

Benchmarking Autonomic Capabilities: Promises and Pitfalls

Aaron B. Brown¹, Joseph Hellerstein¹, Matt Hogstrom², Tony Lau³,
Sam Lightstone³, Peter Shum³, Mary Peterson Yost⁴

¹IBM® T.J. Watson Research Center (Hawthorne, NY), ²IBM® Raleigh Laboratory (Raleigh, NC),

³IBM® Toronto Laboratory (Toronto, ON), ⁴IBM® Autonomic Computing (Hawthorne, NY)

Contact e-mail: abbrown@us.ibm.com

1. Introduction

Benchmarks provide a way to quantify progress in a field. Excellent examples of this are the dramatic improvements in processor speeds and middleware performance over the last decade, driven in part by SPEC® and TPC™ benchmarks. We feel that developing appropriate benchmarks for autonomic computing will similarly drive progress towards self-managing systems. Our goal is to produce a suite of benchmarks covering the four categories of autonomic capabilities: self-configuring, self-healing, self-optimizing, and self-protecting [2]. This is not an easy task, however, and in this paper we identify several of the challenges and pitfalls that must be confronted to extend benchmarking technology beyond its traditional basis in performance evaluation.

Basics of autonomic benchmarks. Benchmarks in the performance space have followed a structure like that depicted in Figure 1(a). A *system under test* (SUT) is deployed in a stable benchmark environment and subjected to a synthetic *workload* designed to be representative of typical system use. Benchmark results are derived from how quickly the SUT can satisfy the imposed workload, as measured by the benchmark driver. During measurement collection, there is no administrator intervention.

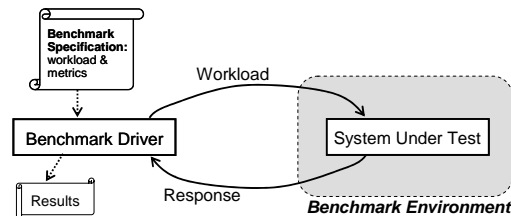


Figure 1(a). Traditional Performance Benchmark

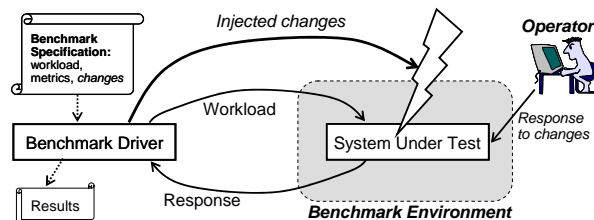


Figure 1(b). Benchmark for Autonomic Capability

Figure 1(b) illustrates in general terms how existing benchmarks should be extended in order to quantify the autonomic characteristics of the SUT. We retain the salient features of a representative workload and a benchmark driver. But to capture the essence of autonomic computing—adaptation—we must now introduce *change* into the heretofore stable benchmarking environment. For example, we might inject faults into the SUT to evaluate its ability to self-heal, as in dependability benchmarks [3] [5]. Another example is modifying the workload driver to inject “flash events” [4]. Still another example is injecting configuration change requests to evaluate self-configuration. Finally, a benchmark for self-protection might employ simulated attacks to quantify this aspect of autonomic capability.

2. Challenges in Autonomic Benchmarking

By injecting changes into the benchmarking environment, we significantly expand the scope of the benchmarking process. An autonomic benchmark must supplement the performance workload with a representative, reproducible set of changes to inject. It needs a much broader set of metrics to capture the SUT’s responses, supplementing traditional performance metrics with quantitative measures of how well the SUT adapts to the injected changes. And, to handle systems that are only partially-autonomic (as are most systems today), the benchmarking process must make accommodation for systems that require active human administrator support, and for unexpected behavior in the SUT, such as crashes and erroneous responses. An overarching challenge is to accomplish this expansion in scope while retaining the familiar structure of performance benchmarks. Not only must we surmount considerable technical hurdles, but we must also educate benchmark consumers about the need for these extensions.

Injecting changes. Besides selecting the set of changes to introduce (itself a nontrivial, albeit practical problem), there are two key challenges in injecting changes. The first is to ensure that the benchmark remains *reproducible* despite injecting the change. We must en-

IBM is a registered trademark of International Business Machines Corporation in the United States, other countries, or both. Other company, product and service names may be trademarks or service marks of others.

sure that individual changes can be injected reproducibly, and must coordinate injected changes with the applied workload. For the benchmark to be useful in cross-system comparisons, changes must also be reproducible across different systems. There is a two-part approach that addresses all of these challenges: (1) use only the subset of the changes that is common to all systems being benchmarked; and (2) synchronize the injector with the performance workload. This approach sacrifices representativeness for reproducibility. It also raises the issue of *gaming*, or undesirable benchmark-specific optimization by the benchmark's users. To address gaming, it may be necessary to develop equivalence classes of injectable changes, choosing among randomly-selected instances in each benchmark run.

