Root Cause Analysis of Anomalies of Multitier Services in Public Clouds

Jianping Weng, Jessie Hui Wang, Jiahai Yang, Yang Yang
Tsinghua National Laboratory for Information Science and Technology
Institute for Network Sciences and Cyberspace, Tsinghua University

***Abstract*—Anomalies of multitier services running in cloud platform can be caused by components of the same tenant 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 cloud providers are in a better position than 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. We design a non-intrusive method to capture the dependency relationships of components, which improves the feasibility of root cause localization system. Our solution can find out root causes no matter they are in the same tenant as the anomaly or from other tenants. Our proposed two-step localization algorithm exploits measurement data of both application layer and underlay infrastructure and a random walk procedure to improve its accuracy. Our real-world experiments of a three-tier web application running in a small-scale cloud platform show a 38.9% improvement in mean average precision compared to current methods.**

I. Introduction

Nowadays, more and more multitier services or cloud applications are adopting IaaS clouds to manage their service infrastructures. But in public clouds, guest VMs will contend for the shared resources, which can result in performance interference [1][2][3] between tenants. So, multitier services running inside the IaaS cloud are prone to performance anomalies due to not only software bugs of multitier services components but also performance interference between tenants.

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][5][6][7]. These solutions assume there is no outside influence that can interfere with their resources, so they cannot work well in public cloud environment. Some researchers try to take performance interference into consideration [8][9]. But the problem has not been well solved. [8] only considers performance interference caused by components of their own service. [9] can only find out root causes of part of anomalies because it only collects and uses resource utilization of infrastructure.

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 we argue that cloud providers are in a better position to diagnose these anomalies. In previous works [4][5][6][7][8], they need to modify the application code or know dependency relationships of 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.

In this paper, we propose a solution for a public cloud provider to help its tenants to locate 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. To determine the root causes, we define a metrics called similarity score, which is calculated based on both the application-level metrics data and resource utilization of underlay infrastructure. We also 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 some experiments on a three-tier web application. By the experiments, we show the rationality and necessity of two steps in our localization algorithm: similarity score and random walk propagation. Experimental results show a 38.9% improvement in mean average precision compared to current methods. Summarily, we make the following contributions in this paper:

- We propose a non-intrusive method to capture the complex dependency relationships of multitier components, which improves the feasibility of root cause localization system.
- Our solution can find out root causes of an anomaly no matter they are in the same tenant as the anomaly or from other tenants.
- We design a two-step localization algorithm, which is based on monitoring of both application layer and underlay infrastructure and a random walk procedure. Experiments demonstrate the algorithm outperforms previous works.

The rest 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. Section IV illustrates how to find out all possible root cause VMs. Section V shows the details of our two-step

Corresponding author: J.H. Wang (hwang@cernet.edu.cn).

978-1-5386-2704-4/17/$31.00 2017 IEEE

localization algorithm. Section VI introduces our experiment settings and evaluation results. Section VII concludes our work.

II. Related Work

In recent years, many solutions have been proposed to solve this problem from the aspect of tenants [4][5][6][7]. These works need to modify the application code or need to know dependency relationships of service components. These solutions can find out root causes of anomalies in a dedicated environment. However, more and more multitier services are deployed in public clouds, and these tenants may interfere with each other due to resource contention. Such performance interference have been studied in [1][2] and [3].

Previous works cannot diagnose anomalies caused by performance interference of other tenants. In 2013, Kim et al. [8] introduces a pseudo-anomaly clustering algorithm on historical data to capture the external factors such as the performance interference, but the work only considers the performance interference caused by components of their own service. 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. Obviously, 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. As far as we know, it is the only work that solves the issue from the aspect of cloud providers before this paper.

But the work by Lin only collects resource utilization of infrastructure and simply uses anomaly propagation distance to rank the possible root causes, so the result is not that accurate. In this paper, we calculate the similarity using metrics data of different levels and run random walk algorithm to determine the root causes.

