Java Performance Troubleshooting and Optimization at Alibaba

Fangxi Yin, Denghui Dong, Sanhong Li, Jianmei Guo and Kingsum Chow
Alibaba Group, Hangzhou, China
{fangxi.yfx,denghui.ddh,sanhong.lsh,jianmei.gjm,kingsum.kc}@alibaba-inc.com

ABSTRACT

Alibaba is moving toward one of the most efficient cloud infrastructures for global online shopping. On the 2017 Double 11 Global Shopping Festival, Alibaba's cloud platform achieved total sales of more than 25 billion dollars and supported peak volumes of 325,000 transactions and 256,000 payments per second. Most of the cloud-based e-commerce transactions were processed by hundreds of thousands of Java applications with above a billion lines of code. It is challenging to achieve comprehensive and efficient performance troubleshooting and optimization for large-scale online Java applications in production. We proposed new approaches to method profiling and code warmup for Java performance tuning. Our fine-grained, low-overhead method profiler improves the efficiency of Java performance troubleshooting. Moreover, our approach to ahead-of-time code warmup significantly reduces the runtime overheads of just-in-time compiler to address the bursty traffic. Our approaches have been implemented in Alibaba JDK (AJDK), a customized version of OpenJDK, and have been rolled out to Alibaba's cloud platform to support online critical business.

CCS CONCEPTS

• **Software and its engineering → Software performance**;

KEYWORDS

Java performance, method profiling, code warmup, overhead

ACM Reference Format:
Fangxi Yin, Denghui Dong, Sanhong Li, Jianmei Guo and Kingsum Chow. 2018. Java Performance Troubleshooting and Optimization at Alibaba. In *ICSE-SEIP '18: 40th International Conference on Software Engineering: Software Engineering in Practice Track, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183519.3183536

1 INTRODUCTION

Alibaba is moving toward one of the most efficient cloud platforms for global online shooping. On the 2017 Double 11 Global Shopping Festival, which has taken place annually on November 11, Alibaba's cloud platform achieved total sales of more than 25 billion dollars and supported peak volumes of 325,000 transactions and 256,000 payments per second. Most of the cloud-based e-commerce transactions were processed by hundreds of thousands of Java applications with above a billion lines of code.

ICSE-SEIP '18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5659-6/18/05.
https://doi.org/10.1145/3183519.3183536

Java performance is critical for dealing with with the enormous volume of online traffic and transactions at Alibaba. To address various performance issues, there is an urgent need for efficient approaches to Java performance troubleshooting and optimization.

In this paper, we focus on two aspects of Java performance tuning: method profiling and code warmup. For each aspect, we first highlight its importance for large-scale online Java applications at Alibaba. Second, we discuss related work and present the limitations of existing methods. Finally, we present our approach and implementation.

2 METHOD PROFILING

Method profiling is important for performance engineers to understand the behavior of Java applications and to evaluate the contribution of each individual method to the overall execution time.

For Java applications, there are many, either open-source or commercial, performance profilers [2, 3]. They mainly fall into two categories: *sampling* approach and *instrument* approach.The sampling approach often captures the contribution of each method to the execution time by sampling the call trace of Java application. The instrument approach chooses the opposite direction and instruments every method enter/exit pair to capture the detailed execution time of every individual method. We discuss a few representative Java profilers here.

HPROF [5] uses Java Virtual Machine Tool Interface (JVMTI) APIs to capture the stack traces of all threads in Java process in a timer periodical way. With the captured stacks, the profiler can build up the information to tell which method contributes the most of application execution time. However, HPROF's sampling is *safepoint-biased* and thus not accurate. The samples of HPROF do not align with the intervals specified as intuitive expectations and lead to the distortion of application profiles.

HonestProfiler [8] uses an Oracle undocumented API AsyncGetCallTrace (AGCT) to sample the Java application. This API overcomes the HPROF's safepoint-biased drawback. HonestProfiler captures exactly what a JVM is doing at precisely the moment of timer signal is sent. This profiler's overhead is lower than the JVMTI sampling approach. But its sampling nature determines that it cannot provide the details of each method's execution process, which are usually critical for troubleshooting performance issues in our production environments.

VisualVM [6] is a popular profiler using the instrument approach by bytecode enhancement technology. It instruments all the methods of a Java application and adds extra bytecodes for every method to capture when the method is called and how long it is executed. Unfortunately, this profiler's overheads are usually high, resulting from two reasons: First, the instrumentation makes Java applications perform more tasks for every method; Second, the change of code size may have negative impact on the inline optimization decisions made by the just-in-time (JIT) compiler.

Different from previous profilers, we implemented a fine-grained, low-overhead method profiler based on the instrument approach. Our approach builds up a call tree that details each method's execution process concretely and intuitively. We modified the Java Hotspot VM to capture every method enter/exit pair of the mutator threads. Specifically, we modified the interpreter and C1/C2 JIT compiler, so that every method compiled can be instrumented with extra instructions of method enter/exit capture.

A distinguished feature of our profiler is that the instrumentation has been implemented at the level of machine code instead of bytecode, so that the overheads are lightweight. Moreover, we added only a few extra instructions for non-inline methods, so that the profiling overheads are reduced as far as possible. Our profiler acquires the detailed execution time of every method with lower overheads, compared to other existing instrument-based profilers.

