2023 IEEE International Conference on Data Mining Workshops (ICDMW) | 979-8-3503-8164-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICDMW60847.2023.00158

A Roadmap towards Intelligent Operations for Reliable Cloud Computing Systems

Yintong Huo, Cheryl Lee, Jinyang Liu, Tianyi Yang, and Michael R. Lyu
Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China.
Email: {ythuo, jyliu, tyyang, lyu}@cse.cuhk.edu.hk, cheryllee@link.cuhk.edu.hk

***Abstract*—The increasing complexity and usage of cloud systems have made it challenging for service providers to ensure reliability. This paper highlights two main challenges, namely internal and external factors, that affect the reliability of cloud microservices. Afterward, we discuss the data-driven approach that can resolve these challenges from four key aspects: ticket management, log management, multimodal analysis, and the microservice resilience testing approach. The experiments conducted show that the proposed data-driven AIOps solution significantly enhances system reliability from multiple angles.**

I. Introduction

IT enterprises have significantly increased the development of cloud applications and services like search engines, messaging apps, and online shopping. The growing complexity and volume of cloud systems make critical failures inevitable, potentially causing service interruptions and performance degradation. For example, on October 4th, 2021, a Facebook outage disconnected Facebook data centers from the Internet globally for nearly six hours [1]. This outage significantly impacted Facebook's market revenue and user experience. The increasing complexity and distributed nature of these services necessities intelligent software reliability engineering. In this paper, we have identified critical reliability challenges in industrial cloud systems and developed a general roadmap for improving cloud reliability using data-driven AIOps.

Challenges to the reliability of cloud microservices originate from both internal and external factors of microservices. *Internal factors* refer to issues within the microservices themselves, such as software bugs and resilience problems. Software bugs are errors or flaws in the design, development, or operation of the microservice that can result in incorrect behavior. On the one hand, a software bug is an error, flaw or fault in the software's design, development, or operation. Bugs lead to erroneous behaviors of the microservice. Service resilience [22], on the other hand, refers to the ability to maintain acceptable performance levels and recover from service failures. Resilience issues can affect the availability of the microservice, which can harm cloud providers' revenue. To ensure service reliability, test engineers conduct resilience tests on microservices, intentionally injecting failures [9] to discover flaws.

External factors refer to the threats from outside the microservice, such as cascading failures and low-quality logs and alerts. Cascading failures that lead to service degradation are prevalent in cloud services. Although cloud management frameworks provide automatic mechanisms for failure recovery, unplanned service failures may still cause severe cascading effects. Therefore, it is crucial to evaluate the impact of service failures rapidly and accurately for efficient operation and maintenance of cloud services. Besides, low-quality logs and alerts are often caused by system-level misconfigurations. When failures occur, On-Call Engineers (OCEs) typically inspect logs and alerts to locate and diagnose failures. If the logs and alerts are of low-quality or misleading, the manual diagnosis process will be impeded.

To tackle the challenges above, we develop intelligent operations to improve the reliability of microservice systems. The roadmap includes (1) proactive measures for internal factors; and (2) reactive measures for external factors. Proactive measures examine the microservice system to detect possible flaws in the system before the occurrence of a failure, including adaptive resilience testing and architectural resilience optimization. Reactive measures assist On-Call Engineers (OCEs) in reducing the impact of a failure during failure mitigation, including log and metric analysis for anomaly detection, postmortem incident ticket analysis, and multimodal root cause localization.

II. Data-driven AIOps for Cloud Computing

Ensuring system reliability is a significant challenge for cloud providers. To achieve the goal, cloud providers gather extensive monitoring data that reflects run-time microservice systems' behavior, which includes logs, traces, and tickets. We describe their details as follows.

- *Traces* record the status of each microservice invocation, such as the return value and the duration of execution.
- *Alerts* are notifications sent to OCEs when the cloud service exhibits abnormal behavior, as defined by the alert strategy.
- *Tickets* are the problem descriptions sent to service providers when customers encounter a technical issue with a product.
- *Logs* are semi-structured text printed by logging statements (e.g., `logger.info()`) in the source code.
- *Metrics* Metrics are fixed-interval time series reflecting the statuses of the cloud system [6].

The cloud monitoring system collects and processes the above monitoring data, and when an abnormal state is de-

