Springer Series in Reliability Engineering

Long Wang · Karthik Pattabiraman · Catello Di Martino · Arjun Athreya · Saurabh Bagchi *Editors*

System Dependability and Analytics

Approaching System Dependability from Data, System and Analytics Perspectives

Springer Series in Reliability Engineering

Series Editor

Hoang Pham, Department of Industrial and Systems Engineering, Rutgers University, Piscataway, NJ, USA

Today's modern systems have become increasingly complex to design and build, while the demand for reliability and cost effective development continues. Reliability is one of the most important attributes in all these systems, including aerospace applications, real-time control, medical applications, defense systems, human decision-making, and home-security products. Growing international competition has increased the need for all designers, managers, practitioners, scientists and engineers to ensure a level of reliability of their product before release at the lowest cost. The interest in reliability has been growing in recent years and this trend will continue during the next decade and beyond.

The Springer Series in Reliability Engineering publishes books, monographs and edited volumes in important areas of current theoretical research development in reliability and in areas that attempt to bridge the gap between theory and application in areas of interest to practitioners in industry, laboratories, business, and government.

Indexed in Scopus and EI Compendex

Interested authors should contact the series editor, Hoang Pham, Department of Industrial and Systems Engineering, Rutgers University, Piscataway, NJ 08854, USA. Email: hopham@rci.rutgers.edu, or Anthony Doyle, Executive Editor, Springer, London. Email: anthony.doyle@springer.com.

Long Wang · Karthik Pattabiraman ·
Catello Di Martino · Arjun Athreya ·
Saurabh Bagchi
Editors

System Dependability and Analytics

Approaching System Dependability from Data, System and Analytics Perspectives

Editors
Long Wang
Tsinghua University
Beijing, China

Karthik Pattabiraman
University of British Columbia
Vancouver, BC, Canada

Catello Di Martino
Nokia Bell Labs
São Paulo, Brazil

Arjun Athreya
Mayo Clinic
Rochester, MN, USA

Saurabh Bagchi
Purdue University
West Lafayette, IN, USA

ISSN 1614-7839 ISSN 2196-999X (electronic)
Springer Series in Reliability Engineering
ISBN 978-3-031-02062-9 ISBN 978-3-031-02063-6 (eBook)
https://doi.org/10.1007/978-3-031-02063-6

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Introduction

The idea of this book was born at the end of 2019, when we celebrated Professor Ravishankar K. Iyer's 70-year-old birthday. Professor Ravishankar K. Iyer is George and Ann Fisher Distinguished Professor in the Department of Electrical and Computer Engineering at University of Illinois at Urbana-Champaign (UIUC), Urbana, Illinois, USA. He has been our Ph.D. or Postdoctoral Advisor, and importantly, a lifelong mentor to us.

Professor Iyer has made seminal contributions to multiple sub-areas within the area of computing system dependability spanning his over 40-year career. And inspiringly, he is continuing to make more path-defining contributions. Therefore, this book took shape as reviewing some of the most important technical achievements in **four dominant themes in dependability**, namely *software dependability*, *large-scale systems and data analytics*, *healthcare and cyber-physical systems*, and *dependability assessment*. Each section is both a look back and a look forward. The look back describes the important milestones, several from the authors of the chapters, as well as detours on the way to the milestones. The look forward defines important open challenges, which are both relevant and technically challenging, needing concerted efforts from the community. Hopefully, this book will serve as a "call to arms" to the community to pick up some of these problems and to solve them.

Fittingly, we have a section with personal reflections from colleagues who have known Prof. Iyer well. The fact that they happen to be towering researchers in their own right adds more weight to these reflections. These reflections offer a view rarely seen in public documents and will, we hope, serve to inspire a fresh generation of researchers in the field of dependability and beyond.

Each section begins with a chapter, written by one of us, introducing the rest of the chapters in that section, and providing a broad perspective on the theme profiled in that section. These introductory chapters can serve as a guidepost for the reader wishing to selectively navigate through the chapters in the book.

Topic of Dependable Computing Systems

Dependability has long been studied in computer science and engineering—our premier conference, **IEEE/IFIP Dependable Systems and Networks**, or **DSN**, had its start in 1970. The importance of this area is understandable since human safety and well-being have long depended on computing and engineered systems. Research on computer system dependability has led to innumerable successes in fields as varied as follows: *aviation and space* (NASA was one of the early organizations that emphasized dependable computing), *supercomputing clusters*, *banking and finance*, *electric power*, *transportation*, and *distributed computing clusters*. As dependability earned more successes, we ventured into the construction of more complex large systems such as cloud platforms, big autonomous IT infrastructures, and the Internet of Things (IoT).

This book is titled *System Dependability and Analytics* to emphasize its focus on system dependability, rather than only of its component pieces, as well as its intersection with data-driven analytics and machine learning. This latter aspect is becoming increasingly important at a rapid pace. The impetus is coming from large amounts of data being generated by our systems, which are being analyzed for understanding dependability weaknesses and for mitigating effects of dependability failures. The field is growing, and we expect many foundational as well as applied advances to come in the next few years. This book is an early attempt to chart that course, though doubtless, there is a good deal of speculation involved in our charting activity.

Staging of Dependability Topics

In the early stage of his research career, Prof. Iyer worked on analysis of dependability data and building of dependability models from the data. Subsequently, he worked on the design of dependability technologies and measurement of system dependability. In the recent decade or so, his research focus has moved onto analytics-driven approaches to dependability, including a prominent focus on dependability in genomics and autonomous transportation. Correspondingly, this book features the four sections that approximately cover these themes. It also makes sense that Prof. Iyer's dependability research started with modeling and measurement and then steered toward application to use cases, as the models and measurement techniques gained maturity. Thus, his career exemplifies the synergistic relationship that should ideally exist between theory and practice. In terms of the target systems for the dependability techniques, Ravi's work spans a long arc. Correspondingly, this book follows such an arc covering dependability of mainframes (early era) to that of supercomputers and software systems, to analytics of healthcare systems, and now to CPS and autonomous systems.

We start off with the theme of software dependability where we look at software that goes in small to large devices. Then, we move to the dependability of large-scale systems and the aspect of data analytics introduced above. Next, we delve into the impact of dependability on healthcare and cyber-physical systems (CPS), two relatively recent but already highly impactful sub-areas. We then come to the topic of how to assess if our dependability design meets its goals or not. We end the book with personal reflections on Ravi from three of his colleagues at the University of Illinois at Urbana-Champaign.

Goals

By reading this book, the reader will obtain an understanding of leading-edge dependability techniques in the diverse areas of software, large-scale systems and data analytics, healthcare and CPS, and dependability assessment techniques. These are grouped into four corresponding sections of the book. The book does not aim for completeness of the coverage of these topics. Rather, it provides influential techniques that have strong theoretical foundations and, in many cases, have proven to be of practical value in real-world systems.

