SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •
Progress of Projects Supported by NSFC

December 2012 Vol. 55 No. 12: 2757–2773
doi: 10.1007/s11432-012-4747-8

Localizing root causes of performance anomalies in cloud computing systems by analyzing request trace logs

MI HaiBo[1]*, WANG HuaiMin[1], ZHOU YangFan[2], LYU Michael R.[2] & CAI Hua[3]

[1]*National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha* 410000*, China;*
[2]*Shenzhen Research Institute, The Chinese University of Hong Kong, Shenzhen* 518000*, China;*
[3]*Computing Platform, Alibaba Cloud Computing Company, Hangzhou* 310000*, China*

Received June 1, 2012; accepted October 30, 2012

Abstract It is hard to localize the primary cause of performance anomalies in cloud computing systems because of the complexity of interactions between components. The hidden connections in the huge number of request execution paths in such systems usually contain useful information for diagnosing performance anomalies. We propose an approach to localize anomalous invoked methods and their physical locations by leveraging request trace logs, which involves two steps: (1) firstly, cluster the requests according to their corresponding call sequences, identify anomalous requests with principal component analysis, and then pick out anomalous methods with Mann-Whitney hypothesis test; (2) secondly, compare the behavior similarities of all replicated instances of the anomalous methods with Jensen-Shannon divergence, and select the ones whose behaviors are different from those of others, which will be chosen as the final culprits of performance anomalies. We conduct experiments with four real-world cases to validate our approach in Alibaba Cloud Computing Inc. The results demonstrate that our approach can locate the prime causes of performance anomalies with the low false-positive rate and false-negative rate.

Keywords cloud computing systems, performance anomalies, request trace logs, fault localization

Citation Mi H B, Wang H M, Zhou Y F, et al. Localizing root causes of performance anomalies in cloud computing systems by analyzing request trace logs. Sci China Inf Sci, 2012, 55: 2757–2773, doi: 10.1007/s11432-012-4747-8

1 Introduction

Due to the continuous growth of the scale and complexity of systems, it becomes more and more difficult to build software with high quality assurance [1–3]. It is beyond the engineers' ability to design sufficient test cases to cover all scenarios for the production environments. Some bugs will not manifest themselves until a specific condition occurs. Therefore, it is almost impossible to deploy a bug-free system.

Compared to the functional bugs that usually cause the breakdown of the systems, performance anomalies are harder to be diagnosed [4]. Performance diagnosis is labor-intensive, especially for the production

*Corresponding author (email: rainmhb@gmail.com)

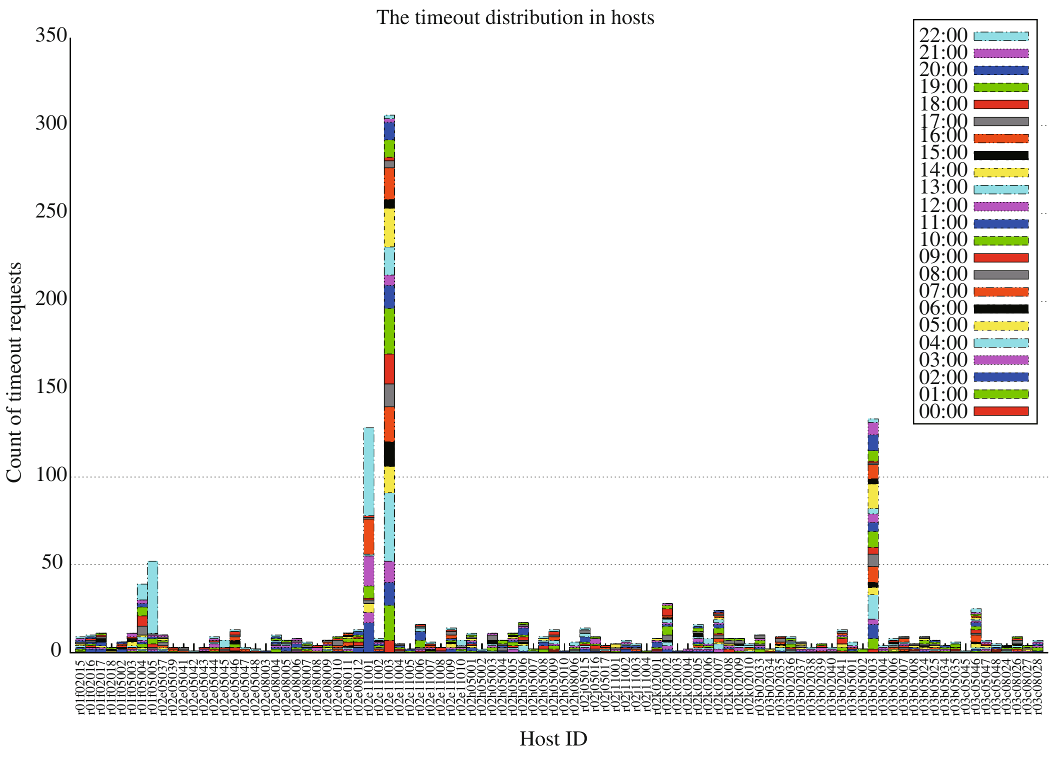

Figure 1 The spatial distribution of the time-out anomaly for one anomalous method in a production cluster for 24 hours.

cloud computing systems. Such systems generally consist of numerous software components and serve tremendous user requests simultaneously. Furthermore, to achieve elasticity, each component generally has a lot of duplicated instances. Replicated instances enhance the scalability and availability of the system, but make performance anomaly diagnosis more complex. Sometimes, when performance anomalies happen, the defect of a component is only manifested in part of instances and anomalous instances are mixed with normal ones . For example, a defective load balance policy in a storage service may result in some overloaded storage component instances, whereas, the remaining instances may behave normally. Figure 1 shows the spatial distribution of the time-out anomaly for one anomalous method in one Alibaba prodution cluster. We can see that most time-out anomalies only happened in five instances in 24 hours. In this situation, operators not only want to know which invoked methods (i.e. logical component) become anomalous, but also need to identify their instances (i.e., physical locations). Hence, when the system suffers performance degradation (e.g., the average response time of user requests increases), locating anomalous instances becomes of critical importance.

However, current approaches generally focus on locating anomalous physical nodes (e.g., [5]) or logical components (e.g., [6–8]). Such coarse-grained results are not enough. In the former case, given an anomalous physical node, system operators have to identify the faulty component among many components hosted in the physical node. In the latter case, given an anomalous logical component, the operators have to identify which ones among its tremendous instances distributed in the cloud are faulty. Consequently, huge human efforts are still required to further pinpoint the subtle root cause.

Furthermore, many existing research investigations utilize rule-based approaches (see, e.g., [9]) or expectation-based approaches (e.g., [7, 10]) to diagnose performance. Engineers are required to manually design detecting rules or expectations according to their specific domain knowledge. However, a production cloud system generally offers a lot of concurrent services and user requests because these services are complicated. Engineers have much difficulty in understanding the characteristics of component interactions. It is beyond the engineers' ability to construct such rules or expectations in cloud computing system.