[1] https://www.facebook.com/business/news/update-about-the-october-4th-outage

2375-9259/23/$31.00 ©2023 IEEE
DOI 10.1109/ICDMW60847.2023.00158

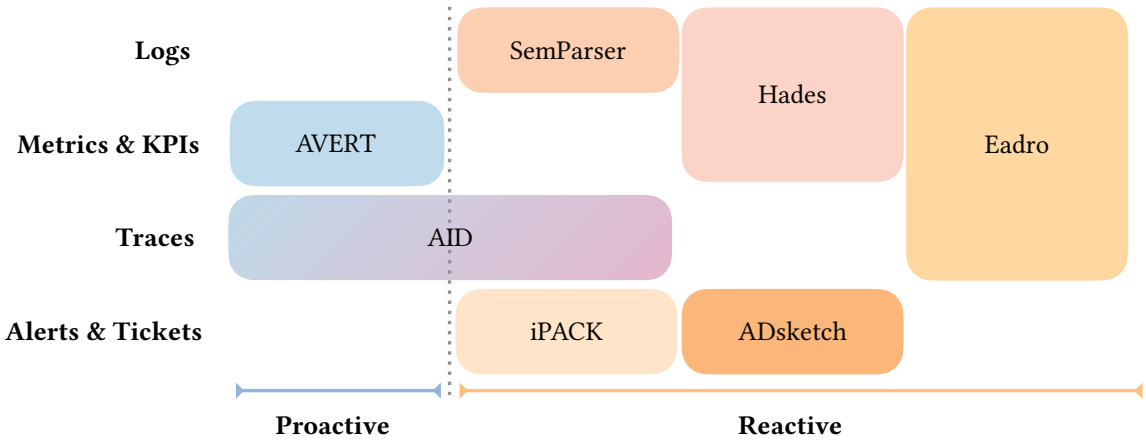

Fig. 1: The Roadmap of Intelligent Operations for Reliable Cloud Systems

tected, the alerting module in the monitoring system will send an alert to OCEs.

Although the monitoring data provide rich information for system status and help engineers intervene in potential faults, they are generated in an overwhelming volume for developers to inspect manually. For example, a high-performance computing (HPC) system generates hundreds of gigabytes of log data in just one week [23]. To utilize the large volume of data, modern AIOps solutions apply a data-driven approach to identify system behavior patterns and performance trends that may not be apparent. Once the normal patterns have been learned from past data, the model can inform developers by effectively identifying the abnormal state of a system in the production environment.

III. Alerts and Tickets

In order to ensure the reliability of cloud systems, cloud vendors rely on comprehensive monitoring mechanisms, which can be divided into two aspects. Firstly, various components within the cloud systems, such as hardware and microservices, are equipped with monitors that raise *alerts* to draw the attention of on-call engineers and enable timely mitigation actions [29]. Our first part on alert research focuses on achieving accurate monitoring of these components to ensure the reliability of cloud systems. Secondly, the customers' experience outside the cloud systems is also closely monitored through a support system, where customers can report encountered problems by issuing *tickets*. Due to the cloud systems' scale, many tickets may be received, including duplicate ones. The second part on ticket research proposes aggregating these duplicate tickets to alleviate the burden on support engineers who handle a high volume of tickets.

A. Adaptive and Interpretable Monitoring for Cloud Systems

Service interruptions, also known as *incidents*, are an inevitable aspect of large-scale cloud platforms [17]. To maintain the reliability of cloud systems, contemporary cloud vendors widely employ monitors to continuously detect anomalies, or unexpected behaviors, of cloud systems. Once an anomaly is detected, the monitor generates *alerts* that provide a description of the anomaly, which promptly notifies on-call engineers to investigate the matter. An established practice is to detect anomalies on key performance indicators (KPIs) to generate alerts. These KPIs capture the runtime states of a system, including various metrics such as CPU usage and service response delay [5].

Although many efforts have been dedicated to detecting anomalies on KPIs [31], most of the existing work lacks interpretability. Specifically, these methods calculate a probability indicating the likelihood of performance anomalies at each timestamp. They then choose a threshold to convert the probability into a binary label, normal or anomaly. However, in practice, a mere recommendation of suspicious anomalies may not be very useful to engineers. This is because they have to manually investigate the problematic metrics (suggested by the model) to locate faults. The issue is compounded by the prevalence of false alerts. Furthermore, many state-of-the-art methods train models with historical metric data in an offline setting. As online services undergo feature upgrades and system renewal continuously, the patterns of metrics may evolve accordingly, resulting in concept drift [7], where Without adaptability, these models cannot accommodate the ever-changing services and user behaviors.

To tackle this issue, we propose ADSketch, an interpretable and adaptive KPI anomaly detection approach based on *pattern sketching*. The core concept is to identify discriminative subsequences from metric time series that can represent classes of different issues. This approach is similar to the problem of shapelet discovery in time series data [30]. Specifically, for multiple subsequences that describe the same type of issue, we compute their average and regard the result as a *metric pattern* for the issue. In this way, ADSketch provides a novel mechanism to characterize service performance issues using metric time series. Our experimental results demonstrate that our design outperforms existing state-of-the-art time series

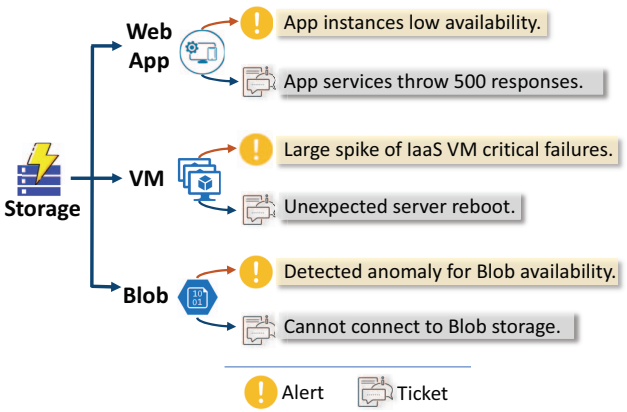

Fig. 2: Alerts and resultant tickets caused by an incident.