The contributors of this book are active researchers and practitioners in leading universities and research laboratories. They conduct research and build real-world systems, services, and technologies in the areas covered in this book. In the book, they bring forward their deep insights and provide their contemporary views and visions on dependability. Thus, researchers, professional practitioners, and graduate students will all obtain a clear-eyed view of the state of the art of the research and real-world practice of system dependability and analytics.

Biographical Note on Prof. Ravishankar K. Iyer

Professor Ravishankar K. Iyer is ACM Fellow, IEEE Fellow, AAAS Fellow, and served as Interim Vice Chancellor of UIUC for research during 2008–2011. He has received several awards, including the IEEE Emanuel R. Piore Award, and the 2011 ACM Outstanding Contributions award. He has supervised about 40 Ph.D. dissertations over his distinguished career.

Long Wang
Karthik Pattabiraman
Catello Di Martino
Arjun Athreya
Saurabh Bagchi

Contents

Software Dependability

Introduction: Software Dependability 3
Long Wang

Intelligent Software Engineering for Reliable Cloud Operations 7
Michael R. Lyu and Yuxin Su

Data Analytics: Predicting Software Bugs in Industrial Products 39
Robert Hanmer and Veena Mendiratta

From Dependability to Security—A Path in the Trustworthy Computing Research 55
Shuo Chen

Assessment of Security Defense of Native Programs Against Software Faults 69
Keun Soo Yim

Multi-layered Monitoring for Virtual Machines 99
Cuong Pham

Security for Software on Tiny Devices 141
Saurabh Bagchi

Large-Scale Systems and Data Analytics

Introduction: Large-Scale Systems and Data Analytics 163
Saurabh Bagchi

On the Reliability of Computing-in-Memory Accelerators for Deep Neural Networks 167
Zheyu Yan, Xiaobo Sharon Hu, and Yiyu Shi

Providing Compliance in Critical Computing Systems 191
Long Wang

Application-Aware Reliability and Security: The Trusted Illiac Experience 207
Karthik Pattabiraman

Mining Dependability Properties from System Logs: What We Learned in the Last 40 Years 221
Marcello Cinque, Domenico Cotroneo, and Antonio Pecchia

Critical Infrastructure Protection: Where Convergence of Logical and Physical Security Technologies is a Must 239
Luigi Coppolino, Salvatore D'Antonio, Giovanni Mazzeo, and Luigi Romano

Health Care and CPS

Introduction: Cyber Physical Systems and Healthcare Analytics 257
Arjun P. Athreya

On Improving the Reliability of Power Grids for Multiple Power Line Outages and Anomaly Detection 259
Jie Wu, Jinjun Xiong, and Yiyu Shi

Domain-Specific Security Approaches for Cyber-Physical Systems 301
Hui Lin

Uniting Computational Science with Biomedicine: The NSF Center for Computational Biotechnology and Genomic Medicine (CCBGM) 323
Liewei Wang and Richard M. Weinshilboum

Data-Driven Approaches to Selecting Samples for Training Neural Networks 327
Murthy V. Devarakonda

Classifying COVID-19 Variants Based on Genetic Sequences Using Deep Learning Models 347
Sayantani Basu and Roy H. Campbell

Twenty-First Century Cybernetics and Disorders of Brain and Mind 361
Gregory Worrell

Dependability Assessment

Introduction: Dependability Assessment 369
Karthik Pattabiraman

Effect of Epistemic Uncertainty in Markovian Reliability Models 371
Hiroyuki Okamura, Junjun Zheng, Tadashi Dohi, and Kishor S. Trivedi

System Dependability Assessment—Interplay Between Research and Practice 393
Mohamed Kaâniche and Karama Kanoun

Assessing Dependability of Autonomous Vehicles 405
Saurabh Jha

Personal Reflections

Foreword: Computing and Genomics at Illinois 425
Gene E. Robinson

An Academic Life Begins and Continues at University of Illinois at Urbana-Champaign 429
Janak H. Patel

Learning from Prof. Iyer 431
Wen-Mei Hwu

Software Dependability

Introduction: Software Dependability

Long Wang

Abstract This is the introduction of the 6 chapters in this "software dependability" section. Threats to software dependability are getting aggravated as more complex software and systems are being used and hardware devices with thinner MOSFET channel lengths are being used. This section presents 6 state-of-the-art work that demonstrate a few trends in software dependability research: popular use of data-driven AI, blurring limits between software dependability and security, and software dependability and security in emerging computing environments. The audience will get an up-to-date view of the software dependability research, especially its ongoing trends, after reading this section.

Keywords Dependability · Security · Blurring limit

Information technology (IT) is rapidly expanding its application scope and spreading into more critical domains such as electric power management, transportation traffic regulation and public health, in addition to the traditional domains of scientific computing, office business, finance and telecommunication, etc. Large computing platforms such as cloud systems and artificial intelligence (AI) platforms, and large networks such as internet-of-things (IoT) network are emerging as key computing infrastructures that host IT services. As a result, the complexities of software running on these modern computing systems have been increasing by a lot.

The rapid spread of software into broader critical domains and the increasing complexities of software demand high dependability of software. Moreover, hardware devices underlying computing systems are using MOSFET (or similar technologies) devices with very thin channel length (5 nm, or thinner expected in near future), which give rise to a much larger amount of soft errors in computing systems. This issue further aggravates the software dependability problem, and demands more focus be placed on software dependability in modern computing systems. However, the rapid progress of IT technologies also brings new capabilities of improving software dependability.

L. Wang (✉)
Tsinghua University, Beijing, China
e-mail: longwang@tsinghua.edu.cn

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_1

This section presents a select set of state-of-the-art work that demonstrate a few trends in software dependability research now. (i) One recent principal thrust addressing software dependability is through data-driven AI, including machine learning based on deep neural network, data analytics, and various classification techniques. (ii) Another trend is the blurring limits between software dependability and software security. Specifically, a number of technologies originally proposed and traditionally applied for software dependability are recently applied for software security and have demonstrated their significance in addressing security issues. Examples include bit flip injection, fuzzing (exploration of various inputs for tests), formal method, distributed consensus and monitoring technologies. As the limits between software dependability and security get blurring a new gate is open, and a number of technology advancements are being proposed and then employed in practice. (iii) Software dependability and security in emerging computing environments such as cloud systems and IoT environments are also hot topics recently.