This paper is an extended work of our previous research [11] that localizes the anomalous invoked methods. In this paper, without any specific domain knowledge, we aim at not only localizing anomalous methods but also the anomalous instances that are the physical locations of the anomalous methods. Since typical cloud computing systems are service-oriented, the response time of user requests naturally reflects the system performance. In this regard, an end-to-end tracing user request approach is a viable means to expose performance data so as to help performance diagnosis. Therefore, based on request trace logs, we solve the problem with two steps. First, the anomalous methods are localized with three sub-steps: (1) cluster the user requests into categories; (2)identify anomalous requests within the same category through the principal component analysis [12] and separate the normal and anomalous requests into two sets; (3) compare the behavior of the same invoked methods in normal and anomalous sets with Mann-Whitney non-parameter statistical hypothesis test [13] and pick out anomalous methods. Second, anomalous instances are localized with two additional sub-steps: (1) group the latencies of an anomalous method by the host addresses of instances and create histograms for each of them; (2) compare the similarities among these histograms with Jensen-Shannon divergence [14] and localize the histograms whose behaviors are the most different from those of others, which are considered to be the culprit of the anomalous methods.

We verify the effectiveness of our approach in the Alibaba cloud computing platform, which is a real-world enterprise-class cloud computing infrastructure providing services to the public in China. The experimental results demonstrate that our approach can locate the primary causes of performance anomalies with a low false-positive rate and false-negative rate. So far, our approach has been successfully applied in the Alibaba cloud computing platform to diagnose performance anomalies in both testing and production clusters.

The remainder of this paper is organized as follows. Section 2 compares our approach with the related work. In Section 3, we briefly introduce the workflow of our approach. Section 4 and Section 5 respectively present how to localize the anomalous methods and service instances in detail. In Sections 6 and 7, we give the experimental scenarios and results. Section 8 concludes this paper.

2 Related work

End-to-end request tracing approaches are efficient for operators to conduct performance debugging. Basically, in order to get request trace data, there are two kinds of instrumentation mechanisms: white-box based mechanism and black-box based mechanism. A white-box based mechanism (e.g., [15–17]) assumes the availability of the source codes and utilizes explicit global identifiers to correlate runtime events; while a black-box-based mechanism (e.g., [18–22]) assumes no knowledge of the source codes and adopts probabilistic correlation methods or statistical regression techniques to infer the casual paths. Since the source codes are available in typical production cloud systems, in this paper, we utilize a white-box instrumented mechanism to trace requests.

Pinpoint [23] traces request call relationship in multi-layers of Web service components and adopts a clustering algorithm to group failure and success logs. It finds out the anomalous components through dependency mining and a probabilistic context free grammar. Chen et al. [24] present a thoughtful discussion on how request tracing can help operators on the process of performance anomaly detection and diagnosis. Magpie [25] uses event schema to correlate requests and clusters requests according to the similarity of structure and timing of requests' paths. X-trace [26] constructs the causal relationship of requests through modifying the transport protocol and detects the anomaly through comparing the structure difference of requests. Pip [7] and Ironmodel [10] apply users' expectation to determine whether a request is anomalous or not. The common ground of these researches is that they use self-definition event schema or expected models to detect requests. It is very hard to construct these models because it requires much specific domain knowledge. However, our approach makes the intrinsic characteristics of trace logs to diagnose the anomalies without domain knowledge.

Spectroscope [8] groups the paths of requests by call structures and finds the anomalous requests through comparing the behaviors of requests in two time periods. This research focuses on performance changes of two time periods, while our work tries to narrow down the space of potential root causes

```
[2011-08-01 16:31:31.690272] [DEBUG] [15330] [./Example.h:117] T_ID:001 FUNC:MethodA_start
[2011-08-01 16:31:31.832724] [DEBUG] [15330] [./Example.h:117] T_ID:001 FUNC:MethodB_start
[2011-08-01 16:31:32.193131] [DEBUG] [15330] [./Example.h:122] T_ID:001 FUNC:MethodB_end
[2011-08-01 16:31:32.200272] [DEBUG] [15330] [./Example.h:117] T_ID:001 FUNC:MethodC_start
[2011-08-01 16:31:33.780275] [DEBUG] [15330] [./Example.h:122] T_ID:001 FUNC:MethodC_end
[2011-08-01 16:31:33.991376] [DEBUG] [15330] [./Example.h:122] T_ID:001 FUNC:MethodA_end
```

Figure 2 Basic structure of tracing logs.

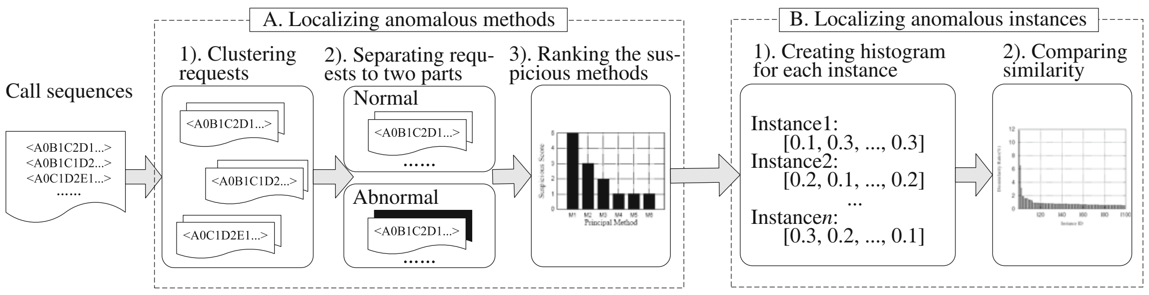

Figure 3 Overview of the approach.

of performance problems with the trace logs in one time period. Furthermore, it does not differentiate the importance of invocation methods in call graphs and exhausts all related methods to localize the abnormal ones. Magnifier [27] performs an empirical study on localizing the performance bottlenecks. Troubleshooters are required to instrument the probes conforming to the naming specifications of the system architecture. This paper is free from this restriction.

Other existing approaches aim at utilizing system resources [28–31] to learn models to detect performance anomalies; whereas, such models cannot detect performance anomalies with finer granularity (e.g., anomalous instances). Furthermore, it takes more time for them to train logs in exchange for the sufficient accuracy of inference.

3 Sketch of the approach

Our tracing mechanism explicitly instruments the target system and associates the activities of requests with global identifiers. Since the execution time of the invoked methods can directly reflect how the system performs a request, we choose to capture such execution time. Once the instrumented methods are invoked, a record with some contextual information will be kept into a log. Figure 2 shows the basic structure of the trace logs. Each line contains the current time stamp, the global identifier for the request, the name of the invoked methods, the label signifying the start or end of the invocation, and other redundant information. The call sequence of a user request can be constructed from the distributed trace logs according to the nested relationships of start/end flags. Usually, performance anomalies are directly reflected from the deviation of request latencies. Hence, these trace logs could be utilized to localize the primary causes of performance anomalies.

Our approach contains two parts. First, we try to localize the anomalous methods with three sub-steps. Second, we try to localize the anomalous instances of service components with two sub-steps. The workflow of our approach is shown in Figure 3.

3.1 Localizing anomalous methods

3.1.1 *Clustering requests*

Usually, user requests for the same service may have different types of call sequences. For example, the call sequence of reading files from the cache is different from that of reading files from the disk. Different call sequences reflect different semantics; hence, we first cluster user requests into categories according to their call sequences. The requests within one category have the same method call sequence.

3.1.2 *Separating requests to normal and anomalous sets*

The latencies of requests will be influenced when they pass through anomalous methods. We hope to localize the anomalous methods through comparing the behavior of normal requests with that of anomalous requests. Therefore, we need to identify anomalous requests within the same category, and then separate the normal and anomalous requests into two sets.

3.1.3 *Ranking suspicious methods*

Then, for invoked methods, the latency distributions in normal sets are compared with those in abnormal sets. A method is defined to be anomalous if the two latency distributions differ obviously. We pick out all suspicious methods and present the top k to operators.

