Toward Fine-Grained, Unsupervised, Scalable Performance Diagnosis for Production Cloud Computing Systems

Haibo Mi, *Student Member*, *IEEE*, Huaimin Wang, *Member*, *IEEE*, Yangfan Zhou, *Member*, *IEEE*, Michael Rung-Tsong Lyu, *Fellow*, *IEEE*, and Hua Cai, *Member*, *IEEE*

Abstract—Performance diagnosis is labor intensive in production cloud computing systems. Such systems typically face many real-world challenges, which the existing diagnosis techniques for such distributed systems cannot effectively solve. An *efficient*, *unsupervised* diagnosis tool for locating *fine-grained* performance anomalies is still lacking in production cloud computing systems. This paper proposes CloudDiag to bridge this gap. Combining a statistical technique and a fast matrix recovery algorithm, CloudDiag can efficiently pinpoint fine-grained causes of the performance problems, which does not require any domain-specific knowledge to the target system. CloudDiag has been applied in a practical production cloud computing systems to diagnose performance problems. We demonstrate the effectiveness of CloudDiag in three real-world case studies.

Index Terms—Cloud computing, performance diagnosis, request tracing

1 Introduction

Performance diagnosis is labor intensive, especially for typical production cloud computing systems. In such systems, a lot of software components bear a large number of replicas (component instances) distributed in different physical nodes in the cloud. They can be assembled into multiple types of services, serving large amounts of user requests. The services provisioned by the cloud are often prone to various performance anomalies (e.g., SLA violations [1]) caused by software faults, unexpected workload, or hardware failures. Such defects may, however, be manifested only in a small part of component replicas, hiding themselves in a large number of normal component replicas.

Our experiences in performance diagnosis for Alibaba Cloud Computing[1] show that troubleshooting performance anomalies in practical production cloud computing systems faces many real-world challenges. It is very difficult to apply existing diagnosis techniques for such distributed systems. We summarize the new design challenges as follows:

1. Performance diagnosis in *fine granularity*. A component typically has a lot of replicas in a production cloud system. Within one component, there are many performance-related private methods (i.e., those invoked inside the component) and public methods (i.e., the interfaces invoked by other components). It is very challenging to localize anomalous methods as well as their corresponding physical replicas. Current approaches generally focus on locating anomalous physical nodes (e.g., [2]) or logical components (e.g., [3]). Such coarse-grained results are not enough. In the former case, given an anomalous physical node, a system operator has to identify the faulty component among many components typically hosted in the same node. In the latter case, given an anomalous logical component, the operator has to identify which one among their numerous replicas distributed in the cloud is faulty. Consequently, huge human efforts are still required to further pinpoint the subtle primary cause. Performance diagnosis in a fine granularity is of high concern to reduce manual efforts.
2. *Unsupervised* performance diagnosis. Many existing performance diagnosis techniques resort to system behavior models in identifying anomalies [4], [5]. Unfortunately, it is hard to manually build such models in production cloud systems, given their complexity in system scale. In addition, cloud services are generally composed of many components developed by different teams, which are independently updated online. It is extremely difficult to maintain the behavior models for such evolutional systems. Hence, a performance diagnosis tool for production cloud systems should be completely unsupervised, without assuming that any prior knowledge about the service should be input.
3. Performance diagnosis with *high efficiency*. Coping with large runtime data generated by a production

1. Alibaba cloud computing is a subsidiary of Alibaba, Inc., one of the largest e-commerce companies in the world. It designs and maintains the data-centric cloud computing facilities for Alibaba, Inc.

- *H. Mi and H. Wang are with National Laboratory for Parallel & Distributed Processing, National University of Defense Technology, Changsha, Hunan 410073, China. E-mail: rainmhb@gmail.com, whm_w@163.com.*
- *Y. Zhou and M.R. Lyu are with Shenzhen Research Institute, The Chinese University of Hong Kong, Ho Sin Hang Engineering Building, Shenzhen, Hong Kong, China. E-mail: {yfzhou, lyu}@cse.cuhk.edu.hk.*
- *H. Cai is with Alibaba Cloud Computing, Alibaba, Inc., Hangzhou, China. E-mail: ch.caih@aliyun-inc.com.*

Manuscript received 1 Mar. 2012; revised 31 Dec. 2012; accepted 31 Dec. 2012; published online 11 Jan. 2013.
Recommended for acceptance by V.B. Misic, R. Buyya, D. Milojicic, and Y. Cui.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDSSI-2012-03-0229.
Digital Object Identifier no. 10.1109/TPDS.2013.21.

cloud system efficiently is a challenging task in performance diagnosis. Many defects only manifest themselves in an online production cloud that involves a large number of component replicas. Unlike a small-scale in-house debugging system, a production cloud system can generate massive performance logs during its runtime, which are recorded in a distributed manner across a large number of cloud nodes. It is, therefore, critical to design a fast, scalable performance diagnosis tool chain that can assemble the relevant logs on demand when performance anomalies occur, and quickly pinpoint the primary cause accordingly. Yet, this is not the focus of the current approaches (e.g., [6]).

Typical production cloud systems are service-oriented in nature. The response time of user requests directly reflects the system performance. In this regard, tracing user requests is a viable means to exposing performance data, so as to help performance diagnosis. Recent work [7], [8], [9], [10] has shown that it is promising to pinpoint performance anomalies with end-to-end request tracing data. However, an *efficient*, *unsupervised* diagnosis tool for locating *fine-grained* performance anomalies is still lacking.

This paper bridges this gap by proposing CloudDiag. CloudDiag periodically collects the end-to-end tracing data (In particular, execution time of method invocations) from each physical node in the cloud. It then employs a customized Map-Reduce algorithm to proactively analyze the tracing data. Specifically, it assembles the tracing data of each user request, and classifies the tracing data into different categories according to call trees of the requests.

When the cloud system is suffering performance degradation (e.g., average response time of user requests is larger than a threshold), a cloud operator can access CloudDiag with its web interfaces to conduct a performance diagnosis. With the request tracing data, CloudDiag will perform a fast customized matrix recovery algorithm to instantly identify the method invocations (together with the replicas they locate) which contribute the most to the performance anomaly. The whole process requires no domain-specific knowledge to the target service.

CloudDiag has been successfully launched in diagnosing performance problems for the production cloud systems in Alibaba Cloud Computing. We report three case studies in our real-world performance diagnosis experiences to demonstrate the effectiveness of CloudDiag in helping the operators localize the primary causes of performance problems.