The first two articles of this section demonstrate two good examples on how data-driven AI is adopted for addressing software dependability issues. *Intelligent Software Engineering for Reliable Cloud Operations*, authored by Prof. Lyu and Prof. Su, describes an AIOps (Artificial Intelligence for IT Operations) framework that employs AI technologies for anomaly detection in cloud systems. The framework leverages existing monitoring data of a cloud, particularly Key Performance Indicators (KPIs) data such as CPU usages of VMs, packet loss rates, packet error rates, etc., and applies neural network models to do anomaly detection and generate system incidents. Then the framework applies Graph Representative Learning algorithms to cluster and aggregate the incidents for failure diagnosis and root cause analysis. Hanmer and Prof. Mendiratta's *Data Analytics: Predicting Software Bugs in Industrial Products* presents a survey of software bug prediction techniques and a case study that employs source code complexity metrics, such as percent branch statements, block depth, line number of deepest block, statements at block level 0, to do bug prediction. The proposed technique in the case study uses Random Forest for the prediction. The two articles show that AI has demonstrated its super powerful capabilities in identifying patterns in complicated data, and such capabilities greatly help with anomaly detection, failure diagnosis, and error prediction.

The following three articles are examples that show blurring limits between software dependability and software security. Dr. Chen's *From dependability to security—a path in the trustworthy computing research* provides enlightenments on the relationships between dependability and security, between faults and attacks, by virtue of the author's own experience. Dependability and security are discussed in context of a common adversary model. Particularly, "bit flips", "formal methods" and "distributed consensus" are discussed as the main instruments used for both dependability and security (actually most of them, if not all, were proposed and applied first for dependability, and then repurposed for security). *Assessment of Security Defense of Native Programs Against Software Faults* by Dr. Yim studies security defense of C/C++ programs against faults. Faults and attacks, though they are two distinct adversaries of programs, are related in that faults, e.g. bit flips, may cause consequences of security breaches. This article conducts experimental studies of

"exploitable software faults", the software faults that can be exploited to result in security breaches, and shows both the capability of the fuzzing technology in finding exploitable software faults and the built-in security defense capability of programs against exploitable software faults. The article exposes interesting insights on how security-oriented exploitation and reliability faults are related. *Multi-layered Monitoring for Virtual Machines* by Dr. Pham describes a solution of VM monitoring for both reliability and security purposes. The solution covers all layers from hardware and hypervisor up to applications. It provides a quite comprehensive description of VM monitoring technologies. The audience will understand the challenges, pros and cons of VM monitoring technologies after reading this article. The three articles are part of the ongoing efforts that combine dependability research and security research.

The last article in this section, Prof. Bagchi's *Security for Software on Tiny Devices*, presents research challenges and potential approaches for providing security to software running on IoT devices. This is a very good introduction on software security on IoT devices. The unique challenges are clearly stated, and the discussions in the article span analysis techniques and algorithms, the enforcement of IoT software security that implements the analysis techniques and algorithms, and measurements, metrics and evaluations of IoT software security. The audience will obtain a clear view of state-of-the-art of the IoT software security from the article.

In summary, this section focuses on software dependability and presents a select set of state-of-the-art work on it. The audience of the section will get an up-to-date view of the software dependability research, especially its ongoing trends. This view is very important today as software dependability is gaining an unprecedented demand while undergoing a drastic change. Both are brought about by the wide and rapid adoption of technology advancements in cloud computing, AI, and other areas: IT services (and software) are growingly supporting more applications and scenarios including many in the critical domains such as public health, transportation traffic regulation and driving of vehicles, where traditionally IT technologies were not largely involved; at the same time, the technology advancements give rise to new approaches, many drastically different from traditional ones, to addressing software dependability issues.

Intelligent Software Engineering for Reliable Cloud Operations

Michael R. Lyu and Yuxin Su

Abstract Reliable Cloud operations are vital to our daily lives because many popular modern software systems are deployed in cloud systems. In this chapter, we discuss our experience in developing an AIOps (Artificial Intelligence for IT Operations) framework to improve the reliability of large-scale cloud systems with intelligence software engineering techniques. The comprehensive AIOps framework includes anomaly detection of key performance indicators, service dependency mining for failure diagnosis, and system incident aggregation for root cause analysis from various information sources like meter data, topology, alert, and incident tickets. We also conduct extensive experiments with production data collected from large-scale Huawei Cloud systems to demonstrate the effectiveness of intelligent software engineering techniques for reliable cloud operations.

1 Introduction

Modern software systems provide convenient services to our daily lives. In particular, IT enterprises start to deploy their applications and services on cloud computing platforms, such as search engines, instant messaging apps, and online shopping. Worldwide public cloud service revenue enjoys an impressive growth, as predicted by Gartner to reach 364 billion US dollars by 2022 [13].

Cloud services are large-scale distributed applications running across thousands of servers within datacenters. The most critical infrastructure of cloud computing is the data centers around the world. Data centers are massive hardware and software systems containing millions of servers, with high-speed interconnection networks. Each server is composed of hardware devices like CPU and memory, which runs an OS or virtual machine on top to manage the hardware resources. Software systems

M. R. Lyu (✉)
The Chinese University of Hong Kong, Hong Kong, Hong Kong
e-mail: lyu@cse.cuhk.edu.hk

Y. Su
Sun Yat-sen University, Guangzhou, China
e-mail: suyx35@mail.sysu.edu.cn

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
L. Wang et al. (eds.), *System Dependability and Analytics*, Springer Series in Reliability Engineering, https://doi.org/10.1007/978-3-031-02063-6_2

as cloud services are large-scale distributed applications running across thousands of servers within data centers.

As modern software systems have grown to an unprecedented scale, the tremendous complexity, scaling, and stringent performance of datacenter operations bring significant reliability challenges. Any cloud outage or breakdown will cause significant revenue loss, and harm customer trust and company reputation to cloud providers and service providers [6, 16]. According to Lloyd's report [21], a major failure that brings cloud outage for 3–6 d could result in a total loss of 19 billion US dollars revenue, most of which is not be covered by insurance. Worst of all, in a society highly dependent on IT infrastructure, cloud outages can affect everybody's life just like power outages. To this end, cloud resilience is of paramount importance.

Unfortunately, cloud failures leading to performance degradation or service interruptions have often occurred to major cloud operators. Cloud reliability issues are mainly due to the fact that tough cloud failures take a long time to mitigate manually. Cloud systems are actively undergoing continuous feature upgrades and system evolution by DevOps [9] paradigm, complex service dependency, load balance, and recovery procedures such as backup and restore; therefore, the statistical properties of system monitoring data may change from time to time. On-call engineers from different sectors equipped with multi-location, multi-source and multi-layer components have their specific responsibilities. Overall, the real root cause of cloud failures is hard to locate.

Traditionally, Software Reliability Engineering (SRE) aims to solve software reliability challenges by providing reliability models to track software failures. The tracked failure rates enable engineers to predict software reliability with analytical models using two or three parameters. The *Handbook of Software Reliability Engineering* [22] examined this process, and introduced the techniques to improve software reliability, including *fault avoidance*, *fault removal*, *fault tolerance*, and *fault prediction*.