Our implementation makes it convenient to turn on/off the profiling function *on the fly* without application restarting or code de-optimization. This feature is very important for our production environments to keep the continuous online services. In contrast, other existing approaches implement the switch on/off feature mainly by application restarting or class retransformation of JVMTI, which is usually not acceptable for online services. Note that class retransformation ofen leads to code de-optimization and degrades the performance seriously.

3 CODE WARMUP

Many online applications at Alibaba often encounter the bursty traffic that is much heavier than the average daily traffic, e.g., the 325,000 peak transactions per second on the 2017 Double 11 Global Shopping Festival. For Java applications, the runtime overheads of JIT compilation is usually affordable, but it becomes non-negligible during the bursty traffic and it may influence the performance of Java applications seriously.

Azul ReadyNow [7] is a commercial solution to Java code warmup. It records accumulated profiling data that is used in the warmup stage before the business-critical window. ReadyNow does code warmup in a black-box way. It aggressively loads or initializes classes and compiles the recorded methods automatically. But it may not well fit all application scenarios, especially when dynamic class loading and flexible class generation are often involved.

Our approach modifies the Java Hotspot VM to help engineers to get insights of code cache and control JIT compilation activities. This way, it is feasible to prepare a well JIT-compiled code cache for all business-critical methods before the coming of bursty traffic, which reduces the JIT runtime overheads significantly [1].

Our approach extends the JCMD utility of OpenJDK [4] to support *code cache dump*, which helps developers to understand if a Java application has been fully warmed up and ready for processing the bursty traffic. The code cache dump provides the summary information of code cache usage, the memory usage of compiled method by class loader, and the distribution of compiled methods at each level (from level 0 to level 4) of code cache. With the code cache dump, we can generate an expected code cache status file that describes the expected JIT-compiled methods, the expected JIT level, and so on. Afterwards, the expected code cache status file is used to perform ahead-of-time warmup before the coming of bursty traffic. Compared to ReadyNow, our approach controls the class loading and initializtion in an explicit way, which facilitates dealing with a wide variety of dynamic class loading and class generation scenarios in production.

4 CONCLUSION

We proposed new approaches to method profiling and code warmup for Java performance troubleshooting and optimization. Our fine-grained, low-overhead method profiling improves the efficiency of Java performance troubleshooting. Moreover, our approach to ahead-of-time code warmup significantly reduces the runtime overheads of JIT compiler to address the bursty traffic. Our approaches have been implemented in Alibaba JDK (AJDK), a customized version of OpenJDK, and have been rolled out to Alibaba's cloud platform to support online critical business.

Our approaches are not only applicable for online services at Alibaba, but also can be adapted to any Java applications. Compared to existing Java profilers, our approach to method tracing instruments each method with a few machine code and traces the fine-grained resource usage of each compiled method with ultra-low overheads. Moreover, our approach enables turning on/off the profiling on the fly without restarting JVM, which is critical for ensuring the reliability and serviceability of online Java applications in production. To deal with the non-negligible JIT runtime overheads at the moment of bursty traffic, we propose a new approach to ahead-of-time code warmup. Our approach reduces the JIT runtime overheads significantly and guarantees the smooth processing of the bursty traffic.

Two key lessons we learned are summarized as follows. First, it deserves to attach more importance to the feature of turning on/off profiling on the fly, since this feature enables online profiling activities that can be performed more frequently in production and can facilitate Java performance troubleshooting with more insightful details. Second, one-size-fits-all solution to Java performance optimization is not always effective, especially for a wide variety of dynamic class loading and class generation scenarios in production. A pragmatic way is to optimize the performance for each particular Java program, given its characteristics of application scenarios.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions.

REFERENCES

[1] Edd Barrett, Carl Friedrich Bolz-Tereick, Rebecca Killick, Sarah Mount, and Laurence Tratt. 2017. Virtual machine warmup blows hot and cold. *Proceedings of the ACM on Programming Languages* 1 (OOPSLA) (2017), 52:1–52:27.
[2] Peter Hofer, David Gnedt, and Hanspeter Mössenböck. 2015. Lightweight Java Profiling with Partial Safepoints and Incremental Stack Tracing. In *Proceedings of ICPE*. 75–86.
[3] Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. 2010. Evaluating the accuracy of Java profilers. In *Proceedings of PLDI*. 187–197.
[4] Oracle. 2006. OpenJDK. (2006). http://openjdk.java.net
[5] Oracle. 2011. HPROF: A Heap/CPU Profiling Tool. (2011). https://docs.oracle.com/javase/7/docs/technotes/samples/hprof.html
[6] Oracle. 2014. Java VisualVM. (2014). https://docs.oracle.com/javase/8/docs/technotes/guides/visualvm/profiler.html
[7] Azul Systems. 2014. Solving the Java warm-up issue in low-latency systems. (2014). https://www.azul.com/products/zing/readynow-technology-for-zing/
[8] Richard Warburton. 2011. Honest Profiler. (2011). https://github.com/jvm-profiling-tools/honest-profiler