VM Performance Evaluation with Functional Models

An Optimist's Outlook

Jan Sinschek
CASED
Technische Universität Darmstadt
christian.sinschek@cased.de

Andreas Sewe Mira Mezini
Software Technology Group
Technische Universität Darmstadt
{sewe, mezini}@st.informatik.tu-darmstadt.de

Abstract

The performance evaluation of virtual machines is notoriously difficult. Therefore, experimental methodology has recently drawn attention, leading to proposals on how to choose benchmarks, interpret results, and detect measurement bias. But this latter task currently relies on the presence of anomalous measurement results, i.e., on outliers, to raise suspicion. We therefore propose the use of functional performance models to detect bias even when benchmark results might appear unsuspicious. Failure to validate the model indicates either bias or a need to refine the model.

Categories and Subject Descriptors D.2.8 [*Software Engineering*]: Metrics—Performance measures; I.6.4 [*Simulation and Modeling*]: Model Validation and Analysis; D.3.4 [*Programming Languages*]: Processors—Run-time environments

General Terms Experimentation, Performance

Keywords Virtual machines, performance modeling, measurement bias, statistics, validation

1. Introduction

Modern process virtual machines (VMs) consist of many interacting subsystems: just-in-time compiler, interpreter, garbage collector, thread scheduler, profiling subsystem, etc. Due to their interaction, the performance of a VM in its entirety is notoriously hard to evaluate. Furthermore, interactions also occur between the VM and the underlying architecture. In both cases, these interactions impede the evaluation of the performance impact of modifications, even those limited to a single subsystem.

Experimental methodology consequently has become a subject of research of its own [4]. Several recent papers [1, 6] point out common flaws in experimental setup and evaluation. However, the countermeasures they propose do not suffice to avoid all pitfalls. Measurement bias in particular can invalidate conclusions [8]; when applied naïvely, statistical methods are oblivious to it.

Only part of the bias manifests itself as outliers and is easily detected. To avoid biased measurements, Mytkowicz et al. propose to use "diverse evaluation workloads" while "randomizing [the] experimental setup" [8]. This, however, puts a strain on the researchers' experimental budget, as many potential sources of bias exist. While it is certainly worthwhile to vary the architecture used in experiments, being a common source of bias, other variations may offer diminishing returns – unless there is cause for suspicion. In that case, Mytkowicz et al. propose the use of causal analysis to detect bias. By devising an intervention and measuring with and without that intervention in place, one can confirm that a given variation indeed affects performance, and thus constitutes bias.

But one question remains: What shall be the cause for suspicion in the first place? In this paper we propose a framework for detecting bias, which attempts to answer this question, namely devising a functional model of the modifications expected impact. This framework to detect and investigate bias is inspired by modeling approaches in the natural sciences. When a model is invalidated, the researchers can either refine it or they can attempt to determine and then eliminate the source of the bias.

Costs of performance-relevant actions established as part of a model can be verified independently of the modifications made. Furthermore, the benchmark characteristics determined as part of the model's validation contribute to the community's knowledge.

2. Background: Prevalent Methodology

Typically, in performance evaluation an account is given of the performance that a VM exhibits before and after modification when running a set of benchmarks. If the modification introduces new functionality, it is also common to perform micro-benchmarking to quantify the cost of using the newly-implemented functionality or to single out a source of overhead.

Attention has recently been drawn to three important aspects of evaluation: the representative choice of benchmarks, the statistically rigorous interpretation of results, and the detection and avoidance of measurement bias. Blackburn et al. criticize the use of dated benchmarks and summarize sources of non-determinism that can be found in managed execution environments, i.e., in VMs [2]. Georges et al. [5] demonstrate how lack of rigour during statistical analysis can lead to erroneous conclusions. Mytkowicz et al. [8] show that measurement bias is an orthogonal issue; it causes benchmark results not to be representative. Sources of bias that have been investigated include code movement [6], thread-scheduling [7], and even the intrinsic characteristics of the VM itself [3].

Bias can only sometimes be perceived through outliers. To counteract many forms of bias, Mytkowicz et al. recommend varying the experimental setup to prevent bias.[1] This, however, is not feasible for all conceivable causes of bias.

To summarize, the best practices presented in the aforementioned papers address some of the intricacies of VM performance evaluation. While not all researchers use these techniques, the

[1] Note that this entails, in the light of a limited experimental budget, preferring varied configurations over many iterations of a single configuration.

awareness for the quirks of VM performance has certainly risen: In a survey of 68 papers from recent first-rate conferences[2] a total of 38 papers employ carefully chosen benchmarks, whereas 25 papers rigorously apply statistics.

Still, the use of these evaluations is often limited, as the results themselves do not establish reusable knowledge about the system under consideration; in particular, it is often unclear whether results carry over to other setups. We believe that performance models may make this possible.

3. Approach: Performance Modeling

The purpose of a physical model is twofold: it identifies the forces at work and quantifies them. Much in the same way, a *functional performance model* describes how a specific functionality in the system works. Its purpose is to predict how the system's performance will react to changes in the workload. Furthermore, the functional performance model should be able to reflect both modified and unmodified systems.