The rest of this paper is organized as follows: Section 2 overviews the design of CloudDiag. In Section 3, we introduce our performance data collection mechanism. Section 4 illustrates how CloudDiag pinpoints the primary causes of the performance anomalies. We demonstrate the effectiveness of CloudDiag with three real-world case studies in Section 5. Section 6 discusses the related work. Section 7 provides some further discussions and concludes this paper.

2 Overview

2.1 Preliminaries

A typical production cloud (e.g.,, the data-centric cloud computing facilities offered by Alibaba Cloud Computing for Alibaba Inc.) generally offers a lot of concurrent services. Services work collaboratively to support a cloud application. For example, an e-mail application hosted in Ablibaba Cloud Computing is supported by many services that handle the e-mail-relevant operations such as sending an e-mail, loading an e-mail, and listing e-mails. From a service-oriented perspective, a service in the cloud is for handling a certain type of user requests (e.g., reading an email).

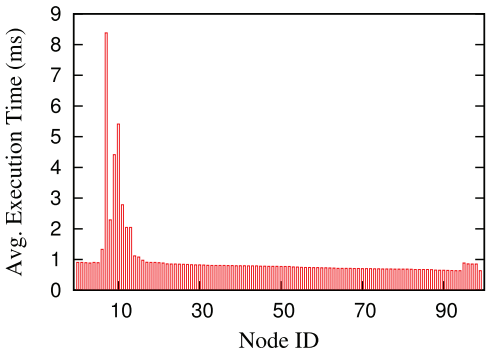


Fig. 1. The execution time of a component method in different replicas in a 100-node cloud system.

A service is typically composed of many components. Each component often contains a large number of replicas (component instances) distributed in different physical nodes in the cloud for fault-tolerance, load balancing, and elasticity considerations.

A user request to a service, therefore, may go through many component replicas, invoking numerous methods they provide. A call tree of a user request is a directed tree describing the method invocation relations, where each node is a method and each edge $e = u \rightarrow v$ from node u to node v denotes that method v is invoked by method u, i.e., u is a caller and v is its callee. Requests to the same service can generate multiple call trees. For example, the call tree of one request reading a file from the cache is different from that of another request reading a file from the disk.

2.2 Framework of CloudDiag

Performance anomalies in cloud systems will manifest themselves as anomalous response time of user requests. Since a service is composed of a lot of components, a service with anomalous performance must have involved some components with performance anomalies. A component typically has a lot of replicas in a production cloud system; however, the performance anomaly of a component may be manifested only in a small part of its replicas. This will cause the performance degradation of the involving service, which is frequently observed in the cloud computing systems of Alibaba Inc.

Fig. 1 shows the execution time of a component method in different replicas in a 100-node cloud system. We can instantly see that only a small part of replicas (e.g., nodes 8 and 12) are anomalous when executing the method.

Such performance problems are the most difficult to locate, because the anomalous methods hide themselves in numerous well-functioning replicas. Therefore, to reduce human efforts in pinpointing performance anomaly, a performance diagnosis tool must first identify which component methods contribute to the performance

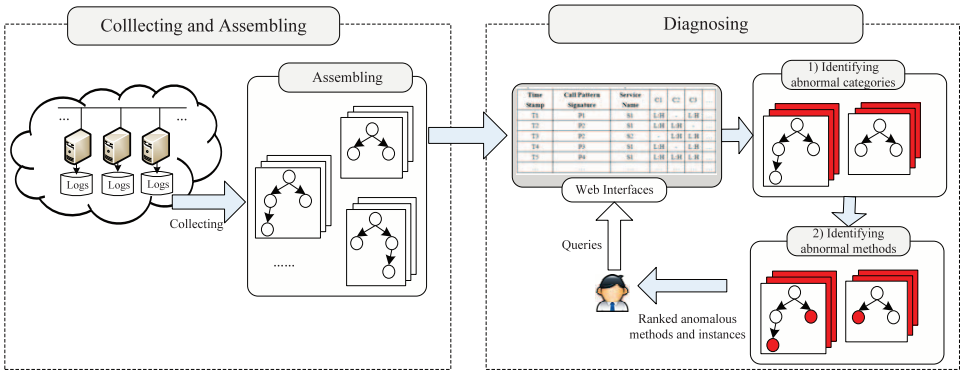


Fig. 2. A System overview of CloudDiag.

anomaly, and then locate the component replicas that execute the methods.

An efficient, unsupervised diagnosing tool that can pinpoint the fine-grained causes of performance anomalies is of critical importance to production cloud computing systems. To this end, we propose CloudDiag, a tool for performance diagnosis in production cloud computing systems. Fig. 2 provides a system-level overview of CloudDiag.

CloudDiag is composed of three major parts, i.e., 1) collecting the performance data; 2) assembling the performance data; and 3) identifying the primary causes of the anomalies. We briefly overview each part as follows:

- *Collect performance data*. CloudDiag traces user requests at a given sampling rate to expose performance data. For the sampled requests, each component replica records the performance data and saves them in its local storage. An important consideration is what kind of performance data CloudDiag should collect and how. CloudDiag adopts an instrumentation-based approach that collects the execution time of each component method. Details are discussed in Section 3.
- *Assemble performance data*. CloudDiag should first assemble the performance data distributed in numerous component replicas in a request-oriented way. In other words, the performance data belonging to the same requests are correlated together. CloudDiag will then analyze such request-oriented performance data and infer the call tree of each sampled request. A customized map-reduce process is utilized to group requests into different categories based on their call trees. Requests within one category share the same call tree.
- *Identify the primary causes of anomalies*. CloudDiag then identifies the anomalous categories according to their latency distribution. Then, for each anomalous category, a fast customized matrix recovery algorithm (i.e., robust principal component analysis (RPCA) [11]) is employed to identify the anomalous method invocations together with the replicas they are located. Details are discussed in Section 4.

Note that Steps 1 and 2 are relatively time-consuming tasks since they work on the massive tracing data generated by the entire production cloud system. Hence, for efficiency considerations, CloudDiag performs these two steps proactively. In other words, CloudDiag conducts the tracing data collection and assembly during the execution of the system. All categories are stored in a BigTable-like storage system [12] for further performance diagnosis when performance anomalies are detected.

A web-based interface to access the data is provided for system operators. When performance anomalies are observed, an operator can conduct the primary cause analysis by accessing the web interface. Step 3 is then triggered and the fine-grained results (i.e., the anomalous method invocations together with the replicas they are associated with) is automatically provided to the operator. Finally, note that such a design also allows the operator to flexibly conduct a tunable Step 3 (e.g., tune the time windows to rerun the diagnosing process) without performing the time-consuming performance data assembly step.