3.2 Localizing anomalous instances

The target of this step is to help operators diagnose whether the behavior of methods becomes abnormal in all replicated instances or it just happens in parts of them. It helps operators further locate the primary causes of problems and understand the extent of the crisis. This process contains two sub-steps: (1) group the latencies of an anomalous method by the host addresses of instances and create histograms for each of them; (2) compare the similarities among these histograms and select the ones whose behaviors are mostly different from those of the others.

4 Localizing anomalous methods

4.1 Clustering requests

Homogeneous replicated instances provide the same instrumented methods for the public. Identical requests may pass through different instances. Although the physical locations (i.e. replicated instances) of instrumented methods are important attributes, we cannot directly consider them during clustering. Suppose a request goes through three instrumented methods. The three methods belong to three kinds of services respectively and each service has one hundred homogenous instances deployed on one hundred hosts. In total, the request has $\mathrm{C}_{100}^1 \times \mathrm{C}_{100}^1 \times \mathrm{C}_{100}^1 = 1 \times 10^6$ kinds of physical paths, except for the failure paths. Actually, the number of instrumented methods that user requests invoke is far more than three. If the host addresses are involved in the process of clustering, it will cause too much computational complexity. Hence, we first cluster requests without considering the physical information.

An incremental clustering algorithm [25] is applied. For a request i, all its relevant methods could be stitched together by the request identifier, and its call sequence $\mathrm{Seq}_i = \langle (m)_{ij} \rangle$ can be constructed. Then a corresponding string representation $m_1 m_2 \cdots m_j$ can be created from the call sequence. Requests within one cluster have the same string representation. The string representations are defined as the centroids for clusters. The distance metric is the string-edit-distance. For a new request, the cluster calculates the distance between the string representation of the request and the centroid of each cluster. The request will be added into the cluster with the zero distance, unless there is no zero distance, in which case a new cluster is created. The whole process of clustering can be finished by traversing all call sequences just one time.

4.2 Separating requests to normal and anomalous sets

When performance anomalies occur, normal and anomalous requests may share the same call sequence and be grouped into one cluster; hence, for each cluster, we hope to first identify the anomalous requests. Since we do not need domain specific knowledge, an unsupervised machine learning algorithm (i.e., principal component analysis) is utilized to achieve the goal.

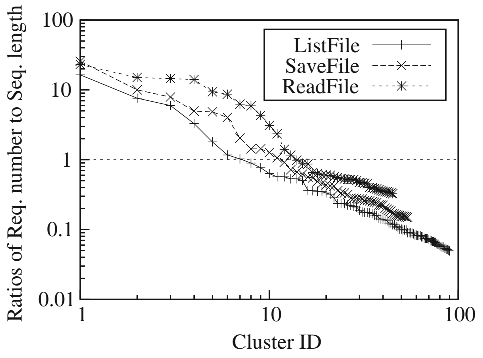

Figure 4 Ratios of the request number in each cluster to the length of corresponding call sequence.

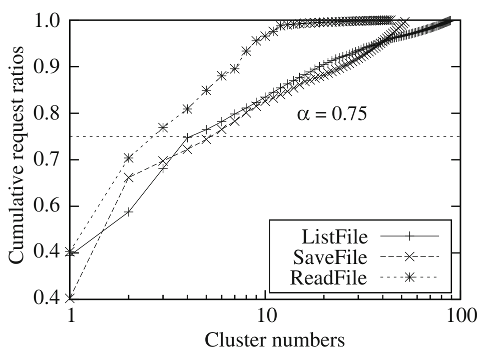

Figure 5 Cumulative ratios of the request number in each cluster to the total requests.

Principal component analysis (PCA) is a useful algorithm for high-dimensional data compression and is widely adopted in anomaly detection [32–34]. Two conditions need to be satisfied for the input data of PCA. First, the number of observations must be greater than or equal to that of their variants, i.e., PCA requires the number of requests to be greater than or equal to the length of the corresponding call sequences. Second, the high-dimensional data should have the low dimensionality. Hence, for each cluster, we have to check whether the trace data satisfies these two conditions.

4.2.1 *Checking the characteristic of the clusters*

From Figure 4, we can see that the request numbers in most clusters are smaller than the lengths of the corresponding call sequences. It means that most clusters cannot satisfy the first condition. In other words, these clusters cannot be directly utilized by PCA. This figure contains three kinds of applications in our study. The x-axis is the cluster identifier and the y-axis represents the ratio of the request number in a cluster to the length of corresponding call sequence.

Although it is statistically meaningless to use PCA for these small clusters, we cannot discard them. These clusters cannot be dropped since the summed effect of the long tail is equally important to those of large clusters. Figure 5 shows that although just a small proportion of clusters (statistically less than 10%) contains over 75% of all requests, the summed request number in left 90% clusters still takes up about 25% of all requests. The x-axis is the cluster number and the y-axis is the cumulative percent ratio. For instance, the ratios of requests for the SaveFile operation in the first two clusters are 0.40 and 0.25 respectively and the sum of the remaining ratios for the other clusters is 0.35, which has a statistically significant influence on the false-negative rate of the diagnosing process (see Section 7). Therefore the clustering result of the step one should be adjusted further in order to utilize all clusters.

The target of adjustment is to select major clusters and merge the minor ones into them. Clusters with the large ratios of requests are defined as major clusters and those with the small ratios are defined as minor clusters. There are many algorithms for us to choose; here a simple heuristic algorithm is adopted.

(1) Selecting major clusters. First, all clusters are ranked in descending order of sizes (i.e., the ratios of requests). Then, clusters are selected until their summed ratios are larger than a threshold α. These selected clusters are considered as the major clusters.

(2) Merging minor clusters into major ones. All the other clusters are traversed in order. The similarities between the centroid of each minor cluster and the ones of major clusters are computed. A minor cluster will be merged into the major one with the largest similarity. The measurement of similarity has two standards: first, the string-edit-distance is the nearest; second, the number of invoked methods in minor clusters is larger than or equal to that of major clusters.

The detailed algorithm is shown in Algorithm 1. We set the threshold α to adjust the cluster numbers, as shown in Figure 5. In our application, there are on average 3 to 5 major clusters for each kind of user request.

Then, we check if the trace data in major clusters meets the second condition. For each major cluster,

Algorithm 1: Adjustment of clusters

```
Input: $C_{seq} = \langle (c, n)_i \rangle$, a call sequence list of clusters, where $c$ is the centroid of cluster and $n$ is the
       ratio of requests.
Output: $C_{main} = \langle (c, n)_i \rangle$, a call sequence list of main clusters.
$C_{seq}.Sort(reverse = \text{True})$
$sum\_ratio = 0$;
$C_{main} = []$;
$main\_token = 0$;
/*Selecting */
for $i$ in range(len($C_{seq}$)) do
    $c\_ratio = C_{seq}[i][1]$;
    $sum\_ratio+ = c\_ratio$;
    if $sum\_ratio > \alpha$ then
        $main\_token = i$;
        $break$;
    end
end
/*Merging */
for $i$ in range($main\_token + 1$,len($C_{seq}$)) do
    $max\_sim = 0$;
    $token = 0$;
    for $j$ in range(len($C_{main}$)) do
        $sim = Similarity(C_{seq}[i][0], C_{main}[j][0])$;
        if $max\_sim < sim$ then
            $max\_sim = sim$;
            $token = j$;
        end
    end
    $C_{main}[token][1]+ = C_{seq}[i][1]$;
end
```

we construct a matrix $\boldsymbol{Y}$ with its requests. The element Y_{ij} denotes the execution time of the jth method in the call sequence of the ith request. We observe that all these matrices have the low intrinsic dimensionality. For example, Figure 6 plots the cumulative variance distribution of the invoked methods for four major clusters in our application. Just as the traffic datasets studied in [35], Figure 6 shows that a small set of principal components captures the large percent of the total variance, which demonstrates that the trace data in major clusters satisfies the requirement of PCA.

