Stability is not Downtime: Comprehensive Stability Evaluation for Large-Scale Cloud Servers in Alibaba Cloud

Haoyu Wang*
Alibaba Cloud Computing
Hangzhou, China
lingzun.why@alibaba-inc.com

Zhicheng Liu*
Alibaba Cloud Computing
Beijing, China
fuxiao.lzc@alibaba-inc.com

Yeliang Qiu
Alibaba Cloud Computing
Hangzhou, China
yeliang.qyl@alibaba-inc.com

Haozhe Li
Alibaba Cloud Computing
Beijing, China
leon.lhz@alibaba-inc.com

Hongke Guo
Alibaba Cloud Computing
Beijing, China
guohongke.ghk@alibaba-inc.com

Zhaoliang Zhu
Alibaba Cloud Computing
Hangzhou, China
rongdi.zzl@alibaba-inc.com

You Zhang
Alibaba Cloud Computing
Hangzhou, China
zhangyou.zy@alibaba-inc.com

Yu Zhou
Alibaba Cloud Computing
Beijing, China
tuhu@alibaba-inc.com

Xudong Zheng
Alibaba Cloud Computing
Hangzhou, China
xudong.zxd@alibaba-inc.com

***Abstract*—As one of the most critical cloud products, the stability of cloud servers is of paramount importance. Traditionally, stability is evaluated based on downtime, i.e., the duration when cloud servers are unavailable. However, through our extensive engagement in stability-related work, we have discovered that only 27% of stability issues are related to unavailability, which is in fact a subset of stability. Therefore, in this paper, we propose the Comprehensive Damage Indicator (CDI), comprising three distinct sub-metrics: the Unavailability Indicator, Performance Indicator, and Control-plane Indicator. The CDI is able to evaluate the stability of large-scale cloud servers more comprehensively. In the production environment of Alibaba Cloud, the CDI has been implemented on top of Apache Spark. Over the past two years, we have employed the CDI to guide our stability-related works, including architecture comparison, potential problem detection and operation action optimization. The overall results are very favorable. Taking Fiscal Year 2024 as an example, the three sub-metrics have respectively decreased by 40%, 80%, and 35%.**

***Index Terms*—Stability, Evaluation Metric, Cloud Server, AIOps**

I. Introduction

Over the past few years, cloud computing has experienced rapid development. The market size escalated to $560 billion dollars in 2023 and is projected to continue with an annual growth rate of about 20% [11]. Among the array of cloud products, cloud servers (referred to as ECS at Alibaba Cloud, EC2 at AWS, and VM at Azure) are one of the most significant ones. They boast a substantial market presence and also act as the foundation for various other cloud services. For instance, at Alibaba Cloud, most other cloud products are built on the basis of ECS. Therefore, the stability of cloud servers is vital.

For cloud vendors, due to the extensive scale of cloud servers under their management, failures are inevitable. To mitigate the impact on customers and decrease operational expenses, Artificial Intelligence for IT Operations (AIOps) [12] [13] has been extensively employed, resulting in a lot of research. Some studies concentrate on the construction of monitoring systems [14], which provide data support for operational tasks. Meanwhile, a significant amount of research is directed toward the prediction and detection of failures, including node failures [15] [16], disk failures [17] [18] [19] [20], memory failures [21] and network traffic anomalies [22] [23]. Moreover, failure recovery strategies [24] are also explored in the face of severe infrastructure failures.

A. Problem

At Alibaba Cloud ECS, in a similar vein, we have constructed an AIOps system, CloudBot, to maintain stability. It collects a vast amount of high-quality data and extracts it into interpretable events, which then enables the automatic execution of operation actions through the operation rule matching.

We believe that the core of AIOPS is data-driven. In addition to data mining, the role of quantitative evaluation cannot be overstated, as it furnishes AIOPS with objective direction. Therefore, a quantitative stability evaluation metric for large-scale cloud servers is required to drive the evolution of stability work.

*Haoyu Wang (https://wanghy.pages.dev/) and Zhicheng Liu contributed equally to this research.

B. Requirements

For cloud vendors such as Alibaba Cloud, stability evaluation needs to satisfy the following requirements:

1) Interpretability: The purpose of the evaluation is to provide guidance for the development of stability initiatives. Considering that stability encompasses the collaboration of numerous teams, an interpretable evaluation is more convincing than a black-box one.

2) Cost-Effectiveness: Considering the vast scale of ECS, a heavyweight evaluation would result in significant extra resource usage. Ideally, this evaluation should rely on already available, small-scale data sources, without the requirement for additional data collection.

3) Non-Invasiveness: As the cloud servers are entirely under customer ownership, we cannot operate them without authorization. Thus, any form of intrusive evaluation is unacceptable.

4) Comprehensiveness: The cloud server is a complex product that integrates capabilities such as computing, storage, and networking to deliver services in the form of virtual machines (VMs). Customers, in turn, use VMs for a diverse range of business purposes. Hence, the stability evaluation should be comprehensive.

C. Intuition

Existing evaluation methods can generally be divided into two categories. The first category measures stability by executing benchmarks within VMs [25]. However, this approach is intrusive and generates considerable computational costs, rendering it unsuitable for large-scale evaluation. Therefore, cloud vendors typically use another method based on downtime [4]. This method focuses on the duration (or frequency) of cloud server unavailability, meeting the requirement of cost-effectiveness and non-invasiveness.

However, in our long-term work on ECS stability, we find that many issues and incidents are unrelated to unavailability, but still have a severe impact on stability. As discussed in Section III-A, our insight is that **stability is not equivalent to downtime**.

In Section III-B, we provide a definition of cloud server stability and categorize the stability issues into unavailability, performance, and control-plane issues. Grounded on this definition, we introduce the Comprehensive Damage Indicator (CDI), which extends the stability evaluation from unavailability to all three categories. Meanwhile, CDI calculation is based on interpretable intermediate events from CloudBot, considering the varying duration and impacts of different events. Therefore, CDI more comprehensively mirrors the complication of cloud servers and offers stronger guidance in the evolution of stability.

D. Contributions

Our major contributions are summarized as follows:

(1) Leveraging our extensive experience in ECS stability, we discover that downtime alone insufficient for a comprehensive stability evaluation. We define the stability of cloud servers as Definition 1, explicitly stating that it encompasses three aspects: unavailability, performance, and control-plane. This definition provides a more holistic view of system stability, which is often overlooked in traditional evaluations.

(2) We propose Comprehensive Damage Indicator (CDI), which is the first comprehensive quantitative stability evaluation metric for large-scale cloud servers, to the best of our knowledge. This metric comprises three sub-metrics: the Unavailability Indicator, Performance Indicator, and Control-Plane Indicator. Unlike existing methods such as Downtime Percentage, CDI provides a more comprehensive and nuanced evaluation of system stability by considering multiple dimensions.

(3) CDI has been deployed in various domains for over two years within Alibaba Cloud ECS. It is utilized to compare the stability of two different architectures and to detect potential issues. Besides, together with A/B test, CDI is also employed to identify the optimal operation actions.

(4) Guided by CDI, the stability of Alibaba Cloud ECS has shown significant improvement. Specifically, in Financial Year 2024, there was a 40% reduction in the Unavailability Indicator, an 80% reduction in the Performance Indicator and a 35% reduction in Control-Plane Indicator.

The frequently used terms in the proposal are listed in TABLE I.

TABLE I: Terminology Explanations

Term	Explanation
Elastic Compute Service (ECS)	Product name of cloud servers in Alibaba Cloud.
Virtual machine (VM)	Software emulation of a physical computer. Customers utilize computing services in the form of VMs.
Node controller (NC)	Physical machine to host VMs.
Internet Data Center (IDC)	Facility used to house physical server systems.
Event	Abstraction of the cloud server's status. It is interpretable and often corresponds to potential stability issues.
Operation rule	Rule that defines the combination of events and the corresponding actions that need to be executed.
Operation action	Action executed after the match of operation rules. It is employed for mitigating risks or recovering services.
Cloud disk	Storage disk on the cloud which provides storage service to ECS.

II. Background of CloudBot

Alibaba Cloud ECS maintains an infrastructure that includes more than a million physical servers and tens of millions of virtual machines. As the responsible team of stability, we have built a system called CloudBot. Utilizing large-scale, highly accurate monitoring data, CloudBot conducts real-time detection and analysis of the stability of both physical and virtual machines, and automatically executes operation actions to ensure the stability of cloud services. In this section, we will introduce the CloudBot system.