3 Performance Data Collection

In this section, we introduce what kind of performance data that CloudDiag should collect and how to collect them.

Our instrumentation-based tracing approach will produce performance data when a sampled request is being processed in each component replica. Specifically, each component method, when being invoked or returning, will generate a log entry.

The data structure of a tracing log entry is shown in Fig. 3a, which contains five items. *Host* indicates the machine where the component replica locates. *Time stamp* records the time of the event occurrence (i.e., a method invocation or a method return). *RequestID* is the global identifier of a request. *MID* is a unique identifier for request

Host	Timestamp	RequestID	MID	Method	Flag
Host1	2012-01-01 16:31:31.690272	169	739	AliStorge.ReadFile	Start
Host1	2012-01-01 16:31:32.991376	169	739	AliStorge.ReadFile	End

(a) Log entry for method execution time

Host	Timestamp	RequestID	Caller MID	Callee MID
Host1	2012-01-01 16:31:31.690724	169	739	991

(b) Log entry for method invocation relationship

Fig. 3. Tracing log formats and examples.

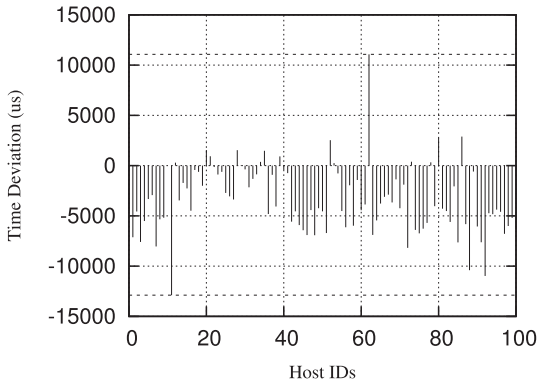


Fig. 4. The time deviation of a 100-node cloud system. The standard time in the figure is the clock of a node randomly selected.

tracing purpose, which will be discussed later. The *Method* field saves the name of the method invoked. Lastly, *Flag* indicates whether this is a method invocation or a method return. Fig. 3a also shows two example log entries that record the invocation of the *AliStorage.readFile* method and its return.

RequestID should be unique for every request. It is assigned when a request arrives the system. Typically, a cloud service may have multiple entry nodes for the same type of requests. To guarantee the uniqueness of *RequestID*, an entry node will assign the *RequestID* (a unique 64-bit integer) as the concatenation of two integers: One is the unique number to identify the entry node *per se*, and the other is incremental with each new request.

CloudDiag uses the invocation relationship between methods to model a request. To obtain such relationship, CloudDiag resorts to the chronological order of the method invocations. Our experiences show that one challenge of request tracing is to cope with the clock drifts of nodes, which are inevitable in production cloud systems. Previous approaches (e.g., [10], [13], [6], [8]) generally assume that the clock drifts are negligible. However, this is not true in production cloud computing systems. Fig. 4 shows an example of clock drifts in a 100-host cluster in Alibaba Cloud Computing. Even with a Network Time Protocol (NTP) [14] to synchronize the clocks, the deviations between the clocks of different cloud nodes are still in millisecond-level. As we can see in the figure, the biggest deviation is larger than 20 milliseconds. Previous approaches (e.g., [8], [10], [13], [6]) simply employ only a global identifier (i.e., *RequestID* in Fig. 5) to mark the distributed tracing record and use their time stamps to infer the invocation relationships with methods. As a result, clock drifts may lead to the wrong order of method invocations when merging the distributed logs generated in processing a request. For example, the start time-stamp of the callee in one host may be earlier than the start time-stamp of the caller in another host.

To guarantee the order of the invoked methods, we design hierarchical identifiers to trace a request. Specifically, before a node calls a method in another node, besides passing the *RequestID* to the callee, the caller also generates an *MID*, a unique integer, for the callee. When a request enters the first component method in the system (i.e., the request entry method), the *MID* of the method is initially set identical to the *RequestID*. CloudDiag then records the *MID*s of the caller and the callee in the logs of the caller as well. Fig. 3b shows such a log format.

Thus, CloudDiag can recover the caller-callee relationships according to the *MID*s. The order of method invocations can then be correctly recovered and an entire performance trace of a request can, thus, be obtained. Fig. 5 shows an example of combining the tracing logs distributed over three hosts and retrieving the call tree.

4 Diagnosing Anomalies without Domain Knowledge

In this section, CloudDiag first employs a statistical technique to detect anomalous categories that contain latency-anomalous requests. Then, from anomalous categories, a fast matrix recovery algorithm, namely, RPCA, is adopted to identify the anomalous methods and instances. Details are as follows:

4.1 Identifying Anomalous Categories

We cannot rely only on the response latency of a request to check whether a request is anomalous. Long response latency does not indicate a failure. For example, the response latency of one request reading a file from hard disk is several times longer than that of another reading a file from cache.

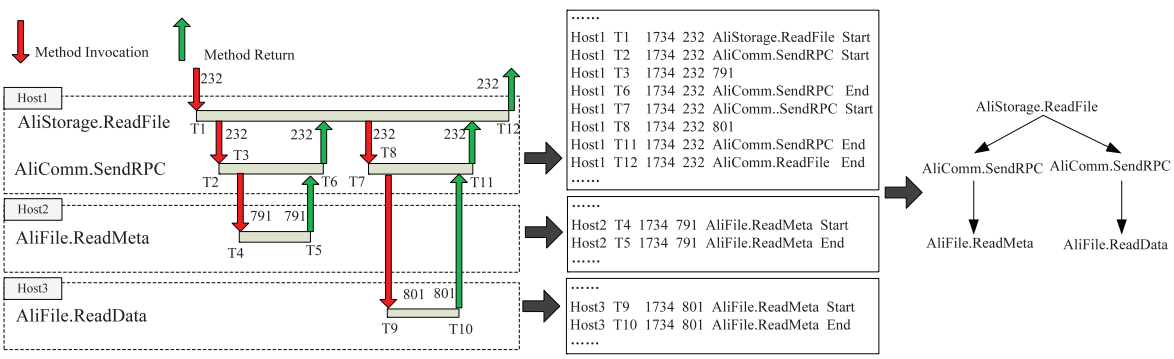


Fig. 5. A request with $requestID = 1734$ passes through three hosts. When the *SendRPC* method at Host 1 invokes the *ReadMeta* method at Host 2 and the *ReadMeta* method at Host 3, it will generate the MIDs (791 and 801) for the two callees. CloudDiag can then recover the caller-callee relationships according to the third and sixth log entries of Host 1.