4.2.2 *Separating requests into two sets*

Determining principal components. For a cluster containing m requests, suppose there are n invoked methods for the corresponding call sequence (i.e., the length of this call sequence is n), we can construct an $m \times n$ latency matrix $\boldsymbol{Y}$. The value of each element is the response latency for the method in the corresponding request. Row i is the n-dimensional latency vector for the ith request while column j is the m-dimensional latency vector of the jth method in all call sequences. After adjusting $\boldsymbol{Y}$ to ensure that each column has the zero-mean, we apply PCA to get k principal components (PCs). Along these k PCs, the requests are captured with a majority of total variance.

Identifying anomalous requests. The set of principal components can be utilized to construct a matrix $\boldsymbol{P}$, which represents the norm subspace of the corresponding cluster. On the contrary, the remaining components can form a matrix $\boldsymbol{I} - \boldsymbol{P}\boldsymbol{P}^{\mathrm{T}}$ denoting the anomalous subspace. For each major cluster, we separate their requests into the normal set and the anomalous set according to the distances of

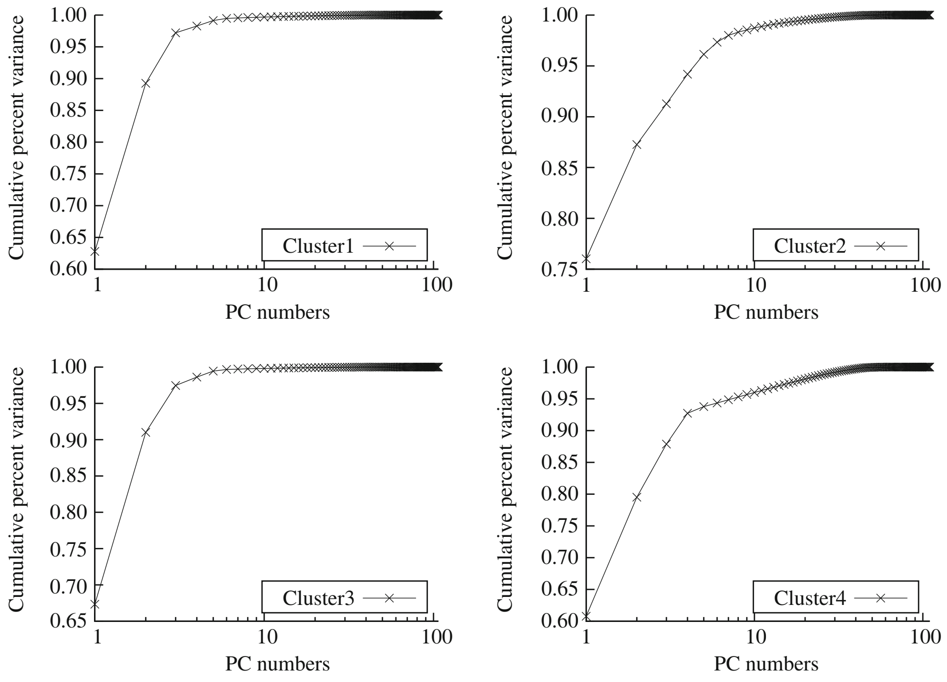

Figure 6 Cumulative percent variance of methods in different major clusters.

their projection onto the anomalous space. A request will be put into the anomalous set if its projection is larger than a given threshold, which can be computed as squared prediction error (SPE):

$$\mathrm{SPE} = \left\|(\boldsymbol{I} - \boldsymbol{P}\boldsymbol{P}^{\mathrm{T}})y\right\| > \delta_\beta, \tag{1}$$

where δ_β is the value of the Q-statistic at the $1 - \beta$ confidence level.

4.3 Localizing anomalous methods

We care about the methods whose response latencies have much fluctuation in the problem periods. However, we observe that the invocation time of many methods in user requests is stable. These methods are not the causes of performance degradation, but the noises for diagnosis. Thus we first need to filter out the noises and keep the methods whose response latencies have much fluctuation. These kept methods are then defined as the principal methods. Then, we localize anomalous methods from the principal methods.

4.3.1 *Selecting principal methods*

In this section, we introduce how to extract principal methods from all methods in major clusters. Note that the PCs (i.e., eigenvectors of $\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y}$) are not the principal methods. If high dimensional matrix can be described by the k PCs, the original matrix can be replaced by a subset of k columns with a relatively small loss of information [12]. Therefore, we utilize the B4 [36] to pick out principal columns (i.e., principal methods) from all candidate columns. The target of the B4 is to select principal methods whose regression coefficients (i.e., weights) are the largest in corresponding PCs. Each PC can be considered as the linear combination of columns (i.e., methods), and the larger the weight of a method is, the more it contributes to its corresponding PC.

For a major cluster, the selection process is as follows: first all PCs are ranked in descending order of variance; then for each principal component, the column with the largest coefficient is retained. If the column with the largest coefficient in one PC has been selected from the former PCs, the column with the second largest coefficient will be considered to pick out. The process will be iterated until all k principal methods are selected.

4.3.2 *Identifying anomalous methods*

By comparing the response latencies of each principal method in the normal and anomalous sets, we can quickly localize the ones whose performance fluctuates the most severely. For each principal method in major clusters, its corresponding latency distributions in the normal and anomalous sets are compared. We employ a standard Mann-Whitney U test [13] to quantify the difference between the two latency distributions. The null hypothesis is that the latency distributions for the normal set and the anomalous set are the same. A principal method is defined to be anomalous if its null hypothesis is rejected, i.e., the difference between the two latency distributions is calculated to be statistically obvious.

There are two reasons for us to choose the Mann-Whitney U test. First, the latency datasets of the principal methods do not belong to any particular distribution; therefore, the statistical hypothesis test should require no specific distributions for the data sets; second, in some major clusters, the number of requests in the anomalous set is small, which requires the statistical hypothesis test to consider the small and large sample volume separately. The characteristic of the Mann-Whitney U test satisfies our requirements.

A scoring scheme is applied to record the suspicious extent of principal methods. For each principal method, we assign it a score to indicate how the method is suspicious, and iterate all clusters to calculate the score. At the beginning, the score for each principal method is zero. For one cluster, if a principal method is judged to be suspicious, the score of this method will be added by one. The top k methods will be selected according to the suspicious scores. Operators are encouraged to examine the methods in order of decreasing suspiciousness.

5 Localizing anomalous instances

Next, we try to localize the anomalous instances that are the physical locations of the anomalous methods. For anomalous method of m_i, we measure the behavior similarity in all related instances. If the behavior of an instance in host h_j differs significantly from the ones in other hosts, the misbehavior of m_i is considered to be caused by the instance in host h_j.