This traditional analytical approach is not enough for today's complicated cloud software systems since modern cloud systems generate more complex and massive amounts of data concerning software reliability issues. To serve various users, cloud provides flexible infrastructure containing three major layers: *application layer*, *platform layer*, and *infrastructure layer*, displayed in Fig. 1. On-call engineers inspect the status of cloud from application and system logs, meter data generated from multiple components, and alerts triggered by rule-based monitor. Besides, top cloud systems provide customer service to collect most incidents, outages, or dissatisfaction from users. Customer service transfers feedback to on-call engineers. In order to obtain a comprehensive understanding of failures, on-call engineers from different sectors establish a war-room to discuss the problem and try to find possible solutions. This process generates incident tickets.

However, humans are not good at solving complex failure diagnosis problems associated with big data generated from large-scale cloud systems. But Artificial Intelligent (AI) algorithms have the opportunity to solve the complicated problems because AI algorithms are superior to human in big data analysis. For example, the

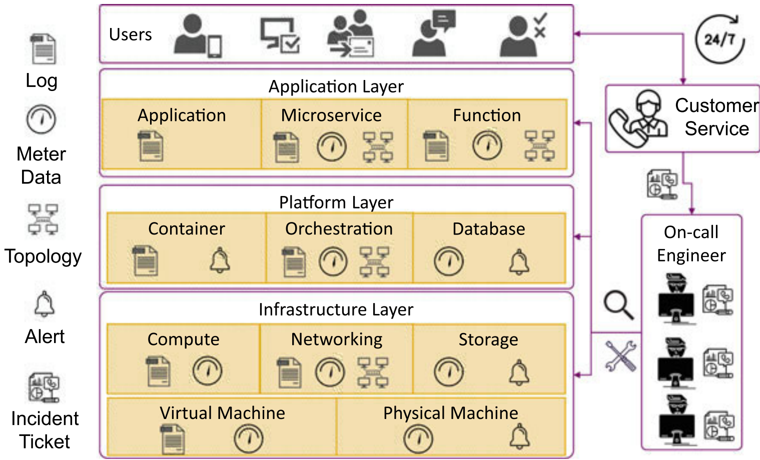

Fig. 1 Cloud systems generate a variety of data

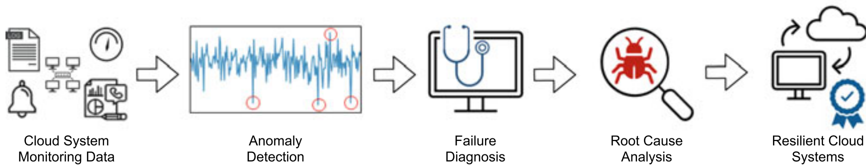

Fig. 2 The overall framework of resilient cloud systems with AIOps

Key Performance Indicator (KPI) "packet number" monitoring the cloud network may suddenly decrease because of anomalies happening in some network services. This may indicate a serious failure in the network. We would like to determine what failures are caused by the anomalies underneath, which is generally indicated by the sudden increase and drop of KPIs. Human maintainers generally assign different importance of system performance to distinct KPIs in the cloud. Generally, when diagnosing failures for large-scale cloud systems, an automated detection model with flexible importance assignment is more precise and quicker to signify the potential root cause than human maintainers.

In this chapter, we describe our experience on the development of AIOps (Artificial Intelligence for IT Operations) framework to tackle several reliability challenges commonly seen in industrial cloud systems. We provide a general end-to-end pipeline of intelligent software engineering illustrated in Fig. 2 to conduct *anomaly detection*, *failure diagnosis*, and *root cause analysis* with multiple sources of heterogeneous information such as meter data, topology, alert and incident ticket. Specifically, the root cause analysis in cloud systems differs from the traditional definition in software reliability engineering that aiming to identify the exact fault of a particular failure. In cloud systems, it is more practical to narrow down the scope of system components

associated with a failure. We also conduct extensive experiments with real-world large-scale cloud systems from Huawei to demonstrate the effectiveness of intelligent software engineering techniques for reliable cloud operations.

2 Anomaly Detection of Key Performance Indicators

2.1 Background

Key Performance Indicators (KPIs) are the most important data in the cloud, which are leveraged to monitor the health status of a machine, like network traffic, response delay and CPU usage. Anomaly detection over the KPIs is a critical tool to ensure the reliability and availability of the system, which aims to discover unexpected events or rare items in data. Different system components (e.g., microservices, servers) are tightly coupled, and cloud failures usually trigger anomaly performance in multiple KPIs. For example, a problematic load balance server is often accompanied by a burst on both round-trip delay and in-bound traffic rate, which will further increase CPU utilization.

Recent studies tackle this problem by constructing an $m \times m$ KPI inner-product matrix [36] or a complete graph [37] for m different KPIs to capture the pairwise KPI interaction, both of which yield an $\mathcal{O}(m^2)$ computation complexity. A real-world example is provided in Fig. 3, which is from a public dataset released by [31]. *CPU LOAD* and *ETH INFLOW* are highly correlated as their curves exhibit a very similar

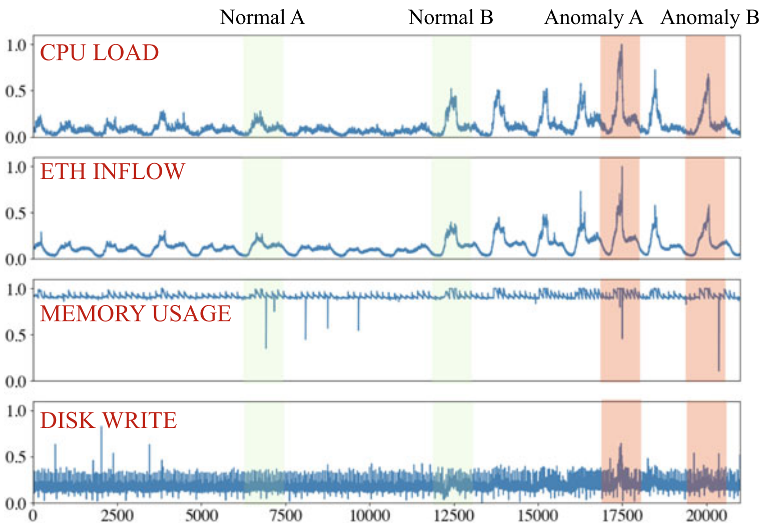

Fig. 3 Multivariate KPIs snippet from server machine dataset

trend. The correlated relationship provides an overall picture of the systems' health status. For example, in the segment marked as *Normal A*, we can see a clear spike in *MEMORY USAGE*, which would be flagged as an anomaly without a glance at the other three KPIs. Similar situation happens to segment *Normal B*, where boots can be clearly seen in both *CPU LOAD* and *ETH INFLOW*. Therefore, we need to consider the full set of multivariate KPIs to pursue an accurate anomaly detection, as shown in segment *Anomaly A* and *Anomaly B*. Besides the dependency between KPIs, we can also leverage historical KPI patterns to reduce false positives. Specifically, in Fig. 3, all KPIs have witnessed some abnormal spikes in history. However, they do not necessarily indicate the occurrence of failures.

