Paper, presented at: Jazoon 2007 - The International Conference on Java Technology

PERFIDIX - A Generic Java Benchmarking Tool

Marc Kramis, Alexander Onea, Sebastian Graf

2007-06-26

Abstract

PERFIDIX shows an easy way to benchmark your code. Unlike heavy-load profilers, PERFIDIX comes in an easy-to-use way based on the proven usage of JUnit Testcases. This short paper describes the current work-in progress of the PERFIDIX 1.0 release.

1 Introduction

In the course of a research project related to prototyping Java-based native XML databases, our team was repeatedly faced with the question which algorithm or data structure performed better for a given workload. Complexity analysis and big-O notations do provide theoretical boundaries and general trends. However, implementations often differ a lot in practice due to implementation details, optimizations, and CPU or RAM availability.

After a tedious array of rudimentary benchmarking efforts ranging from simple hand-coded time measurements to expensive professional profiler sessions, we soon decided to design and implement our own tool which would allow for convenient and consistent benchmarking of arbitrary Java code. Convenience should be guaranteed by a seamless integration with Java language features, existing open source development environments, and tools such as Java annotations, Eclipse [1], and Ant [2]. Consistency should be assured by providing persistent and sound statistical evaluations in several output formats. Releasing PERFIDIX under the Apache License V2.0 [3] to the open source community should eventually allow a bigger audience to work with a uniform benchmarking tool.

The rest of this extended abstract is organized as follows: Section 2 gives a brief market survey of related tools. Section 3 describes our current work, i.e., PERFIDIX 1.0. Section 4 introduces the planned features and extensions for PERFIDIX 2.0. Finally, Section 5 summarizes our preliminary findings and concludes this extended abstract.

2 Related Tools

To the best of our knowledge, the only available tool for generically benchmarking arbitrary Java code is the open source JBench [4]. After a short analysis we decided to start PERFIDIX from scratch and not to extend JBench due to its old and unmaintained code base, the class- instead of method-level benchmarking granularity, the need for better statistical output, and the inflexible configuration through property-files.

JUnitPerf [5] is an open source extension to the unit-testing framework JUnit [6]. JUnitPerf can be applied to any unit test to perform accurate time measurements. Still, it does not comply with our requirements because it just assures that a unit test fails if it exceeds a given time frame. Multiple runs, statistical output, or support in iteratively improving the performance is not in the scope of JUnitPerf.

In contrast to the generic benchmarking tools, there exist a variety of domain-specific benchmarking tools. All of these focus on a very specific set of functions and present the results in various formats, ranging from unstructured console-based output to full-featured charts. JPerf [7] for example is an open source Java port of iperf [8] which measures IP bandwidth using UDP or TCP, amongst other network-related parameters. JBenchmark [9] is a free benchmark suite for mobile 3D APIs and other Java ME-related technologies.

Poleposition [10] is an open source benchmark test suite to compare database engines and object-relational mapping technologies. Most notably, all of these domain-specific benchmarking tools re-implemented the core benchmarking functionality to execute a piece of code multiple times while measuring its execution time and other relevant parameters. Furthermore, each tool has its own way to issue the results or even lacks proper statistics.

Benchmarking is closely related to profiling. Therefore we also looked at Java profilers. Profilers are powerful means to find bottlenecks and optimize existing code. They not only allow analyzing CPU time but also memory consumption, thread execution behavior and plenty of other parameters. However, profilers are complex software requiring quite some skills to run and interpret them. Another disadvantage is that a profiler will not automatically collect or expose statistical evaluations as they appear when running a piece of code multiple times. TPTP [11] is an open source example for a good profiler, JProfiler [12] a commercial one.

3 Current Work

PERFIDIX 1.0 is a preliminary release our research group is currently working with. It provides the core functionality for generic benchmarking but without the alluded convenience features, i.e., Java annotations and Eclipse integration.

Listing [**?**] shows a self-describing excerpt of the Example benchmark. We intentionally designed PERFIDIX 1.0 in the style of JUnit 3.x to foster its adoption by our developers. Example can either be run from Ant using our PERFIDIX 1.0 Ant task or manually by implementing the main() method. In both the Ant task and the main() method, the developer can configure the number of runs on a per-method and per-class basis. PERFIDIX 1.0 basically measures the execution times in either milli- or nanoseconds. If the developer wants to measure other events, e.g., the number of cache misses of a caching algorithm, he can do so via a customized meter. The customized meter has to be incremented manually whenever a specific event is registered and the result appears along with the default timing measurements in the statistical output.

The human-readable console output of the Example benchmark is shown in Listing 2. Other formats, such as XML or GNUPlot [13], are also available. PERFIDIX 1.0 automatically provides statistical output by calculating the minimum, maximum, average, sum, standard deviation, and confidence intervals for the execution times and other customizable meters. The statistics are calculated on a per-method and per-class basis. The XML output can be configured to contain the results of each run together with relevant metadata, e.g., a time stamp.