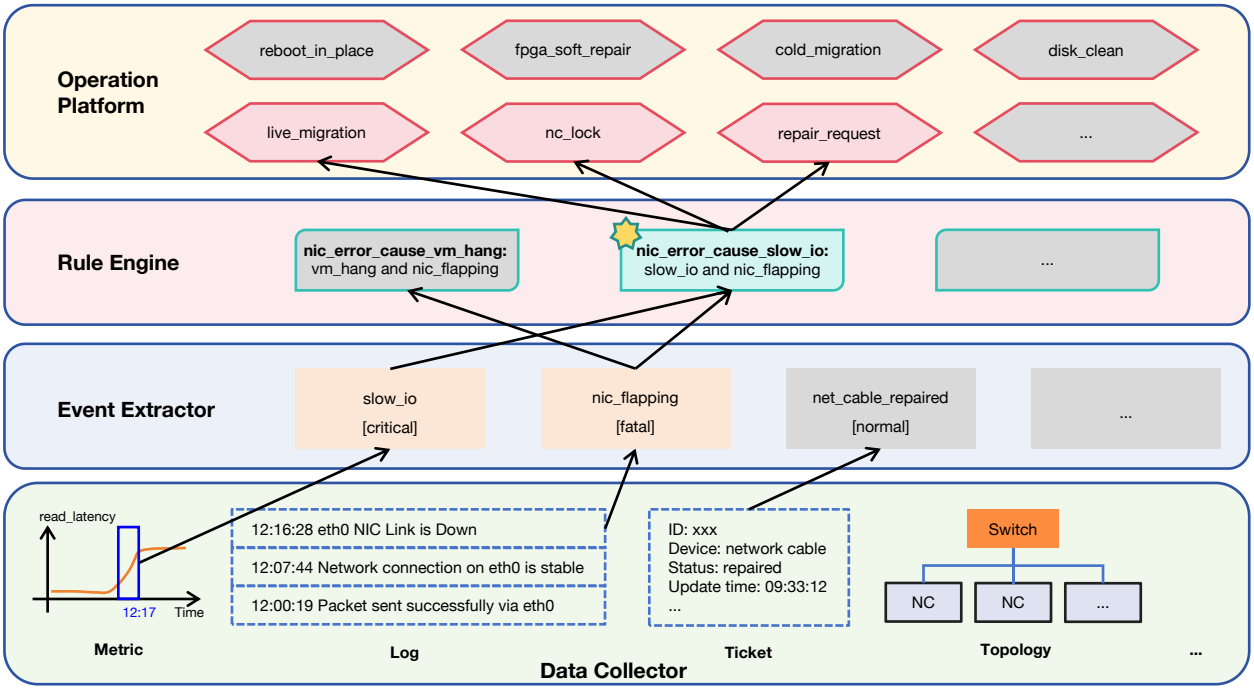


Fig. 1: Example process workflow of CloudBot

A. Overview

CloudBot is designed to reduce the workload of engineers by automating the operations of virtual and physical machines. Example 1 is provided to demonstrate how CloudBot works.

Example 1 (NIC)**.** *Fig. 1 illustrates the architecture of CloudBot and the exemplary process workflow for addressing a network interface card (NIC) issue. Data Collector (Section II-B) gathers various modalities of data, including metrics, logs, tickets, and topology, to support the automated operation of CloudBot. For example, the* ***read_latency*** *metric depicted in the figure is a critical indicator of cloud disk I/O performance. To identify the cause of a sudden increase in latency, we collect various types of data such as NIC logs, IDC tickets of cables, and network topology.*

After that, Event Extractor (Section II-C) transforms the multi-modal raw data into a unified format of events. For instance, at 12:17, the recent measurements from the metric ***read_latency*** *for a VM meets a sudden spike. CloudBot extracts it as a* ***slow_io*** *event with a critical level. Meanwhile, according to a regular expression defined by experts, the log entry* `eth0 NIC Link is Down` *at 12:16:28 is recognized as a* ***nic_flapping*** *event, while other two entries are discarded. Besides, the IDC ticket is transformed to a* ***net_cable_repaired*** *event.*

Following event extraction, Rule Engine (Section II-D) is utilized for matching rules. The co-occurrence of ***slow_io*** *event and* ***nic_flapping*** *event precisely matches the operation rule* ***nic_error_cause_slow_io****. Besides, due to the absence of* ***vm_hang*** *event,* ***nic_error_cause_vm_hang*** *rule cannot be matched with* ***nic_flapping*** *event only.*

Finally, Operation Platform (Section II-E) manages the execution of all operation actions in CloudBot system. The matched ***nic_error_cause_slow_io*** *rule triggers three of the actions. In detail, the VM undergoes a live migration, moving to a new physical machine without a shutdown. At the same time, a ticket to IDC engineers is created for repair request. Throughout the repair duration, the physical machine is locked, preventing the VM creation on it and migration to it, keeping other VMs from potential issues.*

B. Data Collector

Data Collector, a self-developed lightweight component based on eBPF [26], is the cornerstone of CloudBot. This component enables us to execute fine-grained data collection facilitating the precise detection and diagnosis of problems. For instance, in Case 5 (detailed in Section VI-B), we collect the allocation relationships between each VM and physical core, which supports more efficient resource management. Additionally, for power consumption issues addressed in Case 7 (detailed in Section VI-C), we collect power metrics across a spectrum of granularity, including the racks, machines, hardware components, CPU sockets, and individual physical cores.

C. Event Extractor

CloudBot interfaces with hundreds of diverse data sources, each generating substantial quantities of heterogeneous data. Event Extractor standardizes the data into a consistent event-based format. An event describes an anomalous objective phenomenon, but does not necessarily point to a real issue. For

TABLE II: Fields of Event

Field	Description
name	interpretable name of the event, e.g., slow_io
time	timestamp when the event is extracted
target	target of the event, often a VM or a physical machine
expire_interval	time interval between the extraction and the expiration of the event
level	severity level of the event determined by the target, e.g., fatal, critical, warning

example, **slow_io** could be caused by an excessive workload on the VM. There are three ways to extract events as follows:

- **Expert rules:** Based on an understanding of the underlying principles, experts can manually formulate rules to extract events from original data. This approach results in a notably high degree of precision. The transition from logs and metrics to events, illustrated in Fig. 1, serves as an exemplar of such expertly devised rules.
- **Statistic-based methods:** Statistical methods are adept at mining data attributes while providing substantial interpretability. For instance, we combine BacktrackSTL [27] and EVT [28] to detect anomalies within metric time series, which are then considered as events.
- **Deep learning algorithms:** Deep learning techniques excel at extracting sophisticated, high-level events, offering the potential to outperform human experts. Consequently, in particularly challenging situations like server failure prediction [7] [8], we explore the use of neural networks to detect machines that may be at risk, with the aim of reducing the impact of failures on customers.

The core purpose of extraction is to reduce the complexity of the CloudBot system. Firstly, the extraction standardizes the data modals, thus circumventing the need to handle multi-modal data. Secondly, since the vast majority of machines run normally and are not the focus of event extraction, the event data volume is greatly reduced from hundreds of TB to GB per day, significantly enhancing information density. In addition, as interpretable intermediary outcomes, these events also aid in the debugging and update of the CloudBot system.

After extraction, we obtain a series of events for rule matching. Each event is uniquely identified by its name, extraction time, and associated target. To manage the quantity of events effectively, each one is assigned an expiration interval, which is predefined and specific to the event name. In addition, each event is assigned a level that indicates its severity. It is crucial to acknowledge that events with identical names may correspond to varying levels of severity, depending on the particular conditions of the target. TABLE II documents all the fields of the events.

D. Rule Engine

In CloudBot, an operation rule contains a readable boolean expression and several operation actions. When the concurrent occurrence of events fulfills the expression, the operation rule is considered matched. Consequently, the corresponding actions are submitted to Operation Platform for execution.

The establishment of operation rules is a combination of expert knowledge, algorithmic insights, and empirical tests. A portion of the rules are crafted manually by experts. Based on association mining algorithms [29], we can optimize existing rules and discover new rules. Furthermore, as described in Section VI-D, A/B test [30] can further contribute to the rule formulation process.

E. Operation Platform

Since operation actions have a direct effect on virtual and physical machines, and consequently on customers, Operation Platform is designed to centrally control all such actions. This platform determines the execution order for all submitted operation actions and discards the conflicting ones. The action categories are shown in Table III.

F. Discussion on Operation Accuracy

1) Automatically inferred events: CloudBot generates a large volume of events, including those from statistical and deep learning methods. Since a single event only captures one aspect of the cloud server's status, operation actions based solely on individual events may lead to significant noise, i.e., incorrect operation actions.

Thus, the concept of operation rule is introduced to reduce noise, which combines multiple events with meta-information such as product configurations. For example, CPU contention on a shared VM is consistent with the product definition [1] and needs no actions.

Besides, the trend analysis of events can also be utilized for noise reduction. Specifically, we introduce the event-level CDI in Section VI-C to reflect its trend. Even though a single event may not correspond to a rule, anomalous fluctuations in trends can also indicate potential issues.

2) Missing Operation: Due to our limited understanding of the cloud server system, missing operation is a rare but inevitable issue. A small number of missing operations can only be identified through customer complaints. However, for the unexpected surge in events and the potential batch of missing operations it may trigger, we establish an alert mechanism. If the event is unrelated to user behavior or if the surge is influenced by multiple customers, engineers are requested to intervene immediately. Additionally, we regularly review and update the rules to ensure that they cover a wider range of failure conditions and reduce the likelihood of missing operations.