In industrial systems, hundreds or even thousands of KPIs are being monitored. The dependencies among KPIs are very sparse, i.e., most KPIs are not or weakly dependent on other KPIs. Therefore, how to automatically learn the dependencies among different KPIs is critical towards efficient multivariate KPI anomaly detection. In the literature, many studies have shifted to anomaly detection on multivariate KPIs, which mainly resorts to different neural network models. For example, Omni-Anomaly [31] proposes to learn the normal patterns of multivariate time series by modeling data distribution through stochastic latent variables. Anomalies are then determined by reconstruction probabilities. Similarly, Malhotra et al. [24] used an LSTM-based (long short-term memory-base) encoder-decoder network to learn time series's normal patterns and Zhang et al. [36] used an attention-based convolutional LSTM network for the learning purpose. Although tremendous progress has been made, we still observe two major limitations of existing approaches: (1) the interactions among KPIs are not explicitly modeled, and (2) the efficiency falls behind industrial needs. Specifically, previous approaches [28, 31] detect anomalies on multivariate KPIs mainly by stacking different types of KPIs into a feature matrix and feeding it to sophisticated neural network models. Different from previous work, we argue that by properly modeling the interactions of KPIs along with feature and temporal dimensions, cost-effective neural network models can be leveraged for anomaly detection.

To overcome the aforementioned limitations, we introduce CMAnomaly illustrated in Fig. 4, which is an efficient unsupervised model for anomaly detection over multivariate KPIs. CMAnomaly consists of four phases, i.e., *data preprocessing*, *collaborative machine*, *model training*, and *anomaly detection*. The first phase preprocesses the data by applying normalization and window sliding. Particularly, the input types of KPIs can vary depending on the application scenario. In the next phase, the preprocessed data are fed to the proposed collaborative machine, which is the core component of CMAnomaly. The collaborative machine can properly capture the interactions among multivariate KPIs along with both feature and temporal dimensions. In the third phase, we train a forecasting-based anomaly detection model [10, 18], which detects anomalies based on prediction errors. Finally, the trained model will be applied to detect anomalies for new observations.

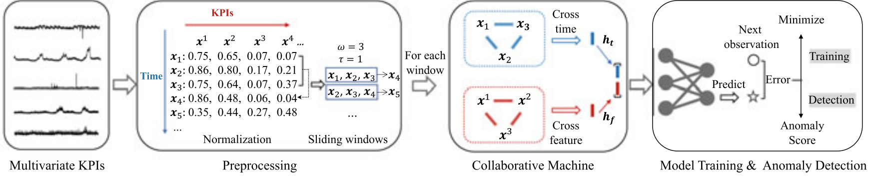

Fig. 4 Overall framework of CMAnomaly

2.2 *Preprocessing*

The input of multivariate KPIs is denoted as $X \in \mathbb{R}^{n \times m}$, where n is the number of different KPIs and m is the number of observations. The t-th row of X, denoted as $x_t = [x_t^1, x_t^2, \ldots, x_t^m]$, is an m-dimensional vector containing the observation of each KPI at timestamp t. Similarly, the k-th column of X, denoted as $x^k = [x_1^k, x_2^k, \ldots, x_n^k]$, is an n-dimensional vector containing the observations of the k-th KPI. Particularly, we denote $x_{i:j}^k = [x_i^k, x_{i+1}^k, \ldots, x_j^k]$ as a consecutive sequence of x^k from timestamp i to j. The objective of anomaly detection for multivariate KPIs is to determine whether or not a given x_t is anomalous, i.e., whether the entity is in abnormal status at timestamp t. For each timestamp t, our model calculates an anomaly score $s_t \in [0, 1]$, which represents the probability of x_t being anomalous. If s_t is larger than a pre-defined threshold θ, x_t will be predicted as an anomaly. The ground truth $\mathbf{y} \in \mathbb{R}^n$ is an n-dimensional vector consisting 0 and 1, where 0 indicates a normal point, and 1 indicates an anomalous one.

Different KPIs may have distinct scales, for example, the KPI monitoring the CPU execution, i.e., *CPU USAGE*, is in the range of 0% to 100%. However, the KPI monitoring the network traffic, i.e., *INBOUND PACKAGE RATE* can range from zero to millions of kilobytes. Therefore, data normalization is performed for each individual KPI to ensure the robustness of our model. We apply max-min normalization to each individual KPI, i.e., x^k, as follows:

$$x_{norm}^k = \frac{x^k - \min(x^k)}{\max(x^k) - \min(x^k)}, \tag{1}$$

where the values of $max(x^k)$ and $min(x^k)$ are computed in the training data, which will then be used for test data normalization. For simplicity, we omit the "norm" subscript in the following elaboration. The sliding window is to partition KPIs along the temporal dimension. Particularly, it consists of two attributes, i.e., window size ω and stride τ. The stride indicates the forwarding distance of the window along the time axis to generate multivariate KPI windows. As the stride is often smaller than the window size, there exists overlapping between two consecutive windows. We denote the s-th sliding window as:.

$$X_s = [x_{s\tau}, x_{s\tau+1}, \ldots, x_{s\tau+\omega-1}] \tag{2}$$

where $s \in [0, 1, 2, \ldots]$. X_s together with the observations at the next timestamp of the window, i.e., $\hat{x}_s = x_{s\tau+\omega}$, constitute a pair $(X_s, \hat{x}_s)$, where $X_s \in \mathbb{R}^{\omega \times m}$ and $\hat{x}_s \in \mathbb{R}^m$.