III. Two Types of Anomaly Propagation in Public Cloud

In this section, we would conduct experiments to see if the performance of one virtual machine vm_i can be affected by applications running on other virtual machines located in the same physical machine as vm_i.

We conduct experiments in a small-scale cloud shown in Figure 1. The cloud consists of 5 physical machines and there are two tenants, *e.g.* T_A and T_B. 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. Both two tenants are runing a 3-tiered web application RUBiS [10] using their VMs.

T_A's LVS (vm_1) receives users' requests, 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.

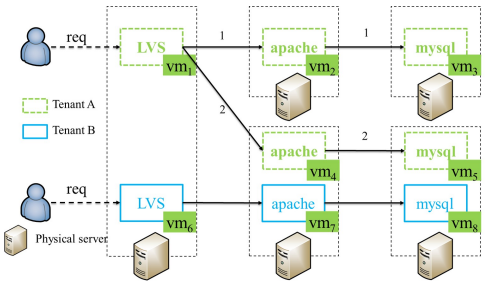

Fig. 1. Two tenants' multitier services in a cloud

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* [15], which is implemented based on the well-known tool *lookbusy* [13].

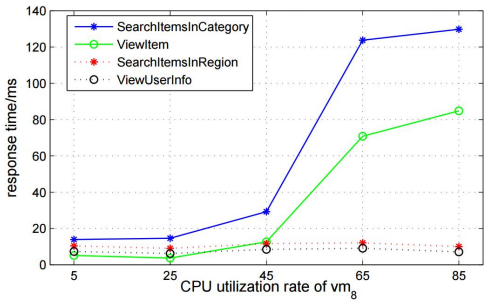

Fig. 2. Response time of tenant A's website

From Figure 2, 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 $vm5$ 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*.

So, 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.

IV. Anomaly Propagation Graph

In this section, for one anomaly under study, say anomaly a, we try to find out all possible causes of the anomaly a and construct the anomaly propagation graph (APG) 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's easy for him to retrieve information about collocated VM pairs, *i.e.*, collocation edges.

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 $Sang$ in [11] 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 the OS kernel or shared libraries, e.g. sending or receiving messages. PreciseTracer uses Systemtap [12] to capture those activities.

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. PreciseTracer records an activity of sending a message as $S_{i,j}^i$, which indicates a process i sends a message to a process j and records an activity of receiving a message as $R_{i,j}^j$, which indicates a process j receives a message from a process i.

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 3 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 $S_{1,2}^1$ and $R_{1,2}^2$ (vm_1 requests service on vm_4); and $S_{2,1}^2$ and $R_{2,1}^1$ (vm_4 responds vm_1's request). The second call is described by $S_{1,2}^{1'}$ and $R_{1,2}^{2'}$; $S_{2,1}^{2'}$ and $R_{2,1}^{1'}$. 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.

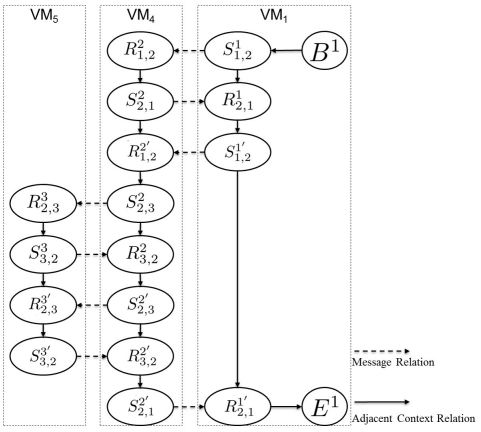

Fig. 3. A causal path graph produced by PreciseTracer

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 3 is shown in Figure 4.

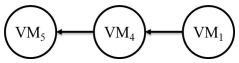

Fig. 4. The VCG of Figure 3

The challenge to transform Figure 3 into Figure 4 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.