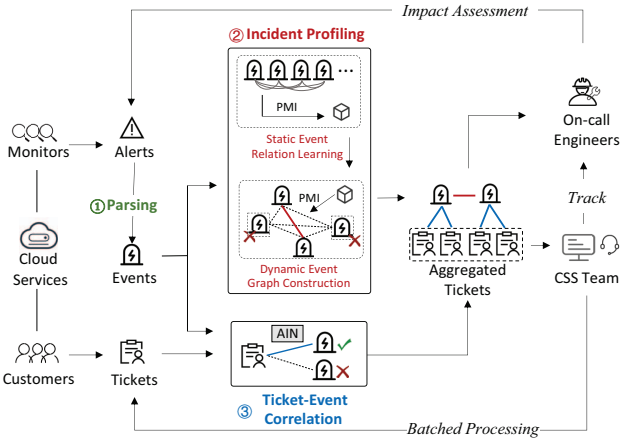

Fig. 3: Overview of iPACK.

anomaly detectors on both public and industrial data. Specifically, we have achieved an average F1 score of over 0.8 in production systems.

B. Incident-aware Duplicate Ticket Aggregation

When customers face technical difficulties with a platform, they usually seek help from cloud providers by submitting a support *ticket*. However, in the case of a large-scale cloud platform with millions of users, an incident could result in a substantial number of tickets, many of which may be duplicates. To alleviate the burden of support engineers, it is crucial to group together duplicate tickets that stem from the same incident [18]. By doing so, the support team can handle the tickets efficiently.

Most existing studies on duplicate issue report detection measure the semantic similarity between two reports based on their textual descriptions, using natural language processing techniques such as word frequency [24], and topic modeling [3]. However, they are suboptimal for aggregating duplicate tickets in cloud systems due to their large-scale and heterogeneous architecture [28]. The primary reason is that customers of cloud systems could encounter different issues with distinct symptoms caused by the same incident. Figure 2 shows an example. When an infrastructure-level service (e.g., a storage service) is interrupted, other services depending on it (e.g., VM and Web application) can also be impacted. As a result, customers using different services may observe different symptoms and submit tickets with dissimilar descriptions. Consequently, solely relying on textual descriptions of tickets is insufficient to tackle this problem.

To address the limitations of existing studies, we propose incorporating cloud-side runtime information, i.e., *alerts*, to facilitate ticket aggregation in cloud systems. As shown in Figure 3, we formulate the ticket aggregation problem in cloud systems as a two-stage linking problem, i.e., alert-alert linking and ticket-alert linking. If multiple interlinked alerts are triggered by the same incident and are further linked to different tickets, we consider these tickets should be aggregated (i.e., caused by the same incident). Thus, it is possible to aggregate semantically different tickets via alert-alert links. Specifically, iPACK consists of three main steps, i.e., *alert parsing*, *incident profiling*, and *ticket-event correlation*. In the *alert parsing* step, we parse alerts as more coarse-grained *events* to reduce redundant alerts. Next, in the *incident profiling step*, we propose a graph-based incident profiling (GIP) method to remove the regular events (i.e., parsed regular alerts) and link correlated indicative events. Then, in the *ticket-event correlation*, we propose an attentive interaction network (AIN) to correlate a ticket to an event. Finally, if two tickets are correlated to the events within the same event graph (i.e., the same incident), we aggregate the tickets as the same cluster. The results of the ticket aggregation are presented to the CSS (Customer Support Services) team to streamline the ticket processing process and improve efficiency. This allows support engineers to send out batch notifications to potentially affected customers and provide quick guidance for service recovery. Additionally, the results can aid on-call engineers in conducting impact assessments, including identifying affected services and determining the extent of customer impact caused by the incident (e.g., number of affected customers). The experimental results on Microsoft Azure show that iPACK can accurately and comprehensively aggregate duplicate tickets, achieving an F1 score of 0.871∼0.935 and outperforming state-of-the-art methods by 12.4%∼31.2%.

IV. Logs

The logging statements, which developers put into the source code, carry run-time information about software systems [11]. By reading these logs, software system operators and administrators can monitor software status [4] or detect anomalies [26]. The overwhelming logs, however, impede developers from reading every line of log files as modern software systems get more complicated than before [10]. Therefore, intelligent software engineering necessitates automated log analysis.

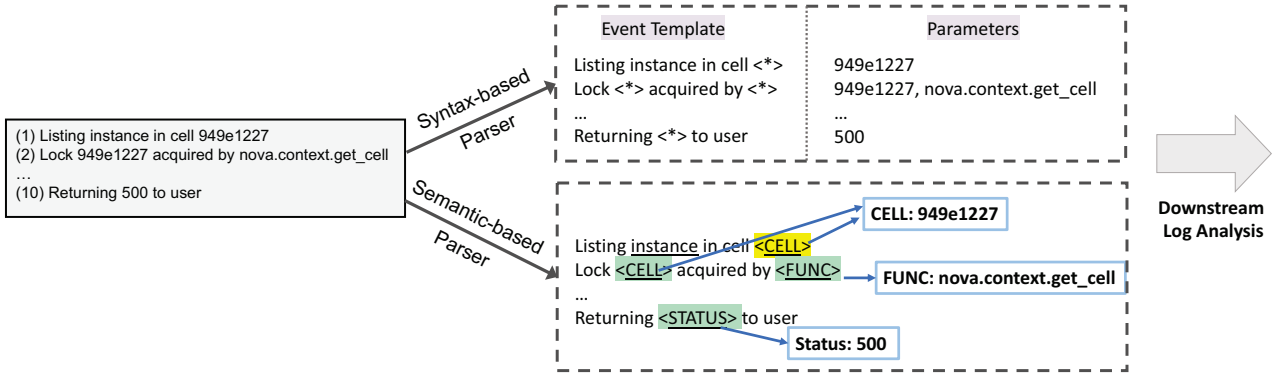

Fig. 4: Difference between syntax-based parsers and semantic-based SemParser.