2.3 Multivariate KPIs Interactions

As shown in Fig. 3, historical patterns of KPIs provide important clues for anomaly detection on multivariate KPIs accurately. To explicitly capture the dependency between multivariate KPIs and their historical patterns, for each sliding window, denoted as $X_s \in \mathbb{R}^{\omega \times m}$, we calculate the pairwise inner product of all KPI feature vectors, i.e., $x^k_{s\tau, s\tau+\omega-1}$, $k \in [1, m]$, and temporal vectors, i.e., x_t, $t \in [s\tau, s\tau + \omega - 1]$.

$$h_f = b_0 + \sum_{i=1}^{m} w_i x^i + \sum_{i=1}^{m} \sum_{j=i+1}^{m} \left\langle x^i, x^j \right\rangle v_i v_j \tag{3}$$

$$h_t = \hat{b}_0 + \sum_{i=1}^{\omega} \hat{w}_i x_i + \sum_{i=1}^{\omega} \sum_{j=i+1}^{\omega} \left\langle x_i, x_j \right\rangle \hat{v}_i \hat{v}_j \tag{4}$$

The cross-feature and cross-time KPI interactions, denoted as h_f and h_t, are formulated as Eqs. 3 and 4, respectively. In Eq. 3, b_0, w_i, v_j, $v_j \in \mathbb{R}$ are trainable parameters, x^i, $x^j \in \mathbb{R}^\omega$ are the i-th and j-th column of X_s with each column representing all the observations of a KPI in the corresponding window, and $< \cdot, \cdot >$ is the operation of inner product. These equations are composed of three terms: the first term is a trainable bias, the second term is a weighted sum of all KPIs without explicit interaction, and the third term is the core part of the proposed collaborative machine, which models the pairwise KPI interactions.

2.4 Collaborative Machine for Anomaly Detection

The last two phases of CMAnomaly are model training and anomaly detection. In the detection phase, the well-trained model predicts the next KPI values given preceding observations. In the training phase, as most multivariate KPIs would reflect the normal status of an entity, the model will learn the normal patterns of KPIs, i.e., what the next observations would be given previous ones. Although there could be anomalies in the training data, they tend to be forgotten by the model as they rarely appear. Consequently, in the detection phase, the model will predict "normal" KPI values based on the learned patterns. If the real observations deviate from the predicted ones by a significant margin, an anomaly may happen, i.e., the entity is not in its normal status. Therefore, such deviation measures the likelihood of the occurrence of the anomaly.

Our framework supports various types of neural network models for anomaly detection. The anomaly detection model can be formulated as follows:

$$\tilde{h}_{i+1} = \sigma(\tilde{h}_i \tilde{X}_i + \tilde{b}_i), i = 0, 1, \ldots, L - 1, \tag{5}$$

where L is the number of layers of the Multilayer Perceptron (MLP) model, $\tilde{W}_i, \tilde{b}_i$ are trainable parameters with customized size, and $\sigma(x) = max(0, x)$ is the ReLU activation function. We simultaneously consider the cross-feature and cross-time KPI interactions by concatenating h_f and h_t, which is the input to the model, i.e., $\tilde{h}_0 = concat(h_f, h_t)$. $\hat{y} = \tilde{h}_L \in \mathbb{R}^m$ is the prediction result produced by the last layer of the MLP model, which contains the predicted values for all KPIs at the next timestamp.

Anomaly detection model is optimized by minimizing the following mean square error (MSE) loss $\mathcal{L}$ between the predictions and the ground truth observations:

$$\mathcal{L} = \sum_{i=1}^{N} \left\| \hat{y}_i - \hat{x}_i \right\|_2, \tag{6}$$

where N is the number of training sliding windows. $\hat{y}_i \in \mathbb{R}^m$ and $\hat{x}_i = x_{i\tau+\omega} \in \mathbb{R}^m$ are the predicted and ground truth observations for the i-th window, respectively.

With the minimization of loss $\mathcal{L}$ during training, CMAnomaly can learn from the normal patterns in the training data by updating all trainable parameters, e.g., $v_i v_j$ denoting the interaction weights. After the model is trained, we compute an anomaly score for each window X_i in the testing data. Then, we first calculate the MSE between the predicted and ground truth observations, and then apply the sigmoid function to rescale the score to the range [0, 1], which represents the probability of the occurrence of an anomaly:

$$s_i = \phi\left(\frac{1}{m} \left\| \hat{y}_i - \hat{x}_i \right\|_2\right) \tag{7}$$

where $\phi(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. To determine whether an anomaly has happened, a threshold θ should be defined for the anomaly score. The timestamps with a large anomaly score, i.e., $s_i \geq \theta$, should be regarded as anomalous points.

In reality, the threshold can be set by on-site engineers based on their experience. A large threshold imposes a strict anomaly detection policy, which may miss important system failures, i.e., low recall. However, a small threshold increases the sensitivity to KPI changes, resulting in false alarms, i.e., low precision.

Table 1 Accuracy comparison on Huawei Cloud dataset

Methods	Precision	Recall	F1
OmniAnomaly	0.6639	0.8382	0.7283
LSTM-VAE	0.8273	0.7436	0.7560
CMAnomlay	**0.9179**	0.8202	**0.8368**

2.5 Experiments

In this part, we evaluate CMAnomaly using both public data and industrial data. We collected real-world KPIs from Huawei Cloud to conduct a more comprehensive evaluation. Huawei Cloud contains a large number of nodes supporting tens of millions of users worldwide. Therefore, to provide a stable 24×7 service, the status of each component of the network is closely monitored with KPIs. The engineers can fix problematic components timely if the anomalies of KPIs can be automatically detected and reported in real-time. To evaluate our method in a practical scenario, we collected a 30-day-long KPIs dataset with 13 network components within Jan. 2021. Each of the network components has 70$\sim$200 different types of KPIs. We use the first 20 d of KPIs as the training data and the rest as the testing data. Then, several experienced engineers were invited to manually label the anomalous points in the testing data.

To study the effectiveness of CMAnomaly, we compare its performance with two most effective open-source anomaly detection methods, i.e., LSTM-VAE [28] and OmniAnomaly [31] on the dataset collected from Huawei Cloud.

The experimental results are shown in Table 1. In particular, the precision of OmniAnomaly is the lowest, but the recall is the highest because the complex architecture of OmniAnomaly incurs more trainable parameters, which makes it easier to overfit the training data. Therefore, OmniAnomaly is more sensitive to capture more anomalies but has the most false positive alarms. LSTM-VAE has a more light-weight design than OmniAnomaly, so LSTM-VAE suffers less overfitting. As a result, LSTM-VAE only raises the anomaly score when the new observation deviates more from the prediction. In this case, although higher precision is achieved, LSTM-VAE has the lowest recall because it cannot effectively find all possible anomalies. CMAnomaly can balance precision and recall better and achieves the best F1 score, $\sim$0.08 higher than the second-best one achieved by LSTM-VAE. The collaborative machine facilitates CMAnomaly to capture the dependency of the training KPIs effectively. Therefore, CMAnomaly avoids overfitting the noisy points existing in the training data, e.g., usual spikes as shown in Fig. 3. CMAnomaly reports a higher anomaly score only when the dependent KPIs are anomalous, thus achieving the highest precision. Moreover, CMAnomaly keeps the sensitivity to detect more true positive samples thanks to its ability to capturing the dependency.