III. Motivation

With the development of stability works including the CloudBot system, evaluating its effectiveness is imperative. In other word, we have to evaluate the stability of large-scale cloud servers. Traditionally, the industry has relied on the server downtime for stability evaluation. This section provides a contemplation on the adequacy of this conventional metric.

TABLE III: Operation actions in CloudBot

Type	Example Actions
VM operation	Live migration (migrate a VM without shut-down), in-place reboot (reboot a VM on the same NC), cold migration (reboot and migrate a VM)
NC software repair	Disk clean, memory compaction, process repair (restart or update the process)
NC hardware repair	device disable (disable the specific device), repair request (create a ticket to IDC engineers for repair), FPGA soft repair (repair the error in FPGA with software or configurations)
NC control	NC reboot, NC lock (halt the creation or migration of new VMs to NC), NC decommission (remove NC from the production environment)

A. Stability ≠ Downtime

When discussing stability, it is often equated with the Downtime Percentage. This metric quantifies the proportion of time during which a cloud server is unavailable relative to the total service time. As an intuitive measure, it is extensively utilized within the industry.

Expanding on this, Azure proposes the concept of Annual Interruption Rate [4]. This approach posits that long-duration of unavailability are rare, and thus, it substitutes the duration of unavailability with the frequency to evaluate customer impact better. Essentially, Annual Interruption Rate remains focused on the unavailability of cloud servers.

However, in our daily work, the customer-reported stability issues are not limited to unavailability. The following is a case.

Case 1. *A customer once submitted a support ticket calling for an investigation. They reported that the latency of an API running on a specific VM had markedly increased during a designated period. Our investigation revealed that the issue stemmed from a recent change. Since the customer's business was more sensitive than we had anticipated, a change we had expected to be non-disruptive actually impacted their operations. As a result, we immediately halted the change and reclassified it as disruptive. Subsequent changes were only deployed within a window established in concordance with the customer, to prevent the recurrence of such impacts.*

Besides, issues other than unavailability can also result in serious incidents. The incident on November 12, 2023, serves as a prime example [31].

Case 2. *On November 12, 2023, Alibaba Cloud experienced a serious incident. Faulty logic within the AccessKey system led to an incomplete whitelist, causing the requests sent from valid sources failed to pass the authentication process. From the data plane, although some encrypted disks were unavailable, most of the existing cloud servers continued to run normally. However, the control plane has encountered more severe issues, including the loss of monitoring metrics for cloud servers, inability to log in to the console, and failures of management API calls. This incident occurred in the early evening, which is the business peaks for many customers, leading to a significant loss in their business.*

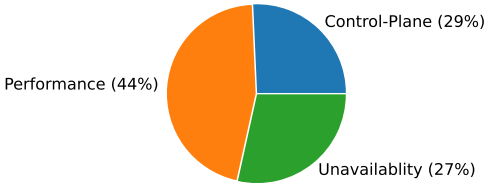


Fig. 2: Distribution of tickets related to ECS stability

In summary, a conclusion can be drawn that **stability is not downtime**. Therefore, an in-depth thinking to stability is necessary.

B. Cloud Server Stability

Based on our long-term practice in the field, we provide the following definition for the stability of cloud servers:

Definition 1. *The stability of cloud servers is defined as its capacity to deliver and manage computational resources in a continuous and consistent manner.*

According to the above definition, we discover that unavailability disrupts the continuity of computational power, which is actually just a subset of stability. Meanwhile, stability concerns also involve performance issues and control-plane issues.

Performance issues undermine the consistency of computational power, including both temporal consistency of a single VM and the cross-VM consistency among VMs with identical specifications. They are common in our routine work, often associated with changes, hardware, and various other factors. For example, Case 1 shows a performance issue.

Control-plane issues obstruct the management of computational power, such as starting, stopping, releasing, or resizing VMs, directly affecting the customers' cloud usage experience. Moreover, the impact of control-plane issues can be global, such as in Case 2.

Quantitatively, we categorize all tickets related to ECS stability from January 2023 to June 2024. As depicted in Fig. 2, tickets pertaining to unavailability constitute merely 27%, whereas those concerning performance and control-plane issues account for 44% and 29%, respectively. It is evident that an evaluation metric centered solely on unavailability cannot fully represent stability. Therefore, to better guide efforts in enhancing stability, it is crucial to employ a more comprehensive metric in place of downtime.

C. Rethink the Events in CloudBot

First, let us revisit the computation of Downtime Percentage in CloudBot system. This calculation is based solely on downtime events, which represent only a subset of all possible events. Therefore, it is a natural consideration to include all events in the stability evaluation.

Based on this idea, we find that CloudBot events align perfectly with the requirements presented in Section I-B. As an interpretable intermediate representation, they comprehensively reflect the various stability issues of cloud servers. By

reusing these events to define stability evaluation metrics, it is possible to evaluate large-scale cloud servers in a low-cost and non-intrusive manner.

Therefore, we propose a novel comprehensive stability evaluation metric based on CloudBot events, which we utilize to guide the evolution of stability. In Sections IV-VI, we provide a detailed explanation of its definition, implementation, and application.

IV. Comprehensive Damage Indicator

Building on the motivation presented in Section III, we employ CloudBot events to define the evaluation metric for large-scale cloud server stability, named Comprehensive Damage Indicator (CDI). In this section, we provide a detailed introduction of CDI.

A. Overview

As discussed in Section III-C, CloudBot events form the foundation of CDI. In the computation process, we represent an event $\mathbf{e}$ as $\mathbf{e} = (t_s, t_e, w)$. Here, t_s and t_e denote the start and end timestamps of the event, as delineated in Section IV-B, while w represents the weight discussed in Section IV-C. The method for calculating CDI based on these events is then presented in Section IV-D.

Additionally, since events can be categorized into three types in Section III-B, our CDI can also be divided into the following three sub-metrics. The calculation process for each is identical, and the only difference lies in the specific events they rely on.

- **Unavailability Indicator**: Unavailability issues denote situations where a VM is completely unavailable to provide computational services, including VM crashing or stalling. Unavailability Indicator quantifies the ratio of the unavailability duration to the total service time.
- **Performance Indicator**: Performance issues refer to situations where a VM performs below expectations though it is still available, including slow IO of cloud disks or frequent network packet loss, etc. Performance Indicator quantifies the ratio of the weighted performance impact duration to the total service time. The weights are determined by the specific impact level of the performance issues.
- **Control-Plane Indicator**: Control-plane issues refer to the inability to execute control operations on a VM, including failures to start, stop, or release the VM. Control-Plane Indicator quantifies the ratio of the weighted uncontrollability duration to the total service time. The weights are determined by the specific impact level of the control-plane issues.

It should be noted that the aforementioned three indicators quantify the impact on three major categories of stability issues, distinct from specific metrics like CPU and IO. As described in Section II-C, fluctuations in CPU and IO metrics will be abstracted as events, and thus, will ultimately be incorporated into the CDI.

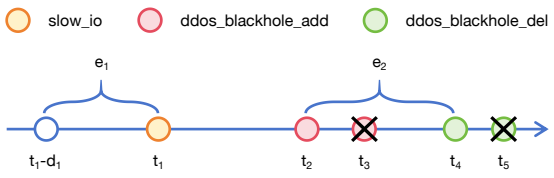


Fig. 3: Example of event period

B. Event Period

Events can be categorized into two types: stateless events and stateful events. The method for calculating their period differs.

1) Stateless Event: Stateless events are independent of each other. A single event represents a specific complete issue during a certain period of time. To simplify the CloudBot system, the majority of events are designed to be stateless. In this case, we consider the event's timestamp as its end time and calculate the start time by tracing backward from the duration.

Some of these events are directly associated with estimated duration. For instance, the event **qemu_live_upgrade_event** depicts the live upgrade of the virtualization component QEMU [32], and the pertinent logs precisely log the impact duration in milliseconds.

However, most stateless events are extracted without estimated duration. For instance, the event **slow_io** indicates that the read latency of VM exceeds a threshold. These types of events are associated with a small time window, e.g., 1 minute. When a VM is persistently compromised, the event will be produced consecutively. Hence, the duration of the event can be approximated using the size of the time window.

2) Stateful Event: Some events from other teams are stateful. They have dependencies among them, representing the start, end, and other states of an issue based on multiple detailed events. For example, **ddos_blackhole** event from security teams is represented by two detailed events, **ddos_blackhole_add** and **ddos_blackhole_del**. Therefore, the start time of **ddos_blackhole** event is the timestamp of **ddos_blackhole_add**, and the end time is the timestamp of **ddos_blackhole_del**.

In practice, each start event is paired with the nearest subsequent end event. To mitigate the impact of dirty data, among all consecutive occurrences of events, only the event with the earliest timestamp is preserved. This procedure ensures that each start event has a unique end event as its pair.