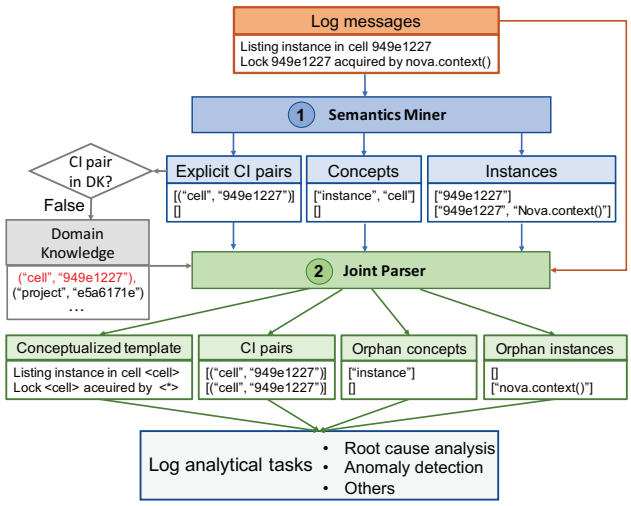

Fig. 5: The pipeline of SemParser.

A. Semantic-based Log Parser

Basically, a log message is a type of semi-structured language comprising a natural language written by software developers and some auto-generated variables during software execution [16]. As most log analysis tools accept a structured input, the fundamental step for automated log analysis is log parsing. Given a raw message, a log parser recognizes a set of fields (e.g., verbosity levels, date, time) and message content, while the latter is represented as structured event templates (i.e., constants) with corresponding parameters (i.e., variables) [12], [13]. For example, for the log message "Listing instance in cell 949e1227", "Listing instance in cell <*>" is the template describing the system event, and "949e1227" corresponds to the parameter indicator "<*>" in the template.

Although automatic log parsing is full of challenges, researchers have made progress leveraging statistical and history-based methods. For instance, SLCT [25] and LFA [20] constructed log templates by counting the number of historical frequently-appearing words. The most widely-used parser in industry, Drain [8], formed log templates by traversing leaf nodes in a tree. However, we argue that all current parsers are *syntax-based* with superficial features (e.g., word length, log length, frequency), and they have limited high-level semantic acquisition from three aspects: (1) individual informative tokens; (2) semantics within a message; and (3) semantics between messages.

To tackle the aforementioned complicated but critical limitations, we propose a novel *semantic-based* log parser, SemParser, the first work to target parsing logs with respect to their semantic meaning as shown in Figure 4. The pipeline of SemParser is exhibited in 5. We first define two-level granularities of semantics in logs, *message-level* and *instance-level semantics*. Message-level semantics refers to identifying technical concepts (e.g., cell) within log messages, while instance-level semantics means resolving what the instance (i.e., parameters) describes. Our framework comprises two parts, an end-to-end semantics miner and a joint parser. To begin with, log messages are sent to the semantic miner for acquiring template-level semantics (i.e., *concepts*) and explicit instance-level semantics (i.e., *explicit CI pairs*) of each log independently. This step mainly solves the first two stated challenges. The unseen explicit CI pairs will be added to the *Domain Knowledge database* to keep the knowledge updated. Moreover, to uncover potential implicit semantics from domain knowledge, *instances* in log messages are kept. Hence, the challenge of missing inter-log relations is addressed. Following that, the joint parser receives outputs from the semantics miner, taking charge of implicit semantics inference with the help of domain knowledge. The newfound implicit instance semantics, coupled with the explicit one, form the instance-level semantics, denoted as *CI pairs*. The remaining concepts and instances that cannot be paired are stored as *orphan concepts* and *orphan instances*, respectively. Besides, the *conceptualized templates* are derived by replacing instances with their related concepts (if available), or "<*>" for else. The final structural outcome of SemParser consists of *conceptualized templates*, *CI pairs*, *orphan concepts*, as well as *orphan instances*. The experimental results demonstrate the effectiveness of our model, which could extract both

high-quality and comprehensive semantics from log messages. SemParser achieves an average F1 score of 0.985 for six systems logs even though it was only fine-tuned the base model on 50 annotated samples with a large portion of templates unseen in the test set.

B. Log-based Anomaly Detection and Failure Identification

After acquiring the semantics from SemParser, we investigate whether these can benefit log downstream applications, i.e., log-based anomaly detection and failure identification. In the anomaly detection task, the detector predicts whether anomalies exist within a short period of log messages (i.e., session). Motivated by previous studies, we decouple the anomaly detection framework into two components, a *log parser* to generate templates, and a *detection model* to analyze template sequences in a session. A dependable parser should perform well as a foundational processor for log analysis, regardless of the down-streaming detection model used. Our experiments compare the performance of different baseline parsers under various anomaly detection techniques. Equipping with the semantic outputs of SemParser, we observe that SemParser outperforms all syntax-based parsers by an average F1 score of 1.22% and 11.71% over state-of-the-art detection models in the HDFS and OpenStack system logs, respectively. While anomaly detection identifies present faults from logs, failure identification looks deeper into the problems and identifies what type of failure occurs. In the more challenging failure identification task, SemParser achieves an average precision score of 0.95, exceeding all baselines of 8.52%.

