Root Cause Analysis of Anomalies of Multitier Services in Public Clouds

Jianping Weng, Jessie Hui Wang, Jiahai Yang, *Member, IEEE, ACM*, and Yang Yang

***Abstract*— Anomalies of multitier services of one tenant running in cloud platform can be caused by the tenant's own components or performance interference from other tenants. If the performance of a multitier service degrades, we need to find out the root causes precisely to recover the service as soon as possible. In this paper, we argue that the cloud providers are in a better position than the tenants to solve this problem, and the solution should be non-intrusive to tenants' services or applications. Based on these two considerations, we propose a solution for cloud providers to help tenants to localize root causes of any anomaly. With the help of our solution, cloud operators can find out root causes of any anomaly no matter the root causes are in the same tenant as the anomaly or from other tenants. Particularly, we elaborate a non-intrusive method to capture the dependency relationships of components, which improves the feasibility. During localization, we exploit measurement data of both application layer and underlay infrastructure, and our two-step localization algorithm also includes a random walk procedure to model anomaly propagation probability. These techniques improve the accuracy of our root causes localization. Our small-scale real-world experiments and large-scale simulation experiments show a 15%–71% improvement in mean average precision compared with the current methods in different scenarios.**

***Index Terms*— Root cause analysis, multitier services, public cloud, performance interference, anomaly propagation graph.**

I. Introduction

NOWADAYS, more and more multitier services or cloud applications are adopting IaaS clouds to manage their service infrastructures. These services often consist of hundreds of software components spread across multiple machines. For a specific request, software components need to communicate with each other and work coordinately to serve it. Ideally, virtualization technology gives tenants the illusion of dedicated hardware access and providing strong isolation between VMs, so they cannot interfere with one another. Unfortunately, IaaS clouds allow multiple tenants to share a common physical computing infrastructure in a cost-effective way and guest VMs will contend for the shared resources, which can result in performance interference [1]–[3] between tenants. So, multitier services running inside an IaaS cloud are prone to performance anomalies due to not only software bugs of multitier services components but also performance interference between tenants. If the performance of a multitier service degrades, it is difficult and time-consuming to find out the actual root causes.

In recent years, researchers have proposed many solutions to solve this problem. Some researchers try to find out root causes of anomalies in a dedicated environment [4]–[8]. These solutions assume there is no outside influence that can interfere with resource utilization of services. Specifically, [4]–[7] only consider software bugs of multitier services components and [8] only considers performance interference between components of tenant's own service. So they cannot work well in public cloud environment. Reference [9] tries to take performance interference between different tenants into consideration. However, it only uses resource utilization of infrastructure, which may not be anomalous during some anomalies of multitier service. Therefore, root causes of some anomalies can not be localized. What's more, the authors rank possible root causes based on anomaly propagation distance, and ignore the influence of anomaly propagation probability. It decreases the accuracy of their analysis results.

Obviously, performance interference is caused by underlay resource contention, and only cloud providers can have the information about underlay resource contention among different tenants. So cloud providers are in a better position to diagnose anomalies caused by performance interference among different tenants. In previous works [4]–[8], cloud providers need to modify the application code or know dependency relationships of tenants' service components. This is not easy or even infeasible, because the dependency relationships among multitier services components are very complicated, and tenants would not like to disclose details of their services. So cloud providers need to find a way to infer these information from their own monitoring.

In this paper, we propose a solution for a public cloud provider to help its tenants to localize the root causes of anomalies of multitier services. Our solution can find out root causes no matter they are in the same tenant as the anomaly or from other tenants, and the solution is non-intrusive to tenants' services. Our solution consists of two parts: a data collection subsystem and a root cause localization subsystem. The data collection subsystem is non-intrusive to tenants and it runs continuously to capture the dependency relationships of components of multitier service and collect necessary metrics data of services. The root cause localization subsystem works when an anomaly occurs, and it is responsible to find out a list of possible root causes of the anomaly and the corresponding probability of each possible root cause. To determine the root causes, we define a metrics called similarity score, which is calculated based on both the metrics data of services and resource utilization of underlay infrastructure. We also

Manuscript received June 29, 2017; revised December 15, 2017 and March 7, 2018; accepted May 24, 2018; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor C. W. Tan. Date of publication June 19, 2018; date of current version August 16, 2018. This work was supported in part by the National Natural Science Foundation of China under Grant 61202356 and in part by the National Key Research and Development Program of China under Grant 2016YFB0801302. *(Corresponding author: Jessie Hui Wang.)*

The authors are with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China (e-mail: jessiewang@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TNET.2018.2843805

1063-6692 © 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

implement a random walk algorithm, which simulates the influence of anomaly propagations in multitier services to improve the accuracy of our solution.

We implement and deploy the system in our real-world small-scale cloud platform and conduct experiments using a three-tier web application and a Storm [34] application to demonstrate that our method can localize root causes at both VM and process level. We also conduct simulation experiments for anomalies that arise from multiple causes. We also show the rationality and necessity of two steps in our localization algorithm: similarity score and random walk propagation. Experimental results show a 15%-71% improvement in mean average precision compared to current methods in different scenarios.

Summarily, we make the following contributions in this paper:

- We propose a solution to help cloud operators accurately localize root causes for any anomaly no matter the root causes are in the same tenant as the anomaly or from other tenants.
- We propose a non-intrusive method to capture the complex dependency relationships of multitier components, which improves the feasibility of our root cause localization system.
- Our solution is able to localize root causes at both VM and process level, and can find out root causes even when the anomaly is resulting from multiple causes.
- We design a two-step localization algorithm based on monitoring of both application layer and underlay infrastructure and a random walk procedure. Experiments demonstrate the algorithm outperforms previous works.

The remaining part of the paper is organized as follows. Section II presents an overview of previous related work. Section III introduces two types of anomaly propagation in public cloud by examples and shows our proposed system architecture. Section IV illustrates how to find out all possible root causes. Section V shows the details of our two-step localization algorithm to determine the corresponding probability of each possible root cause. Section VI introduces our real-world experiments and simulation experiments and evaluation results. Section VII shows an experiment to do root cause analysis on Storm applications at the process level. Section VIII concludes our work.

II. Related Work

In recent years, many solutions have been proposed to solve this problem from the aspect of tenants [4]–[7]. These works need to modify the application code or need to know dependency relationships of service components. For example, Pivot Tracing [7] needs dynamic instrumentation with a novel "happened-before join" operator to identify the root causes of distributed system anomalies.

In addition, these solutions work well only when there is no outside influence that can interfere with resource utilization of services. However, more and more multitier services are deployed in public clouds, where different tenants may interfere with each other due to resource contention. The phenomenon of performance interference has been studied in [1]–[3], but they focus on designing resource allocation algorithm to avoid the effect as well as possible.

In 2013, Kim *et al.* [8] introduces a pseudo-anomaly clustering algorithm on historical data to capture the external factors such as performance interferences between components of tenant's own service, but performance interferences between tenants have not been considered. Therefore, it still cannot work well in public clouds with many tenants.

In 2016, Lin *et al.* [9] proposes a solution that captures anomaly propagation among different tenants. As far as we know, it is the first paper that tries to solve this problem. But the work by Lin only collects resource utilization of infrastructure and simply uses anomaly propagation distance to rank the possible root causes and do not consider the probability of anomaly propagation between components of multitier service, so the result is not accurate.

In our previous work [27], we have proposed a solution to calculate the similarity using metrics data of different levels and run random walk algorithm to determine the root causes. This paper extends our previous work by describing how to collect metrics data of different levels non-intrusively in detail and applying our method to localize root causes at process level. We also conduct large-scale experiments on more complicated scenarios, and demonstrate the ability to deal with anomalies resulting from multiple causes.

As anomalies propagate among system components along the path of the service call, request tracing of multitier services is necessary for root cause analysis of anomalies. Currently there are three ways to do request tracing. The first way is to leverage application-specific knowledge or explicitly declare causal relationships among events of different components [11]–[13]. Its drawback is that users must obtain and modify source code of target applications or middleware. Thus, this approach cannot be used for services of black boxes. For example, it cannot be used by cloud providers since tenants would not like to disclose details of their services. The second way is vPath [26] which can do request tracing out-of-vm. However, vPath is designed in XEN based virtualization environment and is not a portable request tracing tool in KVM based virtualization environment. The third way is to infer the causal paths through kernel instrumentation or traffic monitoring inside the VM without the knowledge of source code [14]–[16]. In this paper, our request traces are inferred from causal path graphs produced by PreciseTracer [16], which falls into the third category. We do not require tenants to disclose any information of their service components, which makes our solution more feasible to be deployed.

Rumor source detection in social networks also aims to find out root causes of abnormal phenomenons from networks [17]–[20]. In these seminal works, the authors design algorithms to solve various issues to detect single or multiple rumor sources in networks of different shapes, *e.g.*, regular trees and general graphs. However, the edge connections in social networks are deterministic and time is an important dimension in their Rumor Spreading Model. While in our scenario, the edge connections between VMs are hard to determine and our model of anomaly propagation between VMs does not take time as one dimension, since the anomaly propagation is almost real-time. Whether an anomaly propagates from one node to the other node is affected by other factors, such as whether there is severe resource contention between them. So the methods proposed in these works cannot be used to solve our problem directly.

In recent years, as clouds are widely used for various services, performance anomaly diagnosis of applications in cloud environment has drawn attentions of many researchers. Dean *et al.* [21], [22] propose PerfCompass, a non-intrusive

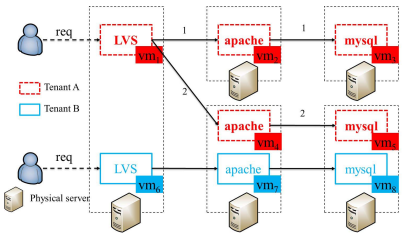


Fig. 1. Two tenants' multitier services in a cloud.