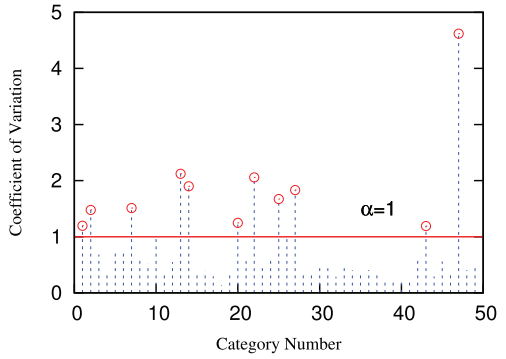


Fig. 6. An Example of detecting anomalous categories.

Normal and anomalous replicas exist simultaneously when performance problems occur. For a component, the same service requests may pass through normal instances as well as anomalous instances. The response latency of a request will be influenced by anomalous instances that it passes through. Normal and anomalous requests may share the same call tree and be grouped into one category; hence, the latency distribution of requests within the same category could be utilized to detect whether it contains latency-anomalous requests or not.

Requests within one category have the same call tree; hence, the response latencies should be close to each other. A category is considered to be normal if the latencies of requests within the category are clustered in a specific range; on the contrary, a category is considered to be anomalous if the latencies are over dispersed. In this regards, we choose the coefficient of variation (CV) [15] to measure the distribution of a set of data. Let α be the threshold. A category is defined to be anomalous if its CV is larger than α. Fig. 6 shows an example of identifying the anomalous categories (α is set to be 1). Finally, note that CV can also be easily replaced by other statistical approaches. But in our field studies, we find that such a simple indicator is good enough.

4.2 Identifying Anomalous Methods

In an anomalous category of requests, our aim now is to pinpoint the anomalous method invocations that are responsive for the performance anomaly of the requests. For such a category of requests, we can create an $m \times n$ matrix M, where n is the number of the invoked methods in the corresponding call tree and m is the number of the requests that bear the same call tree. M_{ij} denotes the execution time of the jth method when depth-first traversing the call tree of the ith request. Column $M(j)$ denotes the invocation time vector of the jth method.

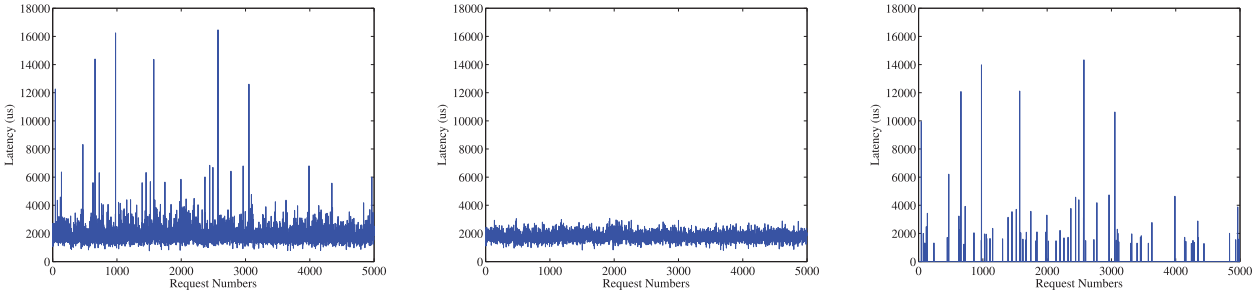
Intuitively, we can identify the anomalous method by measuring the execution time deviation of each method one by one. However, this cannot capture the correlations of the invoked methods, and will, hence, cause imprecise diagnosis results. Furthermore, such a statistical analysis can only identify anomalous methods, but cannot find out on which replicas the anomalous methods are executed. Hence, we design an unsupervised machine learning algorithm to automatically learn the characteristics of the invoked methods and identify which methods are anomalous together with on which replicas they are executed. We discuss the details as follows:

First of all, most requests with the same call tree bear similar performance data. In other words, most of the rows are correlated. As a result, the performance matrix M bears a low rank. Such a property is the basis of many existing approaches [16], [17], [18], [19] to widely employ principal component analysis (PCA) [20] in performance anomaly detection.

Although PCA is one of the most popular algorithms for anomaly detection, it only works well to the data in which the errors (in our problem domain, errors in the data are caused by the performance anomaly) follows the Gaussian distribution. However, the execution time of the anomalous methods is actually corrupted by large errors, which does not follow the Gaussian-distribution assumption of PCA. Such large errors caused by anomalous methods will cause PCA to produce imprecise diagnosing results [11].

Fig. 7a shows a column with large errors in a $5{,}000 \times 117$ matrix. The matrix is composed of the requests of an anomalous category which contains 5,000 requests. This column denotes the invocation time vector of an anomalous method. The horizontal axis represents the request identifier and the vertical axis denotes the microsecond-level execution time. For this method, the normal execution time is about 2,000 microseconds; however, parts of the execution time reach to 16,000 microseconds. These large errors cause the distribution of execution time to deviate from the Gaussian distribution.

To solve the problem, we propose to use the RPCA [11] for the anomaly detection task. RPCA is an algorithm for high-dimensional matrix completion. When a matrix M is corrupted by gross sparse errors, RPCA can decompose the



(a) Real Values (b) Recovered Values (c) Corrupted Values

Fig. 7. An illustration of RPCA recovering a column with gross errors to a noncorrupted column and a corrupted column.

full matrix M as: $M = L + E$, where L is a low-rank matrix with noncorrupted columns and E is a sparse matrix with a few nonzero corrupted columns.

The matrix M is the input of RPCA. Therefore, the problem of identifying anomalous methods in a category is transferred into the process of recovering a matrix with unknown corrupted latency columns.

After obtaining the noncorrupted matrix L and error matrix E, we can identify the corrupted columns (i.e., the anomalous methods) from E. The anomalous methods refer to those columns that are farthest from the true column space. For the ith column in original matrix M and noncorrupted matrix L, the extension of deviation can be measured as:

$$\beta = \cos\theta = \frac{\|M(i) \cdot L(i)\|_1}{\|M(i)\|_2 \|L(i)\|_2}, \quad (1)$$

where θ represents the angle between column $M(i)$ and column $L(i)$. The larger the angle is, the more deviation the column $L(i)$ is away from the true space. A method is defined to be anomalous if β is smaller than a given threshold.

Fig. 7 plots an example of identifying an anomalous method in one anomalous category. The original data of the anomalous method are shown in Fig. 7a. After applying RPCA, we can get a noncorrupted latency column (shown in Fig. 7b) and an error latency column (shown in Fig. 7c).