3 Service Dependency Mining for Failure Diagnosis

3.1 Background

Service reliability is one of the key challenges that cloud providers have to deal with. The common practice nowadays is developing and deploying small, independent, and loosely coupled cloud microservices that collectively serve users' requests. The microservices communicate with each other through well-defined APIs. Under this architecture, microservice management frameworks like Kubernetes will be responsible for managing the life cycles of microservices. Developers can focus on the application logic instead of the bothering tasks of resource management and failure recovery. It enables agile development and supports polyglot programming, i.e., microservices developed under different technical stacks can work together smoothly.

However, the loosely coupled nature of microservices makes it difficult for engineers to conduct system maintenance. Different microservices in a large cloud system are usually developed and managed by separate teams. Each team only has access to their own services as well as services that are closely related, which means they only have a local view of the whole system [32]. As a result, failure diagnosis, fault localization, and performance debugging in a large cloud system become more complex than ever [12, 33]. Despite various fault tolerance mechanisms introduced by modern cloud systems, it is still possible for minor anomalies to magnify their impact and escalate into system outages.

Although microservice management frameworks provide automatic mechanisms for failure recovery, unplanned service failures may still cause severe cascading effects. For example, failures of critical services that provide basic request routing functions will impact the invocation of cloud services, slow down request processing, and deteriorate customer satisfaction. Therefore, evaluating the impact of service failures rapidly and accurately is critical to the operation and maintenance of cloud systems. Knowing the scope of the impact, reliability engineers can emphasis on services that have more significant impacts on others.

3.2 Tracing Analysis

For commercial cloud providers, it is crucial to troubleshoot and fix failures in a timely manner because massive user applications may be affected even by a small service failure [4]. In large-scale cloud systems, a request is usually handled by multiple chained service invocations. As clues to defective services are hidden in the intricate network of services, it is difficult for even knowledgeable SRE personnel to keep track of how a request is processed in the cloud system. All the services and dependencies in a cloud system collectively construct a directed graph of services, which is also called a dependency graph. The dependency graph of a cloud system can be very complicated.

Fig. 5 A span generated by the train-ticket benchmark

Span ID	e22f30bdbfd09134
Parent Span ID	b42a04bf18997d5d
Name	ts-preserve-service
Timestamp (μs)	1618589098705000
Duration (μs)	1126
Result	SUCCESS
Trace ID	c0d17d481f47bdd9
Additional Logs	……

Distributed tracing provides an approach to monitor the execution path of each request in a dependency graph. For chained service invocations, e.g., service `A` invokes service `B` and service `B` invokes service `C`, it is important to know the status of each service invocation, including the result, the duration of execution, etc. By adding hooks to the services and microservices of the cloud system, a distributed tracing system [11] can record the contextual information of each service invocation. Such records are called *span logs*, abbreviated as `spans`. A span represents a logical unit of execution that is handled by a microservice in a cloud system. All the spans that serve for the same request collectively form a directed graph of spans. Such directed graph of spans generated by request is called a piece of *trace log*, abbreviated as a `trace`. With a trace, engineers can track how the request propagates through the cloud system. Collectively analyzing the traces of the entire cloud system can help engineers obtain in-depth latency reports that could assist failure diagnosis, fault localization, and surface performance degradation in the cloud system.

Although the actual implementation of distributed tracing system varies a lot, the types of information they record are similar. For clarity, we formally describe the attributes of spans as follows. Suppose we have a trace T composed of spans $\{s_1, s_2, \ldots, s_n\}$, a span $s_i \in T$ contains the following attributes.

s_i^{id} The ID of span s_i,
s_i^{pid} The ID of the parent span of s_i,
s_i^{tid} The ID of the trace that s_i belongs to,
s_i^{name} The name of service/microservice corresponding to s_i,
s_i^{ts} The time stamp of s_i,
s_i^{d} The duration of execution of s_i, and
s_i^{r} The result of execution of s_i.

Figure 5 illustrates a span generated by the train-ticket benchmark [38]. It means that service `ts-preserve-service` was invoked at 04:58 on April 17, 2020. The duration of execution is 1126 μs and the execution result is SUCCESS.

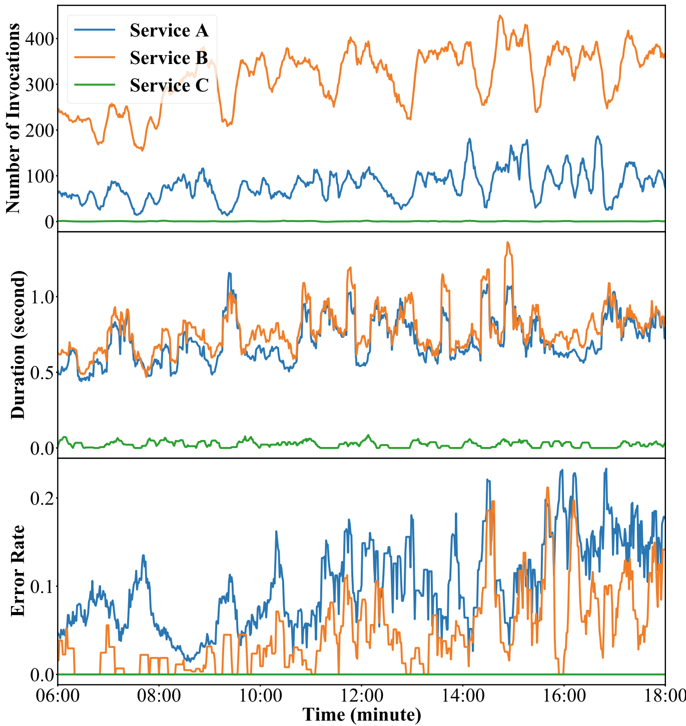

Fig. 6 The statuses of service `A`, `B` and `C`. `A` invokes `B` and `C` but `B` has a greater effect on `A`

3.3 Intensity of Service Dependency

Existing tools treat the dependency as a binary relation, i.e., if the caller service invokes the callee service then the caller is dependent on the callee. We suggest that this binary dependency metric is not fine-grained enough for cloud maintenance. Figure 6 shows the statuses of three services[1] `A`, `B`, and `C` in Huawei Cloud. Service `A` invokes both service `B` and service `C`. Service `B` encountered failures. The x-axis represents time in minute. The y-axes represent the number of invocations per minute, the average duration of invocations per minute, and the error rate per minute of `A`, `B`, and `C`. Although service `A` invokes service `B` and service `C`, it is obvious that the statuses of `B` and `C` influence the status of `A` in different degrees.

The reason is that the functionalities provided by service `A` and `B` are creating virtual machines, and allocating block storage, respectively. Creating a virtual machine requires allocating one or more block storage. Thus, the failure of service `B` inevitably affects service `A`. On the contrary, due to the fault tolerance mechanism of service `A`, the failure of service `C` will not affect service `A` significantly.