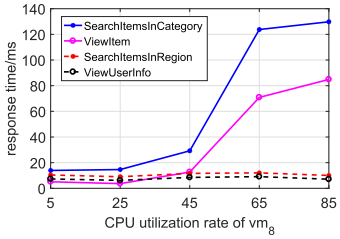


Fig. 2. Response time of tenant A's website.

performance anomaly fault localization tool using system call trace techniques. They also consider two types of anomaly propagations, *i.e.*, external and internal. The external factor is resource contention among co-located VMs, and the internal factor is software bugs. However, PerfCompass is designed for single-node applications. Goel *et al.* [23] and Sharma *et al.* [24] diagnose faults in OpenStack by monitoring network communication and analyzing event sequences. Mi *et al.* [25] use a statistical technique and a fast matrix recovery algorithm to diagnose faults of Aliyun Mail. However, these works are intrusive and tightly coupled with the specific applications they studied. So these works cannot complete root cause analysis for multitier services running on multiple nodes connected via network.

III. Types of Anomaly Propagation and System Architecture

Intuitively, there are two types of factors that can cause performance degradations of a multitier service, namely internal factor and external factor. Internally, anomaly may propagate among components within the multitier service along the path of service call. Externally, anomaly may propagate to one VM of the service from VMs of other tenants because they compete for resources of a same physical server. In this section, we would conduct experiments to demonstrate how these factors interfere with the service performance. Then we summarize the types of anomaly propagation and based on our understanding, we design the architecture of our system to localize root causes of anomalies in multitier services.

A. Two Types of Anomaly Propagation in Public Cloud

We conduct experiments in a small-scale cloud shown in Figure 1. The cloud consists of 5 physical machines. Let us assume two tenants, *e.g.* T_A and T_B, apply to the cloud provider for cloud services. T_A requests for 5 VMs, and T_B requests for 3 VMs. After receiving requests, the cloud provider creates VMs for these two tenants. Obviously, some virtual machines would be allocated in a same physical machine.

In our experiments, we assume the allocation result is that (vm_1, vm_6), (vm_4, vm_7), and (vm_5, vm_8) are VM pairs that are co-located in a same physical machine. We bind all VMs on a physical machine to a same physical CPU core, in order to make sure these VMs are competing for resources of the physical machine. We further assume both tenants are using their virtual machines to run the 3-tiered web application RUBiS [10]. RUBiS is an e-Commerce benchmark developed for academic research. It implements an online auction site loosely modeled after eBay, and adopts a 3-tiered architecture which consists of a frontend tier (a LVS service), a middle tier (apache services) and a data tier (mysql services).

As a software toolset for academic research, RUBiS can create automatically users and commodity items to initialize the simulation of an online shopping website according to researchers' parameter settings. In our experiments, we generate 100,000 users and 100,000 commodity items. RUBiS also provides a client that can emulate user behavior of sending requests. In addition, the client is able to collect statistical results of the response time of users' requests.

T_A's LVS (vm_1) receives users' requests (emulated by the RUBiS client), and further directs requests to one of the two apaches (vm_2 and vm_4) according to the content of requests. In other words, vm_1 implements a task-based load balance. We set the load balance policy as follows: requests with *SearchItemsByRegion* function and *ViewUserInfo* function (denoted by R_1) are served by vm_2, and requests with *SearchItemsByCategory* function and *ViewItem* function (denoted by R_2) are served by vm_4. Then vm_2 depends on vm_3, and vm_4 depends on vm_5, to get necessary information from mysql database to join and decorate results for users' requests. In summary, we can see in our experiments T_A have two call paths, *i.e.*, P_1 ($vm_1 \rightarrow vm_2 \rightarrow vm_3$) and P_2 ($vm_1 \rightarrow vm_4 \rightarrow vm_5$) as labeled in Figure 1, to handle users' requests.

At the beginning of our experiment, every virtual machine of both tenants works well. Then we try to increase the utilization rate of vm_8 (of T_B) gradually and measure if this increase can degrade the quality of service of T_A. We control CPU load by using a tool called *cpu-load-generator* [33], which is implemented based on the well-known tool *lookbusy* [31].

As we increase vm_8's CPU utilization rate, we keep track of T_A's performance. We evaluate T_A's performance by its response time to users' requests of those four functions. We plot the average response time of requests of each function under different vm_8's CPU utilization rate in Figure 2. Obviously, we can see that after vm_8's CPU utilization rate is greater than a certain threshold, *i.e.* 45%, the average response time of R_2's requests keeps increasing as vm_8 is more and more heavy-loaded while the average response time of R_1's requests does not.

The reason is that vm_5 and vm_8 share the CPU resource of the same physical server. Therefore, when vm_8 has a heavier load, it would affect the performance of vm_5. Furthermore, since T_A depends on vm_1, vm_1 depends on vm_4, and vm_4 depends on vm_5 to complement users' requests, vm_5's performance degradation further results in the longer response time of T_A's R_2 requests. We can observe that there are two different types of anomaly propagation, *i.e.*, ($vm_8 \rightarrow vm_5$) due to *collocation relationship between different VMs* and ($vm_5 \rightarrow vm_4 \rightarrow vm_1$) due to *service call*.

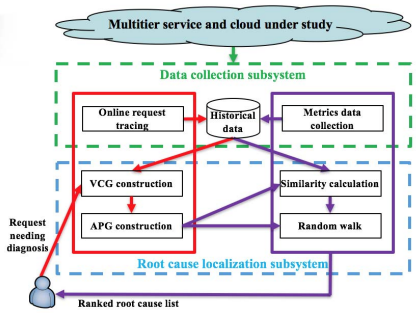


Fig. 3. System overall architecture.

Observation: In public clouds, there are two types of anomaly propagation paths: 1) between co-located VMs because of resource contentions, 2) among multitier service components along the path of service call.

In the public cloud, what T_A can learn is only its own overlay network assigned by the cloud provider. T_A cannot detect or measure collocation relationships among VMs in the public cloud. Therefore, T_A cannot find the root causes of anomalies that propagate due to collocated VMs, *e.g.*, the longer response time to users' requests caused by vm_8's heavy load in Figure 2. Then we have the following proposition.

Proposition 1: It is necessary for cloud providers to provide service to help tenant to determine root causes of anomalies.

B. System Overall Architecture

Figure 3 is the architecture of our system that tries to help cloud providers find out root causes of anomalies experienced by tenants. Assume one tenant observes the request r from one of its users experiences a very long time to be responded. The tenant then goes to the cloud provider for help and submits it as an anomaly a. Taking anomaly a as an example, our system is using a two-step procedure to diagnose as follows.

1) First, find out all possible causes of the anomaly a and construct the anomaly propagation graph (APG) G_a^{APG}. In other words, G_a^{APG} includes both types of anomaly propagation paths into consideration, *i.e.*, collocation propagation and service call propagation.
2) Second, determine the probability of each element in G_a^{APG} as a root cause of the anomaly a and point out the most likely root causes.

The components to complete the first step are shown in the solid-line box in the left of Figure 3. We need to trace online requests, collect and save data for future use (Section IV-A). From the data, we can construct a VM Communication Graph (VCG) of request r, wherein each edge reflects a service call relationship between two VMs (Section IV-B). We further consider anomaly propagations due to resource contention among co-located VMs, and construct the APG which includes two types of edges, *i.e.*, collocation dependency edges and service call edges (Section IV-C).

Now we can go to the second step, *i.e.*, finding the most likely root causes in the APG. As shown in the right box in Figure 3, there are three modules. The challenge here is how to evaluate the likelihood of one node to be a root cause. We propose a metrics, *similarity*, to solve this problem. In order to derive the similarity, we need to compute the service time of each component within the multitier service and monitor the resource utilization of each VM. This job is completed in the module of Metrics data collection (Section V-A). Then we calculate the similarity of each VM in the APG using these metrics data (Section V-B). We further incorporate the probability of VMs propagating their anomalies through the APG in our system by running the random walk algorithm, and finally determine a list of possible root causes for the anomaly a (Section V-C).

Among all modules in the system, two modules, Online request tracing and Metrics data collection, should be always running while the multitier service is serving. They continuously monitor the service and save data in the database. We refer to this subsystem as *Data collection subsystem*, shown in upper part of the figure.

The other four modules are activated in response to anomalies submitted by tenants. We refer to the subsystem including these four modules as *Root cause localization subsystem*. It accepts appeals from tenants, and returns a list of ranked root causes.

IV. Anomaly Propagation Graph

In this section, we would introduce how we accomplish the first task proposed in Section III-B. The nodes in G_a^{APG} are in fact VMs which are possible root causes that can result in the anomaly a. An edge $vm_i \rightarrow vm_j$ in G_a^{APG} means vm_i depends on vm_j during handling a request. There are two types of edges, *i.e.* service call dependency edge and collocation dependency edge. A service call dependency edge $vm_i \rightarrow vm_j$ means vm_i calls vm_j during handling a request. A collocation dependency edge $vm_i \rightarrow vm_j$ means vm_i and vm_j are co-located in a same physical server, and vm_i is a service component of the multitier service under study. In fact, these dependency edges are the reverse directions of anomaly propagation.

Obviously, to construct the graph G_a^{APG}, we need to find out all possible propagation paths, *i.e.*, collocation dependency edges and service call dependency edges. As a cloud provider, it is easy for him to retrieve information about collocated VM pairs, *i.e.*, collocation edges. For example, in OpenStack, a cloud provider can get VM distributions in the physical machines through APIs implemented in the project *nova*.

In terms of service call edges, as we know, for a request r, it would trigger a series of service calls, *i.e.*, communications, among a set of VMs. All these service call edges would form a directed acyclic graph. Let us call this graph as VM Communication Graph (VCG). In terms of graph topology, VCG is in fact a subgraph of the corresponding APG.

However, cloud providers do not have the information directly about call relationships of services run by their tenants. They have to collect and analyze data to infer these service call dependency edges. Considering tenants' privacy concerns and the complexity of multitier services, we argue that cloud providers should construct VCG based on request tracing technique without intrusiveness to multitier services.

