ByCounter: Portable Runtime Counting of Bytecode Instructions and Method Invocations

Manual

May 9th, 2011

Michael Kuperberg Martin Krogmann

Chair for Software Design and Quality
Institute for Program Structures and Data Organisation
Faculty of Informatics, Universität Karlsruhe (TH)

Abstract

For bytecode-based applications, runtime instruction counts can be used as a platform-independent application execution metric, and also can serve as the basis for bytecode-based performance prediction. However, different instruction types have different execution durations, so they must be counted separately, and method invocations should be identified and counted because of their substantial contribution to the total application performance. For Java bytecode, most JVMs and profilers do not provide such functionality at all, and existing bytecode analysis frameworks require expensive JVM instrumentation for instruction-level counting. ByCounter is a lightweight approach for exact runtime counting of executed bytecode instructions and method invocations. ByCounter significantly reduces total counting costs by instrumenting only the application bytecode and not the JVM, and it can be used without modifications on any JVM. It has been successfully applied to multiple Java applications on different JVMs, and is used for bytecode-based performance prediction.

Key words: Java, bytecode, counting, portable, fine-grained

1 Introduction

The performance of a Java application can be described by analysing the execution of the application's Java bytecode instructions. Execution counts of these instructions are needed for bytecode-based performance prediction of Java applications [1–3], and also for dynamic bytecode metrics [4]. As different instruction types have different execution durations, they must be

counted separately. Also, method invocations should be identified due to the substantial contribution of methods to the total application performance. Thus, each method signature (incl. the Java API methods) should have its own counts. To obtain all these runtime counts, static analysis is impractical and too complex, so it is usually faster and easier to use dynamic (i.e. runtime) analysis for counting executed instructions and invoked methods.

However, dynamic counting of executed Java bytecode instructions is not offered by Java profilers or conventional Java Virtual Machines (JVMs). ByCounter's competitors such as (such as JRAF [5]) either have serious shortcoming, or are not publicly available, as outlined in [6]. Many of them rely on the instrumentation of the JVM, however, such instrumentation requires substantial effort and must be reimplemented for different JVMs. In contrast to that, ByCounter it works by instrumenting the application bytecode instead of instrumenting the JVM, making the resulting approach truly portable while neither altering the signatures of instrumented classes nor requiring wrappers. To make performance characterisation through bytecode counts more precise, runtime parameters of some bytecode instructions must be considered, as they can have significant impact on their performance [1]. For these cases, ByCounter provides basic parameter recording (e.g. for the array-creating instructions), and it also offers extension hooks for the recording mechanism.

This manual is structured as follows: Sec. 2 explains how to obtain ByCounter while Sec. 3 describes the structure of the project. Sec. 4 gives an overview of tool's architecture, while Sec. 5 explains how to execute ByCounter examples and tests. Sec. 6 lists and explains limitations of the current version of ByCounter. The manual concludes with a listing of used libraries in Sec. 7.

2 Obtaining ByCounter

To obtain ByCounter, please contact Michael Kuperberg for information (Email: michael.kuperberg@kit.edu). If you are a member of the Palladio research group, you can obtain the current ByCounter release from the SVN repository.

For trouble-free development with ByCounter, we recommend version 3.5 of Eclipse for Java Developers, Eclipse for Java EE or Eclipse Classic to avoid the possibility of conflicts with the ASM library that is included in some Eclipse packages. Important note: the compiler settings in Eclipse should be left to default (which is "checked") for the following compilation settings (Windows → Preferences → Java → Compiler) which describe the contents of compiled class files: adding variable attributes, adding line number attributes, and adding source file names (s. Screenshot). Also check that the project-specific settings (Project → Properties → Java Compiler) do not differ from this. In Eclipse 3.5, "1.6" (aka version 50, see
`http://en.wikipedia.org/wiki/Class_%28file_format%29` for details) is