Example 2 (Event Period)**.** *Suppose the events of a VM are illustrated in Fig. 3. Each event is associated with a timestamp. e_1 is a **slow_io** event ends at t_1. Given the duration of **slow_io** is d_1, the start time of e_1 is $t_1 - d_1$.*

*For **ddos_blackhole** event, we first filter out consecutive occurrences of the same event. Specifically, **ddos_blackhole_add** events at t_2 and t_3 are redundant. Thus, the one at t_3 is discarded. Similarly, **ddos_blackhole_del** at t_5 is also dis-*

carded. After filtering, it is clear that the VM experiences a ***ddos_blackhole*** *event e_2. It starts at t_2 and ends at t_4.*

C. Event Weight

When an unavailability issue occurs, the VM will be completely unavailable to provide computing services. However, this is not the case for performance and control-plane issues. For example, a GPU drop leads to a significant loss of computing power, while slight network packet loss may remain undetected by the customer. It is evident that the seriousness of the former greatly outweighs that of the latter. Therefore, it is necessary to allocate varying weights to distinct events to illustrate this difference in severity.

Based on expert knowledge, during the event extraction process in Section II-C, we have already defined a level for each event, e.g., fatal, critical, warning, etc. Suppose that there are m different levels of increasing severity, the weight assigned to the i-th level is as follows:

$$l_i = \frac{i}{m}, \quad 1 \leq i \leq m \tag{1}$$

However, the severity as perceived by experts may not necessarily equate to that perceived by customers. Consequently, we gather the number of related tickets for each event over the previous year to determine weights reflective of customer perception. The intuitive idea is that the quantity of complaint tickets has a positive correlation with the impact on customers. Therefore, we rank the events based on the related ticket counts, and proportionately distribute them into several levels based on their ranking positions. Similarly, among n levels order by ascend ticket counts, the j-th one has the following weight:

$$p_j = \frac{j}{n}, \quad 1 \leq j \leq n \tag{2}$$

Finally, we employ the Analytic Hierarchy Process (AHP) [3] to integrate the weights of various perspectives. It is a multi-criteria decision-making approach. This method conducts pairwise qualitative comparisons of the importance of perspectives, constructing a judgment matrix, and subsequently calculating the proportion of each perspective in the final weight distribution. Suppose the event is in the i-th expert level and j-th customer level, with the corresponding proportion α_1 and α_2 from AHP, the final weight is as follows:

$$w = \frac{\alpha_1 l_i + \alpha_2 p_j}{\alpha_1 + \alpha_2} \tag{3}$$

In future work, it is possible to include a broader array of perspectives, such as the sensitivity of customers to events. AHP can also be applied to integrate these additional perspectives.

Example 3 (Event Weight)**.** *Suppose there is an event whose level is critical with $m = n = 4$ and $\alpha_1 = \alpha_2 = 0.5$. First, since critical is the third level of severity in expert perception, the expert weight is $l_3 = 3/4 = 0.75$. Then, suppose the related ticket counts of this event is higher than 43% of all events, it falls into the second level. Thus, the customer weight is $p_2 = 2/4 = 0.5$. At last, the final weight is $w = \frac{\alpha_1 l_3 + \alpha_2 p_2}{\alpha_1 + \alpha_2} = 0.625$.*

Algorithm 1: CDI Calculation

Data: VM events $e[1 \dots n]$, service period T_s to T_e
Result: CDI of the VM in the service period Q

```
W[T_s ... T_e] ← 0;
for i ← 1 to n do
    (t_s, t_e, w) ← e[i];
    W[t_s ... t_e] ← max(w, W[t_s ... t_e]);
end
Q ← 1/(T_e − T_s) Σ_{t=T_s}^{T_e} W[t] · Δt;
return Q
```

D. Indicator Calculation

The CDI of a particular VM in a specific service period depends on all associated events. Overlaps can occur between events. When multiple events coincide within a particular time segment, the segment weight is the maximum value of the weights of these events. Algorithm 1 illustrates the procedure for computing the CDI.

For a collection of VMs, since events among VMs do not overlap, the CDI for the entire collection can be straightforwardly calculated by aggregating the CDIs of all individual VMs. The formula is as follows:

$$Q = \frac{\sum_{i \in \mathbf{V}} T_i Q_i}{\sum_{i \in \mathbf{V}} T_i} \tag{4}$$

Here, $\mathbf{V}$ represents the collection of VMs. Moreover, T_i denotes the service time, and Q_i signifies the CDI of VM i. To elucidate the calculation process, the Performance Indicator is employed as an example to illustrate the steps involved in computing the CDI in Example 4.

Example 4 (Indicator Calculation)**.** *Suppose TABLE IV lists the performance events of 3 VMs on a certain day. For VM 1, there are two non-overlapped* ***packet_loss*** *events, both of which lasts 2 minutes with weight $w = 0.3$. Thus, its CDI is as follows:*

$$Q_1 = \frac{2 * 0.3 + 2 * 0.3}{60} = \frac{1.2}{60} = 0.020$$

Similarly, the CDI of VM 2 is $Q_2 = 5 * 0.6/1440 = 0.002$.

For VM 3, it is sure that the weight is $w = 0.5$ at 08:08-08:10 and $w = 0.6$ at 08:12-08:15. Since the ***slow_io*** *event is overlapped with the* ***vcpu_high*** *event at 08:10-08:12, the final weight is the higher one, i.e., $w = 0.6$. Therefore, the CDI of VM 3 is as follows:*

$$Q_3 = \frac{2 * 0.5 + 2 * 0.6 + 3 * 0.6}{1000} = \frac{4}{1000} = 0.004$$

Furthermore, we can also calculate the Performance Indicator for all three VMs:

$$Q_{all} = \frac{60 * 0.020 + 1440 * 0.002 + 1000 * 0.004}{60 + 1440 + 1000} = 0.003$$

TABLE IV: Example of CDI Calculation

VM	Service Time	Event	Period	Weight	CDI
1	60min	packet_loss packet_loss	10:08-10:10 10:10-10:12	0.3 0.3	0.020
2	1440min	vcpu_high	13:25-13:30	0.6	0.002
3	1000min	slow_io slow_io vcpu_high	08:08-08:10 08:10-08:12 08:10-08:15	0.5 0.5 0.6	0.004
All	2500min	-	-	-	0.003

V. Implementation

Since 2022, the CDI has been deployed within Alibaba Cloud ECS to steer our stability efforts. The CDI is computed on a daily basis as depicted in Fig. 4.

In CloudBot, the original event data is stored in the Simple Log Service (SLS) [33] to facilitate rapid searching and is subsequently synchronized to a MaxCompute [6] table for long-term storage. Additionally, a ticket classification model is deployed on Platform For AI (PAI) [34]. Based on the classification results and expert insights, the configurations in MySQL [35] are adjusted as detailed in Section IV-C.

Following that, we developed an Apache Spark [36] application for the computation of the CDI. This application processes events from the MaxCompute table and configuration data from MySQL, yielding two MaxCompute tables. The first table presents the Unavailability Indicator, Performance Indicator, Control-Plane Indicator, and service time for each VM. The second table offers a granular perspective at the event level, with each row recording the CDI of an event on a specific VM.

The input data for the application is about 10 GB volume. We allocate 100 executor instances for this application, each with 8 cores and 12GB of memory. The entire time overhead is around 2 hours, with the majority of the time spent on data cleaning, filtering, integration, and writing. The core CDI computation time is around 500 seconds.

Ultimately, the CDI is visualized on our internal Business Intelligence (BI) system. This system facilitates SQL queries. Utilizing the two aforementioned tables as the foundational data, it is able to aggregate the CDI across diverse dimensions in accordance with Formula 4. For example, we can concentrate on the global CDI and, if required, drill down to the region, availability zone, or even the cluster level.

VI. Application

A. Stability Evaluation

The primary application of the CDI is the quantitative evaluation of the stability of the Alibaba Cloud ECS production environment. Incidents are a critical factor in compromising stability. Based on the CDI, it is able to quantify and visually demonstrate the impact of incidents, as exemplified by the three incidents [37] [38] [39].