A. Request Tracing of Multitier Services

In this paper, we exploit PreciseTracer proposed by Sang *et al.* [16] because it can not only do request tracing without the knowledge of source code but also get more accurate results through kernel instrumentation. In multitier service, a request triggers a series of interaction activities in

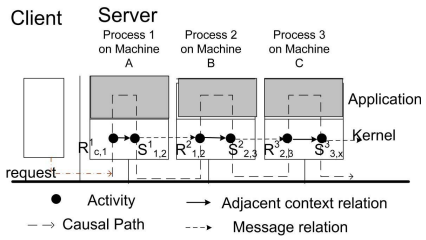


Fig. 4. Activities with causal relations in the kernel [16].

the OS kernel or shared libraries, e.g. sending or receiving messages. PreciseTracer uses Systemtap [28] to capture those activities.

When an individual request is serviced, a series of activities that have causal or happened-before relationship constitute a causal path graph. PreciseTracer provides us the causal path graph which is a directed acyclic graph, wherein vertices are activities of components and edges represent causal relations between two activities. There are four types of activities: *BEGIN, END, SEND,* and *RECEIVE*. The SEND and RECEIVE activities are those of sending and receiving messages. A BEGIN activity marks the start of servicing a new request, while an END activity marks the end of servicing a request. PreciseTracer records an activity of sending a message as $S^i_{i,j}$, which indicates a process i sends a message to a process j and records an activity of receiving a message as $R^j_{i,j}$, which indicates a process j receives a message from a process i.

Figure 4 shows a simple causal path graph which includes only one activity sequence $\{R^1_{c,1}, S^1_{1,2}, R^2_{1,2}, S^2_{2,3}, R^3_{2,3}, S^3_{3,x}\}$. According to the graph, we can calculate the service time of each component *i.e. VM* in servicing an individual request. For example, for the request in Figure 4, the service time of Machine B is $\big(t(S^2_{2,3}) - t(R^2_{1,2})\big)$, where $t(\cdot)$ is timestamp of the corresponding activity. We also define $h(\cdot)$ as the hostname of VM where the corresponding activity run.

B. VCG Construction

Now we need to transform the causal graph into VCG for our further root cause analysis. In a causal path graph, because of the complexity of multitier service function, a service, *i.e.* a process, on one VM might be called many times by one other VM, and the causal path graph records each individual time of the communication between these two VMs, *i.e.*, there are multiple edges between two VMs. Figure 5 gives an example of such causal path graph produced by PreciseTracer in the scenario shown in Figure 1. In this graph, service 1 in vm_1 calls service 2 in vm_4 for two times. The first call is represented by $S^1_{1,2}$ and $R^2_{1,2}$ (vm_1 requests service on vm_4); and $S^2_{2,1}$ and $R^1_{2,1}$ (vm_4 responds vm_1's request). The second call is described by $S^{1'}_{1,2}$ and $R^{2'}_{1,2}$; $S^{2'}_{2,1}$ and $R^{1'}_{2,1}$. Different from the first call, this time service 2 needs to call service 3 in vm_5 for two times to get necessary data and then return results to service 1.

We do not concern the details of these communications among services. To solve the problem in this paper, what we need to know is dependency relations among related VMs, *i.e.*, service call edges at VM level. As an example, the corresponding VCG for Figure 5 is shown in Figure 6.

The challenge to transform Figure 5 into Figure 6 is how to determine the direction of edges between two VMs. As we know, a complete service call usually consists of two stages, *i.e.,* stage 1 during which a VM vm_s sends a request to another VM vm_d, and stage 2 during which vm_d responding to vm_s. The direction of the edge between two VMs should be from the requester to the responder, and the VM who initiates the first communication between two VMs is the requester. So the steps for us to construct the VCG are as follows:

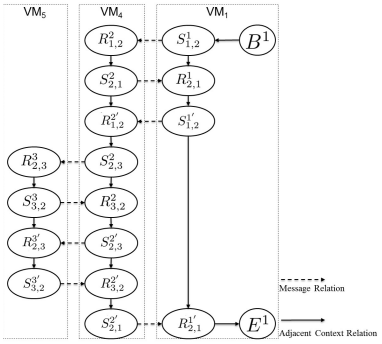


Fig. 5. A causal path graph produced by PreciseTracer.

Fig. 6. The VCG corresponding to Figure 5.

1) We classify activities in the causal path graph into different sets according to *hostname* of each activity, and the activity set $\S_v$ includes all activities related to vm_v.
2) For each VM vm_v, sort the activities in $\S_v$ according to their *timestamp*.
3) For each VM vm_v, try to determine directions of its related edges based on distinguishing the requester and the responder of a service call.
   3.1) For each SEND activity $S^i_{i,j} \in \S_v$, find its corresponding RECEIVE activity $R^j_{i,j}$ in the causal path graph. For example, $S^1_{1,2}$ is a SEND activity and its corresponding RECEIVE activity is $R^2_{1,2}$.
   3.2) If we cannot find a SEND activity $S^j_{j,i}$ that satisfies both $h(S^j_{j,i}) = h(R^j_{i,j})$ and $t(S^j_{j,i}) < t(R^j_{i,j})$, add the edge $h(S^i_{i,j}) \rightarrow h(R^j_{i,j})$ to the VCG.

C. APG Construction

As we stated at the beginning of Section IV, we need APGs to analyze possible root causes of an anomaly, and the APGs should include two types of edges: collocation dependency edges and service call edges. Now we have found all collocation edges based on *nova* APIs and we also obtained VCGs such as Figure 6 that includes all service call edges. Then we can constructs the APG by combining VCG and VM co-location relationship. Take the scenario shown in Figure 1 as an example, assume there is a *ViewItem* request from T_A's users. The request will be handled through path 2, and the APG for this request is shown in Figure 7.

V. Root Cause Localization

In this section, we will try to solve the second task, *i.e.*, pointing out the most likely root causes from all related

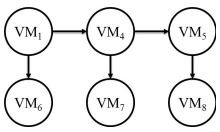


Fig. 7. The APG of *ViewItem* requests in Figure 1.

elements included in G_a^{APG}. We define $\Phi_a(i)$ as the probability of vm_i being the root cause of the anomaly a.

In this paper, an anomaly refers to that the response time of a request of the multitier services becomes longer than users can tolerate. For example, in the scenario shown in Figure 1, assume a user of T_A sends a *SearchItemsByCategory* request r, and it takes the user very long time to receive the response of r. The user will complain to T_A and T_A will regard this slow response as an anomaly. As we have stated in Section III, T_A has to ask the cloud provider for help to find out the most likely root causes of this anomaly.

Which VM is the root cause of this slow response for the request r? Let us denote the root cause VM as vm_{root}. It is reasonable to conjecture that vm_{root} should be very busy when the anomaly occurs. In other words, vm_{root} becomes very busy due to some reasons, *e.g.*, software bugs or resource exhaustion, and it then causes the slower response for the request r. Therefore, there should be a correlation, *i.e.*, similarity, between the metrics data of vm_{root} and the response time. As there are many services provided by the multitier service application and if a request calls a different service, it may form a different causal path graph. We define $\mathbb{R}(r)$ as a collection of requests that form the same causal path graph as r does. Obviously different requests in the $\mathbb{R}(r)$ happen at different time. We define $\mathbb{S}(vm_i, \mathbb{R}(r))$ as the similarity between the metrics data of vm_i and the response time of requests $\mathbb{R}(r)$, *e.g.,* R_2 mentioned in Section III. The similarity can be used to derive the probability of being the root cause to a certain extent.

However, one VM vm_i with a high $\mathbb{S}(vm_i, \mathbb{R}(r))$ is not a sufficient condition to determine that vm_i must be a root cause. For example, in Figure 1, assume both T_A and T_B are providing online auction services, it is highly possible that more users will visit the two websites at weekends and then more requests need to be handled at weekends for services of both T_A and T_B. It is natural that T_A's response time will be longer than weekdays. At the same time, we can see vm_6 will be busier than weekdays. Performance of vm_6 shows a high correlation with the response time of T_A's requests, *i.e.*, $\mathbb{S}(vm_6, R_2)$ is high. Can we conjecture that vm_6 is a root cause of T_A's slow response at weekends? Obviously, we cannot. vm_6 is correlated with T_A's response time only because T_A and T_B share a same periodic user behavior pattern.

How to exclude these VMs during our root cause analysis? In the case mentioned above, vm_6 does not belong to T_A and it appears in APG just because it is co-located with vm_1 of T_A. So vm_6 can interfere with the performance of T_A's services only by its resource contentions with vm_1. If this resource contention really results in longer response time, it must be true that $\mathbb{S}(vm_1, R_2)$ will also be high. Therefore, we can say that if both $\mathbb{S}(vm_6, R_2)$ and $\mathbb{S}(vm_1, R_2)$ are high, the anomaly of T_A might be caused by vm_6; if $\mathbb{S}(vm_6, R_2)$ is high and $\mathbb{S}(vm_1, R_2)$ is small, vm_6 cannot be a root cause of the anomaly. Here we say vm_1 blocks the possibility of vm_6's anomaly propagation through the APG. In our solution, we exploit the random walk algorithm to reflect the possibility of anomaly propagation.

Proposition 2: A VM being the root cause must meet two conditions: 1) the metrics data of the VM must have a high similarity with the response time of the requests $\mathbb{R}(r)$, 2) the VM must have a high possibility to propagate its anomaly through APG.

Obviously, we need to collect related metrics data continuously in order to calculate similarity of metrics data anytime when an anomaly occurs. We will introduce our data collection method in Subsection V-A. In Subsection V-B, we would show how to calculate the similarity of each VM in the APG. In Subsection V-C, we will run a random walk algorithm over the APG to further include the factor of propagation possibility to determine the $\Phi_a(i)$.