For each anomalous method (i.e., the corrupted column), anomalous replicas are located by checking the entries of the corrupted column in Matrix E. With the row and column indices of the corrupted entries, we can get the physical addresses of anomalous replicas from physical paths.

Since the same method (running on the same component replica) may be identified to be anomalous in different categories, we calculate the times that it is identified to be anomalous. The larger the number of times is, the more suspicious the method is. CloudDiag can then rank the methods in descending order of the number of times that they are identified to be anomalous, which can direct the operators to localize the primary cause of performance anomaly.

5 Evaluation

CloudDiag has been launched in Alibaba Cloud Computing Company to perform anomaly diagnosis in its production cloud computing systems. This section reports three case studies during our experiences in using CloudDiag in Alibaba.

Our target cloud system is a cloud facility for Aliyun Mail, a production e-mail system that provides free e-mail service to the public.[2] ListMail, ReadMail, and SendMail are three services that are utilized to handle requests of listing mail titles, reading mail contents, and sending mails, respectively. They are the typical services of our target system, and are the focus of our experimental studies. Services are composed of a series of components (e.g., storage and communication). Each component has many homogeneous replicas that are deployed on different hosts.

2. The system can be accessed via http://mail.aliyun.com.

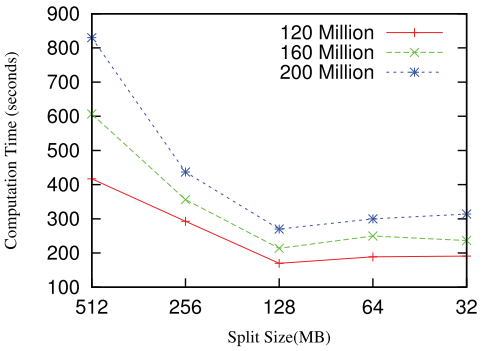


Fig. 8. Computation time under different volumes of split sizes.

Currently, more than 20 million user requests are handled per day. On average, a request will typically go through over 10 hosts, invoking over 100 instrumented methods. By default, requests are sampled with the ratio of 1/200. Generally, the target cloud system would produce about 30-50 gigabytes (around 120-200 million lines) of tracing logs per hour.

CloudDiag is deployed in a small cluster with 10 nodes. Each node is a typical low-end computer running Linux RHEL 5.4. CloudDiag proactively pulls the tracing data from the target cloud in a periodical manner (once every hour in our experiments). It then runs a customized map-reduce process to assemble and classify tracing data. First, map tasks assign correlated tracing logs that belong to the same requests to corresponding reduce tasks. Second, reduce tasks generate and classify requests into categories.

For the map-reduce cluster, one important parameter is the split size, i.e., the volume of data assigned to each *Map* task. The split size determines the number of *Map* tasks. A smaller split size indicates that more *Map* tasks are required to process a given data set. We vary the split sizes from 32 to 512 MB for three data sets (with trace entries sizes being 120, 160, and 200 million lines of trace logs). The computational time of the map-reduce procedure is shown in Fig. 8. We can see the cluster performs the best when the split size is 128 MB. Hence, in the rest of our experiments, we set the split size 128 MB.

Finally, CloudDiag adopts a state-of-the-art RPCA implementation, called inexact ALM algorithm [21] in its performance anomaly detection approach. It is a mature open-source implementation, and can be used in a black-box way for CloudDiag.

5.1 Scalability Evaluation

For efficiency consideration, CloudDiag is required to be scalable to the massive performance data. Since CloudDiag conducts the tracing data collection and assembly proactively, the anomaly diagnosing step is the only issue that will influence the scalability of CloudDiag.

We study the efficiency of the RPCA-based anomaly detection approach. The inputs are the performance data of a typical category of requests to the SendMail service, which bears a critical call tree that contains 117 methods. There are about 4 million of requests following this call tree each day. Fig. 9 plots the computation time of the anomaly detection approach under different request numbers. It shows that the computational time of the approach also

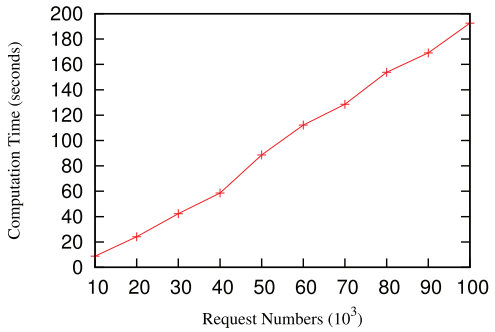


Fig. 9. Scalability of the RPCA based anomaly detection.

scales almost linearly with the performance data volumes of up to 100 thousand requests. This demonstrates the high scalability of the RPCA-based anomaly detection algorithm. The process of computing a $100{,}000 \times 117$ matrix takes less than 200 seconds.

5.2 Evaluation of Diagnosing Results

In this section, we demonstrate how CloudDiag helps operators detect real-world performance anomalies that happened in Alibaba cloud computing platform.

We adopt the following two measures to evaluate the effectiveness of CloudDiag. The first is $precision = \frac{TP}{TP+FP}$, which measures the exactness of our approach. The second is $recall = \frac{TP}{TP+FN}$, which measures the completeness. TP refers to the number of true positives (i.e., the number of anomalous methods); FP refers to the number of false positives (i.e., the number of normal methods that are mistaken for the anomalous); FN refers to the number of false negatives (i.e., the number of anomalous methods that are mistaken for the normal).

To show the advantage of adopting RPCA in CloudDiag, we compare CloudDiag with a recent performance diagnosis approach based on the PCA algorithm [19]. The approach employs PCA and the Mann-Whitney hypothesis test to identify anomalous methods.

5.2.1 Case 1: Misconfiguration of Thread Pool Size

Performance anomalies due to misconfigurations are frequently encountered in large scale systems like a production cloud. But they are very difficult to be troubleshooted [22]. It is hard to correlate the performance degradation to the relevant misconfigured parameters. Even with an approach

Fig. 10. The average latency of ListMail service increases nearly by 1/2 in about 3 hours.

TABLE 1
Summary of Diagnosing Results with CloudDiag

Case	Candidate categories	Anomalous categories	Request ratio	Anomalous req. ratio	Original avg. rank	Recovered avg. rank
1	98	23	47%	9%	129	68
2	87	30	53%	12%	117	56
3	137	27	11%	1%	87	51

that can identify anomalous physical nodes [2] or logical components [3], the operators still have to manually infer the correlation between the performance anomalies and the parameters, which is time consuming and error prone. This case study reports how CloudDiag can greatly help troubleshoot a misconfiguration of the thread pool size, which causes the performance degradation.