Case 3. *Three incidents are considered as examples: The first occurred in Availability Zone C of the Singapore region on April 25, 2024, impacting multiple products, including ECS. The second took place in Availability Zone N of the Shanghai region on July 2, 2024, due to network access abnormalities. The third occurred in the Shanghai region on January 7, 2025, leading to the inability to purchase or modify ECS instances.*

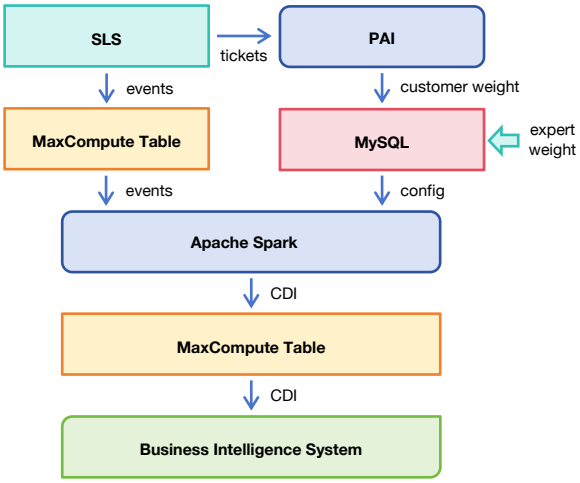


Fig. 4: Deployment of CDI calculation

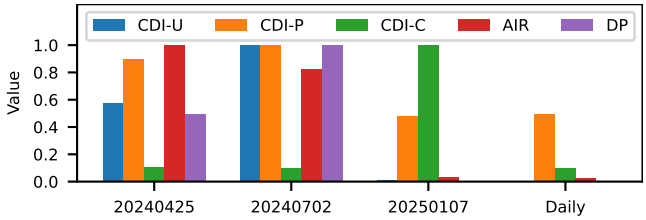
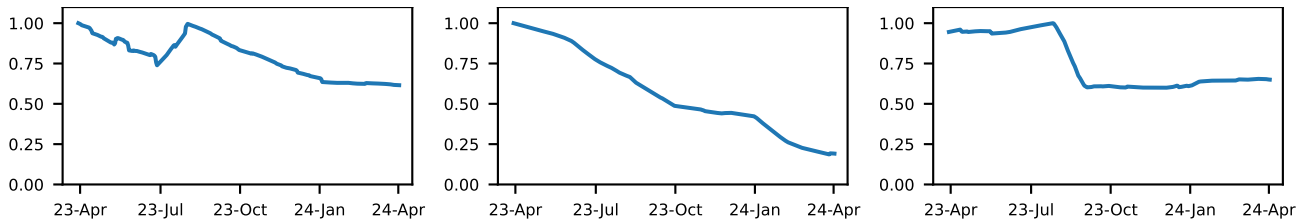


Fig. 5: Stability evaluation on selected incidents

Fig. 5 illustrates the CDI, Annual Interruption Rate (AIR) and Downtime Percentage (DP) on the incident days (named 20240425, 20240702 and 20250107 respectively) compared to that without incidents (named Daily). Due to the data security policies of Alibaba Cloud, all CDI, AIR and DP values referenced in this figure and the subsequent text are normalized.

It can be observed that AIR and DP exhibited very significant changes during the 20240425 and 20240702 incidents, compared to normal times. However, since the 20250107 incident did not affect the existing ECS instances, these indicators could not reflect the damage to stability caused by this incident. In contrast, CDI effectively reflected the impact of all three incidents. The first two were reflected in the unavailability indicator (CDI-U), while the last one was captured in the control-plane indicator (CDI-C). Therefore, CDI provides a more comprehensive evaluation of ECS stability.

Beyond incidents, the CDI is crucial for quantifying overall daily stability. It provides directed guidance for stability-related work. To reduce the daily CDI, many strategies are used. For Unavailability Indicator, fault prediction techniques [7] [8] based on deep learning is utilized, circumventing



(a) Unavailability Indicator (b) Performance Indicator (c) Control-plane Indicator

Fig. 6: Overall CDI from April 2023 to March 2024

the hardware unavailability by migration in advance. For Performance Indicator, virtualization technologies [9] are optimized to reduce performance degradation. For Control-Plane Indicator, we redundantly deploy significant components [10] to achieve high availability. The CDI curves serve to quantify the impact of these measures, enabling stability engineers to monitor stability trends and evaluate the efficacy of distinct stability strategies.

Case 4. *In the evaluation of stability efforts of Fiscal Year 2024 (April 2023 to March 2024), the CDI curve played a significant role. Fig. 6 displays the annual smoothed overall CDI curve for Fiscal Year 2024. The curve has been adjusted to exclude the impact of particularly significant incidents.*

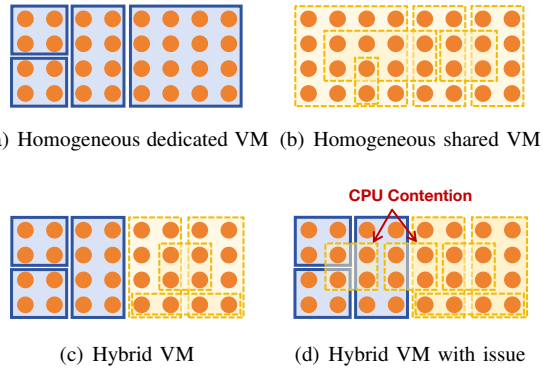
During Fiscal Year 2024, the stability of ECS experienced a significant improvement. The Unavailability Indicator, Performance Indicator, and Control-plane Indicator decreased by approximately 40%, 80%, and 35%, respectively. Among them, the Performance Indicator saw the largest reduction, primarily because its governance work was at an early stage, which typically yields more substantial benefits.

B. Architecture Comparison

Confronted with intense market competition, our technological architecture is in a state of constant evolution. Despite the existence of the impossible trinity in architectural design, we still strike a balance across various dimensions, including stability, security, performance, cost, and elasticity. During this process, the CDI is utilized to compare the stability of two architectures. Specifically, we expect the CDI of the new architecture to be comparable with that of the old one. Consequently, we persistently monitor the CDI for both architectures. Once a significant discrepancy arise, immediate action was taken to avert any substantial impact on stability.

Case 5. *In ECS, VMs are categorized into dedicated and shared types. Dedicated VMs are assigned to specific physical cores for exclusive resource use, ensuring consistent performance. In contrast, shared VMs are allocated across multiple cores, sharing resources with other users, which can lead to performance degradation during peak times.*

As shown in Fig. 7 (a) and (b), in the homogeneous-deployment architecture, dedicated and shared VMs are hosted on separate physical machines, creating two independent resource pools. This can result in inefficiencies, such as shared VMs being fully utilized while dedicated VMs have idle resources.



(a) Homogeneous dedicated VM (b) Homogeneous shared VM

(c) Hybrid VM (d) Hybrid VM with issue

Fig. 7: Schematic diagram of architectural transition

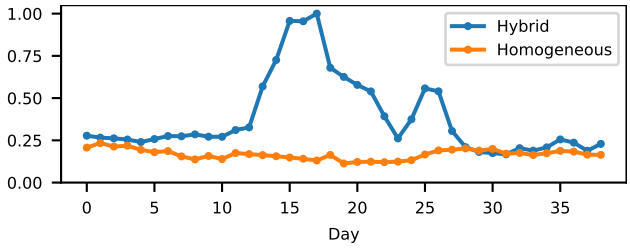
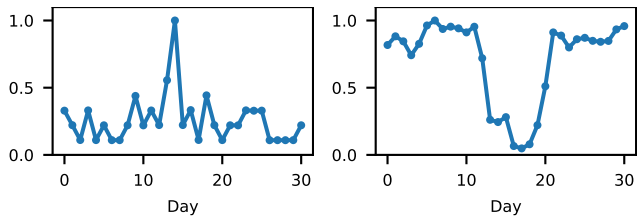


Fig. 8: Performance Indicator of deployment architectures

To improve elasticity and cost efficiency, we transitioned to a hybrid-deployment model, where both VM types are deployed on the same physical machine but on different cores, as shown in Fig. 7 (c). This merges the resource pools without affecting stability.

During the transition, we tracked the CDI of VMs within both architectures. Fig. 8 displays their Performance Indicators over a period of time. Initially, there was minimal variance, so we expanded the hybrid deployment. However, from Day 13, the Performance Indicators of the hybrid-deployment increased rapidly, prompting us to halt further expansion and investigate.

We found that the issue was due to an incompatibility between the hybrid architecture and certain virtualization components on a specific machine model. As shown in Fig. 7 (d), this caused CPU contention when the core allocation ranges



(a) vm_allocation_failed (b) inspect_cpu_power_tdp

Fig. 9: Event-level CDI for potential problem detection

of shared and dedicated VMs overlapped.

Given the complexity of fixing this, we temporarily avoided hybrid deployment on the affected machine model to sidestep the problem. We locked all of the affected physical machines, migrated the VMs, and steadily rolled them back to the homogeneous-deployment architecture. The Performance Indicators differences decreased, and by Day 28, the curves converged, indicating the issue was mitigated. This allowed us to continue with our architectural evolution.

C. Potential Problem Detection

The release of changes is a significant contributor to stability problems. Despite having implemented a system for gradual releases and circuit breaking, this system falls short in recognizing non-fatal problems that require an extended period to emerge. In these situations, the CDI offers valuable aid in detecting such potential problems.