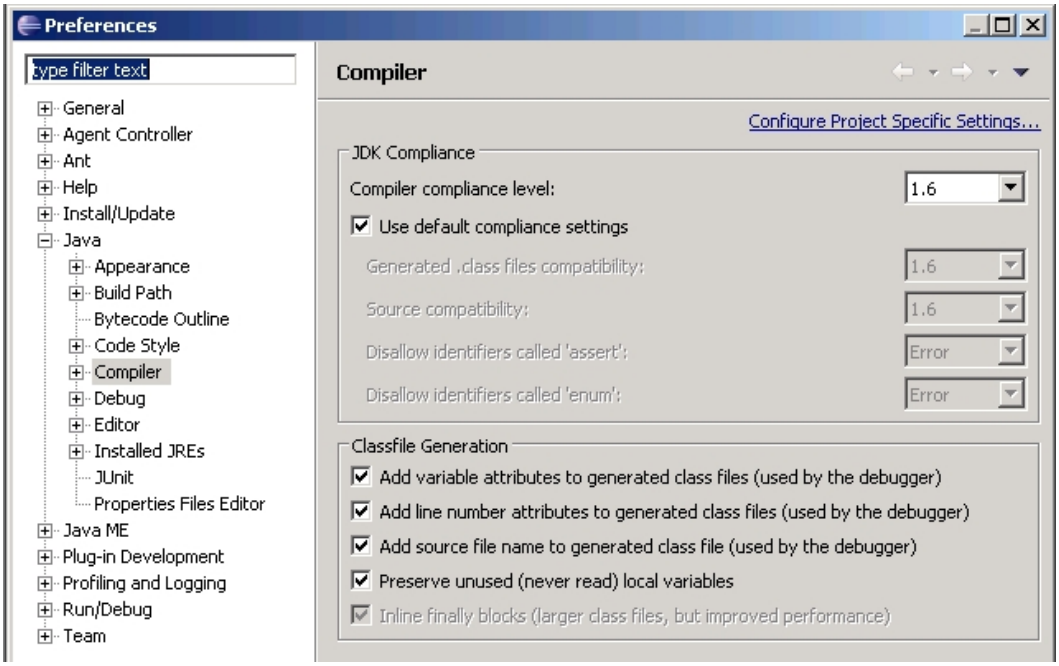

Fig. 1. Bytecode instrumentation and instruction counting using ByCounter

the default classfile version to which Java classes are compiled. When changing this to "1.5" (aka Java 5), Eclipse displays a warning that the Javassist library contains classes with version 50. This seems to be a false positive, under investigation by Javassist authors and us.

3 Description of file structure

- `/bin` : compiled ByCounter classes
- `/bin_instrumented` : instrumented classfiles written by ByCounter on request
- `/ByCounter_logged_counting_results` : the location of textual counting logs written when ByCounter is run in collectorless mode (see below)
- `/doc` : a small entry-level manual, as well as the location for generated Javadoc HTML files
- `/lib` : libraries (both end-user requirements and developer dependencies, see Section 7
- `/src` : source files

4 ByCounter Architecture

In Figure 2, the data flow in ByCounter is detailed (for overall ByCounter architecture, see [6]).