After a series of software upgrade, the cloud operators found the performance of ListMail service decreased nearly by 1/2 in about 3 hours, as shown in Fig. 10. CloudDiag quickly finds that the method with the highest suspicious score was one relevant to the queuing time before a request is handled by the storage service. The queuing time is long because the requests are waiting for available handlers, while all the handlers are busy processing requests. We can instantly know that this is because the number of handlers (implemented as threads) are set too small. This number is directly controlled by a system-wide configuration file. After setting the configuration of thread pool size to a more reasonable number, the system is cured.

In Table 1, we can see that about one-fourth of categories become anomalous. The number of requests in these anomalous categories accounts for 47 percent of total ListMail requests and about 9 percent requests are directly impacted by the performance anomaly. On average these anomalous categories pass through 129 invoked methods (some are invoked many times in one request.). The average rank of original and recovered matrices are 129 and 68, respectively, which confirms the low dimensionality of original categories.

The comparisons between CloudDiag and the PCA-based approach are shown in the first row of Table 2. We can see that the precision of the PCA-based approach is only 67 percent. It indicates that the PCA-based approach has mistaken more normal methods for the anomalous. More false positives require more human effort to examine the methods that are actually correct. Furthermore, the recall of the PCA-based approach is 8 percent lower than that of CloudDiag, which means that fewer anomalous methods can be successfully identified by the PCA-based

TABLE 2
Comparison of CloudDiag and PCA-Based Approach

Case	CloudDiag		PCA-based approach	
	Precision	Recall	Precision	Recall
1	95%	93%	67%	85%
2	98%	96%	71%	89%
3	97%	91%	76%	81%

approach. Hence, it requires much additional human effort in diagnosing the defect.

Note that in Fig. 10, we can observe that the performance of SendMail and ReadMail services are not influenced by configuration as much as that of the ListMail service. Without conducting a separation-of-concerns analysis, the diagnosing result will be disturbed by the performance data of SendMail and ReadMail services.

5.2.2 Case 2: Performance Bottleneck Caused by Random Number Generator

Sometimes minor design defects will become the performance bottleneck of the entire systems. Not like fail-stop faults that will directly cause the breakdown of the system, these subtle design defects that will, however, cause severe performance degradation will not manifest until systems are under specific conditions, which increases the complexity of detection. This case study reports how CloudDiag helps operators identify a minor design defect.

After the load was increased by 300 percent, the average latencies of SendMail service rose by 35 percent in about 6 hours. Operators spent much time on checking the design logic in the server and performance-relevant parameters; however, nothing wrong was detected. Hence, CloudDiag was then recommended to locate the primary cause of this performance degradation.

The methods with the highest suspicious scores were the front-end related ones. The most anomalous method contains a process of waiting for a random number. Every new mail was tagged with a unique string, which is obtained with a random number generator */dev/random*. It generates random numbers through collecting the environmental noises from hardware devices, which inevitably suffers poor efficiency. When load increases, some front-end nodes cannot generate random numbers timely. It causes a considerable portion of the SendMail requests to spend much more time in the frond-end nodes.

CloudDiag is effective to identify these fine-grained defects that are not easily noticed. We can see that without CloudDiag, even experts will have much difficulty in locating the cause of such a performance defect. Furthermore, massive performance data is generated in 6 hours, without CloudDiag proactively assembling the data, operators could not efficiently conduct the diagnosing process. They need to wait extra time for the diagnosing result.

The quantitative results are shown in the second row of Table 1. In this service, about one-third of the categories become anomalous and 12 percent of the requests are influenced. Table 2 shows that the precision of CloudDiag is 98 percent (27 percent higher than that of the PCA-based approach), which again shows the advantage of CloudDiag in eliminating the false positives.

5.2.3 Case 3: Periodic Request Bursts

Testing cannot cover all possible occasions in production systems given the complexity of the real world, some performance anomalies triggered by user access behaviors are inevitable. This case study shows how CloudDiag can help operators conveniently diagnose these performance anomalies.

In Alibaba, the Service Level Agreement (SLA) for the execution time of SendMail service was that the fraction of requests with response latencies larger than 300 milliseconds could not exceed 0.05 percent. However, on every Wednesday and Thursday, the ratio rose to 1 percent and this situation lasted for several weeks.

To debug this anomaly, we ran CloudDiag for the two time windows, i.e., Wednesday and Thursday. Both results revealed that the most anomalous methods were related to the transaction begin operations. Through checking the physical addresses of the anomalous methods, we found that most of them were just located in a few replicas. In the mail service, mails of the same user account would be sequentially stored by one storage component replica. Hence, we inferred that some user accounts in this replica might be called to receive massive mails simultaneously.

Then, we presented the results to one operator. After checking the replica, he confirmed our judgment. Two user accounts belonging to the security department of Alibaba cloud company would receive thousands of mails from 3:00 PM to 5:00 PM on every Wednesday and Thursday, which caused a fraction of SendMail service to wait more time for disk access.

When performance anomalies occur, it is hard for operators to efficiently identify which teams should be responsible for these anomalies. With CloudDiag, they can quickly identify related causes and assign the bug reports to the "culprits."

The quantitative results are shown in the third row of Table 1. We can see that about one-sixth of categories become anomalous. These anomalous categories account for 11 percent of total SendMail requests. On average these anomalous categories invoke 87 instrumented methods. As shown in Table 2, the precision and recall are 97 and 91 percent, respectively, which again demonstrates the effectiveness of CloudDiag.

5.2.4 Discussions

In the above three case studies, we compare CloudDiag with the PCA-based approach in terms of precision and recall. From Table 2, we can see that CloudDiag outperforms the PCA-based approach in both measures, especially in term of precision. Hence, CloudDiag can save more effort in troubleshooting the primary cause of the performance anomalies. This shows that the PCA-based approach cannot well handle the performance data with gross errors. It consequently generates more false positives and false negatives. The RPCA-based approach CloudDiag, on the other hand, can work well for such non-Gaussian performance data.

There are two thresholds in our approach. α and β are, respectively, utilized to detect whether categories or invoked methods are anomalous or not. α is conventionally set to 1 [15]. In our experimental study, we have set β in range [0.4, 0.8]. Although anomalous methods can be successfully identified by CloudDiag in our three case studies, we observe that the value of beta can slightly influence the precision. Empirically, the value 0.5 is a best choice for β.