Listing 1: PERFIDIX 1.0 Example benchmark code excerpt

```
public class Example extends Benchmarkable {
  public void setUp() { ... }        // Setup method called before each method run.
  public void tearDown() { ... }     // Cleanup method called after each method run.
  public void benchMethod() { ... }  // Method to benchmark.
}
```

Listing 2: Human-readable console output of the Example benchmark

```
|             | unit |  sum | min |  max |    avg |  stddev |     conf95     | runs |
| Example ..................................................................|
| benchMethod |   ms |  438 |   0 |    3 |  00.44 |   00.50 | [00.41,00.47]  | 1000 |
```

4 Future Work

While PERFIDIX 1.0 proved itself as a generic benchmarking tool, it still lacks important convenience features such as Java annotations and Eclipse integration. Especially the requirement to extend the class Benchmarkable and to configure it manually by implementing the main() method or run an external tool such

as Ant heralded the brainstorming for PERFIDIX 2.0. While the new features still have to be implemented, the design is already stable.

The most notable change will be the use of Java annotations as they are used with JUnit 4.x. Figure 3 shows how the Example benchmark code excerpt will look like with PERFIDIX 2.0. The annotations do no longer require the methods to follow a fixed naming, are more expressive, and even will allow configuring the benchmark on a per-method or per-class basis right in the code. With the annotations, a developer could use the same class as a unit test and a benchmark.

Listing 3: PERFIDIX 2.0 Example benchmark code excerpt

```
@BenchClass(runs=1000) public class Example {
  @BeforeBench public void setUp() { ... }    // Setup method called before each method run.
  @AfterBench public void tearDown() { ... }  // Cleanup method called after each method run.
  @Bench public void benchMethod() { ... }    // Method to benchmark.
}
```

Only the tight integration with an integrated development environment - we chose Eclipse because it is widely used and open source - will bring the desired convenience. Therefore, PERFIDIX 2.0 will come along with an Eclipse plugin that allows the developer to right-click on any package or class to find an entry in the context menu similar to JUnit, i.e., *Run as ¿ PERFIDIX Benchmark*. The plugin will then run all classes containing @Bench annotations with the class-local configuration. An Eclipse view will display the benchmarking progress to give an immediate feedback. This is especially useful for long-running benchmarks and to produce intermediate results. The configuration of each benchmark can be customized through the Eclipse run configuration.

5 Preliminary Conclusion

PERFIDIX is a tool for researchers and developers to conveniently and consistently benchmark Java code. It can quickly answer the question which implementation is faster without the need to repeatedly launch a full-fledged profiling application or rewrite the same Java code to measure the execution time again and again. Even though PERFIDIX is simple to use, it still provides a sound statistical output for taking and documenting a decision.

Our first experiences with PERFIDIX show that researchers and developers quickly integrate it in their standard tool set for every-day use because it facilitates the research and development process and supports decision making as well as persistent documentation. Still, successfully benchmarking a piece of code requires quite some artistry [14]. PERFIDIX users must as well carefully consider several aspects and find a common sense of *do's and dont's*. The JVM for example with its garbage collector, heap allocation internals, threads, object pools, and buffers, as well as the environment outside the JVM can influence the benchmark results in manifold ways.

A general advice for the developer is to first profile the whole application to find its runtime-bottlenecks. Having identified the pièce de résistance one can iteratively trim the equivocal code and document the progress with PERFIDIX. A final run with the profiler should then reflect the improvements.

PERFIDIX 2.0 will be released as an open source project under the Apache License V2.0. The primary project site will be hosted under http://perfidix.org. Sourceforge will host sources and downloads under http://perfidix.sourceforge.net.

References

[1] Eclipse. http://www.eclipse.org.

[2] Ant. http://ant.apache.org.

[3] Apache License V2.0. http://www.apache.org/licenses/LICENSE-2.0.

[4] JBench. http://www.yoda.arachsys.com/java/jbench.

[5] JUnitPerf. http://clarkware.com/software/JUnitPerf.html.

[6] JUnit. http://www.junit.org.

[7] JPerf. http://dast.nlanr.net/projects/jperf.

[8] IPerf. http://sourceforge.net/projects/iperf.

[9] JBenchmark. http://www.jbenchmark.com.

[10] Poleposition. http://www.polepos.org.

[11] TPTP. http://www.eclipse.org/tptp.

[12] JProfiler. http://www.ej-technologies.com/products/jprofiler/overview.html.

[13] GNUPlot. http://www.gnuplot.info.

[14] Raj Jain, editor. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley-Interscience, New York, 1991.