ETH Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Department of Computer Science
Laboratory for Software Technology

Profile Caching for the Java Virtual Machine

BSc Thesis for Marcel Mohler

Feb 15, 2015

Introduction

(Language) Virtual Machines like the Java Virtual Machine (JVM) are used as the execution environment of choice for many modern programming languages. The VMs interpret a suitable intermediate language (e.g., Java Byte Code for the JVM) and provide the runtime system for application programs and usually include a garbage collector, a thread scheduler, interfaces to the host operating system. As interpretation of intermediate code is time-consuming, VMs include usually a Just-in-Time (JIT) compiler that translates frequently-executed functions or methods to "native" code (e.g., x86 instructions).

The JIT compiler executes in parallel to a program's interpretation by the VM, and as a result, compilation speed is a critical issue in the design of a JIT compiler. Unfortunately, it is difficult to design a compiler such that the compiler produces good (or excellent) code while limiting the resource demands of this compiler (the compiler requires storage and cycles – and even on a multi-core processor, compilation may slow down the execution of the application program). Consequently, most VMs adopt a *multi-tier* compilation system. The first tier is the interpretation of a method. If this method is "hot", the Tier-1 compiler translates this method into a native code. The Tier-1 compiler implements only a small set of the know optimization techniques and as result, it had good compilation speed but the generated code is far from the output of an optimizing compiler. Such a compiler is usually the Tier-2 compiler, which takes a longer amount of time and produces *optimized* native code. To determine which methods should be compiled by the Tier-1 (or Tier-2) compiler, the VM profiles the execution of all application programs to identify "hot" methods.

In current VMs, at program startup, all methods are interpreted by the VM (execution at Tier-0). The interpreter performs profiling, and if a method is determined to be "hot", this method is then compiled by the Tier-1 compiler (fast compilation, few optimizations). Methods compiled to Tier 1 are then profiled further (and more extensively). Based on that profiling information, some methods are eventually compiled at Tier 2 (slow compilation, many optimizations).

One of the drawbacks of this setup is that for all programs, all methods start in Tier 0, with interpretation and profiling by the VM. However, for many programs the set of "hot" methods does not change from one execution to another and there is no reason to gather again and again the profiling information. Cached profiles would allow the VM to compile some methods right away.

Goal

This thesis is supposed to explore the design space for caching profiles in the context of a state-of-the-art Java Virtual Machine (Java HotSpot). The student should investigate

1. What is a good way to select methods to be compiled right away;
2. To which tier (Tier 1 or Tier 2) should the VM compile selected methods.
3. What information should be kept with the profile (i.e, what kind of profile information should be recorded).

Tasks

The thesis is divided in five main tasks:

0. Reach familiarity with basic operation of the multi-tier compilation system of the Java HotSpot JVM.

1. What is a good way to select methods to be compiled right away; What information should be kept with the profile (i.e, what kind of profile information should be recorded)?

2. Devise a simple strategy to determine to which tier (Tier 1 or Tier 2) should the VM compile selected methods.

3. (Optional) Investigate the stability of profiles (i.e, is there a difference in the set of hot methods for application A when we compare the execution for input I_1 and I_2?). Define suitable metrics.

4. Take a selection of industry-standard benchmarks (will be supplied by the advisor) and evaluate the performance aspects (start-up time, execution time, storage, etc).

5. Document the design, the implementation, and the evaluation and discuss these issues in the thesis.

General instructions

- The project starts on Mar 1st, 2015 and should be completed by September 1st, 2015.
- The course of work should be discussed regularly with a responsible project advisor and the thesis supervisor.
- The deliverables of this work are (1) the source code, (2) a report, and (3) a one page summary.

- Where applicable, the source code should be provided in the form of a patch relative to the modified initial code (e.g., in the case of HotSpot modifications).
- The report should be phrased as a scientific essay and should be submitted as three paper copies, and in PDF format. The preferred language is English, but at the preference of the student the report can be written in German.
- The one page summary should be written in English and provided as an XML file. A template is provided at `http://www.lst.inf.ethz.ch/teaching/sada_template.xml`.
- There should be an oral presentation at the end of the project.

Grading scheme

Grading will be based on the following criteria:

Criterion	Weight ($\pm$ 0.5)
Implementation (functionality, extensibility, documentation)	3
Report (content, illustration, writing)	2
Presentation (content, illustration, quality of talk)	1
Approach (organization, approaching problems, independence, involvement)	1
Contribution to the state of the art research	1
Completion of optional tasks	2
Completion of all tasks	3

Each criterion will be graded in accordance with the "Merkblatt zur Bachelor-Arbeit in Informatik nach Studienreglement 08".

Professor: Prof. T. Gross