V. Multimodal Data

As introduced in I, software operators must closely monitor the system status via multi-source run-time information to discover and tackle potential failures in their earliest efforts. Yet, the explosion of monitoring data makes automated troubleshooting techniques imperative. Many efforts have been devoted to troubleshooting automation. Generally, they focus either on anomaly detection (AD) [19] or on root cause localization (RCL) [21].

AD tells whether an anomaly exists, and RCL identifies the culprit microservice upon the existence of an anomaly. However, unlike operation teams that closely monitor diverse sources of run-time information, existing efforts mainly focus on a single information source, which is insufficient to precisely depict the system status. We argue that leveraging multimodal monitoring data can contribute to more effective troubleshooting approaches. Hence, we propose two works to study using multimodal data to deal with AD and RCL.

A. Anomaly Detection for General Distributed Software Systems

We first intensively study system anomalies resulting from typical faults in Apache Spark. We find that logs and metrics complement each other and also collaborate in revealing system health. While both logs and metrics respond to anomalies, neither alone is sufficiently informative [15]. This results in Hades, a heterogeneous anomaly detector via semi-supervised learning for large-scale software systems equipped with a novel cross-modal attention mechanism, as shown in Figure 6. Hades involves four components: 1) We model lexical semantics and sequential dependencies of logs by adopting FastText and Transformer. 2) For metrics, we employ a hierarchical encoder to jointly learn aspect-oriented temporal dependencies, cross-metric relationships, and inter-aspect correlations. 3) We design novel cross-modal attention to learn meaningful intra- and inter-modal properties. 4) Finally, the framework infers the system status and triggers an alarm upon detecting anomalies. We also present a two-phase semi-supervised training strategy to reduce labor-intensive annotation: 1) train the model with a small amount of labeled data and apply pseudo-labeling on the unlabeled data; 2) update the model using both labeled and high-confidence pseudo-labeled data until convergence.

Hades is evaluated on one simulated dataset from Spark and two datasets from the cloud services of Huawei Cloud. The experimental results demonstrate the superiority of Hades, which achieves an average F1-score of 0.933 and outperforms all state-of-the-art competitors by 9.12%$\sim$174.41%.

B. Root Cause Localization for Microservices

RCL aims to identify which microservice is initially experiencing a functional anomaly. An anomaly in one microservice could propagate to others and magnify its impact, so the monitoring data exhibit complex patterns and relationships, making RCL extremely difficult [14]. We identify that existing data-driven localizers suffer two main limitations: 1) Existing research deeply relies on traces only, which is demonstrated to be insufficient. Other sources, such as logs and metrics, are underutilized, though they provide valuable clues into presenting abnormal patterns. 2) In the context of microservice troubleshooting, RCL follows AD since we must discover an anomaly before analyzing it. However, current research treats them as independent with little consideration for their shared inputs and knowledge of the microservice status.

To overcome the limitations, we propose Eadro, the first end-to-end framework integrating AD and RCL to troubleshoot microservices based on multi-source monitoring data, as shown in Figure 7. Specifically, Eadro consists of three components: 1) Modal-wise learning: It contains three modality-specific modules for learning intra-service behaviors from logs, metrics, and traces. We apply Hawkes process and dilated causal convolution to model the log event occurrences, temporal dependencies and inter-series associations of metrics, and meaningful fluctuations of latency in traces. 2) Dependency-aware status learning: This fuses the multi-modal representations via gated concentration and a graph attention network, where the topological dependency is built on historical invocations. 3) Joint detection and localization: It consists of an anomaly detector and a root cause localizer sharing representations. The detector predicts the existence of anomalies, and the localizer predicts the probability of each microservice being the culprit upon an anomaly alarm.