A. Metrics Data Collection

For one VM vm_i, there can be various metrics to evaluate its performance. Which metrics is better to be used in calculating $\mathbb{S}(vm_i, \mathbb{R}(r))$? If vm_i is included in the APG because vm_i provides a service which is necessary to complete the request r, the service time of vm_i spent on vm_i to handle the request would be a good choice. If vm_i is included into the APG through collocation dependency relationship, resource contention is the reason that vm_i can interfere with the performance of multitier services, then resource consumption would be a good choice.

So we need to collect different types of metrics data *i.e.,* the service time and the resource consumption, for different types of VMs :

1) Service Time Computation: As we know, one multitier service needs many VMs to work coordinately to serve a request r, so the response time of r is the sum of the service time spent on each VM to handle the request r. The service time η_i of each vm_i for the request r can be calculated from the corresponding causal path graph as follows:

$$\eta_i = \sum_x \left(t(S^i_{i,x}) - t(R^i_{x,i})\right) - \sum_y \left(t(R^i_{y,i}) - t(S^i_{i,y})\right)$$

In this formula, x stands for the VM which sends request to vm_i and y stands for the VM to which vm_i sends request. The first item is the sum of all intervals that vm_i spends in handling the requests from other VMs. Because the first item includes the intervals that vm_i spends in waiting for the response from other VMs, so we subtract the sum of waiting intervals in the second item.

2) Resource Utilization Collection: We use Ganglia monitoring system [29] to collect resource utilization data. Ganglia is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids and it can achieve very low per-node overheads and high concurrency. By default, Ganglia only collects metrics data of physical machines. Fortunately there is a plugin sFlow [30] which can collect metrics data of virtual machines through hypervisor in physical machines. So we use Ganglia and sFlow to collect CPU and memory consumption, I/O and network throughput for every VM.

B. Similarity Calculating

If an anomaly a of request r occurs, we can construct the APG. Given the APG, our goal is to locate the VM in the APG

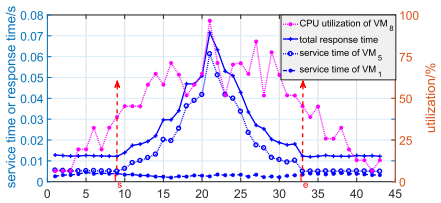


Fig. 8. The changing trend of metrics data of different VMs.

that causes a. Our intuition is that the correlation between metrics data of one VM and the response time of $\mathbb{R}(r)$ can be used to measure the probability of the VM being the root cause to a certain extent.

Again, let us take the scenario shown in Figure 1 as an example. We gradually increase vm_8's CPU utilization rate and send *SearchItemsByCategory* requests continuously. We monitor metrics data of each VM and the response time of users' requests. Figure 8 shows the changing trend of the response time of *SearchItemsByCategory* requests which are handled through the path 2, the service time of vm_5, the service time of vm_1 and the CPU utilization of vm_8. From the figure, we can observe that except for the curve of the service time of vm_1, the other three curves are of strong correlation when the performance of vm_5 is interfered with, *i.e.*, from t_s to t_e. We will show how to determine t_s and t_e later. But, in the beginning, the curves, *i.e.*, before t_s, the response time and CPU utilization of vm_8 are of weak correlation because the CPU utilization of vm_8 is low and the performance of vm_5 hasn't been interfered with at that time. This phenomenon has been illustrated in Figure 2, where the response time does not increase with the CPU utilization of vm_8 when vm_8's CPU utilization is smaller than a certain threshold. In Figure 8, the root cause is vm_8 according to our experiment setting, and the experiment result shows that metrics data of vm_8 and vm_5 are both of strong correlation with the response time. It proves our Proposition 2 is reasonable.

If vm_i is a component of the multitier service, we calculate the similarity according to its service time η_i. For a co-located VM, we first calculate the correlations between the response time of requests and its contentions of different resource types, *e.g.* CPU and memory, I/O and network throughput, and then we select the maximum of these correlations to denote its similarity. This is because the service performance can be affected due to contentions of different types of resources.

We define $t(r)$ as the timestamp when request r is issued and $\tau(r)$ as the response time of request r. Then we define a function $\mathcal{R}(i, M, \mathbb{R}(r), t_s, t_e)$ that calculates the correlation between the metric M of vm_i and response time of requests $\mathbb{R}(r)$ which are issued from t_s to t_e based on Pearson Correlation Coefficient. The calculation formula is as follows:

$$\mathcal{R}(i, M, \mathbb{R}(r), t_s, t_e) = \frac{Cov\left(\mathbb{M}_{t_s}^{t_e}(M, i), \mathbb{T}_{t_s}^{t_e}(\mathbb{R}(r))\right)}{\sigma_{\mathbb{M}_{t_s}^{t_e}(M,i)} \sigma_{\mathbb{T}_{t_s}^{t_e}(\mathbb{R}(r))}}$$

wherein $\mathbb{M}_{t_s}^{t_e}(M, i)$ is a series of metric data M of vm_i from t_s to t_e and $\mathbb{T}_{t_s}^{t_e}(\mathbb{R}(r))$ is a series of response time of related requests (share a same causal path graph) issued from t_s to t_e. The similarity $\mathbb{S}(vm_i, \mathbb{R}(r))$ is calculated based on the correlation defined above as follows:

$$\mathbb{S}(vm_i, \mathbb{R}(r)) = \begin{cases} \mathcal{R}(i, \eta, \mathbb{R}(r), t_s, t_e), & \text{if } vm_i \in VCG \\ max\{\mathcal{R}(i, \upsilon, \mathbb{R}(r), t_s, t_e) | \upsilon \in \Upsilon\}, & \text{otherwise} \end{cases}$$

wherein Υ is the set of types of resource competed for by VMs. For example, Υ can be $\{$*CPU, Memory, I/O, Network*$\}$.

Given the formula above, we now need to solve two problems. The first one is to collect history data of service time η_i of vm_i when vm_i handles requests $\mathbb{R}(r)$. The second problem is to determine the time point t_s and t_e.

For the first one, because the service time can be calculated based on the causal path graph according to Equation 1, we only need to find out all causal path graphs of requests in $\mathbb{R}(r)$, then we can calculate the history data of service time η_i of vm_i. As $\mathbb{R}(r)$ are requests that can form the same causal path graph as r does, we need to find out all the causal path graphs that are the same as the causal path graph of request r.

To find out all the same causal path graphs, we need to extract the sequence of a causal path graph. Specifically, given a causal path graph, we can get the activity sequence $\S$ according to the *topological sorting algorithm*. We then use the attribute tuple (*activity type*, *program name*) of activity $\mathcal{A}$ to represent the activity $\mathcal{A}$ in $\S$. Using the method above, we can get the sequence of (*activity type*, *program name*) tuple for every causal path graph. For example, the sequence of causal path graph in Figure 5 are $\{(B, P_1), (S, P_1), (R, P_2), (S, P_2), (R, P_1), (S, P_1), (R, P_2), (S, P_2), (R, P_3), (S, P_3), (R, P_2), (S, P_2), (R, P_3), (S, P_3), (R, P_2), (S, P_2), (R, P_1), (E, P_1)\}$, wherein B, E, S and R stand for activity type *BEGIN, END, SEND,* and *RECEIVE*, P_i stands for the name of process i. At last, we only need to find out causal path graphs that have the same sequence of (*activity type*, *program name*) tuple as the causal path graph of r does.

The second problem is to determine the time point t_s and t_e. We set t_e as the time point when a tenant submits the root cause analysis job to our system. We set t_s as the time point when the performance of multitier service starts to degrade. It is reasonable to assume the tenant knows t_s and can provide it to the system. If the system cannot get t_s from tenants, we can find out it from historical data as follows:

$$t_s = min\{t | \frac{\tau(r_t)}{E\left(\mathbb{T}_{t-w}^{t}(\mathbb{R}(r))\right)} > \delta\}$$

wherein r_t is the request whose timestamp is t and w is the length of the time window. That is we set t_s as the earliest time point t when the ratio of response time of request that was issued at t to the average response time of requests that are issued during the previous time window is bigger than a threshold δ (e.g. 1.2).

C. Random Walk Over APG

As we have discussed at the beginning of this section, high $\mathbb{S}(vm_i, \mathbb{R}(r))$ is a necessary but not sufficient condition to conclude that vm_i is a root cause of the anomaly of request r. We now need to further consider the probability that those VMs propagate their anomaly through the APG.

We propose to conduct a random walk over the APG to emulate the procedure of one cloud provider tracing back to the root causes of the anomaly. The walker starts from the VM where the anomaly occurs, and moves forward from one

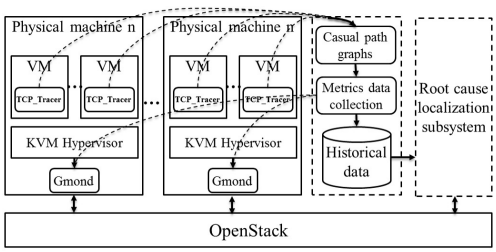


Fig. 9. Experimental environment

node to one of its neighbors. Given an APG $G^{APG}(V, E)$, let us define a matrix Q as follow.

- Forward-edges: as we described in Section IV, the edges in APG have the reverse directions of anomaly propagations. That is if edge $e_{ij} \in E$, vm_j may propagate anomalies to vm_i. Recall that a node with higher similarity is more likely to be a root cause. Therefore the walker prefers to move to vm_j with higher similarity to trace back to the anomaly source. As a result, we set Q_{ij} as $\mathbb{S}(vm_j, \mathbb{R}(r))$ if edge $e_{ij} \in E$.
- Backward-edges: it is possible that the walker arrives at VMs with low similarity. In that case, he may want to go back to the previous node and then he can move to other neighbors from that node. We add backward-edges for such cases. If $e_{ij} \in E$ and $e_{ji} \notin E$, Q_{ji} is set as $\rho\mathbb{S}(vm_i, \mathbb{R}(r))$, where ρ is a parameter set by the administrator and $\rho \in [0, 1)$.
- Self-edges: when the walker stands in a VM with a higher similarity than the neighbors of the VM, it is an indication that the VM is likely to be one of root causes, so the walker should stay at the VM. As a result, we add self-edges, if we set Q_{ii} as the result of the similarity of vm_i minus the maximum similarity score of the neighboring VMs.

