Accurate Compilation Replay via Remote JIT Compilation

Andrej Pečimúth
Charles University
Prague, Czech Republic
pecimuth@d3s.mff.cuni.cz
Oracle Labs
Prague, Czech Republic
andrej.pecimuth@oracle.com

David Leopoldseder
Oracle Labs
Vienna, Austria
david.leopoldseder@oracle.com

Petr Tůma
Charles University
Prague, Czech Republic
petr.tuma@d3s.mff.cuni.cz

Abstract

When a JIT compiler crashes in a production deployment, compiler developers wish to reproduce the problem locally. However, existing approaches to replay compilation lack the necessary accuracy for this use case, or they introduce too much of a maintenance burden. We propose to achieve accurate compilation replay by running a remote compilation, recording the input to the remote compiler, and replaying the compilation using the recorded data. The benefit is significantly reduced iteration times for compiler developers when such an issue occurs.

Keywords: replay compilation, remote compilation, JIT compilation, virtual machines

Introduction

Applications in production deployments typically connect to external services and run on a platform with particular CPU features. When the JIT compiler crashes in this setup, a compiler developer may wish to rerun it with diagnostic options [3] or attach a debugger to the compiler. However, many practical barriers exist to rerunning the application on the compiler developer's hardware. When such a crash occurs, the developers' options to diagnose it are usually restricted to information such as logs provided by the team that services the application, which is a lengthy and inefficient process.

The existing approaches to replay compilation [7, 3, 2, 8] do not fit this use case well. The majority of them are based on recording and reusing the profiles, the methods selected for optimization, and similar data. Although this leads to more stability in JIT compilation, it is not guaranteed that the compiler repeats the same sequence of steps leading to a crash — the replay is not accurate, and these approaches require executing the application. An alternative approach is instrumenting the compiler's source code [6] to intercept the accesses to the VM's data. However, this reduces the readability of the code and poses an impractical maintenance overhead.

We propose a solution to replay compilations accurately by leveraging the infrastructure for remote JIT compilation [4, 1, 5], which is already supported by several runtimes. These runtimes disaggregate the compiler from the virtual machine (VM), allowing it to target a different platform and VM than the one it is hosted on. Moreover, remote compilation clearly separates the input to the compiler, which is usually transmitted over the network and is thus serializable. Therefore, in our approach, we perform a remote compilation (which may be hosted on the same machine) and record the inputs to the remote compiler. To replay a compilation, we invoke the remote compiler on the recorded inputs, except perhaps with extra diagnostic options. The desired result is the process of the compilation rather than the compiled code itself.

When a JIT compilation crashes in a production deployment, the running VM could immediately attempt to recompile the same method and record this compilation for a future replay — the VM would save all the inputs needed for replay in a file also holding the description of the target platform and the VM's configuration. The attempted recompilation is likely to exhibit the same issue that led to the original crash. Another option is to record all compilations for replay, but the overhead of recording may be high. Compiler developers can then replay the offending compilation using the recorded file while utilizing many of the debugging tools they are accustomed to. The proposed workflow would significantly shorten the time it takes to resolve these issues.

References

[1] Azul. 2024. Cloud Native Compiler. https://docs.azul.com/optimizer-hub/about/cloud-native-compiler. Accessed 2024-07-31.

[2] Andy Georges, Lieven Eeckhout, and Dries Buytaert. 2008. Java performance evaluation through rigorous replay compilation. *SIGPLAN Not.*, 43, 10, (Oct. 2008), 367–384. DOI: 10.1145/1449955.1449794.

[3] Tobias Hartmann. 2020. Debugging the Java HotSpot VM. https://cr.openjdk.org/~thartmann/talks/2020-Debugging_HotSpot.pdf. Accessed 2024-07-31.

[4] Alexey Khrabrov, Marius Pirvu, Vijay Sundaresan, and Eyal de Lara. 2022. JITServer: Disaggregated Caching JIT Compiler for the JVM in the Cloud. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, (July 2022), 869–884. https://www.usenix.org/conference/atc22/presentation/khrabrov.

[5] Han B. Lee, Amer Diwan, and J. Eliot B. Moss. 2007. Design, implementation, and evaluation of a compilation server. *ACM Trans. Program. Lang. Syst.*, 29, 4, (Aug. 2007), 18–es. DOI: 10.1145/1255450.1255451.

[6] Kazunori Ogata, Tamiya Onodera, Kiyokuni Kawachiya, Hideaki Komatsu, and Toshio Nakatani. 2006. Replay compilation: improving debuggability of a just-in-time compiler. 41, 10, (Oct. 2006), 241–252. doi: 10.1145/1167515.1167493.

[7] Oracle. 2023. Profile Replay Support in GraalVM. https://github.com/oracle/graal/blob/graal-24.0.2/compiler/src/jdk.graal.compiler/src/jdk/graal/compiler/hotspot/ProfileReplaySupport.java. Accessed 2024-07-31.

[8] Narendran Sachindran and J. Eliot B. Moss. 2003. Mark-copy: fast copying GC with less space overhead. *SIGPLAN Not.*, 38, 11, (Oct. 2003), 326–343. doi: 10.1145/949343.949335.