C. APG Construction

Now we have found all collocation edges based on *nova* APIs, and we also obtained VCGs such as Figure 4 that includes all service call edges. Therefore, now we can construct 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 5.

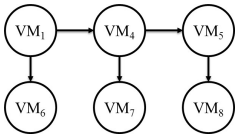

Fig. 5. The APG of *ViewItem* requests in Figure 1

V. Root Cause Localization

In this section, we will try to point out the most likely root causes from all related elements included in G_a^{APG}. We define $P_{vm_i}^a$ as the probability of vm_i being the root cause of the anomaly a.

Let us denote the root cause VM as vm_{root}. 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.

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 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. In our solution, we exploit the random walk algorithm to reflect the possibility of anomaly propagation.

A. Similarity Calculating

We define a function $\mathcal{R}(i, M, \mathbb{R}(r), t_s, t_e)$ that calculate the correlation of the metric M of vm_i and response time (RT) 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:

$$\mathcal{R}(i, M, \mathbb{R}(r), t_s, t_e) = \int_{t_s}^{t_e} \frac{\left(M_i^t - \mu_{M_i}\right)\left(RT_{\mathbb{R}(r,t)} - \mu_{RT}\right)}{\sigma_{M_i}\sigma_{RT}}\, dt,$$

wherein M_i^t means the metric M data of vm_i at time t, $RT_{\mathbb{R}(r,t)}$ means the response time of a request in $\mathbb{R}(r)$ that was issued at time t.

We then calculate the similarity $\mathbb{S}(vm_i, \mathbb{R}(r))$ based on the correlation defined above. If vm_i is a part of the multitier service components, we calculate the similarity according to the service time $\mathcal{T}_i$ of it. And for co-located VMs, we first calculate the correlations according to the different types of resource assumption, *e.g.* CPU and memory consumption, I/O and network throughput, and then select the maximum of these correlations to denote its similarity. This is because the service performance can be interfered due to different types of resources contention. The calculation formula of similarity is as follows:

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

wherein RA means the collection of different types of resources contention.

Given the formula above, we need 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 T_d when the performance of multitier service starts to degrade. If the tenant knows the T_d, we set t_s as the time point the tenant provides to us. Otherwise, we can find out T_d according to the formula as follows:

$$T_d = min\{t_p | \frac{RT_{\mathbb{R}(r,t_p)}}{f(t_p - w, t_p)} > \delta\}$$

wherein $f(t_p - w, t_p)$ is a function that can calculate the average response time of requests in $\mathbb{R}(r)$ that are issued during the time window which lasts from $(t_p - w)$ to t_p and w is the length of the time window.

B. Random Walk over APG

After identifying those VMs with high $\mathbb{S}(vm_i, \mathbb{R}(r))$, we need to further consider the possibility that those VMs propagate their anomaly through the APG. We conduct a random walk over the APG as follows. A random walker starts from a random VM, say vm_i, and moves from vm_i to one of the neighbor vm_j for some time according to a transition probability matrix Q. Q_{ij} is the probability of the random walker moving from vm_i to vm_j. The problem here is how to derive the value of Q_{ij}.

Given a $G^{APG}(V, E)$, we can construct a real value adjacency matrix A for it. A_{ij} will be proportional to Q_{ij}. We can get Q by normalizing every row of the matrix A. There are three cases to determine the value of A_{ij}. For the first case, if edge $e_{ij} \in E$, we set A_{ij} as $\mathbb{S}(vm_j, \mathbb{R}(r))$ as the random walker needs to move to vm_j proportionally to the similarity of vm_j. For the second case, if $e_{ij} \in E$ and $e_{ji} \notin E$, A_{ji} is set as $\rho\mathbb{S}(vm_i, \mathbb{R}(r))$ for some constant $\rho \in$ [0, 1). That is we add backward-edges. If without backward-edges, once the random walker falls into VMs that have low similarity, there is no way to escape out. For the last case, if $i = j$, we set A_{ij} as the similarity of vm_i subtracted by the maximum similarity score of the neighboring VMs. That is we add self-edges. If without self-edges, the random walker is enforced to move to the neighbors even if the current node shows a higher similarity and all the neighboring VMs do not. In summary, we can calculate matrix A according to the following formula.