[1] For confidentiality reasons, we cannot reveal the names of the related services.

Thus, it is more accurate to say that the intensity of dependency between service A and service B is higher than the intensity of dependency between service A and service C. Ideally, if the development team of every cloud microservice accurately provide the intensity of dependencies for every dependent service, the failure diagnosis could be accelerated. On-call engineers (OCEs) can prioritize the services that exhibit the higher intensity of dependency instead of inspecting all the dependent services if they have accurate intensity information. However, due to the complexity and the fast evolving nature of cloud systems [2], manually maintaining the dependency relations with intensity is very difficult. As a result, OCEs often struggle in diagnosing failures due to the lack of intensities.

3.4 Dependency Strength Mining

In order to relieve the pressure on OCEs, we introduce a framework called AID [35] to predict the Aggregated Intensity of service Dependency in large-scale cloud systems. The intuition is that direct service invocation incurs direct dependency to some degree. To properly capture service dependency, AID consists of three steps: candidate selection, status generation, and intensity prediction. We will introduce the details in the following parts:

3.4.1 Candidate Selection

Given the raw traces, AID first generates a set of candidate service pairs (P, C) where service P directly invokes service C. In general, direct service invocations can be divided into two categories, i.e., synchronous invocations and asynchronous invocations. Modern tracing mechanisms can keep track of both synchronous and asynchronous invocations [27]. Given all the raw traces of a cloud system, in this step, we generate a candidate dependency set $Cand$. The candidate dependency set $Cand$ contains service invocation pairs $(P_1, C_1), (P_2, C_2), \cdots, (P_n, C_n)$. Each pair (P_i, C_i) in the candidate dependency set denotes that the service named P_i invokes the service named C_i at least once. Therefore, service P_i depends on service C_i. This step is to shrink the search space of possible dependent pairs because the service invocations indicate direct dependencies.

3.4.2 Service Status Generation

The status of one service is composed of three aspects of dependency, i.e., number of invocations, duration of invocations, and error of invocations. Each aspect of the service's status contains one or more KPIs, depending on the actual implementation of the distributed tracing system. As service invocations occur repeatedly, the three statuses of service invocations can derive three aspects of service dependency:

Number of Invocations	The number of invocations from the caller to the callee.
Duration of Invocations	The duration of invocations.
Error of Invocations	The number of successful invocations from the caller to the callee.

Inspired by the common practice in cloud monitoring [1], we distribute the spans of one service into many bins according to the spans' timestamps. Each bin accepts spans whose timestamp is in a short, fixed-length period. We denote the length of the short period as τ. For example, the span shown in Fig. 5 will be put in the bin of `ts-preserve-service` at time 04:58, 17 April 2020. We can then represent the status of a cloud service in a short period by the statistical indicators of all the spans in the corresponding bin. Formally, given all the spans in the cloud system over a long period T, we first initiate $\mathbf{M} \times \mathbf{N}$ empty bins of the predefined size τ. $\mathbf{M}$ is the number of microservices. $\mathbf{N}$, determined by $\frac{T}{\tau}$, is the number of bins. Then we distribute all spans into different bins according to their timestamp s^{ts} and service name s^{name}. After that, we can calculate the following three types of indicators as the KPIs for each bin.

$invo_t^m$	Total number of invocations (spans) in the bin;
err_t^m	Error rate of the bin, i.e., the number of errors divided by the number of invocations;
dur_t^m	Averaged duration of all spans in the bin;

where t is the time of the bin and m is the service name of the bin. If a service is not invoked in a particular bin (i.e., the corresponding bin is empty), all the KPIs will be zero. In this scenario, we obtain the KPIs of every service S at every period t. Ordering the bins by t, we get three time series of KPIs for each cloud service, denoted as $invo^S$, err^S, and dur^S as the status of each cloud service.

3.4.3 Intensity Prediction

The intensity prediction steps quantitatively predict the intensity of dependency by measuring the similarity between two service's statuses. The similarity between two service's statuses is a normalized and weighted average of the similarity of all the KPIs of the two services. We calculate the similarity between two KPIs by a dynamic time warping algorithm (DTW) [19] and aggregate all the similarities to get the overall similarity.

DTW automatically warps the time in chronological order to make the two status series as similar as possible and get the similarity by summing the cost of warping. It utilizes dynamic programming to calculate an optimal matching between two status series. Given two services P, C, and their status series $invo^P$, $invo^C$, err^P, err^C, dur^P, and dur^C, the warping from the callee C to the caller P is specially designed for the cloud environment.

For all $(P_i, C_i) \in Cand$, we calculate similarities between their status series, denoted as $d_{invo}^{(P_i,C_i)}$, $d_{err}^{(P_i,C_i)}$, and $d_{dur}^{(P_i,C_i)}$. We normalize the similarity across the

whole candidate set with a min-max normalization with Eq. 8, where $status \in \{invo, err, dur\}$.

$$d_{status}^{(P_i,C_i)} = \frac{d_{status}^{(P_i,C_i)} - \min(d_{status}^{(P,C)})}{\max(d_{status}^{(P,C)}) - \min(d_{status}^{(P,C)})} \tag{8}$$

The intensity of dependency between P_i and C_i is the average similarity of all three similarities between their status series.

$$I^{(P_i,C_i)} = \frac{1}{3} \sum_{status \in S} d_{status}^{(P_i,C_i)}, S = \{invo, err, dur\} \tag{9}$$

Finally, we can build the dependency graph with intensity from the candidate set and the corresponding intensity values.

3.5 Experiments

In this part, we evaluate AID on both simulated dataset and industrial dataset from Huawei Cloud system. For the simulated dataset, we deploy train-ticket [38], an open-source microservice benchmark, for data collection. Apart from the simulated dataset, we also collected a 7-day-long trace dataset with 192 microservices in April 2021 from a region of Huawei Cloud to evaluate AID. Table 2 displays the detailed information about these two datasets.

Since there is no existing work that measures the intensity of service dependency, we employ Pearson correlation coefficient, Spearman correlation coefficient, and Kendall Rank correlation coefficient as the baseline. Particularly, we calculate correlation on the status series of a candidate dependency pair (P, C). For the baselines, we directly use the implementation from Python package `scipy`.[2] We map the correlation to [0, 1] with the function $f(x) = (x + 1)/2$. The intensities of dependencies are then produced in the same way as Eq. 9.

We employ Cross Entropy (CE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE), as calculated in Eq. 10 to evaluate the effectiveness of AID in predicting the intensity of dependency. Specifically, cross entropy calculates the difference between the probability distributions of the label and the prediction. Mean absolute error and root mean squared error measure the absolute and squared error. Lower CE, MAE, and RMSE values indicate a better prediction.

[2] https://www.scipy.org/.