The CDI and its associated drill-down curves are expected to change gradually. Therefore, a sudden and sharp change within a short period may signal a potential problem, necessitating further investigation. Specifically, in addition to the original CDI curve, we also pay attention to the drill-down curves at the event level. The computation is still consistent with Algorithm 1, except that the input is narrowed down from all events to specific events. For these curves, we apply techniques like K-Sigma and EVT [28] for detecting anomalies, and we leverage root cause analysis algorithms [40] to aid engineers in identifying the source of the problems. A specific example follows:

Case 6. *The scheduling system is responsible for the creation, resource allocation, and management of VMs. After a software change, some resource data within the scheduling system became erroneous. For example, suppose a physical machine has 104 CPU cores, but due to data errors in the scheduling system, the created VMs actually requires 108 cores in total. Thus, the last VM created would experience a allocation failure, unable to obtain the required computing resources. Without exclusive cores, its performance suffered a loss.*

*Fig. 9(a) illustrates the CDI for the **vm_allocation_failed** event over a period of one month. Upon detecting the spike on Day 14 through the algorithm, an immediate investigation is undertaken. We corrected the resource data and migrated the excessive VMs. As a result, the CDI for Day 15 reverted to expected levels.*

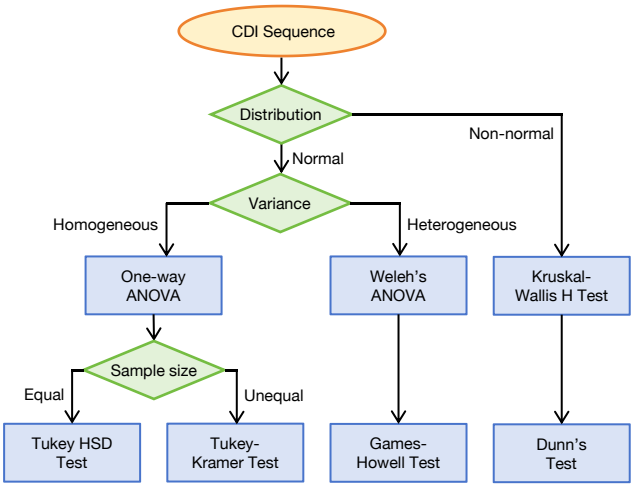


Fig. 10: Workflow of hypothesis test

Spikes in the CDI correspond to a decline in stability and necessitate heightened scrutiny. On the other hand, although dips are theoretically representative of an improvement in stability, a detailed analysis in practice is also required. These dips could result from the successful implementation of stability strategies or may signal the presence of a potential problem. An example is provided below:

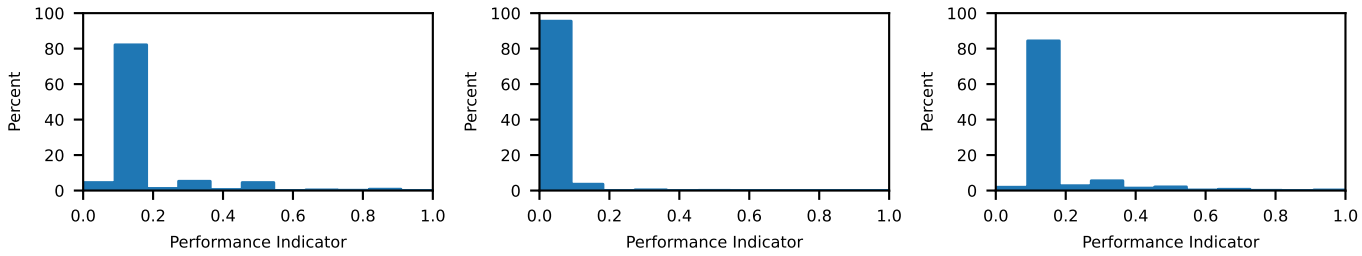
Case 7. *Thermal Design Power (TDP) is a term used to describe the maximum amount of heat generated by a CPU under normal operational conditions. To prevent overheating, if the CPU's actual energy consumption reaches this value, the CPU may reduce its frequency (operating speed), which can affect the performance of VMs running on it. Therefore, we collect CPU energy consumption data and compare it with the TDP to generate **inspect_cpu_power_tdp** events, which helps in monitoring performance.*

As shown in Fig. 9(a), the CDI began to decline on Day 13. Initially, we thought this was a success of the optimized scheduling strategy. However, by Day 17, the CDI had dropped to a very low level, far below our expectations. We discovered that the decline was due to an error in power collection module on physical machines, which resulted in the collected power being zero. We immediately fixed this issue.

Starting from Day 18, the CDI gradually returned to normal levels. This incident highlighted the importance of data quality monitoring and the limitations of focusing only on CDI increases. Consequently, we have since allocated equal scrutiny to both spikes and dips in the CDI.

D. Operation Action Optimization

For each operation rule, it is necessary to associate an appropriate operation action. Besides, in situations where there are multiple candidate actions, especially when candidates are similar, it is challenging for even experts to determine the optimal action. In such cases, the CDI integrated with A/B test, can serve a pivotal role in directing the optimization of operation actions.



(a) Action A (b) Action B (c) Action C

Fig. 11: Performance Indicator of each operation action

TABLE V: Hypothesis Test Results

Sub-metric	Omnibus Test		Post-hoc Analysis		
	P-value	Sign.	Pair	P-value	Sign.
Unavailability	0.47	False			
Control-plane	0.89	False			
Performance	0	True	A-B	0	True
			A-C	0.03	True
			B-C	0	True

Specifically, we initiate A/B test for operation rules requiring optimization. When a VM is hit by a rule, it will randomly carry out one of the potential actions, following a predefined probability distribution. For each VM, we calculate its CDI for the subsequent two days after the operation. Thus, for every action, we have a sequence of CDI values, with each element within the sequence corresponding to a VM which has implemented that specific action.

We employ hypothesis testing to analyze whether there are differences between actions. Firstly, omnibus test [41] is used to verify whether the differences between groups are statistically significant. If a significant difference is identified and the number of sample groups exceeds two, paired comparisons, known as post-hoc analysis [42], are applied to identify which specific groups exhibit significant differences.

The choice of tests for the omnibus test and post-hoc analysis varies according to the distribution, variance homogeneity, and the number of samples. We use one-way ANOVA [43], Welch's ANOVA [46] or Kruskal-Wallis H test [48] as the omnibus test. Meanwhile, Tukey HSD test [44], Tukey-Kramer test [45], Games-Howell test [47] or Dunn's multiple comparison test [49] are utilized for the post-hoc analysis. Fig. 10 presents the complete procedure for hypothesis testing.

It should be noted that the CDI contains three sub-metrics. Consequently, we need to carry out hypothesis testing three times, one for each sub-metric. Alternatively, it is possible to aggregate the three sub-metrics into a single one using techniques like weighted summation before proceeding with the test. Moreover, this methodology can also serve to evaluate the effectiveness of the operation rules if a null action is included as a comparison in the A/B test.

Case 8. *Based on a series of performance events, the operation rule* ***nc_down_prediction*** *is designed to forecast NC failures. Upon the prediction of a potential failure, an operation action is executed to prevent the NC failure from leading to VM unavailability.*

However, the selection of operation actions meets challenges. There are three candidates; although each action involves a live migration of all VMs on the NC, differences exist in migration parameters and sequences. In order to determine the most effective operation action, an A/B test is conducted utilizing the CDI.

This test spans three months. Based on the CDI sequences of three actions, the hypothesis test results are presented in Table V. At a significance level of 0.05, significant differences are only observed in the Performance Indicator. Moreover, post-hoc analysis indicates significant differences between each pair of actions. Fig. 11 displays the distribution of Performance Indicators for the three operation actions, with their mean values at 0.40, 0.08, and 0.42, respectively. Evidently, Action B stands out as the superior choice. Consequently, it is selected as the designated action for the rule ***nc_down_prediction****.*

VII. Related Work

A. Gray Failure

For cloud services, it is common for a service to remain available while experiencing performance degradation. This phenomenon is known as gray failure [50] or fail-slow [51]. The research community has recognized the significance of such issues and has initiated efforts to mitigate them. In the realm of distributed storage systems, the IASO method [52] detects slowdown nodes via inter-node timeout signals. Concurrently, Perseus [20] employs non-intrusive metrics like IO latency and throughput for problem identification. Furthermore, in the context of large-scale cloud computing clusters, SORN [53] leverages the distribution of task execution times, integrating deep learning with classical transportation optimization techniques, to pinpoint cluster-level slowdowns.

Inspired by these works, we introduce the Performance Indicator and Control-Plane Indicator within CDI, which quantitatively evaluates the gray failures of cloud servers. These indicators offer valuable guidance for the operational and maintenance activities associated with gray failures.

B. Stability Evaluation

1) Algorithm Stability: For an algorithm, especially a machine learning algorithm, achieving stable output in response to input perturbations is a key objective. To achieve this, an array of techniques such as data cleaning and repairing [54] [55] [56], ensemble learning [57], and cross-validation [58] are utilized to enhance stability. When it comes to evaluating stability, certain studies offer theoretical assurances [59]. Meanwhile, repeated trials and datasets from a multitude of domains [60] are leveraged in experiments. Additionally, for randomized algorithms, the influence of random seeds [61] must also be taken into account in the stability evaluation.

2) System Stability: It is the ability to consistently deliver services as expected. Given the complexity of entire systems, stability evaluation often requires sophisticated, high-level approaches, which can be categorized into two types: The first is the non-invasive methods based on monitoring data, such as mean time between failures (MTBF), mean time to repair (MTTR), and availability [62]. The second is invasive methods that involve providing the system with carefully crafted inputs and comparing the outputs with expected results. Benchmark methods [25] are a prime example.

