Scaling Performance Issue Detection and Diagnosis in Cloud Infrastructures

Yigong Hu[1], Ze Li[2], Peng Huang[1], Suhas Pinnamaneni[2], Francis David [2], Yingnong Dang[2]
Murali Chintalapati[2]
[1]Johns Hopkins University [2]Microsoft Azure
{zeli, suhasp, frdavid, yidang, muralic}@microsoft.com hyigong1@jhu.edu huang@cs.jhu.edu

Abstract

Cloud infrastructures today consist of heterogeneous software and hardware environments to serve different workloads. Efficiently and accurately analyzing the performance for these systems is of critical need. However, current performance analysis methods scale poorly to handle the performance data from modern cloud platforms and do not consider the effect of system heterogeneity. In this paper, we present SDiag, a scalable performance analysis tool to uncover performance issues in complex heterogeneous cloud platform. Specifically, SDiag first breaks down tracing records into different buckets. SDiag then analyzes and compares different buckets to find the performance bottleneck and critical path. Finally, SDiag provides supporting evidence to help developers effectively troubleshoot the reported performance issues.

1 Introduction

Large cloud computing platforms, such as Microsoft Azure, serve millions of users. Rapidly uncovering and fixing performance issues of the system infrastructures is critical for ensuring user satisfactions and meeting the Service Level Agreement (SLA). However, the ever-increasing scale and complexity of cloud infrastructures pose significant challenges for service engineers to efficiently understand, analyze, and troubleshoot performance issues.

Although extensive solutions have been proposed to help uncover performance issues in distributed systems (Sigelman and et al. 2010; Barham and et al. 2004; Kaldor and et al. 2017; Fonseca and et al. 2007; Mace and et al. 2015; Veeraraghavan and et al. 2016; Wang and et al. 2013), they primarily focus on how to effectively trace or profile systems performance across layers of software stack, e.g., through propagating request identifiers. Analyzing the large volume of complex performance data generated from the tracing or profiling solutions remains challenging. Classic performance analysis techniques (e.g., critical path analysis) and recent cloud performance analysis solutions (Chow and et al. 2014) assume that the execution environment of software are stable so the systems exhibit similar performance behavior under the same workloads. However, many dynamic factors can influence system performance, such as continuous deployment of new code, changing configurations, user-specific experiments, and the system environment. For example, the performance of a virtual disk creation operations might incur different execution times among different hardware generations. As a result, a slow VM creation performance issue might only exhibit in the requests that are executed on the certain types of hardware. Therefore, without considering the impact of dynamic factors and different execution environments, the performance analysis can lead to false positives or missing complex performance issues.

In this paper, we propose SDiag, a performance analysis tool that can efficiently and accurately detect complex performance issues in large cloud platforms. SDiag analyzes the tremendous amount of performance logs and tracing results of a cloud platform in real time. It first breaks down data into different buckets and builds the statistical call stack for each bucket. Then SDiag applies critical path analysis in each bucket to understand how statistically each component impact the end-to-end latency. Afterwards, SDiag performs data mining techniques such as anomaly detection and correlation analysis to find the responsible software or hardware component. Finally, SDiag reports the abnormal performance behavior, the culprit component and the critical operations in the callstack to developers.

2 Problem Statement

Modern cloud platform collects billions of performance trace records and execution logs every day. State-of-art performance analysis tools typically analyze all the performance data together, e.g., finding the most time-consuming operation among all the requests. However, as the performance is impacted by diverse, dynamic factors in the execution environments, blindly aggregating all the data would hide the performance variability on different software packages, hardware configurations, and application idiosyncrasies, and cause some complex performance misbehavior to escape detection. We argue that to analyze performance in a dynamic environment, the performance data needs to be properly grouped in a way that can expose performance anomalies within some specific group. Thus, we propose to first use data mining techniques to group all the requests that are impacted by the same factors into the same buckets. Then we can apply performance analysis techniques to each bucket to find the performance anomalies.

3 Design of SDiag

SDiag is a performance analysis tool for uncovering performance issues resulting from complex runtime environments in cloud platforms. The objective of SDiag is to analyze a large volume of performance events in real time, correlate them, identify anomalies that appear in a subset of these events, and alert developers about the anomalies with supporting evidence, and trigger actions (e.g., stopping the bad software deployments).

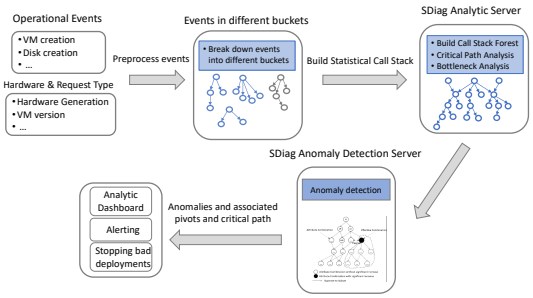

Figure 1: Workflow of SDiag.

