-scale online service systems in Microsoft. Their results reveal the fact that incorrect assignment of incident reports occurs frequently, especially for the incidents with high severity. Dang et al. [12] summarized the real-world challenges in building AIOps solutions and proposed a roadmap of AIOps related research directions. Gunawi et al. [16] conducted a cloud outage study of 32 popular Internet services. They provided answers to why outages still take place in cloud environments by analysing 1,247 headline news and public post-mortem reports which detail 597 unplanned outages. While these work only studies one specific aspect of cloud system’s incident management, we present a comprehensive characterization for it.

### 7 CONCLUSIONS

With years of efforts, cloud incident management has become much more automated and faster. However, some critical incidents still occur in an unexpected manner and thus require intensive human effort. In this paper, we summarize two main challenges incurring such inefficient and error-prone workflow: (1) the lack of a fine-grained service/resource dependency graph; and (2) the imprecision of health assessment for cloud resources. Particularly, the dependency graph is dramatically complicated by system modularity and visualization technology. While fault tolerance mechanism could sometimes impede the detection of unhealthy resources, the imperfection of monitor design and distribution further compound the problem. We conduct quantitative analysis of incidents from six core services at Microsoft and provide five real-world incident examples as well as four lessons learned. We also present BRAIN, our AIOps framework, which is able to effectively reduce the time cost in different incident management phases.

We believe our work could shed light on future research and engineering effort towards failure-resilient cloud systems, for example, high-performance algorithms for accelerating different incident management phases, design of efficient incident management workflow, and more advanced cloud architecture.

### ACKNOWLEDGMENTS

The work was supported by the National Natural Science Foundation of China (No. 61702107), the Research Grants Council of the Hong Kong Special Administrative Region, China (CUHK 14210717), and the Guangdong Key Research Program (No. 2020B010165002).
REFERENCES