In summary, we calculate matrix Q according to the following formula.

$$Q_{ij} = \begin{cases} \mathbb{S}(vm_j, \mathbb{R}(r)), & \text{if } e_{ij} \in E \\ \rho\mathbb{S}(vm_j, \mathbb{R}(r)), & \text{if } e_{ji} \in E, e_{ij} \notin E \\ max\left(0, \mathbb{S}(vm_i, \mathbb{R}(r)) - max_{k:e_{jk}\in E}\mathbb{S}(vm_k, \mathbb{R}(r))\right), & \\ & \text{if } j = i \end{cases}$$

We normalize every row of the matrix Q, and get our transition probability matrix $\bar{Q}$ as follows.

$$\bar{Q}_{ij} = \frac{Q_{ij}}{\sum_j Q_{ij}}$$

Now we can do random walk over the APG, and the probability of the random walker moving from vm_i to vm_j is $\bar{Q}_{ij}$. Let the walker move a lot of steps, and we count the number of visits on each VM. More visits on a certain VM implies that the VM is more likely to be a root cause.

VI. Experiment and Evaluation

A. Experimental Environment

As shown in Figure 9, we implement and deploy our proposed system to do root cause analysis in our real cloud platform. We use OpenStack [32] which is a well-known enterprise-class cloud computing stack used for both private and public cloud computing infrastructure to do VM management and resource allocation. We use PreciserTracer [16] to do request tracing of multitier services to construct causal path graph of requests. PreciserTracer needs to deploy an agent called *TCP_Tracer* on each VM to record interaction activities of interest. Then PreciserTracer would correlate those activity logs of different VMs into causal path graphs. We then calculate the metrics data about service time of each VM from the causal path graphs, and store the results in the database. We also use the Ganglia which is deployed on each physical server to collect resource utilization of each VM. The root cause localization subsystem can find out the root cause list and corresponding probability if an anomaly of a request occurs.

B. Performance Evaluation

Baseline methods. For the purpose of comparison, we firstly introduce three baseline methods:

- Random Selection (RS): A human without any domain knowledge will examine VMs in random order. We mimic this behavior by issuing random permutations.
- Sudden Change (SC): It compares the metrics in the current and previous time windows and checks if there is any sudden change between the two time windows. It then calculates the ratio between average metrics in the current and previous time windows and refers to this ratio as the root cause score of each VM.
- Distance Based Rank (DBR) [9]: In DBR, for every component c, it forms a propagation graphs wherein nodes are a set of anomalous components that can be reached from c. Then the problem of finding out the root cause can be transformed into selecting the best propagation graph. The rank of a propagation graph is determined by the minimum total distance from the source entity to all other anomaly entities.

Evaluation metric. We use the following two evaluation metrics proposed by [8] to quantify the performance of each method on a set of anomalies $\mathbb{A}$, where $\psi_a(i)$ means the rank of vm_i as the root cause of an anomaly a and $I_a(i)$ represents whether vm_i actually is the root cause of an anomaly a (that is, either 0 or 1):

- Precision at top K ($PR@K$) indicates the probability that top K VMs given by each algorithm actually are the root causes of each anomaly case.

$$PR@K = \frac{1}{|\mathbb{A}|}\sum_{a\in\mathbb{A}}\frac{\sum_{i:\psi_a(i)\le K} I_a(i)}{min\left(K, \sum_i I_a(i)\right)}$$

- Mean Average Precision (MAP) quantifies the overall performance of a method, where N is the number of VMs:

$$MAP = \frac{1}{|\mathbb{A}|}\sum_{a\in\mathbb{A}}\sum_{1\le k\le N} PR@k$$

We also use two metrics to quantify the cost for a cloud provider to diagnose an anomaly with the help of our system. Assume the operator gets the list from our system, and then he checks VMs one by one, from VM with higher probability to lower probability, to recover the service.

We define $N_a(i)$ as the number of VMs that are not true root causes but have been checked before checking the root

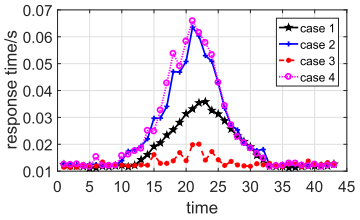


Fig. 10. Response time of different cases.

cause vm_i of the anomaly a:

$$N_a(i) = \begin{cases} \sum_{1 \le j \le \psi_a(i)} (1 - I_a(j)), & \text{if } I_a(i) = 1 \\ 0, & \text{if } I_a(i) = 0 \end{cases}$$

When $I_a(i)$ is 0, $N_a(i)$ is defined as 0 for convenience.

Based on the definition of $N_a(i)$, we define our two metrics, AFP (average false positive) and MFP (max false positive) as follows:

$$AFP = \frac{1}{|\mathbb{A}|} \sum_{a \in \mathbb{A}} \frac{\sum_{1 \le i \le N} N_a(i)}{\sum_{1 \le i \le N} I_a(i)}$$

$$MFP = \frac{1}{|\mathbb{A}|} \sum_{a \in \mathbb{A}} max\{N_a(i) | 1 \le i \le N\}$$

C. Root Cause Analysis at the VM Level

In the first experiment, we use the same scenario as shown in Section III-A. There are two tenants in the cloud, each of which runs a 3-tiered web application RUBiS. In this scenario, the mapping between server processes and the VMs is one-to-one, and we only need to find out root causes at the VM level.

We consider 4 cases for our performance evaluation under this scenario. The first case is to emulate the anomaly caused by one component within the service and propagated through service calls. The second case is to emulate the anomaly caused by collocated VMs of other tenants and propagated through collocation edges. The third case is to emulate the situation that resource insufficiency on collocated VMs cannot propagate to affect the service. The forth case is to emulate a more complex situation where resource insufficiency occurs on two collocated VMs and one of them cannot propagate. Technically, we inject anomalies to the RUBiS of T_A as follows:

- Case 1: We inject delays into PHP "SearchItemsByCategories" function on vm_4, and the delay is a random value between 2ms and 25ms for each request.
- Case 2: We orchestrate the CPU utilization of vm_8 from 10% up to 90% linearly and then from 90% down to 10% linearly using the tool lookbusy [31]. It would interfere with the performance of mysql on vm_5, and finally results in an anomaly of the service.
- Case 3: We orchestrate the CPU utilization of vm_6 from 10% up to 90% linearly and then from 90% down to 10% linearly. It would not interfere with the performance of LVS on vm_1 because vm_1 only requires little CPU resource.
- Case 4: We combine case 2 and case 3. That is, we orchestrate the CPU utilization of both vm_8 and vm_6 at the same pace.

Figure 10 shows the response time of *SearchItemsByCategory* requests of T_A's website in different cases. We can find that except for case 3, the performance of the website degrades seriously. For case 3, because the CPU requirement of LVS on vm_1 is so low that vm_6 cannot interfere with the performance of vm_1 by CPU resource contention, the performance of the website doesn't degrade.

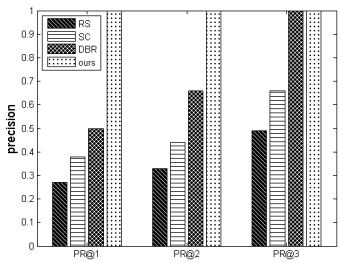


Fig. 11. Precision at top K of different methods in scenario 1.

TABLE I
MAP OF DIFFERENT METHODS

method	RS	SC	DBR	ours
MAP	0.36	0.49	0.72	1.00

For later comparison of methods, let us first point out the root causes of above three anomalies as ground truth. The root cause of case 1 is vm_4. For case 2, the performance of the website degrades because the performance of vm_5 is interfered with vm_8, so the root causes of case 2 are vm_8 and vm_5. For case 4, as the increase of the CPU utilization of vm_6 does not interfere with the performance of the website, the root causes of case 4 are vm_8 and vm_5.

Next we conduct different root cause analysis methods to localize the root causes of the three anomalies *i.e.,* case 1, 2 and 4, to compare their accuracies. In this experiment, the $\mathbb{A}$ in the definition of our metrics includes the above three anomalies.

Experiment Results. We evaluate our method and all the three baseline methods on three cases. Figure 11 shows $PR@1$, $PR@2$, $PR@3$ of different methods. Table I shows the average MAP metric of different methods. In every evaluation metric, our method outperforms the baseline methods by a large factor. More specifically, in terms of MAP, the improvement over the DBR method is approximately 38.9%.

D. Root Cause Analysis of Anomalies Arising From Multiple Causes

To evaluate our method in large-scale cloud platform and more complicated scenario, we conduct simulation experiments. CloudSim [37], [38] is a generalized simulation framework that allows modeling, simulation and experimenting the cloud computing infrastructure and application services. However, CloudSim can only simulate very simplistic application models without any communicating tasks within the data center. Networkcloudsim [39] extends the function of CloudSim. It can simulate applications with communicating elements or tasks such as MPI (message passing interface) and workflows. So we can use Networkcloudsim to simulate multitier services in cloud environment conveniently.

Networkcloudsim defines a basic and general structure *i.e.*, a Java class, called AppCloudlet to simulate complex parallel and distributed services. Each AppCloudlet object

consists of several communicating elements, *i.e.*, NetworkCloudlet. Each NetworkCloudlet runs in a single virtual machine and consists of communicating and computing stages.

We firstly design an algorithm to construct a cloud environment and simulate the allocation of VMs to physical servers when a tenant applies for VMs. We also design an algorithm to construct a topology of multi-tier service for cloud tenants. Then we use Networkcloudsim to simulate the running of the cloud and services of tenants.