The similarities between two instances are measured by the square roots of Jensen-Shannon divergence (JSD) [14]. JSD is widely adopted to quantify the similarity between two data sets. The latencies of the suspicious method m_i in all anomalous sets are categorized according to the instance addresses and the histogram of latencies for each category is generated. Here a latency histogram of the method m_i in instance j is defined as

$$h_{ij} = (r_1, r_2, \ldots, r_n)_{ij}, \tag{2}$$

where r_k is the ratio of the invoked times of m_i in the kth bin to the total invoked times in the instance j. For simplicity, the width of each bin is considered to be the equal and computed by the formula

$$\left\lceil \frac{\max(l) - \min(l)}{\text{binNum}} \right\rceil,$$

where l is the latency of m_i. The histograms of the method m_i for all instances have the same size of bins. For method m_i, the dissimilarity between the instance j and k is defined as

$$\begin{cases} \text{dissim}_{j,k} = \text{JSD}(h_{ij}, h_{ik}) = \frac{1}{2}D(h_{ij}, v_{ij}) + \frac{1}{2}D(v_{ij}, h_{ik}), \\ v_{ij} = \frac{1}{2}(h_{ij} + h_{ik}), \end{cases} \tag{3}$$

where $D(\boldsymbol{X}, \boldsymbol{Y})$ represents the Kullback-Leibler divergence [14] and can be computed as

$$D(\boldsymbol{X}, \boldsymbol{Y}) = \sum_i \boldsymbol{X}(i) \ln \frac{\boldsymbol{X}(i)}{\boldsymbol{Y}(i)}. \tag{4}$$

The larger the $\text{dissim}_{j,k}$ is, the more dissimilar the two instances are.

Algorithm 2: Process of computing the dissimilarity ratio for each instance

```
Input: $S = \{(h, l)_i\}$, a set of 2-tuple, where $h$ is the host address of the instance and $l$ is the response
    latency.
Output: $dissimRatio\_dic$, a dictionary of (instance, dissimilarity ratio) pair.
/* Categorizing the requests by the host addresses of instances.*/
$host\_latencyList\_dic = GroupByHostIP(S)$;
$host\_histogram\_dic = \{\}$;
for $h_i$, $latencyList_i$ in $host\_latencyList\_dic$ do
    /*Computing the histogram for each host.*/
    $hist = Histogram(latencyList_i)$;
    /*Computing the ratio of the number in each bin to total counts.*/
    $host\_histogram\_dic[h_i] = hist/len(latencyList_i)$;
end
/* Computing the JSD of histograms between the host*/
$total = 0$;
$hist\_dissimilarity\_dic = \{\}$;
for $h_i$ in $host\_histogram\_dic$ do
    $dissim_i = 0$;
    for $h_j$ in $host\_histogram\_dic$ do
        $dissim_{i,j} = JSD(h_i.hist, h_j.hist)$;
        $dissim_i += dissim_{i,j}$;
    end
    $hist\_dissimilarity\_dic[h_i] = dissim_i$;
    $total += dissim_i$;
end
/*Computing the dissimilarity ratio for each instance*/
$dissimRatio\_dic = \{\}$;
for $h_i$, $dissim_i$ in $host\_dissimilarity\_dic$ do
    $dissimRatio\_dic[h_i] = dissim_i/total$;
end
```

Then, we use the dissimilarity ratio to measure the dissimilarity extent of one instance with the others. For instance j, the dissimilarity ratio is defined as

$$\text{dissim\_ratio}_j = \frac{\sum_{k \in H} \text{dissim}_{j,k}}{\sum_{j \in H} \sum_{k \in H} \text{dissim}_{j,k}}. \tag{5}$$

Ideally, the dissim_ratio for all instances should be close to each other. If a given instance's dissim_ratio differs significantly from those of the other instances, the instance is considered to the location of the anomalous methods, which help operators narrow down the diagnosing scope. The detailed measuring algorithm is described in Algorithm 2.

6 Evaluation

Our approach has been applied in Alibaba Cloud Computing Inc. to diagnose anomalies when the performance of systems degrades. In this section, we mainly describe the experimental environment and four scenarios of performance anomalies.

Table 1 Key factors influencing the diagnosing results

Factor	Value	Description
α	0.75	The ratio of the request numbers in main clusters to the total requests.
γ	0.95	The required variance for the trace logs in PCA.
β	0.05	The significant level for the anomalous request detection in PCA.
λ	0.05	The significant level for Mann-Whitney U test.

6.1 Experimental setup

6.1.1 *Cluster sizes and workloads*

The following experiments are based on the Alibaba cloud computing platform, which contains a series of service components, such as distributed scheduler, storage, communication, and monitor. The application that we use is a file-sharing service, which is deployed on the distributed computing infrastructure. When the performance (i.e., response latencies) of the application degrades under the steady load, we try to detect the primary causes. The effectiveness of our approach is studied in a testing cluster of 100 nodes in Alibaba Cloud Computing Company. Three types of user operations are applied, i.e., SaveFile, ListFile, and ReadFile.

6.1.2 *Performance problems scenarios*

Many factors can cause performance degradation of cloud computing systems. We will evaluate our approach by four case scenarios. All cases are the typical real-world performance anomalies that happened in Alibaba cloud computing platform. The first three performance anomalies are selected from the bug repository of the Alibaba cloud computing platform. We re-inject these anomalies into the system and validate whether our approach can diagnose these problems or not. The last performance anomaly occurs in the production cluster. With our approach, an operator conducts the diagnosing process.

1) Misconfiguration. In a configuration file, the parameter controlling ON/OFF of merging small files is set to OFF, which causes the performance of the ListFile to degrade sharply.

2) Failover. A small portion of service instances used for storing files is killed. In the process of service recoveries, the performance of the SaveFile decreases significantly.

3) Code bug. When a service component is upgraded, a deprecated method that adds an extra cost of the ReadFile is accessed due to the mistake of a developer. It causes the average response latencies of the ReadFile to rise obviously.

4) Design defect. Most SaveFile requests are centralized to a small portion of hosts for a period of time due to a defect of the load balance mechanism. Hence, the average response latencies increase about two times larger than the normal.

7 Evaluation of results

We use the metrics of false-positive rate and false-negative rate to evaluate the results of localizing anomalous methods. The false-positive rate is defined as

$$\frac{\text{num of misunderstood\_methods}}{\text{num of normal\_methods}}, \tag{6}$$

i.e., the ratio of the number of normal methods which are mistaken for the anomalous to the total normal methods. The false-negative rate is defined as

$$\frac{\text{num of leaked\_methods}}{\text{num of anomalous\_methods}}, \tag{7}$$

i.e., the ratio of the number of abnormal methods which are mistaken for the normal to the total anomalous methods.