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

Given matrix A, the transition probability matrix Q of our random walk is represented as follows.

$$Q_{ij} = \frac{A_{ij}}{\sum_j A_{ij}}$$

VI. Experiment and Evaluation

A. Experimental Environment

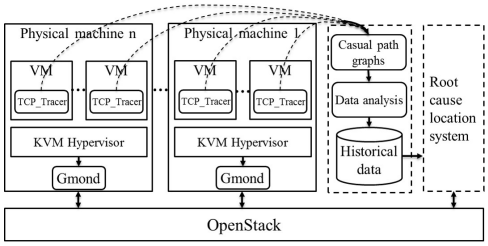

Fig. 6. Experimental environment

As shown in Figure 6, we use OpenStack [14] to do VM management and resource allocation. We use PreciserTracer [11] 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 correlates those activity logs of different VMs into causal path graphs. We then calculate the metrics data about service time of each VM through doing data analysis on the causal path graphs, and save the results to the database. We also use the Ganglia which is deploy on each physical server to collect resource utilization of each VM. The root cause localization system can find out the possible root cause list and corresponding probability if an anomaly of a request occurs.

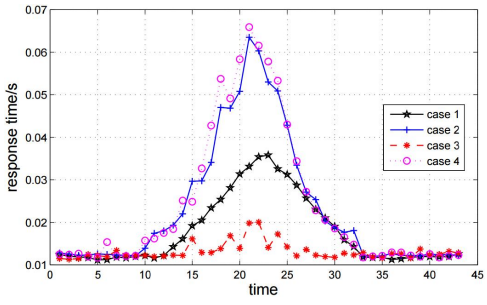

Fig. 7. Response time of different cases

B. Experiment Setup

Experiment scenario. In the experiment, we use the same scenario as shown in Section III.

Anomaly injection. We attempt to do anomaly injection to the RUBiS of T_A by 4 cases:

- Case 1: We inject some delays last from 2ms to 25ms into PHP "Search-ItemsByCategories" function on vm_4.
- Case 2: we control the CPU utilization of vm_8 from 10% up to 90% and from 90% down to 10% with lookbusy [13] to interfere the performance of mysql on vm_5.
- Case 3: Similar to case 2, we control the CPU utilization of vm_6 from 10% up to 90% and then from 90% down to 10% to interfere the performance of LVS on vm_1.
- Case 4: We combine case 2 and case 3. That is, we orchestrate both the CPU utilization of vm_8 and vm_6 at the same time in order to interfere the performance of mysql on vm_5 and LVS on vm_1.

Figure 7 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 utilization of LVS on vm_1 is so low that vm_6 cannot interfere the performance of vm_1 by CPU resource contention, the performance of the website doesn't degrade. So next we use different root cause analysis methods to find out the root causes of the three cases *i.e.,* case 1,2 and 4, to compare the accuracy of different methods.

As the 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 by vm_8, the root causes of case 2 are vm_8 and vm_5. For case 4, as shown in case 3, the increase of the CPU utilization of vm_6 doesn't interfere the performance of the website. The root causes of case 4 are vm_8 and vm_5.

C. Performance Evaluation

Baseline methods. For the purpose of comparison, we 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): A natural way for a human to find root causes is to compare the metrics in the current and previous time windows and check any sudden change between the two time windows. In SC, it calculates the ratio of average metrics on both the time periods and refers to this ratio as the root cause score of each VM.
- Distance Based Rank (DBR)[9]: DBR regards the root causes as a set of misbehaving entities whose propagation graphs best capture all the observed anomalies. And it uses a ranking method for selecting the best propagation graph. The rank of a propagation graph is determined by the minimum total distance from the source anomaly entity to all other 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 given by every method and $\Psi_a(i)$ represent 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.

