Importance and novelty

The key contribution of this dissertation is the regression benchmarking methodology described in Chapters 3–7. In addition to introducing regression benchmarking as a quality assurance tool, the methodology also includes a useful statistical model for measuring performance in the face of the non-determinism prevalent in modern systems. This addresses a very real issue faced by researchers and practitioners in the systems area. Beyond the application in regression benchmarking, the model is also relevant for researchers designing new implementations or optimizations of compilers, runtime systems, and middleware frameworks. Ultimately these researchers evaluate their innovations based on their performance improvements, and the measurement of such improvements is subject to the non-determinism covered by the statistical model introduced in this thesis. I have encountered several research projects where non-determinism in the initial state during the evaluation of a performance improvement caused significant problems. The work presented in this dissertation helps to prevent such problems in the future.

Overall structure of the thesis

Continuity is the unifying theme that fits all the components of the thesis. However, the dependency tool (Chapter 10) and the component model work (Chapter 11), which are integrated into this theme in Section 1.2 of Chapter 1, look rather isolated and are not reflected in the dissertation title or the other unifying chapters of the thesis (Chapter 2 "Regression Benchmarking", Chapter 12 "Contribution", or Chapter 13 "Related Projects and Methods").

While Chapters 10 and 11 feel rather isolated, they do document the author's skills to go beyond theory and engineer and implement the complex systems necessary for experimental performance research. Additionally, Chapter 11 allowed the author to gather extensive experience in distributed systems, which was undoubtedly helpful for inventing the architecture and design of the benchmark execution environment described in Chapters 8 and 9.

Regression benchmarking methodology

As stated before, the regression benchmarking methodology is the very valuable key contribution of this dissertation. The Mono benchmarking project can be seen as the ultimate evaluation of the methodology from Chapters 3–7. The fact that the Mono developers adopted this project proves the usefulness of the regression benchmarking methodology.

It would have been interesting to see more extensive quantitative information about the performance of the various aspects of the methodology. Such results could have included additional benchmarks, hardware platforms, and metrics:

- *Benchmarks.* For example, Table 1 in Chapter 4 includes 7 benchmarks, of which 4 are instances of the FFT benchmark on different platforms. The Mono benchmarking web site includes over 100 regression test "benchmarks". Integrating them into the evaluation would have made the results even stronger.

- *Hardware platforms.* The main focus was on the Pentium 4, which can be a source of rather significant non-determinism (e.g. due to its trace cache). Would the problem be as significant for other processors (even other IA32 implementations)? The one IA64 experiment in Table 1 in Chapter 4 indicates this might be the case.

- *Metrics.* The evaluation of the methodology would benefit from additional metrics, such as the average *number of software versions that are flagged as changes*, or an extension of the evaluation in Chapter 7, which could show the *false positive rates of change detection*, for all of the three dimensions (number of compilations, executions, measurements) of the parameter space (not just for different numbers of

compilations). It would also have been helpful to get a more concrete view of the magnitude of the problem, for example by showing the actual *averages and variances* in addition to the impact factors.

Chapter 6 describes why the two-sample approach using the overlap of confidence intervals is a good method for change detection. However, the approach may not be able to detect certain kinds of changes. For example, it is not able to detect gradual changes that happen over several versions (e.g. when more and more code is added to a critical method). Do such gradual changes happen in any of the Mono regression test "benchmarks" (and if yes, how often), or is such a gradual change scenario unrealistic?

In Chapter 4, the impact factor of the Marshaling/P4/FC2 benchmark goes down from 2.61 to only 1.2 for the transformed data. Could it be that sometimes differences between runs can't just be attributed to "static" initial state, e.g. for cases where initialization happens lazily throughout the execution, maybe beyond the "startup" phase?

The methodology introduced in this thesis is built on the idea that benchmarking means running a benchmark multiple times, where each run takes multiple measurements of the duration of a specific operation. This setup is not always the case. (1) Sometimes, especially when analyzing the performance benefits of changes to a compiler or runtime system, there are no clearly delineatable "operations", but performance improvements or regressions are distributed throughout the entire benchmark execution (e.g. in dynamic profiling, compilation, optimization, and execution of the optimized code). In such cases it is necessary to measure the entire duration of the run, instead of getting multiple samples of the duration of a specific operation. (2) In other situations, one is not interested in the average duration of an operation, but in the minimum or maximum of some metric (e.g. the minimum mutator utilization, or maximum pause time, with respect to the performance of garbage collectors).

Generic regression benchmarking environment

The BEEN regression benchmarking environment introduced in Chapters 8 and 9 is a generification of the specific environment used in the Mono benchmarking experiments. The specific environment was crucial for the evaluation of the practicability of the regression benchmarking research of Chapters 3–7, and has proven its usefulness by its adoption by the Mono project. The focus of the generic environment is on the sound engineering of such an infrastructure, and on the reduction of the effort of integrating new benchmarking experiments.