We simulate a cloud with $\mathbb{N}_u$ tenants. We randomly select a tenant from the $\mathbb{N}_u$ tenants as our target tenant u^* for our study. Let U' denote the set of other tenants, *i.e.*, $U' = (U - u^*)$.

The cloud has $\mathbb{N}_p$ physical servers, and the number of CPU cores of every physical servers is set as a value randomly selected from integers between $\mathbb{C}_l$ and $\mathbb{C}_h$. If the number of CPU cores of one physical server is c, the max number of VMs allocated to the physical server is set to be $(2 * c)$. We can derive that one tenant in U' can have $((\mathbb{C}_l+\mathbb{C}_h)*\mathbb{N}_p/(\mathbb{N}_u-1))$ vms in average.

The frequency of every CPU core of physical servers is $\mathbb{F}$. Each vm has only one CPU core and its frequency is set to be $\mathbb{F}/2$. The VM allocation procedure is shown in Algorithm 1. In our simulation, the parameter $\mathbb{N}_p$, $\mathbb{N}_u$ and $\mathbb{F}$ are set as 10000, 500 and 3GHz respectively.

Algorithm 1 vm Allocation in the Cloud

Parameters: $\mathbb{N}_p$: the number of physical servers
$\mathbb{C}_l$,$\mathbb{C}_h$: min and max num of cpu cores of PM
$\mathbb{N}_u$: the num of tenants, $\mathbb{N}_t$: vm number of u^*
Output: $result$: $\{(tenantId, vmId) \rightarrow pmId\}$

$result \leftarrow \{\}$, $count \leftarrow 0$, $index\_u^* \leftarrow randomInt(\mathbb{N}_u)$
$avg\_num \leftarrow ((\mathbb{C}_l + \mathbb{C}_h) * \mathbb{N}_p/(\mathbb{N}_u - 1))$
while $count < \mathbb{N}_u$ **do**
  $i \leftarrow randomInt(\mathbb{N}_u)$
  $num \leftarrow randomInt(avg\_num - 10, avg\_num + 10)$
  if $tenant_i$ has not been allocated **then**
    $count \leftarrow count + 1$
    if $i == index\_u^*$ **then**
      $num \leftarrow \mathbb{N}_t$
    for $j \in range(0, num)$ **do**
      find a PM k with free resource
      add $(i,j) \rightarrow k$ to $result$
return $result$

Now we generate multitier services topology for u^*, since we need to know the service call edges of u^*. There are three parameters for this generation. The number of VMs of the tenant u^* is $\mathbb{N}_t$. In each layer of the multi-tier service, the number of VMs is set as a random integer between $\mathbb{L}_l$ and $\mathbb{L}_h$, where $\mathbb{L}_l$ is the lower bound and $\mathbb{L}_h$ is the upper bound. Every VM in ith layer is connected to every VM in $(i+1)$th layer. In our simulation, we generate topologies using four settings in Table II.

Given the cloud and the service topology of the target tenant, we run one NetworkCloudlet in every VM of u^* and set up the communication relationships among NetworkCloudlets based on the generated topology.

We define $\mathbb{O}_s(t_s, t_e, \eta_1, \eta_2, linear)$ on NetworkCloudlet C as the operation that we orchestrate the service time of NetworkCloudlet C from η_1 up to η_2 linearly in the time window (t_s, t_e). And we define $\mathbb{O}_s(t_s, t_e, \eta_1, \eta_2, random)$ on NetworkCloudlet C as the operation that we set the service times of NetworkCloudlet C as values randomly selected from integer between η_1 and η_2 in the time window (t_s, t_e). Similarly, $\mathbb{O}_f(t_s, t_e, \mathbb{F}_1, \mathbb{F}_2, linear)$ on VM vm is the operation that we orchestrate the CPU frequency of vm from $\mathbb{F}_1$ up to $\mathbb{F}_2$ linearly in the time window (t_s, t_e). $\mathbb{O}_f(t_s, t_e, \mathbb{F}_1, \mathbb{F}_2, random)$ on VM vm is the operation that we orchestrate the CPU frequency of vm as values randomly selected between $\mathbb{F}_1$ and $\mathbb{F}_2$ in the time window (t_s, t_e).

TABLE II
SETTINGS OF TOPOLOGY GENERATION

No.	$\mathbb{N}_t$	$\mathbb{L}_l$	$\mathbb{L}_h$	No.	$\mathbb{N}_t$	$\mathbb{L}_l$	$\mathbb{L}_h$
1	60	2	4	2	90	3	6
3	120	4	7	4	150	5	9

We attempt to do anomaly injection to the multitier service of u^* as the following four cases, and each anomaly arises from multiple causes. In this experiment, the set $\mathbb{A}$ includes the following four anomalies.

- Case 1: We firstly select $\mathbb{N}_v^f$ NetworkCloudlets of u^* and apply operation $\mathbb{O}_s(0, 20, 1500, 2000, linear)$ on these NetworkCloudlets synchronously. Let us denote the set of these selected NetworkCloudlets as V_f. Secondly, to increase the complexity for our performance evaluation, we select another $\mathbb{N}_v^f$ NetworkCloudlets of u^* to simulate normal fluctuations of service time, that is we apply operation $\mathbb{O}_s(0, 20, 100, 150, linear)$ on these NetworkCloudlets synchronously. Let $\hat{V}_f$ denote the second set of selected NetworkCloudlets. The root causes of this case are VMs which run NetworkCloudlets in V_f.
- Case 2: Similar to case 1, we apply operation $\mathbb{O}_s(0, 20, 1500, 2000, random)$ on NetworkCloudlets in V_f synchronously and we apply operation $\mathbb{O}_s(0, 20, 100, 150, random)$ on NetworkCloudlets in $\hat{V}_f$ synchronously. The root causes of this case are VMs which run NetworkCloudlets in V_f.
- Case 3: We firstly select $\mathbb{N}_v^f$ VMs of tenants in U' to apply operation $\mathbb{O}_f(0, 20, \mathbb{F}/2, F, linear)$ on these selected VMs synchronously. Let V_f denote the set of these VMs. Secondly, to increase the complexity of finding out root causes, we select another $\mathbb{N}_v^f$ VMs of U' to simulate normal fluctuations of resource utilization rate, and we apply the operation $\mathbb{O}_f(0, 20, \mathbb{F}/20, \mathbb{F}/2, linear)$ on these VMs synchronously. We denote the second set of selected VMs as $\hat{V}_f$. The root causes of this case are VMs in V_f and $\{H(i)|\forall vm_i \in V_f\}$, where $H(i)$ is the set of VMs of u^* which are collocated with vm_i.
- Case 4: Similar to case 3, we apply operation $\mathbb{O}_f(0, 20, \mathbb{F}/2, F, random)$ on VMs in V_f synchronously and we apply operation $\mathbb{O}_f(0, 20, \mathbb{F}/20, \mathbb{F}/2, random)$ on VMs in $\hat{V}_f$ synchronously. The root causes of this case are VMs in V_f and $\{H(i)|\forall vm_i \in V_f\}$.

In summary, we use the first two cases to simulate an anomaly caused by abnormal components of the multitier service and there are some components with normal fluctuations of service time. And we use the last two cases to simulate an anomaly caused by performance interference of abnormal

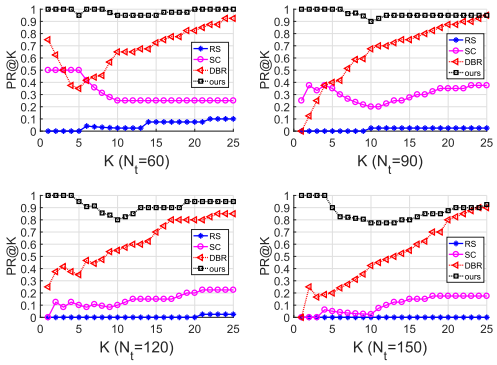


Fig. 12. PR@K of different methods (different $\mathbb{N}_t$).

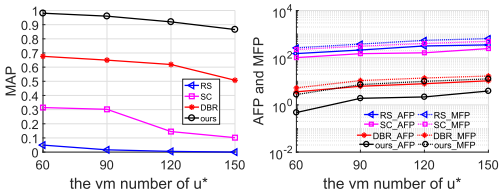


Fig. 13. MAP, AFP and MFP of methods (different service sizes).

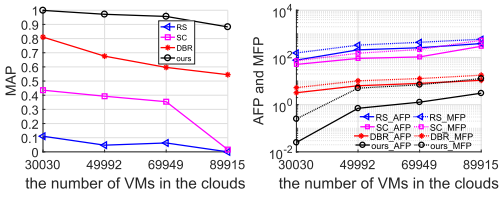


Fig. 14. MAP, AFP and MFP of all methods (different cloud sizes).

collocated VM of other tenants and there are collocated VMs with normal fluctuations of resource utilization rate.

We conduct our method and all baseline methods under different parameter settings. For each setting, we construct the cloud and the topology of u^*'s service, inject anomalies as described above, and then run methods to localize root causes. For each case and each method, we repeat 100 times and calculate the average of performance metrics.

• **Performance and the scale of service**. In this experiment, $\mathbb{N}_t$, the number of VMs u^* has in its service, is set to be 60, 90, 120 and 150 respectively. Additionally, ($\mathbb{C}_l$, $\mathbb{C}_h$) is set as (2, 3) and $\mathbb{N}_v^f$ is set as 5. Figure 12 shows $PR@K$, and Figure 13 shows MAP, AFP and MFP of all methods. We can see that our method outperforms other methods in terms of any performance metrics in all settings. As $\mathbb{N}_t$ gets bigger, which means the service becomes larger scale, AFP and MFP increase and the MAP decreases for all methods.

We calculate the MAP improvement of our method compared with DBR by the formula $\frac{MAP_{ours}-MAP_{DBR}}{MAP_{DBR}}$, and the MAP improvements are 45.21%, 47.76%, 48.45% and 70.96% in different $\mathbb{N}_t$. We can see that our method achieves more precision improvement than the DBR method for tenants with more VMs.

