Statistically Rigorous Java Performance Evaluation

Andy Georges Dries Buytaert Lieven Eeckhout
Department of Electronics and Information Systems, Ghent University, Belgium
{ageorges,dbuytaer,leeckhou}@elis.ugent.be

Abstract

Java performance is far from being trivial to benchmark because it is affected by various factors such as the Java application, its input, the virtual machine, the garbage collector, the heap size, etc. In addition, non-determinism at run-time causes the execution time of a Java program to differ from run to run. There are a number of sources of non-determinism such as Just-In-Time (JIT) compilation and optimization in the virtual machine (VM) driven by timer-based method sampling, thread scheduling, garbage collection, and various system effects.

There exist a wide variety of Java performance evaluation methodologies used by researchers and benchmarkers. These methodologies differ from each other in a number of ways. Some report average performance over a number of runs of the same experiment; others report the best or second best performance observed; yet others report the worst. Some iterate the benchmark multiple times within a single VM invocation; others consider multiple VM invocations and iterate a single benchmark execution; yet others consider multiple VM invocations and iterate the benchmark multiple times.

This paper shows that prevalent methodologies can be misleading, and can even lead to incorrect conclusions. The reason is that the data analysis is not statistically rigorous. In this paper, we present a survey of existing Java performance evaluation methodologies and discuss the importance of statistically rigorous data analysis for dealing with non-determinism. We advocate approaches to quantify startup as well as steady-state performance, and, in addition, we provide the JavaStats software to automatically obtain performance numbers in a rigorous manner. Although this paper focuses on Java performance evaluation, many of the issues addressed in this paper also apply to other programming languages and systems that build on a managed runtime system.

Categories and Subject Descriptors D.2.8 [*Software Engineering*]: Metrics—Performance measures; D.3.4 [*Programming Languages*]: Processors—Runtime environments

General Terms Experimentation, Measurement, Performance

Keywords Java, benchmarking, data analysis, methodology, statistics

OOPSLA'07, October 21–25, 2007, Montréal, Québec, Canada.

1. Introduction

Benchmarking is at the heart of experimental computer science research and development. Market analysts compare commercial products based on published performance numbers. Developers benchmark products under development to assess their performance. And researchers use benchmarking to evaluate the impact on performance of their novel research ideas. As such, it is absolutely crucial to have a rigorous benchmarking methodology. A non-rigorous methodology may skew the overall picture, and may even lead to incorrect conclusions. And this may drive research and development in a non-productive direction, or may lead to a non-optimal product brought to market.

Managed runtime systems are particularly challenging to benchmark because there are numerous factors affecting overall performance, which is of lesser concern when it comes to benchmarking compiled programming languages such as C. Benchmarkers are well aware of the difficulty in quantifying managed runtime system performance which is illustrated by a number of research papers published over the past few years showing the complex interactions between low-level events and overall performance [5, 11, 12, 17, 24]. More specifically, recent work on Java performance methodologies [7, 10] stressed the importance of a well chosen and well motivated *experimental design*: it was pointed out that the results presented in a Java performance study are subject to the benchmarks, the inputs, the VM, the heap size, and the hardware platform that are chosen in the experimental setup. Not appropriately considering and motivating one of these key aspects, or not appropriately describing the context within which the results were obtained and how they should be interpreted may give a skewed view, and may even be misleading or at worst be incorrect.

The orthogonal axis to experimental design in a performance evaluation methodology, is *data analysis*, or how to analyze and report the results. More specifically, a performance evaluation methodology needs to adequately deal