Table 2 Results of localizing the anomalous methods

Scenario	Ratio of major clusters to total clusters (%)	Cluster adjustment (Y/N)	Ratio of principal methods to total methods (%)	Ratio of involved requests to total requests (%)	False negative rate (%)	False positive rate (%)
Case scenario 1	3.3	N	3.0	73.1	26.1	2.7
		Y	4.1	100	5.9	2.1
Case scenario 2	5.6	N	2.3	69.0	31.4	5.1
		Y	4.5	100	7.6	4.2
Case scenario 3	7.1	N	2.9	78.3	21.7	3.9
		Y	3.7	100	7.6	2.1
Case scenario 4	5.2	N	4.7	69.6	21.4	4.3
		Y	6.1	100	12.7	5.4

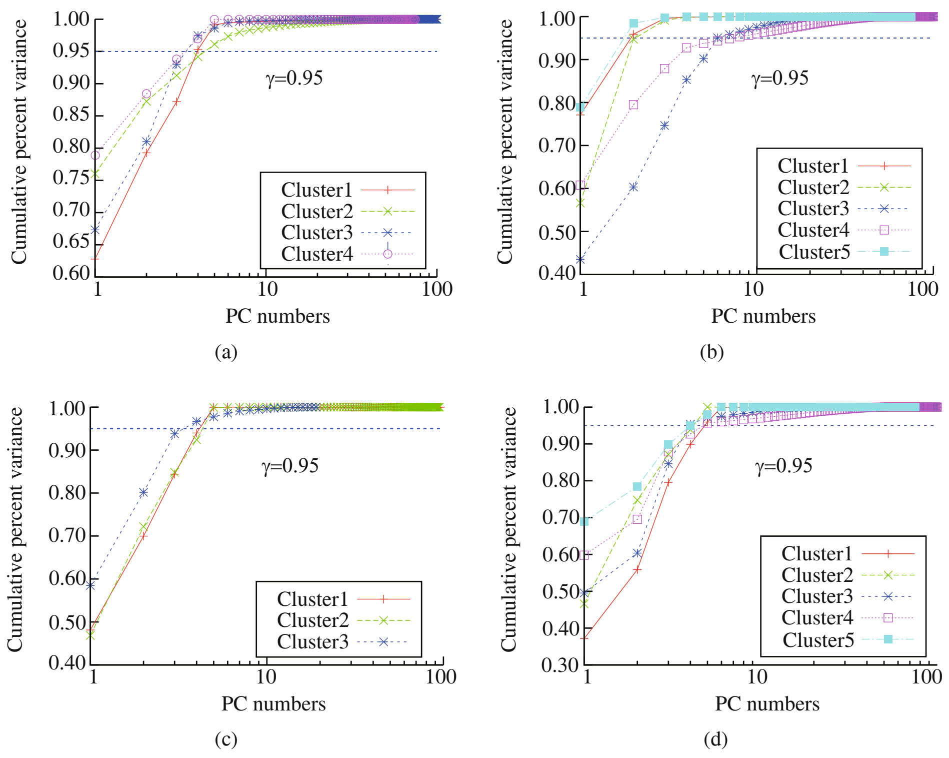

Figure 7 Principal methods selection in different scenarios. (a) Scenario 1; (b) scenario 2; (c) scenario 3; (d) scenario 4.

There are four factors influencing the diagnosing results, as shown in Table 1. Empirically, we set α to 0.75. How to select the most suitable values for the other factors has been well studied in their relevant research areas [13, 35]. Here, we just use the recommended values for them.

Table 2 summarizes the results of localizing the anomalous methods in different scenarios. From Table 2 we can see that about 30% requests will be discarded without adjusting clusters. It causes the high false-negative rate (in each case, the false-negative rate is above 20%). However, after merging the minor clusters into the major ones, the false-negative rate decreases obviously. For instance, in case 2, the false-negative rate decreases by 23.8%.

Figure 7 shows the cumulative percent variance of components for the four scenarios. The top k components whose summed variance is larger than the threshold (i.e., 0.95) are considered as the principal components. From these principal components, we can extract the principal methods. The number of principal methods equals that of principal components.

Figure 8 lists the suspicious scores of the principal methods for each scenario. All related method

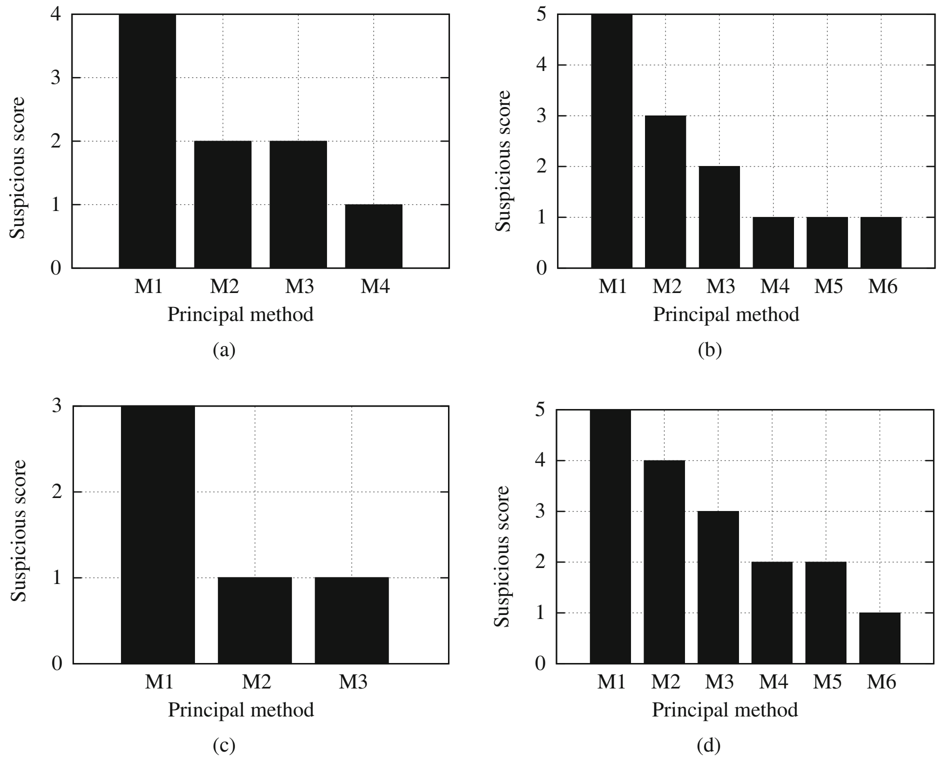

Figure 8 Suspicious scores of principal methods in different scenarios. (a) Scenario 1; (b) scenario 2; (c) scenario 3; (d) scenario 4.

names have been replaced by the numbers due to the confidentiality. For space consideration, we just list the dissimilarity ratios of the instances for the top one anomalous method in each scenario, as shown in Figure 9. The ratios have been ranked in a descending order.

7.1 Case scenario 1

Performance anomalies due to misconfigurations are frequently encountered in large scale systems like a production cloud. But they are very difficult to be troubleshot. It is hard to correlate the performance degradation to the relevant misconfigured parameters. This case reports how our approach helps engineers debug misconfiguration-related performance anomalies.

In our target system, the distributed file system is built with Hadoop-like architecture. A mechanism is provided to merge small files into a bigger one in case of excessively accessing disks. In our experiment, we disable this merging mechanism through setting the relevant parameter of the configuration file to be OFF. Therefore, the meta data of users increases over time, which causes the response latency of the ListFile to rise significantly. We try to validate whether our approach can localize the method of reading user meta data that correlates with this performance degradation.