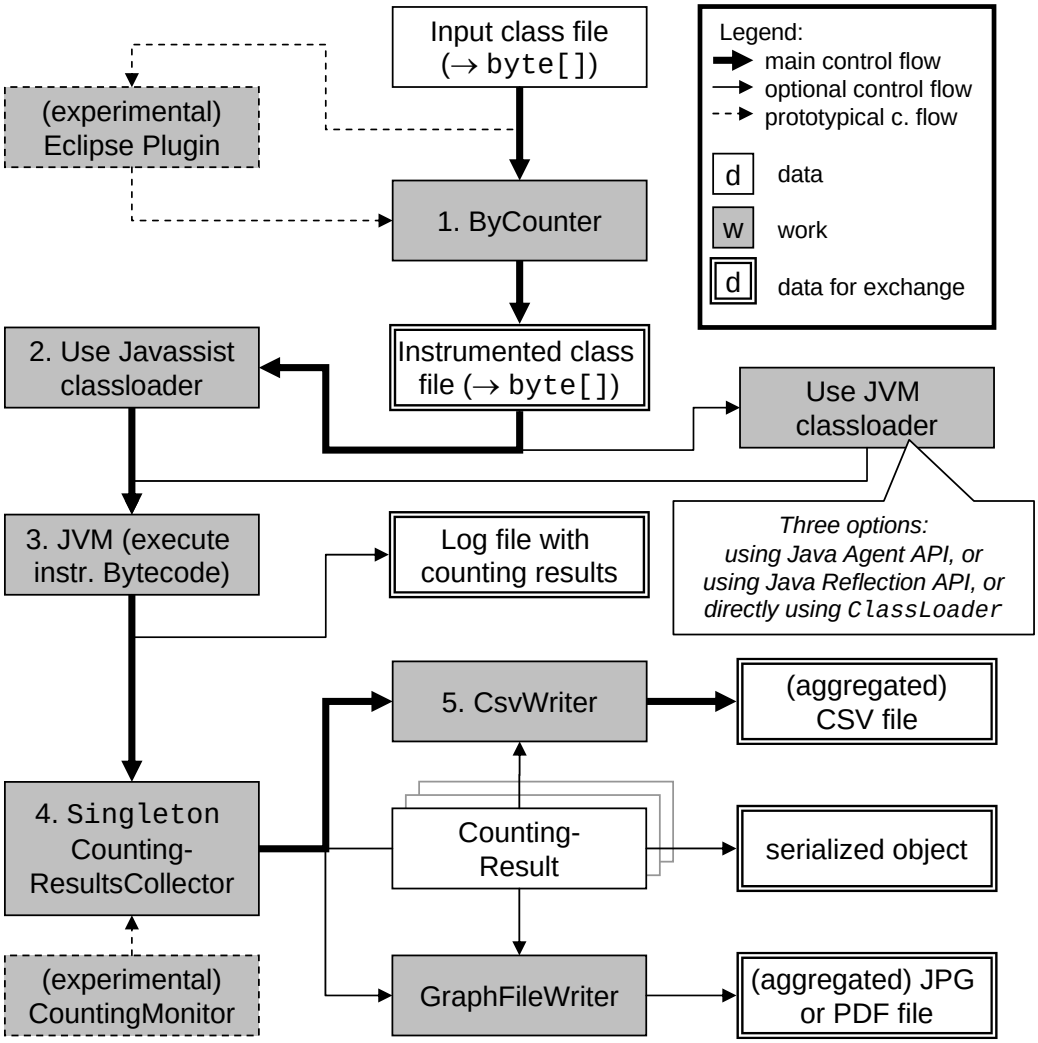

Fig. 2. Bytecode instrumentation and instruction counting using ByCounter

5 Examples and Running ByCounter

The class `ByCounterExample` in package `de.uka.ipd.sdq.ByCounter.example` can be executed directly. It instruments itself and runs the instrumented version, printing the counting results to standard out (i.e. to the console). The implementation of the class is documented so that the class can be used as an entry example. Further examples can be found in the `test` subpackage. **Both examples and test cases should be run with JVM parameters set to increase memory bounds, e.g. `-Xmx512M -Xss1M`.** Note that ByCounter creates a significant number of CSV files in the root project directory during test runs.

The Swing dialogue window that pops up when instrumentation (step) is failing is to be reworked soon; it is meant to accentuate the failure which otherwise gets easily lost in the console logging output. For logging, log4j was discarded in favor of Java platform API's logging facilities because log4j is

known to cause problems with bytecode instrumentation.

6 Limitations of ByCounter

The current version of ByCounter has troubles instrumenting classes in the default package (i.e. no containing package). Therefore make sure to have classes that have to be instrumented in packages until this issue is resolved

7 Description of Libraries

The following libraries, listed in alphabetical order, are found in the `/lib` directory of ByCounter. The JARs in bold are required, the others are needed for experimental purposes (and are partially not included in the ZIP file on BSCW).