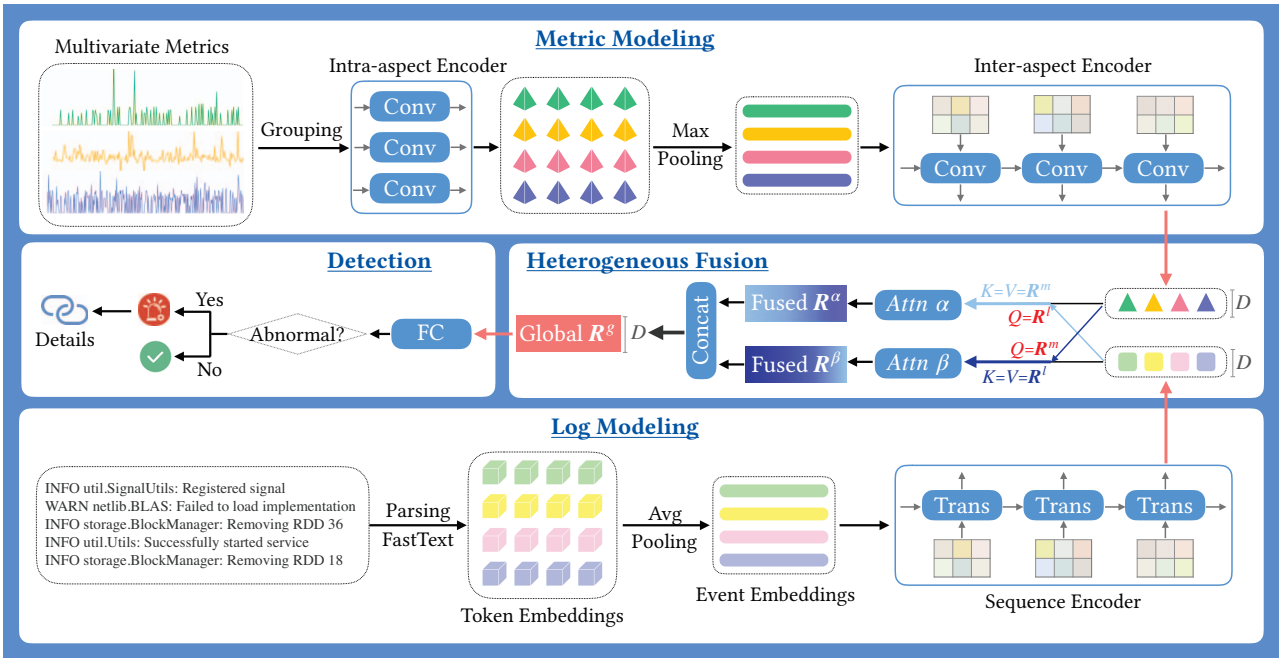

Fig. 6: Overview of Hades.

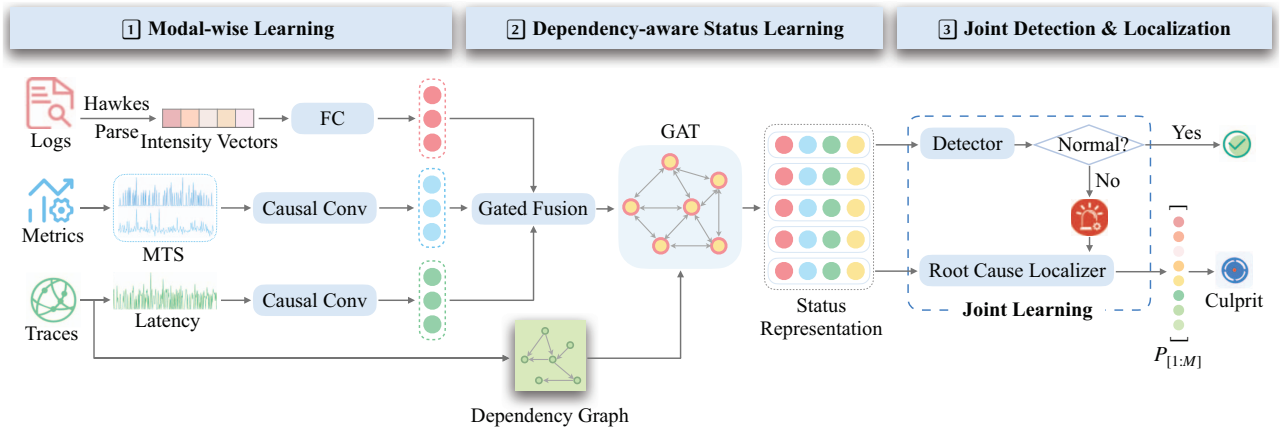

Fig. 7: Overview of Eadro.

Experimental results on two widely-used benchmark microservices demonstrate the effectiveness of Eadro, which surpasses all compared anomaly detectors by 53.82%∼92.68% in F1-score and achieves state-of-the-art RCL results with 290%∼5068% higher in Top-1 Hit Rate than five advanced baselines.

VI. Microservices

Modern online services are moving towards the microservice architecture [1], where a monolithic online service is split into fine-grained, independently-managed microservices which collectively serve user requests. A *microservice* is a small independent program that communicates over well-defined APIs. Multiple microservices serve users' requests as a whole.

The microservice architecture exhibits three prominent attributes [2]: (1) highly decoupled, (2) highly dynamic, and (3) specialized. Their further clarifications are described as follows. First, a microservice system is highly decoupled. Each microservice in a microservices system can be developed, deployed, operated, and scaled without affecting the functioning of other services. The microservices communicate with each other through well-defined APIs. Second, the microservice architecture is highly dynamic. New features and updates are delivered continuously and frequently. Last, microservices are specialized. Different from other existing distributed systems (e.g., Hadoop, Spark, and Blockchain), each microservice is designed for a set of capabilities and focuses on serving a specific problem. If developers contribute

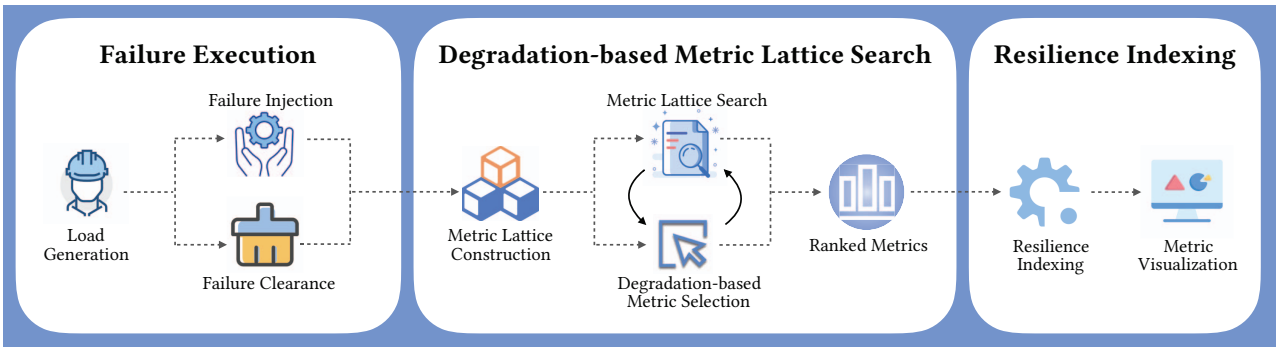

Fig. 8: Overview of AVERT.