• **Performance and the size of cloud**. In this experiments, ($\mathbb{C}_l$, $\mathbb{C}_h$) is set to be (1, 2), (2, 3), (3, 4) and (4, 5). These settings result in four clouds and the number of VMs in the clouds are 30030, 49992, 69949 and 89915 respectively.

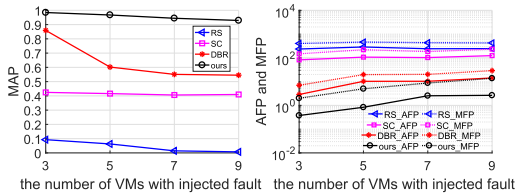


Fig. 15. MAP, AFP and MFP of all methods (different $\mathbb{N}_v^f$).

TABLE III
RUNNING TIME OF EXPERIMENTS IN SECTION VI-D

	Expt. 1	Expt. 2	Expt. 3	Expt. 4
service size	3.3/3.5	8.4/8.6	14.6/15.2	20.4/21.6
cloud size	1.8/1.9	8.2/8.7	17.2/18.9	29.4/30.1
fault number	8.4/8.8	8.3/8.4	8.4/8.5	8.5/8.6

Additionally, $\mathbb{N}_t$ is set as 90 and $\mathbb{N}_v^f$ is set as 5. Figure 14 shows MAP, AFP and MFP of all methods under different cloud sizes. Obviously, the performance degrades as the cloud size increases for all methods. The MAP improvement of our method compared with DBR is 23.45%, 43.55%, 60.02% and 63.12% in different cloud sizes. Again, we see that our method achieves more precision improvement as cloud size increases compared with DBR.

• **Performance and the number of faults**. In this experiments, $\mathbb{N}_v^f$ is set as 3, 5, 7 and 9 respectively. Additionally, $\mathbb{N}_t = 90$, $\mathbb{C}_l = 2$, and $\mathbb{C}_h = 3$. Figure 15 shows MAP, AFP and MFP of all methods under different $\mathbb{N}_v^f$. We can see that the performance degrades for all methods as there are more and more synchronous faults. The MAP improvement of our method compared with DBR is 14.28%, 60.87%, 70.47% and 70.89% in different $\mathbb{N}_v^f$. We see that our method achieves more precision improvement as $\mathbb{N}_v^f$ increases.

E. Diagnose Time Analysis of Our Method

We analyze the complexity of our method as follows. In the worst scenario, every VM of u^* is allocated to a different physical server, that is the number of physical servers associated with the APG is $\mathbb{N}_t$. And the average number of VMs in one physical server is $(\mathbb{C}_l + \mathbb{C}_h)$. So the number of nodes in the APG is approximately $\mathbb{N}^{APG} \approx ((\mathbb{C}_l + \mathbb{C}_h) * \mathbb{N}_t)$.

Our method needs two kinds of computations:

- Similarity calculation: for every node in the APG, we need to calculate its similarity. The complexity of this procedure is $O(\mathbb{N}^{APG})$.
- Random walk over APG: according to [40], the complexity of the random walk algorithm is $O(\mathbb{N}^{APG} + \mathbb{N}^{APG^2})$.

So the total complexity of our method can be about $O(\mathbb{N}^{APG})$+$O(\mathbb{N}^{APG} + \mathbb{N}^{APG^2})$.

Besides the above theoretic analysis on time complexity of our method, in Table III we also show the running times of our method for all experiments we conducted in Section VI-D. Take the first row as an example. It reports the running times of the four experiments with different service sizes, *i.e.*, the experiments in Figure 13, and the corresponding service sizes are 60, 90, 120, and 150. We can find that the running time increases as the service size increases. Similarly, we can also find that the running time increases with cloud

TABLE IV
SIMILARITY AND PROBABILITY OF VM IN CASE 4 OF SUBSECTION VI-C

VM	vm_1	vm_4	vm_5	vm_6	vm_7	vm_8
similarity (no rw)	0.052	0.31	0.993	0.567	0.053	0.599
visited prob. (with rw)	0.0536	0.132	0.325	0.067	0.017	0.403

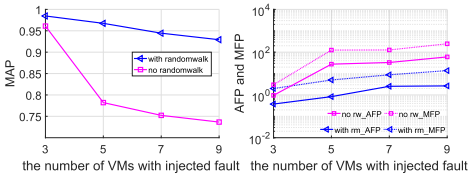


Fig. 16. MAP, AFP and MFP with and without random walk.

size, while the number of faults does not have a clear impact on the running time. In almost all experiments we conducted, our method can get results within 30 seconds. Our code is written in Python and run on a server with dual-core 3.6GHz CPU and 4G ram.

F. Discussion of Our Approach

In this paper, we argue that the high similarity between metrics data of one VM and the response time of the multitier service is a necessary but not sufficient condition for the VM to be a root cause. Besides the similarity metrics, we also include a random walk algorithm to exclude those VMs which are highly correlated with the multitier service only because of same periodic user behavior patterns.

We show the necessity of this random walk algorithm by a data analysis for case 4 in Subsection VI-C. The first row of Table IV shows the similarity score of each VM (without random walk), while the second column shows the root cause probability of each VM, *i.e.*, considering both similarity and random walk. In case 4, we use the same tool and algorithm to increase the CPU utilization of vm_6 and vm_8, so they have approximately same similarity score, *i.e.*, 0.567 and 0.599. We can also see that the similarity score of vm_1 is 0.052 which is very low. It indicates that vm_1 is not affected by vm_6's high CPU utilization. In other words, vm_6 is not a root cause for T_A's slow response. This is consistent with our result of random walk. From the second row of the table, we can see that the root cause probabilities of them given by random walk algorithm are 0.067 and 0.403. By the random walk algorithm, we exclude the vm_6 from the possible root cause list.

The necessity and value of random walk is also evaluated in large-scale clouds. We use the same simulation parameters as the experiments in Figure 15. Figure 16 shows MAP, AFP and MFP of the system with random walk and without random walk. Obviously, the system with random walk outperforms the other one in all cases. The MAP improvement when using random walk is 2.46%, 23.69%, 25.54% and 26.14% in different $\mathbb{N}_v^f$. It also proves the necessity of the random walk over APG.

Therefore, it is necessary for us to use the random walk to determine the probability of being the root cause, because it can consider the possibility that a VM propagates its anomaly through the APG and return the probability more precisely.

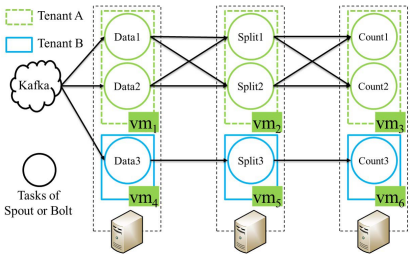


Fig. 17. Two tenants' Storm applications in a cloud.

VII. ROOT CAUSE ANALYSIS AT THE PROCESS LEVEL

In the previous sections, we mainly focus on finding out root cause at the VM level, that is we can only localize which VM is anomalous. But in some scenarios, there are VMs that host multiple server processes. For example, Figure 17 shows two tenants T_A and T_B both run a Storm [34] application Word Count Topology(stream version) [35] in a cloud. In this scenario, there are two processes *i.e.*, tasks running on each VM of T_A at the same time. If the performance of topology of T_A degrades, we need to find out root cause at the process level. In other words, we need to localize which process behaves anomalously.

A Storm application is modeled as a directed graph called a topology, which usually includes two types of components: spouts and bolts. A spout is a source of data stream, while a bolt consumes tuples from spouts or other bolts, and processes them in the way defined by user code. Spouts and bolts can be executed as many tasks in parallel on multiple virtual machines *i.e.*, worker nodes in a cluster.

The application in Figure 17 has a chain-like topology with one spout and two bolts. The Data spout is a consumer of the distributed streaming platform *Kafka* [36]. The Data spout subscribes one of the topics of Kafka, Kafka will send data to the spout automatically. The Data spout is connected to a SplitSentence bolt which splits each line into words and feeds them to a WordCount bolt using fields grouping. The WordCount bolt increments counters based on distinct input word tuples.

T_A sets the parallelism of the spout and bolts as 2, that is for every spout or bolt, there are two tasks running at the same time in vm_1, vm_2 and vm_3. While T_B set the parallelism of the spout and bolts as 1.

In this scenario, as there are two processes running on the same VM, the cloud provider cannot use PreciseTracer to do request tracing and collect metrics data directly. Instead, Storm provides the Thrift API for users to monitor the topology running in it without intrusiveness to tenants' service.

Firstly, the cloud provider can use function *getExecutors* to get process list of T_A's Word Count Topology. For each process, cloud provider can use function *getComponent_id* to get name of the process and use function *getHost* to find out the VM where the process runs. Based on the process name, cloud provider can construct the communication relationship of these processes. For example, there will be an edge between the process named SplitSentence and the process named WordCount because the SplitSentence bolt will send words to WordCount bolt. And based on the VM name, cloud provider can find out co-located VMs of T_B by the API of the cloud platform. Given these two relationships, cloud provider can construct the APG as shown in Figure 18. We need to notice

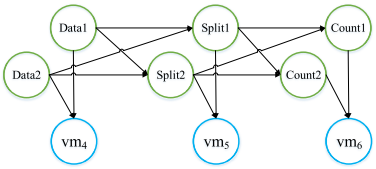


Fig. 18. The APG corresponding to Figure 17.