6 Related Work

Extensive work has employed explicit annotation-based instrumentation to conduct performance monitoring,

tuning, and debugging for distributed systems, which is surveyed as follows:

Magpie [23] applies application-specific event schemas to correlate the resource consumption of individual requests with the goal of understanding performance. Pip [4] and Ironmodel [24] compare users' actual behavior with self-defined expectation to determine whether a request is anomalous or not. It is very hard to construct these models because they require much specific domain knowledge. Compared to them, CloudDiag considers the intrinsic characteristics of request latencies to determine the anomalous method invocations, which requires no specific domain knowledge.

Pinpoint [13] traces request call relationship in multi-layers of web service components and adopts a clustering algorithm to group failure and success logs. It can only find out the anomalous components. In comparison, CloudDiag can identify the latency-anomalous methods together with corresponding physical replicas.

Dapper [25] introduces an infrastructure of performance monitoring. It keeps tracing logs into Bigtable [12]. Yet this approach does not describe how to use these logs to diagnose performance problems. Spectroscope [7] aims to find the primary cause of performance changes between two time periods, while our work does not assume the existence of a "correct" time period when the system performs normally. P-Tracer [26] can be utilized to identify the performance anomalies that manifest themselves as the change in the ratios of the chosen call trees, while CloudDiag can localize the latency-anomalous methods within call trees.

There are also many performance diagnosing techniques that rely on specific types of data. Xu et al. [27] combine console logs with source codes to construct performance features and conducts the PCA [20] to detect problems. Similarly, Nagaraj et al. [28] model event logs and state logs into performance features and engage the t-statistic to infer the anomalies between components. However, these techniques generally focus on locating anomalous logical components instead of replicas.

7 Discussion and Conclusion

Request tracing technologies have been proven effective in performance debugging. In this paper, CloudDiag resorts to a white-box instrumentation mechanism to trace service requests, because the source codes of services are generally available in typical production cloud systems. Note that such a white-box performance data acquisition component of CloudDiag can also be substituted with another tracing mechanism if it can obtain the latency data of method invocations.

Another way to trace requests is via black-box mechanisms. Black-box tracing mechanisms (e.g., [29]) assume no knowledge of the source codes. But, existing approaches generally cannot directly obtain the latency data of method invocations. In this regard, a black-box tracing mechanism can be deemed as a tracing mechanism with the large granularity (e.g., in node level). There is a tradeoff between tracing granularity and debugging effort. As a result, more effort will be increased in troubleshooting the performance anomalies if a black-box tracing mechanism is applied. To incorporate black-box tracing mechanisms with CloudDiag, a future direction is to explore black-box tracing mechanisms so that a fine granularity (i.e., in method invocation level) can be achieved. To this end, the runtime instrumentation (e.g., [30]) can be a promising technique.

In conclusion, this paper proposes CloudDiag, an efficient, unsupervised diagnosis tool for locating fine-grained performance anomalies. The experimental results demonstrate that our approach scales well to massive tracing data. We also report our experiences that CloudDiag can effectively and conveniently help operators diagnose three real-world performance problems with high precision and recall.

Acknowledgments

This research is supported by the National Basic Research Program of China under Grant No. 2011CB302600, the National High Technology Research and Development Program of China under Grant No. 2012AA011201, the National Natural Science Foundation of China (NSFC) under Grant No. 90818028, 61161160565, 60903043, 61100077, the Basic Research Program of Shenzhen (Grant No. JCYJ20120619152636275 and JC201104220300A), and the Research Grants Council of Hong Kong (Project No. N_CUHK405/11).

References

[1] V. Emeakaroha, M. Netto, R. Calheiros, I. Brandic, R. Buyya, and C. De Rose, "Towards Autonomic Detection of SLA Violations in Cloud Infrastructures," *Future Generation Computer Systems,* vol. 28, pp. 1017-1029, 2011.

[2] Z. Lan, Z. Zheng, and Y. Li, "Toward Automated Anomaly Identification in Large-Scale Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 21, no. 2, pp. 174-187, Feb. 2010.

[3] H. Malik, B. Adams, and A. Hassan, "Pinpointing the Subsystems Responsible for the Performance Deviations in a Load Test," *Proc. IEEE 21st Int'l Symp. Software Reliability Eng. (ISSRE),* pp. 201-210, 2010.

[4] P. Reynolds, C. Killian, J. Wiener, J. Mogul, M. Shah, and A. Vahdat, "Pip: Detecting the Unexpected in Distributed Systems," *Proc. USENIX Third Symp. Networked Systems Design and Implementation (NSDI),* pp. 115-128, 2006.

[5] E. Thereska and G. Ganger, "Ironmodel: Robust Performance Models in the Wild," *ACM SIGMETRICS Performance Evaluation Rev.,* vol. 36, no. 1, pp. 253-264, 2008.

[6] E. Thereska, B. Salmon, J. Strunk, M. Wachs, M. Abd-El-Malek, J. Lopez, and G. Ganger, "Stardust: Tracking Activity in a Distributed Storage System," *ACM SIGMETRICS Performance Evaluation Rev.,* vol. 34, no. 1, pp. 3-14, 2006.

[7] R. Sambasivan, A. Zheng, M. De Rosa, E. Krevat, S. Whitman, M. Stroucken, W. Wang, L. Xu, and G. Ganger, "Diagnosing Performance Changes by Comparing Request Flows," *Proc. USENIX Eighth Symp. Networked Systems Design and Implementation (NSDI),* pp. 43-56, 2011.

[8] A. Chanda, A. Cox, and W. Zwaenepoel, "Whodunit: Transactional Profiling For Multi-Tier Applications," *ACM SIGOPS Operating Systems Rev.,* vol. 41, no. 3, pp. 17-30, 2007.

[9] R. Fonseca, G. Porter, R. Katz, S. Shenker, and I. Stoica, "X-Trace: A Pervasive Network Tracing Framework," *Proc. USENIX Third Symp. Networked Systems Design and Implementation (NSDI),* pp. 271-284, 2007.

[10] M. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer, "Path-Based Faliure and Evolution Management," *Proc. USENIX Symp. Networked Systems Design and Implementation (NSDI),* pp. 23-36, 2004.

[11] E. Candes, X. Li, Y. Ma, and J. Wright, "Robust Principal Component Analysis?" *Arxiv Preprint arXiv:0912.3599,* 2009.