more code to a service over time and the service becomes complex, it can be broken into smaller services. As a result, the microservice failures are usually cascaded due to the multi-layer deployment and inter-service dependencies architecture. Such three attributes lead to the challenges specific to the microservice architecture. Our research work focuses on two closely-related tasks towards reliable microservices. First, we propose predicting the intensity of microservice dependencies, by which engineers can identify the potential risk factors that can lead to cascading failures and take proactive measures to prevent them. microservice systems use runtime metrics during testing to identify potential resilience issues. It is another proactive way to ensure the reliability of cloud services. The task details are listed as follows.

A. Predicting the Intensity of Microservice Dependency

Service invocations create dependencies between services. Online service systems have binary dependency tracing frameworks, but using binary-valued dependency for failure diagnosis and recovery is inefficient. The callee microservice impacts the caller microservice in different ways. Hence, the procedure of failure recovery can be sped up by skipping those unimportant services. In microservice systems, examining different dependencies manually without any priority is inefficient, especially when the microservice components are highly decoupled and dynamic. Therefore, measuring the dependency as a continuous value indicating the dependency's intensity could be useful. Specifically, by checking microservices dependent on the failed microservice with large intensity values, OCEs can find the root cause of a failure with a higher probability [28]. By recovering the services strongly dependent on the failed one, the whole system could be restored faster. To this end, we propose AID, the first method to quantify the intensity of dependencies between different services. The evaluation results on both the simulated and industrial environments show the proposed method's effectiveness and efficiency. Additionally, our method has been successfully applied in a leading public cloud provider, and helped greatly reduce manual maintenance effort.

B. Resilience Testing of Microservice Systems

The resilience of a microservice system refers to the ability to maintain the performance of services at an acceptable level and recover the service back to normal when a failure in one or more parts of the system causes service degradation. Resilience testing is one of the primary ways to measure the resilience of software. By purposefully introducing failures into the system, the test engineers can monitor how the microservice system performs and improve the architectural design according to the discovered flaws. Automation of the resilience testing procedure is possible, but manual standardization of test parameters is still required, which is burdensome and unscalable. This is due to microservice systems' decoupled and specialized nature. For the resilience testing of microservice systems, [27] identifies the scalability and adaptivity issues of current industrial practice for resilience testing. Then we conduct the first empirical study on the failures' manifestations on resilient and unresilient microservice systems. The empirical study demonstrates the feasibility of self-adaptive resilience testing. We propose AVERT, shown in Figure 8, the first self-adaptive resilience testing framework that can automatically index the resilience of a microservice system to different failures. AVERT measures the degradation propagation from system performance metrics to business metrics. The higher the propagation, the lower the resilience. The evaluation on two open-source and one industrial benchmark microservice systems indicates that AVERT can effectively and efficiently produce accurate test results.

VII. Conclusion

This paper presents a roadmap toward intelligent operations for reliable cloud computing systems. To do so, we identify two challenges to cloud microservice reliability: internal and external factors. To mitigate the two challenges, the roadmap illustrates four approaches to ensure software reliability: tickets management, logs management, multimodal data analysis, and microservice resilience testing approach.

VIII. Acknowledgement

The work described in this paper was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14206921 of the General Research Fund).

REFERENCES

[1] Amazon: What are microservices? (2022), https://aws.amazon.com/microservices/

[2] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley (Feb 2009)

[3] Budhiraja, A., Reddy, R., Shrivastava, M.: Lwe: Lda refined word embeddings for duplicate bug report detection. In: Proceedings of the 40th ICSE-C. pp. 165–166 (2018)

[4] Chen, M., Zheng, A.X., Lloyd, J., Jordan, M.I., Brewer, E.: Failure diagnosis using decision trees. In: International Conference on Autonomic Computing, New York, NY, USA, May 17-19, 2004. pp. 36–43. IEEE Computer Society (2004)

[5] Chen, Z., Liu, J., Su, Y., Zhang, H., Ling, X., Yang, Y., Lyu, M.R.: Adaptive performance anomaly detection for online service systems via pattern sketching. In: Proceedings of the 44th ICSE. pp. 61–72 (2022)

[6] Dickson, C.L.: A working theory-of-monitoring. Tech. rep., Google, Inc. (2013), https://www.usenix.org/conference/lisa13/working-theory-monitoring

[7] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM computing surveys (CSUR) **46**(4), 1–37 (2014)

[8] He, P., Zhu, J., Zheng, Z., Lyu, M.R.: Drain: An online log parsing approach with fixed depth tree. In: 2017 IEEE International Conference on Web Services, Honolulu, HI, USA, June 25-30, 2017. pp. 33–40. IEEE (2017)

[9] Heorhiadi, V., Rajagopalan, S., Jamjoom, H., Reiter, M.K., Sekar, V.: Gremlin: Systematic resilience testing of microservices. In: 36th IEEE International Conference on Distributed Computing Systems, ICDCS 2016, Nara, Japan, June 27-30, 2016. pp. 57–66. IEEE Computer Society (2016)