Chapters 8, 9, and 13 contrast and compare the proposed architecture and design to existing systems, highlighting the shortcomings of existing systems. Given the positive evaluation of the existing *Condor* batch execution environment (in Chapter 13), a more concrete rationale for why BEEN was not built on top of Condor (but implements its own execution framework instead) would have been informative.

In Chapter 9, BEEN is evaluated using one comparison analysis for one benchmark only. This evaluation is rather preliminary and can only serve as an indication of the true value of this generic environment. It would be nice to see an extended set of benchmarks, including regression analyses, showing the benefits of BEEN by measuring the reduction in effort (such as the lines of code metric already shown, or the number of hours spent) for adding a new experiment.

Minor specific issues, questions, and comments

This section contains some minor points related to various parts of the dissertation.

Chapter 1 (page 9) states that "Low-level performance metrics, such as ... duration of the application startup ... are of interest in specialized systems with limited system resources". In my experience, some of these metrics, in particular application startup times, are also of interest in systems without specific resource constraints, for example in runtime environments that execute interactive desktop applications.

Section 2.1 in Chapter 1 (page 20) states that "... measurement code should be simple and non-intrusive. In particular, the code should not analyze the results, but it should report them in raw format". If measurements are frequent enough (e.g. in an extreme case, when instrumenting every method call), then the large amount of raw data reported (or written to secondary storage) may significantly perturb the system. In such situations it can be beneficial to analyze and aggregate the data online.

Section 3.3 in Chapter 5 (page 70) reports on a performance regression of almost 24% in the FFT benchmark. It states that this regression was caused by a Mono JIT loop optimization introduced between August 10 and August 11. What did this optimization do to cause such a drastic decrease of performance? Was the decrease caused by the time needed to perform the optimization, or did the optimized code run slower than the code without that loop optimization?

In Section 3.3 in Chapter 5 (page 70) in the discussion of the 99% improvement of TCP Ping, the meaning of the statement "similar dependencies between benchmark results could also be described and tested" is unclear.

Section 3.3 in Chapter 5 (page 70) proposes separate regression benchmarking of changes that influence the compiler, the runtime system, or the libraries. This may not work for changes in the interfaces between these three subsystems. For example, in the JikesRVM virtual machine, some changes (such as the introduction of new magic methods) impact both the runtime library and the implementation of the compiler. If only the library part of the change was considered, the system would not build.

Section 4.1 of Chapter 4 (page 48) states "... dependency of measured times on memory cache misses ... Pentium 4 processor and shows a weak positive correlation, however the results for this platform are only approximate due to a bug in the processor which prevents precise counting of memory cache events". It is unclear which cache's performance was measured (level 1 data cache, trace cache, level 2 cache). Could it be that the measurements do not include the trace cache misses (the Pentium 4 trace cache being one of the most significant sources of non-determinism)?

The Mono benchmarking project web page for regression test "benchmarks" specifies the number of compilations (60), executions per compilation (1), and number of samples per execution (60, of which the first 15 are considered warmup-samples). Are these numbers arbitrarily chosen, or was the statistical model used to determine these numbers (by bounding the time for experiments, or the precision of the results)?

The main focus of the thesis is how to live with the non-determinism in the initial state, and how to deal with the many sources of variance in modern systems. It stresses that identifying and removing these sources of variance is hard. However, these sources of variance are not only a problem for benchmarking. With the ever increasing non-determinism in modern systems, they may also become a problem with respect to the system performance observed by end users. For example, a user running a productivity application will most probably be dissatisfied if the response time to user events changes drastically whenever the application is restarted. Research into approaches that help detecting such sources of variance should thus not be lightly discounted. Such approaches could allow designers of future systems to take more informed implementation decisions (e.g. by trading off slightly higher performance for a significant reduction in non-determinism).

Recommendation

Overall, I believe that the thesis represents a solid contribution to the research in the area of system performance and clearly demonstrates the author's ability for creative scientific work. I recommend the thesis to be passed unconditionally.

Università
della
Svizzera
italiana

**Facoltà
di scienze
informatiche**

Matthias Hauswirth, Assistant Professor
Faculty of Informatics
University of Lugano
Via Giuseppe Buffi 13
CH-6904 Lugano
Switzerland

July 29, 2006

Prof. Jiří Anděl, Vice-Dean
Faculty of Mathematics and Physics
Charles University
Ke Karlovu 3
CZ-121 16 Prague 2
Czech Republic

Dear Professor Anděl,

Herewith I submit my review of the doctoral thesis of Mgr. Tomáš Kalibera, entitled "Performance in Software Development Cycle: Regression Benchmarking". In summary, I believe that the thesis represents a solid contribution to the research in the area of system performance and clearly demonstrates the author's ability for creative scientific work. I recommend the thesis to be passed unconditionally.

Yours sincerely,

[redacted]

Complete review: Pages 2–4