Developing Software Performance Training at Alibaba

Kingsum Chow*, Wanyi Zhu, Chengdong Li, Si Chen, Tongbao Zhang, Chenggang Qin and Sanhong Li
Alibaba Inc.
Hangzhou, China
*kingsum.chow@gmail.com

ABSTRACT
Effective software performance analysis needs to be conducted by crossing multiple disciplines such as algorithms, data structures, effective coding, performance data collection and its associated overheads, computer architecture, operating systems, containers and virtual machines, statistical analysis, machine learning and applied mathematics. However, no students are prepared to learn all these subjects in school. There is a need to develop software performance training at work. We need a training program that targets the different needs of new and old employees. We are working on developing such a program here at Alibaba. This paper describes our focus on practical aspect of mastering various subjects to aid software performance analysis.

CCS Concepts
• Computer systems organization → Architectures → Distributed architectures • Software and its engineering → Software organization and properties → Extra-functional properties → Software performance.

Keywords
Datacenter efficiency; software performance; capacity planning; analytics

1. INTRODUCTION
The emergence of large-scale software deployments in the cloud has led to challenges in measuring software performance, and optimizing software deployments. This vision paper addresses the two challenges by bringing the knowledge of software performance monitoring in the data center to the world of applying performance analytics. We developed an approach called Performance Improvement and Planning Analytics (PIPA). In our approach, we bridge together 4 key layers for (1) performance data collections, (2) data transformations, (3) analytics, and (4) decisions. The PIPA approach is a cost-effective performance engineering solution in the datacenter. However, the approach alone is not sufficient. We need software performance engineers that can work along the PIPA approach. These are the engineers that can continue to develop new analytics based on strong understand of how to do software analysis. Thus, it brings to the next question: where can you hire software performance engineers?

To do a good job in software performance analysis, the engineers need to have a breadth of knowledge in math, statistics, scientific thinking, performance tools, data collections, performance scaling, and tuning software applications. As there not many people with these backgrounds, developing a training program to develop performance engineers in house would be needed.

ICPE'17 Companion, April 22–26, 2017, L'Aquila, Italy.
ACM ISBN 978-1-4503-4899-7/17/04.
DOI: http://dx.doi.org/10.1145/3053600.3053640

2. SOFTWARE PERFORMANCE TRAINING
Many textbooks and papers describe a collection of aspects of software performance analysis. Students can certainly read them and learn from them. What is lacking seems to be why these software performance analytical methods are needed. Furthermore, there are certain industrial practices that we do for practical results. We would like to get students excited by learning these practices. We thought it would be good to collect a list of mistakes in software performance and use them to highlight why proper software performance analysis is needed.

We describe each of the software performance myths in each subsection below.

2.1 Myth: My performance data collection tool has low overheads because the throughput has not changed when the tool is used.
One of the frequently claimed successes of performance data collection tool is low overhead. To draw the conclusion, we would run an in-house workload. We would report the configuration of the experiments such as the number of users, the throughput and response time of the transactions. We would run it with and without the performance data collection tool. We would collect the data indicating there is little change in both throughput and response time. Then we would show that by collecting additional data, the story could be quite the opposite. We are developing two case studies here: (1) the performance tool affects all transactions and (2) the performance tool affects just some of the transactions. The second case study exposes another problem – if the number of samples is small compared to all the transactions, then the average throughput may appear to be stable even through the tool has a huge impact to the sampled transactions.

2.2 Myth: Performance scaling experiments are limited by the stress-testing tool failing to saturate the system.
In the industry, we sometimes reach the limitation of the load testing tools. The experiments can only be run with limited loads. One may be tempted to conclude that no result can be obtained. We will run some experiments and demonstrate that despite the limitations we encounter, we can still identify performance-scaling issues. We can study the workload characteristics at multiple levels and study the scaling behavior, even though we are not saturating the software system under test.

2.3 Myth: Performance measurement units are not important.
We have been amazed by how often measurement units are skipped in analysis. Among them, CPU utilizations seem to be leading the misleading analysis. In this lab exercise, we will have

the students run some workloads and collect system performance counters. We would illustrate how we can draw misleading conclusions. Then we would show how to do it right. There are two parts of this case study: (1) the performance unit itself is misleading, e.g. CPU utilizations and (2) the missing units of measurements prevent further analysis.

2.4 Myth: Incomplete software performance logs lead to delay in analysis.

Different software applications may choose to log various amount of information in the form of data logs. But during early application development, the logs can be incomplete. Some students would collect some performance data and claim that analysis cannot be done due to missing data. We would demonstrate that a lot of useful analysis can still be done despite incomplete performance logs.

2.5 Myth: The same software application must have the same behavior.

We will conduct experiments to show that even the same software application can behave quite differently in different configurations, containers or under-lying hardware. The experiments will comprise of several cases: (1) changes in the hardware configurations, (2) changes in the use of the hardware components, and (3) changes in the software configurations. The software performance behavior can be captured in the software logs or common performance tools such as perf.

2.6 Myth: In Java applications, longer garbage collections are always bad.

We will conduct experiments with varying amount of garbage collections, and show that in some cases, they don't matter. It is important to identify what those cases are. We need to design a couple of case studies that can help the performance engineer develop the relationship between the amount of garbage collections and the two key performance metrics: throughput and response times.

2.7 Myth: Hardware options are optimized for us and they cannot be changed.

We will conduct some experiments by changing how hardware options are used. We would demonstrate a significant performance improvement is possible with hardware options tuning.

3. REPRODUCIBLE ANALYSIS

When the experts do software performance manually, the analysis cannot be easily reproduced as the assumptions made, the process of analysis and how the conclusion is drawn cannot be easily documented. Here we propose the use of scripts to document the analysis. We need reproducible analysis to advance software performance analysis in the industry.

Our reproducible analysis is consisted of accessing the raw data that are generated by the performance monitoring tools, our scripts that process the raw data and transform the raw data into something we apply analysis, the results of the analysis such as graphs and tables can be generated from the scripts, and the decisions that we make are documented. As the whole analysis is documented, any assumption made during the analysis is revealed in the script. If any assumption is deemed invalid, we can redo the analysis by just changing that assumption.

4. SUMMARY

We covered only a small list of experiment driven analysis that we hope to train new performance engineers. This is not an exhaustive list. But we believe this is a good start. We also highlight the importance of reproducible analysis as a documented communication among performance engineers.

It is a daunting task to deal with large-scale datacenter performance and efficiency analytics. In PIPA, we describe our early thought in approaching this problem. We are just scratching the surface of large-scale datacenter efficiency. After we realize the need for a better performance analysis education, we have started working on a training program. We believe we have many years to learn from our mistakes, refine our techniques and perhaps developing a much better approach.

ACKNOWLEDGMENTS

We thank multiple teams in Alibaba that has developed multiple data collectors, storage and analysis portals for multiple projects.