[10] Huo, Y., Lee, C., Su, Y., Shan, S., Liu, J., Lyu, M.: Evlog: Evolving log analyzer for anomalous logs identification. Proceedings of IEEE 34th ISSRE (2023)

[11] Huo, Y., Li, Y., Su, Y., He, P., Xie, Z., Lyu, M.R.: Autolog: A log sequence synthesis framework for anomaly detection. Proceedings of IEEE/ACM 38th ASE (2023)

[12] Huo, Y., Su, Y., Lee, C., Lyu, M.R.: Semparser: A semantic parser for log analysis. In: Proceedings of IEEE 45th ICSE. IEEE (2023)

[13] Huo, Y., Su, Y., Lyu, M.: Logvm: Variable semantics miner for log messages. In: 2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). pp. 124–125. IEEE (2022)

[14] Lee, C., Yang, T., Chen, Z., Su, Y., Lyu, M.R.: Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. In: Proceedings of IEEE 45th ICSE. IEEE (2023)

[15] Lee, C., Yang, T., Chen, Z., Su, Y., Yang, Y., Lyu, M.R.: Hades: Heterogeneous anomaly detection for software systems via semi-supervised cross-modal attention. In: Proceedings of IEEE 45th ICSE. IEEE (2023)

[16] Li, Y., Huo, Y., Jiang, Z., Zhong, R., He, P., Su, Y., Lyu, M.R.: Exploring the effectiveness of llms in automated logging generation: An empirical study. arXiv preprint arXiv:2307.05950 (2023)

[17] Liu, H., Lu, S., Musuvathi, M., Nath, S.: What bugs cause production cloud incidents? In: Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS). pp. 155–162 (2019)

[18] Liu, J., He, S., Chen, Z., Li, L., Kang, Y., Zhang, X., He, P., Zhang, H., Lin, Q., Xu, Z., et al.: Incident-aware duplicate ticket aggregation for cloud systems. arXiv preprint arXiv:2302.09520 (2023)

[19] Liu, P., Xu, H., Ouyang, Q., Jiao, R., Chen, Z., Zhang, S., Yang, J., Mo, L., Zeng, J., Xue, W., Pei, D.: Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. In: 31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020, Coimbra, Portugal, October 12-15, 2020. pp. 48–58. IEEE (2020)

[20] Nagappan, M., Vouk, M.A.: Abstracting log lines to log event types for mining software system logs. In: Proceedings of the 7th International Working Conference on Mining Software Repositories, Cape Town, South Africa, May 2-3, 2010. pp. 114–117. IEEE, IEEE Computer Society (2010)

[21] Pan, Y., Ma, M., Jiang, X., Wang, P.: Faster, deeper, easier: Crowdsourcing diagnosis of microservice kernel failure from user space. In: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. p. 646–657. Association for Computing Machinery, New York, NY, USA (2021)

[22] Samir, N., Kyle, B.: Production software application performance and resiliency testing (2020)

[23] Shilpika, Lusch, B., Emani, M., Vishwanath, V., Papka, M.E., Ma, K.: MELA: A visual analytics tool for studying multifidelity HPC system logs. In: 3rd IEEE/ACM Industry/University Joint International Workshop on Data-center Automation, Analytics, and Control, DAAC@SC, Denver, CO, USA, November 22, 2019. pp. 13–18. IEEE (2019)

[24] Sun, C., Lo, D., Wang, X., Jiang, J., Khoo, S.C.: A discriminative model approach for accurate duplicate bug report retrieval. In: Proceedings of the 32nd International Conference on Software Engineering (ICSE). pp. 45–54 (2010)

[25] Vaarandi, R.: A data clustering algorithm for mining patterns from event logs. In: Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IEEE Cat. No. 03EX764), Kansas City, MO, USA, Oct 3, 2003. pp. 119–126. IEEE (2003)

[26] Xu, W., Huang, L., Fox, A., Patterson, D.A., Jordan, M.: Large-scale system problems detection by mining console logs. Tech. rep., EECS Department, University of California, Berkeley (Jul 2009)

[27] Yang, T., Lee, C., Shen, J., Su, Y., Yang, Y., Lyu, M.R.: An adaptive resilience testing framework for microservice systems. CoRR **abs/2212.12850** (2022)

[28] Yang, T., Shen, J., Su, Y., Ling, X., Yang, Y., Lyu, M.R.: Aid: Efficient prediction of aggregated intensity of dependency in large-scale cloud systems. In: Proceedings of the 36th International Conference on Automated Software Engineering (ASE). pp. 653–665 (2021)

[29] Yang, T., Shen, J., Su, Y., Ren, X., Yang, Y., Lyu, M.R.: Characterizing and mitigating anti-patterns of alerts in industrial cloud systems. In: Proceedings of the 52st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE (2022)

[30] Yeh, C.C.M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H.A., Silva, D.F., Mueen, A., Keogh, E.: Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In: 2016 IEEE 16th international conference on data mining (ICDM). pp. 1317–1322. IEEE (2016)

[31] Zhao, G., Hassan, S., Zou, Y., Truong, D., Corbin, T.: Predicting performance anomalies in software systems at run-time. ACM Transactions on Software Engineering and Methodology (TOSEM) **30**(3), 1–33 (2021)