The second challenge concerns the *representativeness* of the injected changes. Unlike a performance workload, which can be simulated in isolation, environmental changes might require a large-scale simulation infrastructure approaching the complexity of a real deployment. Such a requirement arises in multiple aspects of autonomous computing: self-protection may require simulating a distributed denial-of-service (DoS) attack; self-healing may require injecting faults in a distributed systems infrastructure; and self-configuration may require a complex multi-system deployment request. Significant resources may be needed to successfully inject these changes unless tricks can be found to simulate their effects with fewer resources. Furthermore, there may be additional concerns if some forms of injected changes are made too realistic: containment risks and legal issues with maintaining the viruses and DoS attacks needed to test self-protection; the risk of permanent system damage from fault injection into the hardware; and so on. Finding the balance between representative changes and benchmark practicality reflects the art of benchmark design.

Metrics and scoring. Autonomous benchmarks must quantitatively capture four dimensions of a system's autonomous response: the *level* of the response (how much human administrative support is still needed), the *quality* of the response (how well it accomplishes the necessary adaptation), the *impact* of the response on the system's users, and the *cost* of any extra resources needed to support the autonomous response. The quality and impact dimensions can be quantified relatively easily by measuring end-user-visible performance, integrity, and availability both during and after the system's autonomous response. The level dimension is harder to quantify because it requires an assessment of human involvement with the system. Scorecard-based approaches—where the benchmarker assigns the system a score based on a qualitative assessment of its response—offer promise, but need further investigation to determine if they are reproducible and effective at differentiating among systems at different levels of autonomous maturity.

Other issues related to metrics and scoring involve the challenge of synthesizing multiple sub-metrics (e.g., for

the different autonomous capabilities) into an overall measure of autonomy, and calibrating scores across different systems. The latter issue must be solved before competitive autonomous benchmarking is possible; we are exploring an approach that requires systems to provide a basic predefined set of baseline autonomous mechanisms for establishing common cross-system comparison points.

Handling partially-autonomous systems. Partially-autonomous systems include some autonomous capabilities, but still require some human administrative involvement to adapt fully. For example, a system might diagnose a problem and suggest a course of action, but wait for an administrator's OK before completing the self-healing process. An autonomous computing benchmark should provide useful metrics for such systems so that we can quantify the steps towards a fully autonomous system. But the benchmark cannot easily simulate a "representative, reproducible human administrator" to complete the SUT's autonomous loop. Human involvement in the benchmark process itself may need to be considered, as in [1], in which case the benchmark scope expands to include aspects of human user studies, with statistical techniques used to provide reproducibility. An alternative approach is to break the benchmark into separate phases such that human intervention is only required between phases. Each phase would then be scored individually, with a penalty applied according to the amount of inter-phase human support needed. This could be a simple time penalty added to the benchmark result, or a more complex scoring scheme. The phase-based approach also may help with benchmarking of systems that crash or fail in response to injected changes, because failure in one phase can be treated independently from behavior in other phases.

3. Conclusion

While the promise of autonomous benchmarks is clear, a key question facing their developers is how we will address the challenges enumerated above. We hope to open a dialog with others working in the areas of autonomous computing and non-traditional benchmarking to discuss these issues and potential solutions, and to understand how various solutions will impact the value and applicability of an autonomous benchmark.

4. References

- [1] A. Brown, L. Chung, et al. Dependability Benchmarking of Human-Assisted Recovery Processes. *Submitted to DSN 2004*.
- [2] IBM. *Architectural Blueprint for Autonomous Computing*. <http://www.ibm.com/autonomic/pdfs/ACwpFinal.pdf>, 2003.
- [3] K Kanoun, H. Madiera, and J. Arlat. A Framework for Dependability Benchmarking.. *2002 Workshop on Dependability Benchmarking (DSN 2002)*. Washington, D.C., June 2003.
- [4] E. Lassetre, DW Coleman, Y. Diao et al. Dynamic Surge Protection: An Approach to Handling Unexpected Workload Surges With Resource Actions that Have Lead Times. *DSOM*, 2003.
- [5] H. Madeira and P. Koopman. Dependability Benchmarking: making choices in an n-dimensional problem space. *Proc. EASY '01*, Göteborg, Sweden, 2001.