Figure 1 shows the workflow of SDiag. SDiag takes operational events, hardware information and request type information as the input and generates a regression alert as the output to the developers. SDiag first groups the events with attribute metrics that describe the origins of the events and breaks down all the events into different buckets based on the metrics. SDiag analytics server then aggregates all the events in same bucket to build the statistical call stack and apply critical path analysis and bottleneck analysis in each buckets to locate the buggy function call stack. Finally, SDiag detection server performs anomaly detection to find the abnormal component and generates a report to the developer with the critical path, bucket metrics and bottleneck.

SDiag first combines each operational event with its associated metrics such as hardware generation, software component version and request type to generate a comprehensive operational events table. Each entry in the comprehensive operational event tables contains *function name, execution time, acttivityId, parentId, rootId, hardware generation, VM type, software version*. Then SDiag breaks down the event table into separate small buckets whose events have same hardware generation and workload type.

Since each bucket contains millions of events, SDiag needs to aggregate all the events before performing further analysis. SDiag first builds the calling relationship for each request based on the activityId, parentid and rootId. Then SDiag aggregates the call stack of each request to build a *statistical call stack forest*. The call stack forest is represented as a directed acyclic graph, in which the function calls are vertices with weight equal to 95% of the execution time of the associated function calls. Since each bucket contains an enormous amount of events, the statistical call tree can precisely reflect the performance of the cloud platform under a given system environment. After creating the statistical call stack for each bucket, SDiag performs two types of analysis on top of the call stack forest.

For each bucket, SDiag further aggregates the performance data into different groups based on the version combination of all the software packages. SDiag performs anomaly detection by comparing different groups in the same bucket to find regression. In particular, SDiag calculates the mean of end-to-end request latency for each group and uses the Z-score to identify the outlier groups and requests. Z-score is a numerical measurement of relationship of the latency of different versions of software components. If a Z-score is close to 0, it indicates that the roll-out does not introduce any performance regression. A Z-score of 1.0 would indicate that the latency is one standard deviation away from the mean of latency of the older versions. Currently, SDiag defines outliers as the group whose Z-score is greater than 3 (which is a commonly-used threshold that distinguishes "unusual" values in a sample).

After the bucketing and aggregations, SDiag applies critical path analysis in each bucket to find the function call stacks that contribute most to the long latency. Critical path analysis is a classic technique for understanding how individual function calls of a parallel execution impact end-to-end latency (Wang and et al. 2013). SDiag calculates the critical path on a per-bucket basis. It finds the longest-path (in terms of total duration) from the first event in the request (the initiation of the request) to the last event.

4 Deployment Status

SDiag is currently deployed and being used by an engineering team in Microsoft Azure to analyze the performance of the tenant deployment operations. It has been running for more than two months and successfully found a performance regression. In one release of an Azure agent package, SDiag found that the VM creation request has a significantly high latency in a specific hardware generation. SDiag reported the issue to the developer team with the supporting critical path. The developer team confirmed that the root cause is that the creation of a specific type of disk is done sequentially rather than in parallel. The performance regression has already existed in the Azure environment for one month but since it only manifested in a specific hardware generation, the current performance analysis tool can not effectively detect it. This result shows that the SDiag can effectively uncover complex performance behavior.

SDiag currently only uses predefined metrics to bucket performance events. However, the performance relevant factors are usually diverse and dynamic, which means predefined metrics are not enough to uncover complex performance issues. Our next step is to automatically find the bucket metrics that can best identify and explain the performance regression and perform further analysis.

References

Barham, P., and et al. 2004. Using Magpie for request extraction and workload modelling. In *OSDI '04*.

Chow, M., and et al. 2014. The mystery machine: End-to-end performance analysis of large-scale internet services. In *OSDI '14*.

Fonseca, R., and et al. 2007. X-trace: A pervasive network tracing framework. In *NSDI '07*.

Kaldor, J., and et al. 2017. Canopy: An end-to-end performance tracing and analysis system. In *SOSP '17*.

Mace, J., and et al. 2015. Pivot tracing: Dynamic causal monitoring for distributed systems. In *SOSP '15*.

Sigelman, B. H., and et al. 2010. Dapper, a large-scale distributed systems tracing infrastructure. Technical report, Google, Inc.

Veeraraghavan, K., and et al. 2016. Kraken: Leveraging live traffic tests to identify and resolve resource utilization bottlenecks in large scale web services. In *OSDI '16*.

Wang, X. S., and et al. 2013. Demystifying page load performance with WProf. In *NSDI '13*.