3) Cloud Server Stability: As a complex production system, cloud server stability evaluation can also be categorized into two types. For non-intrusive methods, in addition to traditional metrics like MTBF and MTTR, Levy et al. [4] introduced the Annual Interruption Rate, which focuses on the frequency of unavailability incidents. Zhou et al. [63] proposed a method based on classified statistics and hierarchical variable weights, incorporating non-technical aspects like service request timeliness, but still focusing on technical unavailability scenarios.

In the realm of intrusive methods, Xiong et al. [25] developed SuperBench, a benchmark suite for evaluating AI infrastructure which includes both end-to-end AI workload benchmarks and hardware component micro-benchmarks. The SuperBench, however, incurs a relatively high cost and is therefore only applied to machines with potential issues as identified by the selector, rather than being used for comprehensive testing.

In contrast, our proposed CDI is more cost-effective and suitable for large-scale cloud server stability evaluation. CDI is designed for general scenarios, addressing not only unavailability issues but also performance and control-plane concerns, providing a more comprehensive assessment.

VIII. Discussion and Exploration

A. Generality

The original intent of CDI is to offer a universal evaluation framework to assist cloud providers in quantifying the stability of their cloud servers. The core concept of CDI is an event-driven stability evaluation mechanism, a design that grants it good generality. For business scenarios, though our existing events are designed for generic use cases, they can be customized for particular scenarios via configuration adjustment. For example, due to the sensitivity to network fluctuations, Redis instances might necessitate a higher warning level.

Additionally, the sub-metric categorization provides insights into cloud servers and can be deployed across different platforms. In implementations, for instance, the Azure Narya platform [4] catalogs over 2000 features, akin to our events, which enables the stability evaluation in a similar way.

Furthermore, this event-based evaluation philosophy can be extended to other cloud products. For instance, storage services (such as OSS [5]) may prioritize elements such as data availability, read/write latency, and configuration management. In contrast, computing services (like MaxCompute [6]) may emphasize concerns related to computing latency, resource allocation, fault tolerance, and control aspects. The key lies in the precise definition and classification of events. Since these definitions and categorizations involves the implementation details of cloud products, the deep involvement of product experts is required.

B. Customer-Perspective Event

As discussed in Section III-A, apart from the unavailability of cloud servers, Alibaba Cloud customers are also sensitive to performance degradation and control-plane problems. ECS instance health diagnosis [2] discloses a subset of system events to customers, assisting them in problem diagnosis. For example, if a customer receives an alert about slow I/O performance from the underlying VM, they can correlate this with their own API timeout errors and take appropriate actions.

Besides, since the existing CDI is designed for internal stability engineers at Alibaba Cloud, it still poses a technical barrier that prevents its direct application to customers. With the event subset from instance health diagnosis, we can also compute a Customer-Perspective Indicator using the same framework as the CDI. We leave this as our future work.

C. Operation Platform Optimization

Although the CDI primarily evaluates stability retrospectively, its components like event weights can be utilized to improve operational efficiency within a real-time operation platform. By applying this approach, the system can prioritize actions based on event severity. For instance, when deciding between migrating two VMs, the system would give precedence to the VM with higher event weights, as its migration would more positively influence overall CDI. Furthermore, for issues of different severities, the platform can automatically choose the most appropriate action. That is, low-severity issues might result in a ticket being filed, while high-severity issues could trigger immediate actions such as VM migration. This aspect is designated for future research and development.

IX. Conclusion

This study explores the stability evaluation of large-scale cloud servers. Existing technology is limited to downtime, which is just a subset of stability. Thus, we introduce the CDI. Based on the interpretable CloudBot events, it extends stability evaluation beyond system unavailability to include performance and control-plane issues, yielding a more comprehensive evaluation and providing significant support for the evolution of stability.

REFERENCES

[1] "Shared instance types," 2024. [Online]. Available: https://www.alibabacloud.com/help/en/ecs/user-guide/shared-instance-families

[2] "Instance health diagnosis," 2024. [Online]. Available: https://help.aliyun.com/zh/ecs/user-guide/examples-of-health-diagnosis-1

[3] E. H. Forman and S. I. Gass, "The analytic hierarchy process—an exposition," *Operations research*, vol. 49, no. 4, pp. 469–486, 2001.

[4] S. Levy, R. Yao, Y. Wu, Y. Dang, P. Huang, Z. Mu, P. Zhao, T. Ramani, N. Govindaraju, X. Li *et al.*, "Predictive and adaptive failure mitigation to avert production cloud {VM} interruptions," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 1155–1170.

[5] "Object storage service (oss)," 2025. [Online]. Available: https://www.alibabacloud.com/en/product/object-storage-service

[6] "Maxcompute," 2024. [Online]. Available: https://www.aliyun.com/product/odps

[7] L. Deng, Y. Wang, H. Wang, X. Ma, X. Du, X. Zheng, and D. Wu, "Time-aware attention-based transformer (TAAT) for cloud computing system failure prediction," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, R. Baeza-Yates and F. Bonchi, Eds. ACM, 2024, pp. 4906–4917. [Online]. Available: https://doi.org/10.1145/3637528.3671547

[8] X. Lu, Y. Wang, Y. Fu, Q. Sun, X. Ma, X. Zheng, and C. Zhuo, "MISP: A multimodal-based intelligent server failure prediction model for cloud computing systems," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, R. Baeza-Yates and F. Bonchi, Eds. ACM, 2024, pp. 5509–5520. [Online]. Available: https://doi.org/10.1145/3637528.3671568

[9] Y. Wang, B. Luo, and Y. Shen, "Efficient memory overcommitment for {I/O} passthrough enabled {VMs} via fine-grained page meta-data management," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 769–783.

[10] "[resolved] service outage in zone c of the china (hong kong) region," 2025. [Online]. Available: https://www.aliyun.com/noticelist/articleid/1061819219.html

[11] "Gartner forecasts worldwide public cloud end-user spending to surpass 675 billion dollars in 2024." [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2024-05-20-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-surpass-675-billion-in-2024

[12] Y. Dang, Q. Lin, and P. Huang, "Aiops: real-world challenges and research innovations," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 4–5.

[13] L. Rijal, R. Colomo-Palacios, and M. Sánchez-Gordón, "Aiops: A multivocal literature review," *Artificial Intelligence for Cloud and Edge Computing*, pp. 31–50, 2022.

[14] P. Huang, C. Guo, J. R. Lorch, L. Zhou, and Y. Dang, "Capturing and enhancing in situ system observability for failure detection," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 1–16.

[15] Y. Li, Z. M. Jiang, H. Li, A. E. Hassan, C. He, R. Huang, Z. Zeng, M. Wang, and P. Chen, "Predicting node failures in an ultra-large-scale cloud computing platform: an aiops solution," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 2, pp. 1–24, 2020.

[16] Q. Lin, K. Hsieh, Y. Dang, H. Zhang, K. Sui, Y. Xu, J.-G. Lou, C. Li, Y. Wu, R. Yao *et al.*, "Predicting node failure in cloud service systems," in *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2018, pp. 480–490.

[17] S. Lu, B. Luo, T. Patel, Y. Yao, D. Tiwari, and W. Shi, "Making disk failure predictions {SMARTer}!" in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, 2020, pp. 151–167.

[18] C. Luo, P. Zhao, B. Qiao, Y. Wu, H. Zhang, W. Wu, W. Lu, Y. Dang, S. Rajmohan, Q. Lin *et al.*, "Ntam: Neighborhood-temporal attention model for disk failure prediction in cloud platforms," in *Proceedings of the Web Conference 2021*, 2021, pp. 1181–1191.

[19] Y. Xu, K. Sui, R. Yao, H. Zhang, Q. Lin, Y. Dang, P. Li, K. Jiang, W. Zhang, J.-G. Lou *et al.*, "Improving service availability of cloud systems by predicting disk error," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 481–494.

[20] R. Lu, E. Xu, Y. Zhang, F. Zhu, Z. Zhu, M. Wang, Z. Zhu, G. Xue, J. Shu, M. Li *et al.*, "Perseus: A {Fail-Slow} detection framework for cloud storage systems," in *21st USENIX Conference on File and Storage Technologies (FAST 23)*, 2023, pp. 49–64.

[21] C. Lou, C. Chen, P. Huang, Y. Dang, S. Qin, X. Yang, X. Li, Q. Lin, and M. Chintalapati, "{RESIN}: A holistic service for dealing with memory leaks in production cloud infrastructure," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 109–125.

[22] P. Sekwatlakwatla, M. Mphahlele, and T. Zuva, "Traffic flow prediction in cloud computing," in *2016 International Conference on Advances in Computing and Communication Engineering (ICACCE)*. IEEE, 2016, pp. 123–128.

[23] B. L. Dalmazo, J. P. Vilela, and M. Curado, "Performance analysis of network traffic predictors in the cloud," *Journal of Network and Systems Management*, vol. 25, pp. 290–320, 2017.