- ant.jar (Ant is a standard plugin in Eclipse, but the ant.jar may be needed in other environments to compile the ByLoader.jar "JVM instrumentation agent")
- ant-launcher.jar
- **asm-all-3.3.jar** [1] , needed by end users and for building
- ByLoader.jar: GUI/"instrumentation JVM agent" that wraps ByCounter, self-created (`build.xml` in `loader` package)
- **commons-math-1.1.jar** [2] needed for mathematical evaluations
- derby.jar
- **gnujaxp.jar** [3] Needed for classpath/classloading operations
- **itext-2.1.7.jar** [4] needed for creating charts as PDF files
- janino.jar
- javac.jar
- javassist.jar [5] needed for Javassist-based classloading
- jcommon-1.0.10.jar [6]
- **jfreechart-1.0.13.jar** [7] , needed for displaying charts summarising counting results
- jfreechart-1.0.13-experimental.jar
- jfreechart-1.0.13-swt.jar

[1] http://asm.objectweb.org/
[2] http://commons.apache.org/math/
[3] http://www.gnu.org/software/classpath/
[4] http://www.jfree.org/jfreechart/
[5] http://javassist.org/, currently using version 3.8
[6] http://jcommons.sourceforge.net/
[7] http://www.jfree.org/jfreechart/

- **junit-4.4.jar** Needed to run the tests, e.g. outside of Eclipse(JUnit is a standard plugin)
- ocutil-2.5.1.jar
- servlet.jar
- swtgraphics2d.jar
- Tidy.jar

References

[1] M. Kuperberg and S. Becker, "Predicting Software Component Performance: On the Relevance of Parameters for Benchmarking Bytecode and APIs," in *Proceedings of the 12th International Workshop on Component Oriented Programming (WCOP 2007)*, R. Reussner, C. Czyperski, and W. Weck, Eds., July 2007. [Online]. Available: http://sdqweb.ipd.uka.de/publications/pdfs/kuperberg2007a.pdf

[2] M. Kuperberg, K. Krogmann, and R. Reussner, "Performance Prediction for Black-Box Components using Reengineered Parametric Behaviour Models," in *Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE 2008), Karlsruhe, Germany, 14th-17th October 2008*, vol. 5282, October 2008, pp. 48–63. [Online]. Available: http://sdqweb.ipd.uka.de/publications/pdfs/kuperberg2008c.pdf

[3] M. Hauck, M. Kuperberg, K. Krogmann, and R. Reussner, "Modelling Layered Component Execution Environments for Performance Prediction," in *Proceedings of the 12th International Symposium on Component Based Software Engineering (CBSE 2009)*, ser. LNCS, no. 5582. Springer, 2009, pp. 191–208. [Online]. Available: http://www.comparch-events.org/pages/present.html

[4] B. Dufour, K. Driesen, L. Hendren, and C. Verbrugge, "Dynamic Metrics for Java," in *OOPSLA '03: Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications*. New York, NY, USA: ACM, 2003, pp. 149–168.

[5] W. Binder and J. Hulaas, "Using Bytecode Instruction Counting as Portable CPU Consumption Metric," *Electr. Notes Theor. Comput. Sci.*, vol. 153, no. 2, pp. 57–77, 2006.

[6] M. Kuperberg, M. Krogmann, and R. Reussner, "ByCounter: Portable Runtime Counting of Bytecode Instructions and Method Invocations," in *Proceedings of the 3rd International Workshop on Bytecode Semantics, Verification, Analysis and Transformation, Budapest, Hungary, 5th April 2008 (ETAPS 2008, 11th European Joint Conferences on Theory and Practice of Software)*, 2008. [Online]. Available: http://sdqweb.ipd.uka.de/publications/pdfs/kuperberg2008a.pdf