Figure 7(a) shows that there are four major clusters in this case. In each cluster, the summed variance of the first three principal components captures more than 0.95 of variance in the data. Therefore, for each major cluster, about four principal methods are picked out. The suspicious scores of the top four principal methods are listed in Figure 8(a). The top one method is used to fetch the user meta data. Its suspicious score is four, which means that it is considered to be suspicious in each major cluster. The experimental results confirm that our approach localizes the anomalous method precisely. Furthermore, the dissimilarity ratios of instances for the top one anomalous method are shown in Figure 9(a). We can see that there are three instances whose behavior are significantly different from others. In other words, the behavior of the top one method manifests to be anomalous in these three replicated instances. It gives operators very meaningful clues to diagnose the problems more effectively. Without this information,

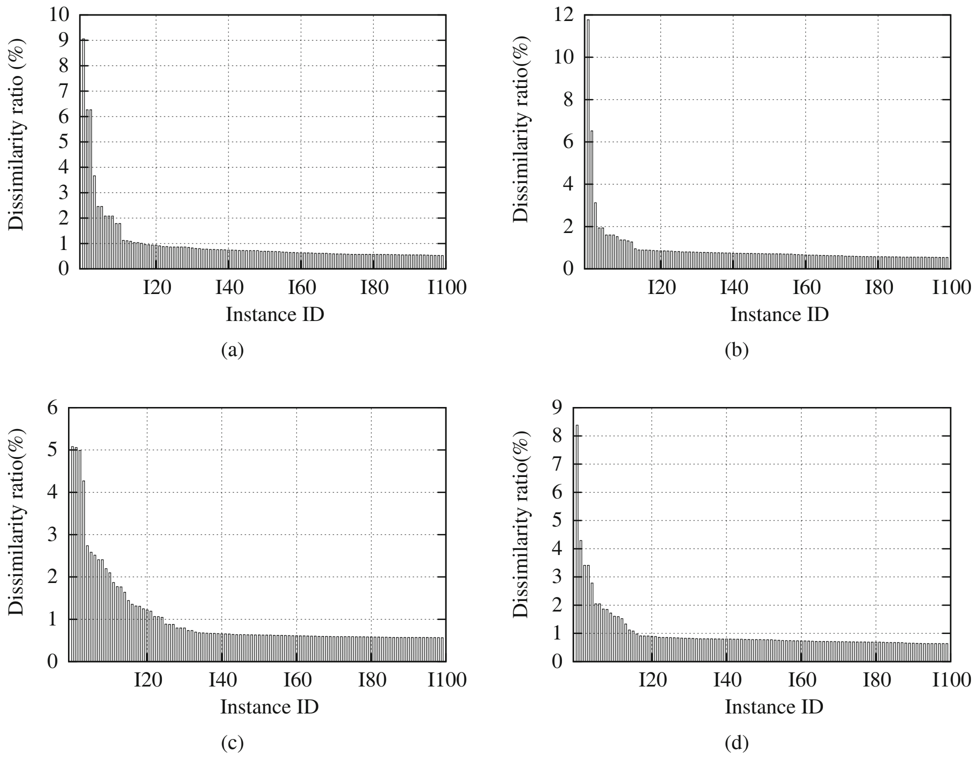

Figure 9 Dissimilarity ratios of instances in different scenarios. (a) Scenario 1; (b) scenario 2; (c) scenario 3; (d) scenario 4.

operators have to check all corresponding instances, which will suffer poor efficiency.

As is shown in Table 2, the number of principal methods takes up only 4.1% of the total invoked methods, which sharply decreases the localization scopes. Without adjusting the clusters, the false-negative rate and false-positive rate are 26.1% and 2.7% respectively; however, after merging the minor clusters into the major ones, the false-negative rate decreases to 5.9% and the false-positive rate declines to 2.1%.

7.2 Case scenario 2

In large-scale cloud computing systems, component failures are the norm rather than the exception [37]. In case of service inaccessibility, the distributed platform provides a failover mechanism to recovery the killed instances. It will influence the system performance during the recovering process. In our experiment, we manually kill one fourth of chunk-server service instances that store files into disks. Then, the response latencies of most SaveFile operations clearly increase. We try to check whether our approach can detect the methods that reflect the recovery process.

As is shown in Figure 7(b), about three to five principal methods can be picked out from more than one hundred of methods for each major cluster. There are in total eight different principal methods and Figure 8(b) lists the suspicious scores of the top six methods. The top three methods have direct relationship with chunk server services. Because the rest chunk server instances suffer more load pressures than before, excessive writing requests cannot be handled in time. Our approach finds out that these methods play a major role in the performance degradation for the SaveFile. The scores of the right three methods all equal one, which are the false-positives of our approach. Figure 9(b) lists the dissimilarity ratios of instances for the top one anomalous method. Among them, the first two instances are the most questionable because they suffer more accesses; thus our approach helps operators narrow down the localization scope effectively.

The quantitative results are shown in Table 2. Through adjusting the clusters, the false-negative rate decreases from 31.4% to 7.6% and the false-positive rate declines from 5.1% to 4.2%, which again demonstrates the effectiveness of our approach.

7.3 Case scenario 3

In cloud systems, components are usually developed by different teams. When engineers with one team invoke public interfaces of another component developed by another team, they may invoke unsuitable methods that cause extra performance costs. It is hard to detect such faults especially when the function is correct. In this case study, we investigate how our approach can be utilized to help the developer diagnose such performance anomalies.

During the running of the system, we manually inject a fault through the hot patch. The fault causes clients of the storage service to call an interface with extra authorization logic. It costs more than two times extra overheads due to unnecessarily adding an authorization process for clients. In the process of performance regression tests, the performance of the ReadFile decreases dramatically.

In this scenario, the number of major clusters is three and each of them contains about four principal methods, as is shown in Figure 7(c). Figure 8(c) shows that the relevant method is precisely found out (i.e., the one with the highest score). The developer can easily find the mistake according to the clue. Furthermore, the dissimilarity ratios of instances for this method are shown in Figure 9(c). The first three instances differ significantly from others. We can know that the influence scopes of faults in cloud computing system are different, and that without finding out the exact anomalous instances, the debuggers will have to spend more efforts on pinpointing the culprit.

In Table 2, we can see that if requests within the minor clusters are dropped, it will cause the increase of the false-negative rate obviously. It means that the anomalous methods within the minor clusters will not be fully identified without adjusting the clusters.

7.4 Case scenario 4

This scenario is a real-world diagnosing process in one production cluster in Alibaba Cloud Computing Company. Performance bugs related to user behaviors are hard to detect in the testing environment. These bugs will not be trigged until specific user behavior occurs. Our approach is applied by an operator to diagnose this problem.

The average latencies of the SaveFile increase about two times and this situation lasts several hours. With our approach, the most suspicious methods are picked out, as is shown in Figure 8(d). The first one is used to lock the file ID before the transaction of saving a file begins. Figure 9(d) shows that the dissimilarity ratio of the first instance is sharply high. It means that the instance is the physical locations of the top one anomalous method. Along the clue, he finds the root cause efficiently. Because the older load balance mechanism does not consider the access patterns adequately, it causes more than 60% of accesses to be centralized into that instance. This causes the performance of the SaveFile operation to decline significantly in that period. The diagnosis process again helps operators localize the key issues effectively.

From Table 2, we can see that through adjusting the clusters, the false-positive rate increases slightly; however, the suspicious scores of the normal methods that are mistaken for the anomaly are low (just rank top 5 and 6 in Figure 8(d)); therefore, it will not influence the diagnosing result. Furthermore, the false-negative rate decreases from 21.4% to 12.7%.

8 Conclusions

When a system performance anomaly occurs, it is generally a labor-intensive task for operators to locate anomalous parts of the system. Isolating the physical locations (i.e., instances) of anomalous methods could tremendously reduce the overall manual work in identifying the root cause.

Performance anomalies always cause the change in response latencies of user requests. The hidden connections among the huge amount of runtime request execution paths usually contain useful information for diagnosing performance problems. In this paper, we propose an approach to localize the anomalous methods as well as their physical locations by engaging request trace logs. The approach requires no specific domain knowledge for the operators. To highlight the effectiveness of the approach, we report our experiences to diagnose four real-world performance anomalies that occurred in Alibaba cloud computing platform. The experimental results show that our approach can locate the primary causes of performance anomalies with low false-positive rate and false-negative rate.