Mytkowicz et al. [8] abstractly speak of a system S and its optimization O that together form a new system $S + O$. We generalize this by having O denote any modification, whether optimizing or not, and considering S and O to be functions that describe VM performance. (This notation must obviously be taken with a grain of salt, as completely independent subterms for system and optimization might be unattainable.) We can then categorize the terms in such a functional performance model as follows:

Dimensionless quantities n are under the researcher's control. These are, e.g., the number of optimization opportunities, the number of instrumentation points, or the sampling rate.

Constants γ vary with the VM configuration or the underlying architecture. These are the benefits of an optimization opportunity taken, the cost of code instrumentation or of taking a sample.

Opaque terms correspond to the performance-relevant actions of the VM that do not vary when n is changed.

As an example, consider modeling the cost of code instrumentation. The set I of instrumentation points can be established for each benchmark setup. The researchers can easily vary it by performing only a fraction of the instrumentations. As every instrumentation point i is reached a different number of times m_i, these coefficients must be established for all i. The instrumentation's total execution count n is thus only indirectly controlled by the researchers. Still, the cost γ of encountering any such point can be derived from multiple measurements with varying n by considering them jointly as a system of overdetermined equations. These γ-values constitute what we analogize to physical constants. By not instrumenting anything, i.e., $n = 0$, we arrive at $\big(S + O\big)(n) = S$; the performance of the rest of the system becomes an opaque term. If this is not true, bias has already manifested. The above model likely needs refinement; it might be necessary to distinguish types of program points at which instrumentation is more expensive.

As there is always noise in performance measurements, the correctness of a model's predictions needs to be established statistically, e.g., by a location test, which indicates whether the hypothesis implied by the function is in accordance with the data. If the measurements contradict the prediction, bias has been detected. Its source must then be found and accounted for. This can be done in two ways: by suppressing the bias' source on subsequent measurements or by making it a part of the model in its own right.

[2] We surveyed the conferences on Architectural Support for Programming Languages and Operating Systems (ASPLOS); Object-Oriented Programming, Systems, and Applications (OOPSLA); Programming Language Design and Implementation (PLDI); Virtual Execution Environments (VEE).

4. Discussion and Outlook

Using functional performance models as a complement to representative benchmarks and rigorous analysis has two key advantages: The first advantage is its ability to detect some sources of bias that affect a modification. The second is its reusability and its ability to make predictions. When a modification introduces a feature, the model can give an idea of the cost of using the feature before actually implementing it.

Our proposal also has a number of limitations. The modeled performance must be reproducible within the boundary of statistical limitations. This requirement is challenged by nondeterminism as caused, e.g., by the VM's or the modification's intrinsic complexity or by thread-parallelism [7].

The framework as proposed also cannot detect every kind of bias. For instance, the use of a certain architecture can easily influence the outcome of the evaluation, but a functional performance model validated on that architecture alone will not be falsified by this. It is therefore useful to publish the model, the n-values and the empirical γ-values as part of an evaluation. However, such a model must not be too specialized so to not overfit the data and describe measurement noise rather than performance-relevant phenomena.

Our suggestion has not yet explored the statistical methods that should be applied to account for the unavoidable variations of the model's constants and opaque terms.

Despite these obstacles, we are optimistic that using a functional performance model can increase the researchers' understanding of a modification's performance impact. Its publication helps the community to assess what optimizations are still possible and which are the benchmarks' relevant characteristics.

Acknowledgments

This work was supported by CASED (www.cased.de).

References

[1] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann. The DaCapo benchmarks: Java benchmarking development and analysis. In *Proceedings of the 21st OOPSLA Conference*, pages 169–190, 2006.

[2] S. M. Blackburn, K. S. McKinley, R. Garner, C. Hoffmann, A. M. Khan, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanovik, T. VanDrunen, D. von Dincklage, and B. Wiedermann. Wake up and smell the coffee: evaluation methodology for the 21st century. *Communications of the ACM*, 51(8):83–89, 2008.

[3] L. Eeckhout, A. Georges, and K. De Bosschere. How Java programs interact with virtual machines at the microarchitectural level. In *Proceedings of the 18th OOPSLA Conference*, pages 169–186, 2003.

[4] A. Georges. *Three Pitfalls in Java Performance Evaluation*. PhD thesis, Universiteit Gent, 2008.

[5] A. Georges, D. Buytaert, and L. Eeckhout. Statistically rigorous Java performance evaluation. In *Proceedings of the 22nd OOPSLA Conference*, pages 57–76, 2007.

[6] D. Gu, C. Verbrugge, and E. M. Gagnon. Relative factors in performance analysis of Java virtual machines. In *Proceedings of the 2nd VEE Conference*, pages 111–121, 2006.

[7] P. Kulkarni, M. Arnold, and M. Hind. Dynamic compilation: the benefits of early investing. In *Proceedings of the 3rd VEE Conference*, pages 94–104, 2007.

[8] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney. Producing wrong data without doing anything obviously wrong! In *Proceedings of the 14th ASPLOS Conference*, pages 265–276, 2009.