[24] Z. Guo, S. McDirmid, M. Yang, L. Zhuang, P. Zhang, Y. Luo, T. Bergan, M. Musuvathi, Z. Zhang, and L. Zhou, "Failure recovery: When the cure is worse than the disease," in *14th Workshop on Hot Topics in Operating Systems (HotOS XIV)*, 2013.

[25] Y. Xiong, Y. Jiang, Z. Yang, L. Qu, G. Zhao, S. Liu, D. Zhong, B. Pinzur, J. Zhang, Y. Wang, J. Jose, H. Pourreza, J. Baxter, K. Datta, P. Ram, L. Melton, J. Chau, P. Cheng, Y. Xiong, and L. Zhou, "Superbench: Improving cloud ai infrastructure reliability with proactive validation," in *USENIX ATC*. USENIX Association, July 2024, pp. 835–850. [Online]. Available: https://www.microsoft.com/en-us/research/publication/superbench/

[26] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, "Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, 2020.

[27] H. Wang, H. Guo, Z. Zhu, Y. Zhang, Y. Zhou, and X. Zheng, "Backtrackstl: Ultra-fast online seasonal-trend decomposition with backtrack technique," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, R. Baeza-Yates and F. Bonchi, Eds. ACM, 2024, pp. 5848–5859. [Online]. Available: https://doi.org/10.1145/3637528.3671510

[28] A. Siffer, P. Fouque, A. Termier, and C. Largouët, "Anomaly detection in streams with extreme value theory," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, 2017, pp. 1067–1075. [Online]. Available: https://doi.org/10.1145/3097983.3098144

[29] C. Borgelt, "An implementation of the fp-growth algorithm," in *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, 2005, pp. 1–5.

[30] R. Kohavi and S. Thomke, "The surprising power of online experiments," *Harvard business review*, vol. 95, no. 5, pp. 74–82, 2017.

[31] "[exception (recovered)] abnormalities in alibaba cloud product console access and api calls." [Online]. Available: https://status.alibabacloud.com/#/eventDetail?eventId=16

[32] "Qemu." [Online]. Available: https://www.qemu.org/

[33] "Simple log service (sls)," 2024. [Online]. Available: https://www.alibabacloud.com/help/en/sls/

[34] "Platform for ai," 2024. [Online]. Available: https://www.alibabacloud.com/help/en/pai

[35] "Mysql," 2024. [Online]. Available: https://www.mysql.com/

[36] "Apache spark," 2024. [Online]. Available: https://spark.apache.org/

[37] "[exception (recovered)] exception has been identified for certain products in availability zone c of singapore region," 2024. [Online]. Available: https://status.alibabacloud.com/#/eventDetail?eventId=18

[38] "[exception (recovered)] abnormalities in network access for certain products in availability zone n of shanghai region," 2024. [Online]. Available: https://status.alibabacloud.com/#/eventDetail?eventId=20

[39] "[exception (recovered)]abnormalities in purchase and modify for certain products in shanghai region," 2025. [Online]. Available: https://status.alibabacloud.com/#/eventDetail?eventId=22

[40] Z. Li, D. Pei, C. Luo, Y. Zhao, Y. Sun, K. Sui, X. Wang, D. Liu, X. Jin, and Q. Wang, "Generic and robust localization of multi-dimensional

root causes," in *30th IEEE International Symposium on Software Reliability Engineering, ISSRE 2019, Berlin, Germany, October 28-31, 2019*, K. Wolter, I. Schieferdecker, B. Gallina, M. Cukier, R. Natella, N. R. Ivaki, and N. Laranjeiro, Eds. IEEE, 2019, pp. 47–57. [Online]. Available: https://doi.org/10.1109/ISSRE.2019.00015

[41] C. M. Urzúa, "On the correct use of omnibus tests for normality," *Economics Letters*, vol. 53, no. 3, pp. 247–251, 1996.

[42] D. G. Pereira, A. Afonso, and F. M. Medeiros, "Overview of friedman's test and post-hoc analysis," *Communications in Statistics-Simulation and Computation*, vol. 44, no. 10, pp. 2636–2653, 2015.

[43] H. M. Park, "Comparing group means: t-tests and one-way anova using stata, sas, r, and spss," 2009.

[44] H. Abdi and L. J. Williams, "Tukey's honestly significant difference (hsd) test," *Encyclopedia of research design*, vol. 3, no. 1, pp. 1–5, 2010.

[45] W. C. Driscoll, "Robustness of the anova and tukey-kramer statistical tests," *Computers & Industrial Engineering*, vol. 31, no. 1-2, pp. 265–268, 1996.

[46] M. Delacre, C. Leys, Y. L. Mora, and D. Lakens, "Taking parametric assumptions seriously: Arguments for the use of welch's f-test instead of the classical f-test in one-way anova," *International Review of Social Psychology*, vol. 32, no. 1, p. 13, 2019.

[47] S. A. Rusticus and C. Y. Lovato, "Impact of sample size and variability on the power and type i error rates of equivalence tests: A simulation study," *Practical Assessment, Research, and Evaluation*, vol. 19, no. 1, p. 11, 2019.

[48] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[49] O. J. Dunn, "Multiple comparisons using rank sums," *Technometrics*, vol. 6, no. 3, pp. 241–252, 1964.

[50] P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, and R. Yao, "Gray failure: The achilles' heel of cloud-scale systems," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems, HotOS 2017, Whistler, BC, Canada, May 8-10, 2017*, A. Fedorova, A. Warfield, I. Beschastnikh, and R. Agarwal, Eds. ACM, 2017, pp. 150–155. [Online]. Available: https://doi.org/10.1145/3102980.3103005

[51] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey *et al.*, "Fail-slow at scale: Evidence of hardware performance faults in large production systems," *ACM Transactions on Storage (TOS)*, vol. 14, no. 3, pp. 1–26, 2018.

[52] B. Panda, D. Srinivasan, H. Ke, K. Gupta, V. Khot, and H. S. Gunawi, "{IASO}: A {Fail-Slow} detection and mitigation framework for distributed storage services," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 47–62.

[53] F. Chen, Y. Zhang, L. Fan, Y. Liang, G. Pang, Q. Wen, and S. Deng, "Cluster-wide task slowdown detection in cloud system," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, R. Baeza-Yates and F. Bonchi, Eds. ACM, 2024, pp. 266–277. [Online]. Available: https://doi.org/10.1145/3637528.3671936

[54] Z. Liu, Y. Zhang, R. Huang, Z. Chen, S. Song, and J. Wang, "Experience: Algorithms and case study for explaining repairs with uniform profiles over iot data," *Journal of Data and Information Quality (JDIQ)*, vol. 13, no. 3, pp. 1–17, 2021.

[55] H. Wang, A. Zhang, S. Song, and J. Wang, "Streaming data cleaning based on speed change," *VLDB J.*, vol. 33, no. 1, pp. 1–24, 2024. [Online]. Available: https://doi.org/10.1007/s00778-023-00796-y

[56] Y. Sun and S. Song, "From minimum change to maximum density: On s-repair under integrity constraints," in *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 2021, pp. 1943–1948. [Online]. Available: https://doi.org/10.1109/ICDE51399.2021.00181

[57] Z. Liu, W. Cao, Z. Gao, J. Bian, H. Chen, Y. Chang, and T. Liu, "Self-paced ensemble for highly imbalanced massive data classification," in *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 2020, pp. 841–852. [Online]. Available: https://doi.org/10.1109/ICDE48307.2020.00078

[58] M. Stone, "Cross-validation: A review," *Statistics: A Journal of Theoretical and Applied Statistics*, vol. 9, no. 1, pp. 127–139, 1978.

[59] Z. Chen, S. Song, Z. Wei, J. Fang, and J. Long, "Approximating median absolute deviation with bounded error," *Proc. VLDB Endow.*, vol. 14, no. 11, pp. 2114–2126, 2021. [Online]. Available: http://www.vldb.org/pvldb/vol14/p2114-song.pdf

[60] H. Wang and S. Song, "Frequency domain data encoding in apache iotdb," *Proc. VLDB Endow.*, vol. 16, no. 2, pp. 282–290, 2022. [Online]. Available: https://www.vldb.org/pvldb/vol16/p282-song.pdf

[61] D. Picard, "Torch.manual_seed(3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision," *CoRR*, vol. abs/2109.08203, 2021. [Online]. Available: https://arxiv.org/abs/2109.08203

[62] P. A. Kullstam, "Availability, mtbf and mttr for repairable m out of n system," *IEEE Transactions on Reliability*, vol. 30, no. 4, pp. 393–394, 1981.

[63] P. Zhou, L. Meng, X. Qiu, Z. Wang, Z. Wang, and Z. Chen, "Evaluation of cloud service reliability based on classified statistics and hierarchy variable weight," *J. Signal Process. Syst.*, vol. 91, no. 10, pp. 1115–1126, 2019. [Online]. Available: https://doi.org/10.1007/s11265-018-1407-2