Acknowledgements

This research was supported by National Basic Research Program of China (Grant No. 2011CB302600), National High Technology Research and Development Program of China (Grant No. 2012AA011201), National Natural Science Foundation of China (Grant Nos. 61161160565, 90818028, 91118008, 60903043). Yangfan Zhou and Michael Lyu's work was supported by National Natural Science Foundation of China (Grant No. 61100077), Basic Research Program of Shenzhen (Grant No. JC201104220300A), and Research Grants Council of Hong Kong (Project No. N_CUHK405/11).

References

1 Lu X, Wang H, Wang J, et al. Internet-based virtual computing environment: beyond the data center as a computer. Futur Gener Comp Syst, 2013, 29: 309–322
2 Han S, Dang Y, Ge S, et al. Performance debugging in the large via mining millions of stack traces. In: Proceedings of the 34th International Conference on Software Engineering, Zurich, 2012. 176–186
3 Chilimbi T, Liblit B, Mehra K, et al. Holmes: Effective statistical debugging via efficient path profiling. In: 31st IEEE International Conference on Software Engineering, Vancouver, 2009. 34–44
4 Killian C, Nagaraj K, Pervez S, et al. Finding latent performance bugs in systems implementations. In: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM, 2010. 17–26
5 Lan Z, Zheng Z, Li Y. Toward automated anomaly identification in large-scale systems. IEEE Trans Parallel Distrib Syst, 2010, 21: 174–187
6 Malik H, Adams B, Hassan A. Pinpointing the subsystems responsible for the performance deviations in a load test. In: Proceedings of 21st IEEE International Symposium on Software Reliability Engineering, San Jose, 2010. 201–210
7 Reynolds P, Killian C, Wiener J, et al. Pip: Detecting the unexpected in distributed systems. In: Symposium on Networked Systems Design and Implementation, San Jose, 2006, 115–128
8 Sambasivan R, Zheng A, De Rosa M, et al. Diagnosing performance changes by comparing request flows. In: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2011. 43–56
9 Jin G, Song L, Shi X, et al. Understanding and detecting real-world performance bugs. In: The 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2012. 77–88
10 Thereska E, Ganger G. Ironmodel: Robust performance models in the wild. ACM SIGMETRICS Perform Eval Rev, 2008, 36: 253–264
11 Mi H B, Wang H M, Yin G, et al. Performance problems diagnosis in cloud computing systems via analyzing request trace logs. In: The 13th International Conference on Network Operations and Management Symposium (NOMS), Maui, 2012. 893–899
12 Jolliffe I. Principal Component Analysis, 2nd ed. New York: Springer, 2002
13 Fay M, Proschan M. Wilcoxon-Mann-Whitney or t-test on assumptions for hypothesis tests and multiple interpretations of decision rules. Stat Surv, 2010, 4: 1–39
14 Melville P, Yang S, Saar-Tsechansky M, et al. Active learning for probability estimation using jensen-shannon divergence. In: Proceedings of the 16th European Conference on Machine Learning. Berlin/Heidelberg: Springer-Verlag, 2005. 268–279
15 Sigelman B, Barroso L, Burrows M, et al. Dapper, a large-scale distributed systems tracing infrastructure. Technical Report dapper-2010-1, Google, 2010
16 Park I, Buch R. Event tracing-improve debugging and performance tuning with ETW. MSDN Mag, 2007. 81–92
17 Thereska E, Salmon B, Strunk J, et al. Stardust: tracking activity in a distributed storage system. ACM SIGMETRICS Perform Eval Rev, 2006, 34: 3–14
18 Sang B, Zhan J, Lu G, et al. Precise, scalable, and online request tracing for multi-tier services of black boxes. IEEE Trans Parallel Distrib Syst, 2010, 99: 1–16

19 Tak B, Tang C, Zhang C, et al. Vpath: precise discovery of request processing paths from black-box observations of thread and network activities. In: Proceedings of the 2009 Conference on USENIX Annual Technical Conference. Berkeley: USENIX Association, 2009. 19–32

20 Koskinen E, Jannotti J. Borderpatrol: isolating events for black-box tracing. ACM SIGOPS Operat Syst Rev, 2008, 42: 191–203

21 Reynolds P, Wiener J, Mogul J, et al. Wap5: black-box performance debugging for wide-area systems. In: Proceedings of the 15th International Conference on World Wide Web. New York: ACM, 2006. 347–356

22 Aguilera M, Mogul J, Wiener J, et al. Performance debugging for distributed systems of black boxes. ACM SIGOPS Operat Syst Rev, 2003, 37: 74–89

23 Chen M, Kiciman E, Fratkin E, et al. Pinpoint: Problem determination in large, dynamic internet services. In: Proceedings of 32nd IEEE International Conference on Dependable Systems and Networks, Bethesda, 2002. 595–604

24 Chen M, Accardi A, Kiciman E, et al. Path-based faliure and evolution management. In: Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation, Vol. 1. Berkeley: USENIX Association, 2004. 23–36

25 Barham P, Donnelly A, Isaacs R, et al. Using Magpie for request extraction and workload modelling. In: Proceedings of the 6th Conference on Symposium on Opearting Systems Design and Implementation. Berkeley: USENIX Association, 2004. 259–272

26 Fonseca R, Porter G, Katz R, et al. X-trace: A pervasive network tracing framework. In: Proceedings of the 4th USENIX Conference on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2007. 20–33

27 Mi H, Wang H, Yin G, et al. Magnifier: Online detection of performance problems in large-scale cloud computing systems. In: Proceedings of 8th IEEE International Conference on Services Computing, Washington DC, 2011. 418–425

28 Wang C, Schwan K, Talwar V, et al. A flexible architecture integrating monitoring and analytics for managing large-scale data centers. In: Proceedings of the 8th ACM International Conference on Autonomic Computing. New York: ACM, 2011. 141–150

29 Wang C, Viswanathan K, Choudur L, et al. Statistical techniques for online anomaly detection in data centers. In: Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management, Dublin, 2011. 385–392

30 Bodik P, Goldszmidt M, Fox A, et al. Fingerprinting the datacenter: automated classification of performance crises. In: Proceedings of the 5th European Conference on Computer Systems. New York: ACM, 2010. 111–124

31 Wang C, Talwar V, Schwan K, et al. Online detection of utility cloud anomalies using metric distributions. In: Proceedings of the IEEE Network Operations and Management Symposium, Osaka, 2010. 96–103

32 Lakhina A, Crovella M, Diot C. Diagnosing network-wide traffic anomalies. ACM SIGCOMM Comput Commun Rev, 2004, 34: 219–230

33 Xu W, Huang L, Fox A, et al. Detecting large-scale system problems by mining console logs. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. New York: ACM, 2009. 117–132

34 Oliner A, Aiken A. Online detection of multi-component interactions in production systems. In: 41st IEEE/IFIP International Conference on Dependable Systems & Networks (DSN), Hong Kong, 2011. 49–60

35 Ringberg H, Soule A, Rexford J, et al. Sensitivity of PCA for traffic anomaly detection. ACM SIGMETRICS Perform Eval Rev, 2007, 35: 109–120

36 King J, Jackson D. Variable selection in large environmental data sets using principal components analysis. Environmetrics, 1999, 10: 67–77

37 Ghemawat S, Gobioff H, Leung S. The Google file system. ACM SIGOPS Operat Syst Rev, 2003, 37: 29–43