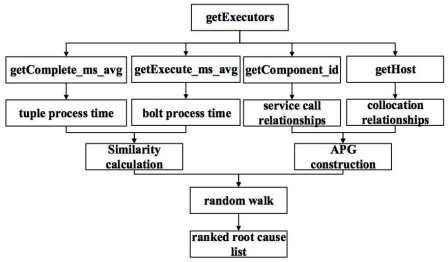


Fig. 19. **Root** cause analysis procedure at the process level.

that some nodes, *i.e.*, the green circles in the APG stand for processes, so the cloud provider can do root cause analysis at the process level. Then the cloud provider can use the function *getComplete_ms_avg* to get total tuple process time (the time from Data spout receiving the sentence to WordCount bolt finishing counting of the words of the sentence) and use the function *getExecute_ms_avg* to get the time spent by every bolt process. Using these metrics data, cloud provider can calculate the similarity score of those nodes, *i.e.*, the green circles which stand for processes in the APG. In the meantime, cloud provider can calculate the similarity score of the rest nodes, *i.e.*, the blue circles which stand for co-located VMs of T_B based on resource assumption of these VMs. At last, cloud provider can do the random walk over APG to find out the root cause list and corresponding probability. The root cause analysis procedure at the process level is shown in Figure 19.

The experiments and their results are similar to the experiments in Section VI. Due to page limitations, we skip the details of experiments and results.

VIII. Conclusion

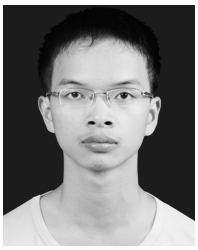
In this paper, we propose a solution for a public cloud provider to help its tenants to localize the root causes of anomalies of multitier services. Our solution consists of two parts: a data collection subsystem and a root cause localization subsystem. The data collection subsystem is running continuously to collect data to be used when anomalies occur. Since our solution is proposed from the aspect of cloud providers, we particularly design the data collection subsystem to be non-intrusive to tenants. This makes our root cause analysis system more feasible to be deployed in public clouds. The root cause localization subsystem is responsible to find out a possible root cause VM list when an anomaly occurs. Our solution is able to find both factors which can cause anomalies in public clouds: software bugs of the anomalous service components and performance interference from other tenant. We argue that cloud providers are in a better position to diagnose anomalies caused by performance interference. Our consideration about interference among tenants is essentially valuable for tenants to localize the root causes of anomalies and improve the quality of their services.

We implement and deploy the system in our real-world small-scale cloud platform and conduct experiments on a three-tier web application and a Storm topology to show that our method can find out root causes at both VM and process level. We also conduct simulation experiments to do root cause analysis of anomalies that arise from multiple causes. Experimental results demonstrate that our solution outperforms previous works.

References

[1] Y. Koh *et al.*, "An analysis of performance interference effects in virtual environments," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Apr. 2007, pp. 200–209.
[2] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, "Workload-aware provisioning in public clouds," *IEEE Internet Comput.*, vol. 18, no. 4, pp. 15–21, Jul. 2014.
[3] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, "Bobtail: Avoiding long tails in the cloud," presented at the 10th USENIX Symp. Networked Syst. Design Implement. (NSDI), 2013, pp. 329–342.
[4] H. Nguyen, Z. Shen, Y. Tan, and X. Gu, "FChain: Toward black-box online fault localization for cloud systems," in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2013, pp. 21–30.
[5] N. Marwede, M. Rohr, A. van Hoorn, and W. Hasselbring, "Automatic failure diagnosis support in distributed large-scale software systems based on timing behavior anomaly correlation," in *Proc. 13th Eur. Conf. IEEE Softw. Maintenance Reeng. (CSMR)*, Mar. 2009, pp. 47–58.
[6] K. Wang, "A methodology for root-cause analysis in component based systems," in *Proc. IEEE 23rd Int. Symp. Quality Service (IWQoS)*, Jun. 2015, pp. 243–248.
[7] J. Mace, R. Roelke, and R. Fonseca, "Pivot tracing: Dynamic causal monitoring for distributed systems," in *Proc. ACM 25th Symp. Oper. Syst. Princ.*, 2015, pp. 378–393.
[8] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 41, no. 1, pp. 93–104, 2013.
[9] J. Lin, Q. Zhang, H. Bannazadeh, and A. Leon-Garcia, "Automated anomaly detection and root cause analysis in virtualized cloud infrastructures," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Apr. 2016, pp. 550–556.
[10] (2009). *RUBiS: Rice University Bidding System*. [Online]. Available: http://rubis.ow2.org/
[11] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using magpie for request extraction and workload modelling," in *Proc. OSDI*, vol. 4, 2004, p. 18.
[12] E. Thereska *et al.*, "Stardust: Tracking activity in a distributed storage system," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 34, no. 1, pp. 3–14 , 2006.
[13] B. H. Sigelman *et al.*, "Dapper, a large-scale distributed systems tracing infrastructure," Google, Mountain View, CA, USA, Tech. Rep. dapper-2010-1, Apr. 2010.
[14] P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, and A. Vahdat, "WAP5: Black-box performance debugging for wide-area systems," in *Proc. ACM 15th Int. Conf. World Wide Web*, 2006, pp. 347–356.
[15] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance debugging for distributed systems of black boxes," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 74–89, 2003.
[16] B. Sang *et al.*, "Precise, scalable, and online request tracing for multitier services of black boxes," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 6, pp. 1159–1167, Jun. 2012.
[17] Z. Wang, W. Dong, W. Zhang, and C. W. Tan, "Rumor source detection with multiple observations: Fundamental limits and algorithms," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, pp. 1–13, 2014.
[18] Z. Wang, W. Dong, W. Zhang, and C. W. Tan, "Rooting our rumor sources in online social networks: The value of diversity from multiple observations," *IEEE J. Sel. Topics Signal Process.*, vol. 9, no. 4, pp. 663–677, Jun. 2015.
[19] W. Dong, W. Zhang, and C. W. Tan, "Rooting out the rumor culprit from suspects," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2013, pp. 2671–2675.
[20] P.-D. Yu, C. W. Tan, and H.-L. Fu, "Rumor source detection in finite graphs with boundary effects by message-passing algorithms," in *Proc. IEEE/ACM Int. Conf. Adv. Soc. Netw. Anal. Mining*, Aug. 2017, pp. 86–90.

[21] D. J. Dean *et al.*, "PerfCompass: Toward runtime performance anomaly fault localization for infrastructure-as-a-service clouds," in *Proc. HotCloud*, 2014, pp. 16:1–16:14.
[22] D. J. Dean *et al.*, "PerfCompass: Online performance anomaly fault localization and inference in infrastructure-as-a-service clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 6, pp. 1742–1755, Jun. 2016.
[23] A. Goel, S. Kalra, and M. Dhawan, "GRETEL: Lightweight fault localization for OpenStack," in *Proc. ACM 12th Int. Conf. Emerg. Netw. Exp. Technol.*, 2016, pp. 413–426.
[24] D. Sharma, R. Poddar, R. Poddar, M. Dhawan, and V. Mann, "Hansel: Diagnosing faults in openStack," in *Proc. 11th ACM Conf. Emerg. Netw. Exp. Technol.*, 2015, p. 23.
[25] H. Mi, H. Wang, Y. Zhou, M. R.-T. Lyu, and H. Cai, "Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1245–1255, Jun. 2013.
[26] B.-C. Tak *et al.*, "vPath: Precise discovery of request processing paths from black-box observations of thread and network activities," in *Proc. USENIX Annu. Tech. Conf.*, 2009, pp. 19:1–19:14.
[27] J. Weng, J. H. Wang, J. Yang, and Y. Yang, "Root cause analysis of anomalies of multitier services in public clouds," in *Proc. IEEE/ACM 25th Int. Symp. Quality Service (IWQoS)*, Jun. 2017, pp. 1–6.
[28] *SystemTap*. Accessed: Jan. 2017. [Online]. Available: https://sourceware.org/systemtap/
[29] *Ganglia*. Accessed: Jan. 2017. [Online]. Available: http://ganglia.info/
[30] *sFlow*. Accessed: Jan. 2017. [Online]. Available: http://www.sflow.org/
[31] *Lookbusy*. Accessed: Jan. 2017. [Online]. Available: http://www.devin.com/lookbusy/
[32] *OpenStack*. Accessed: Jan. 2017. [Online]. Available: http://docs.openstack.org/
[33] *CPU-Load-Generator*. Accessed: Jan. 2017. [Online]. Available: https://github.com/beloglazov/cpu-load-generator
[34] *Storm*. Accessed: May 2017. [Online]. Available: http://storm.apache.org/
[35] *WordCountTopology*. Accessed: May 2017. [Online]. Available: http://t.cn/RajHPwZ
[36] *Kafka*. Accessed: May 2017. [Online]. Available: http://kafka.apache.org/
[37] R. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011.
[38] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities," in *Proc. Int. Conf. High Perform. Comput. Simulation (HPCS)*, 2009, pp. 1–11.
[39] S. K. Garg and R. Buyya, "NetworkCloudSim: Modelling parallel applications in cloud simulations," in *Proc. 4th IEEE Int. Conf. Utility Cloud Comput. (UCC)*, Dec. 2011, pp. 105–113.
[40] *Complexity Analysis*. Accessed: Dec. 2017. [Online]. Available: https://stackoverflow.com/questions/12474398/what-is-pageranks-big-o-complexity



Jianping Weng received the B.Sc. degree from the Beijing University of Posts and Telecommunications in 2015 and the M.Sc. degree from Tsinghua University in 2018. He will join Alibaba Co., Ltd., in 2018. His research interests include cloud computing and big data applications.

Jessie Hui Wang received the Ph.D. degree in information engineering from The Chinese University of Hong Kong in 2007. She is currently an Assistant Professor with Tsinghua University. Her research interests include Internet routing, traffic engineering, network measurement, cloud computing, and Internet economics.

Jiahai Yang (M'99) received the B.Sc. degree in computer science from the Beijing Technology and Business University, and the M.Sc. and Ph.D. degrees in computer science from Tsinghua University, Beijing, China. He is currently a Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include network management, network measurement, network security, Internet routing, cloud computing, and big data applications. He is a member of the ACM.

Yang Yang received the Ph.D. degree from Tsinghua University in 2017. His research interests include computer network, routing protocols, traffic engineering, and software defined network.