[12] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A Distributed Storage System for Structured Data," *ACM Trans. Computer Systems,* vol. 26, no. 2, pp. 1-26, 2008.
[13] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem Determination in Large, Dynamic Internet Services," *Proc. IEEE Int'l Conf. Dependable Systems and Networks (DSN),* pp. 595-604, 2002.
[14] D. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis," 1992.
[15] H. Abdi, "Coefficient of Variation," *Encyclopedia of Research Design,* N. Salkind, ed., pp. 1-5, SAGE, 2010.
[16] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Online System Problem Detection by Mining Patterns of Console Logs," *Proc. IEEE Int'l Conf. Data Mining (ICDM),* pp. 588-597, 2009.
[17] A. Oliner and A. Aiken, "Online Detection of Multi-Component Interactions in Production Systems," *Proc. IEEE/IFIP 41st Int'l Conf. Dependable Systems and Networks (DSN),* pp. 49-60, 2011.
[18] H. Ringberg, A. Soule, J. Rexford, and C. Diot, "Sensitivity of PCA for Traffic Anomaly Detection," *ACM SIGMETRICS Performance Evaluation Rev.,* vol. 35, no. 1, pp. 109-120, 2007.
[19] H. Mi, H. Wang, G. Yin, H. Cai, Q. Zhou, and T. Sun, "Performance Problems Diagnosis in Cloud Computing Systems by Mining Request Trace Logs," *Proc. IEEE Network Operations and Management Symp. (NOMS),* pp. 893-899, 2012.
[20] I. Jolliffe, *Principal Component Analysis.* Springer, 2002.
[21] Z. Lin, M. Chen, L. Wu, and Y. Ma, "The Augmented Lagrange Multiplier Method for Exact Recovery of Corrupted Low-Rank Matrices," *Arxiv preprint arXiv:1009.5055,* 2010.
[22] K. Nagaraja, F. Oliveira, R. Bianchini, R. Martin, and T. Nguyen, "Understanding and Dealing with Operator Mistakes in Internet Services," *Proc. USENIX Sixth Conf. Symp. Operating Systems Design and Implementation (OSDI),* pp. 5-20, 2004.
[23] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using Magpie for Request Extraction and Workload Modelling," *Proc. USENIX Sixth Conf. Symp. Operating Systems Design and Implementation (OSDI),* pp. 259-272, 2004.
[24] E. Thereska and G. Ganger, "Ironmodel: Robust Performance Models in the Wild," *ACM SIGMETRICS Performance Evaluation Rev.,* vol. 36, no. 1, pp. 253-264, 2008.
[25] B. Sigelman, L. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, "Dapper, a Large-Scale Distributed Systems Tracing Infrastructure," Technical Report dapper-2010-1, Google, 2010.
[26] H. Mi, H. Wang, Y. Zhou, M.R. Lyu, and H. Cai, "P-tracer: Path-Base Performance Profiling in Cloud Computing Systems," *Proc. IEEE 36th Ann. Computer Software Applications Conf. (COMPSAC),* pp. 509-514, 2012.
[27] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Detecting Large-Scale System Problems by Mining Console Logs," *Proc. ACM SIGOPS 22nd Symp. Operating Systems Principles,* pp. 117-132, 2009.
[28] K. Nagaraj, C. Killian, and J. Neville, "Structured Comparative Analysis of Systems Logs to Diagnose Performance Problems," *Proc. USENIX Ninth Conf. Networked Systems Design and Implementation (NSDI),* pp. 26-39, 2012.
[29] B. Sang, J. Zhan, G. Lu, H. Wang, D. Xu, L. Wang, and Z. Zhang, "Precise, Scalable, and Online Request Tracing for Multi-Tier Services of Black Boxes," *IEEE Trans. Parallel and Distributed Systems,* no. 99, pp. 1-16, 2010.
[30] E. Gonina et al., "Fay: Extensible Distributed Tracing From Kernels to Clusters," *Proc. ACM 23rd ACM Symp. Operating Systems Principles (SOSP),* vol. 13, pp. 5-20, 2011.

Haibo Mi received the BEng and MEng degrees from Communication Command University of Wu Han, in 2005 and 2008, respectively, and is currently working toward the PhD degree in National Laboratory for Parallel & Distributed Processing, National University of Defense Technology, Changsha, China. His thesis focuses on performance maintenance in large-scale distributed systems. He was a engineer at Alibaba Cloud Computing Company for two years. His research interests include distributed computing, cloud computing, performance monitoring, and fault localization. He is a student member of the IEEE.

Huaimin Wang received the PhD degree in computer science from the National University of Defense Technology (NUDT) in 1992. He is currently a professor and the chief engineer in department of educational affairs, NUDT. He has been awarded the "Chang Jiang Scholars Program" professor by Ministry of Education of China, and the Distinct Young Scholar by the National Natural Science Foundation of China (NSFC), and so on. He has worked as the director of several grand research projects and has published more than 100 research papers in international conferences and journals. His current research interests include middleware, software agent, trustworthy computing. He is a member of the IEEE.

Yangfan Zhou received the BSc degree from Peking University in 2000, and the MPhil and PhD degrees from CUHK in 2006 and 2009, respectively. He is currently a research staff member with the Shenzhen Research Institute, The Chinese University of Hong Kong (CUHK). His research interests include distributed computing and networking, particularly in the Internet of Things, cloud computing, and sensor networks. His current research is on their software engineering issues (e.g., fault management, fault tolerance, reliability engineering, testing, and debugging) and their applications. Before joining CUHK, for many years he has been working as an engineer in information technology industry, where he is now also active in technology consulting. He is a member of the IEEE.

Michael Rung-Tsong Lyu received the PhD degree in computer science from the University of California, Los Angeles, in 1988. He is currently a professor in the Department of Computer Science & Engineering. He initiated the First International Symposium on Software Reliability Engineering (ISSRE) in 1990. He was the program chair for ISSRE 1996, the general chair for ISSRE 2001, the program cochair for PRDC 1999, WWW 2010, SRDS 2005, and ICEBE 2007, the general co-chair for PRDC 2005, and a program committee member for many other conferences. His research interests include software reliability engineering, distributed systems, fault-tolerant computing, data mining, and machine learning. He has published more than 400 refereed journal and conference papers in these areas. He received IEEE Reliability Society 2010 Engineer of the Year Award. He is fellow of the IEEE and AAAS.

Hua Cai received the BS degree from the Shanghai Jiaotong University, Shanghai, China, in 1999, and the PhD degree from the Hong Kong University of Science and Technology (HKUST) in 2003, all in electrical and electronic engineering. He joined Microsoft Research Asia, Beijing, China, in December 2003, and was an associate researcher in the Media Communication Group. He is currently a senior expert in Alibaba Cloud Computing Company and leads the teams of cloud monitoring and storage. His research interests include distributed system, cloud computing, digital image/video signal processing, image/video coding and transmission, multiview video coding and transmission, and mobile media computing. He is a member of the IEEE and ACM.