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

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

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

Experiment Results. We evaluate our method and all the three baseline methods on that three cases. Figure 8 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%.

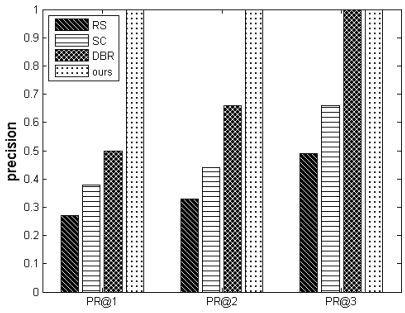

Fig. 8. Precision top K of different methods

TABLE I
MAP OF DIFFERENT METHODS

method	RS	SC	DBR	ours
MAP(%)	0.36	0.49	0.72	1.00

D. Discussion of Our Approach

We show the necessity of this random walk algorithm by a data analysis for case 4. The first row of Table II 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 VM list.

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.

TABLE II
SIMILARITY AND PROBABILITY OF EACH VM IN CASE 4

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
probability (with rw)	0.0536	0.132	0.325	0.067	0.017	0.403

VII. CONCLUSION

In this paper, we propose a solution for a public cloud provider to help its tenants to locate the root causes of anomalies of multitier services. Our method is non-intrusive to tenants and more feasible to be deployed in public clouds. Our solution is able to find both factors which can cause anomalies in public clouds: component bugs in the anomalous service, and performance interference from other tenant. 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. By the experiments, we show the rationality and necessity of two steps in our localization algorithm: similarity score and random walk propagation. Experimental results demonstrate that our solution outperforms previous works.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under Grant No. 61202356.

REFERENCES

[1] Koh, Younggyun, et al. "An analysis of performance interference effects in virtual environments." 2007 IEEE International Symposium on Performance Analysis of Systems & Software. IEEE, 2007.
[2] Xu, Yunjing, et al. "Workload-Aware Provisioning in Public Clouds." IEEE Internet Computing 18.4 (2014): 15-21.
[3] Xu, Yunjing, et al. "Bobtail: Avoiding long tails in the cloud." Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13). 2013.
[4] Nguyen, Hiep, et al. "FChain: Toward black-box online fault localization for cloud systems." Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on. IEEE, 2013.
[5] Marwede, Nina, et al. "Automatic failure diagnosis support in distributed large-scale software systems based on timing behavior anomaly correlation." Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on. IEEE, 2009.
[6] Wang, Kui, et al. "A methodology for root-cause analysis in component based systems." 2015 IEEE 23rd International Symposium on Quality of Service (IWQoS). IEEE, 2015.
[7] Mace, Jonathan, Ryan Roelke, and Rodrigo Fonseca. "Pivot tracing: dynamic causal monitoring for distributed systems." Proceedings of the 25th Symposium on Operating Systems Principles. ACM, 2015.
[8] Kim, Myunghwan, Roshan Sumbaly, and Sam Shah. "Root cause detection in a service-oriented architecture." ACM SIGMETRICS Performance Evaluation Review. Vol. 41. No. 1. ACM, 2013.
[9] Lin, Jieyu, et al. "Automated anomaly detection and root cause analysis in virtualized cloud infrastructures." Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP. IEEE, 2016.
[10] (2009) RUBiS: Rice university bidding system. http://rubis. ow2.org/
[11] Sang, Bo, et al. "Precise, scalable, and online request tracing for multitier services of black boxes." IEEE Transactions on Parallel and Distributed Systems 23.6 (2012): 1159-1167.
[12] Systemtap, http://sourceware.org/systemstap/.
[13] lookbusy, http://www.devin.com/lookbusy/
[14] OpenStack, http://docs.openstack.org/
[15] cpu-load-generator, https://github.com/beloglazov/cpu-load-generator