Measuring the Effectiveness of Self-Healing Autonomic Systems

Aaron B. Brown[1] and Charlie Redlin[2]
[1]*IBM T.J. Watson Research Center (Hawthorne, NY),* [2]*IBM Rochester Laboratory (Rochester, MN)*
{abbrown,redlin}@us.ibm.com

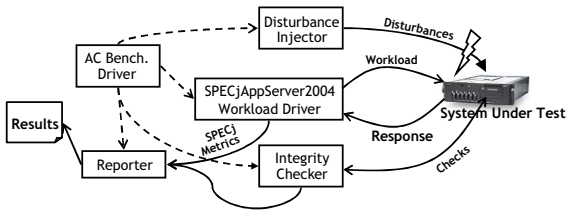
1. Introduction

Benchmarks are a central force in engineering progress, providing the objective ability to quantify improvement and justify design decisions. In previous work, we brought together the concepts of benchmarks and autonomic computing, describing a vision of benchmarks that quantitatively evaluate a computing system along the four core autonomic dimensions of self-healing, self-configuration, self-optimization, and self-protection [1,5]. In this paper, we describe our experience with implementing a practical benchmark for the self-healing dimension of autonomic capability, which goes beyond simple measures of fault tolerance [4] by including a measure of autonomic maturity. Our benchmark is capable of quantifying the autonomic self-healing capability of complex, production-scale enterprise solutions based on J2EE middleware (and indeed is currently being used for such purposes).

2. Architecture of self-healing benchmark

Our benchmark architecture, depicted in the figure below, follows the basic pattern described in [1] and injects *disturbances* into a System Under Test (SUT) subjected to a performance workload. The SUT includes all components necessary to run the SPECjAppServer® 2004 performance workload [6], which simulates a realistic enterprise-class e-commerce application with manufacturing, supply chain, and inventory components and web, Enterprise Java™ Bean, messaging, and database tiers. Note that we use only the workload driver component of SPECjAppServer2004, and thus we are not conducting compliant SPECjAppServer runs or reporting SPECjAppServer results. Around that workload driver we wrap four additional components: a disturbance injector, a data integrity checker, a reporting module, and a coordination driver.

Our benchmark injects 30 different types of disturbances representing common expected failure modes for multi-tier enterprise application systems, including component shutdowns, data loss, resource exhaustion, load surges, operator errors, and restart failures. By adding more types of disturbances to this set in the future (such as security attacks and workload shifts), we expect to extend the benchmark to other autonomic capabilities.



Each disturbance is injected individually in a *slot*, which is a fixed-length interval that includes a period of steady-state operation preceding the injection, a detection and recovery period, and a period of steady-state operation following recovery. If the SUT cannot autonomically detect or recover from a disturbance, we apply a fixed time penalty then use scripts to reproducibly simulate the manual actions needed for detection or recovery.

We quantify the SUT's self-healing capability with two metrics: (1) a measure of how effectively the SUT heals itself in response to the injected disturbances, and (2) a measure of how autonomic that healing response is. In our initial implementation, the first metric is calculated quantitatively as the ratio of the number of SPECjAppServer2004 requests that complete successfully during the injection slot to the number that complete successfully during a baseline slot with no disturbance injected. The second metric is calculated via a 90-question survey that assigns points to the SUT based on the level of automation present in its response to each disturbance, following the 5-point classification defined in the IBM Autonomic Computing Maturity Model [3]: 0 points for a basic manual response, 1 point for a managed response, 2 for predictive, 4 for adaptive, and 8 for autonomic. Both metrics are calculated separately for each disturbance to provide detailed feedback on the SUT's behavior, but can be aggregated across disturbances (via a weighted average) to produce summary scores in the [0,1] range.

3. Example result

To explore the capabilities of our prototype benchmark, we built two SUT environments based on clustered J2EE middleware and single-instance database, web, and messaging servers. SUT #1 did not include autonomic functionality; SUT #2 included a set of system manage-

ment technologies that we were evaluating for degree of autonomic capability (the details of the SUT configurations are elided for space reasons).

Our baseline run on SUT #1 resulted in an average healing effectiveness score of 0.79 and an autonomic maturity score of 0.15 (both out of 1.0), indicating a relatively low level of autonomic self-healing capability. In comparison, SUT #2 attained an effectiveness score of 0.83 and a maturity score of 0.22. Comparing the two results indicates that SUT #2's system management technology provided a small—but measurable—improvement in autonomic capability. The detailed per-disturbance results provide insight into where SUT #2's technology was beneficial (in this case, in improving the autonomic handling of component shutdowns), as well as in identifying gaps in the technology (for example, highlighting classes of disturbances that confuse the SUT's self-healing mechanisms, resulting in worse behavior). These example results indicate the power of our benchmark in providing quantitative technology comparisons and in steering autonomic improvement efforts toward current problem areas.

4. Issues and discussion

While our results (such as the example above) demonstrate the utility of our initial self-healing benchmark implementation, there still many areas where the benchmark could benefit from additional sophistication, and a set of challenging issues to address in these areas.

Quantifying autonomic maturity. Since autonomic computing is still in its infancy, most systems today have low levels of autonomic maturity. More granularity is needed in the benchmark's maturity score to differentiate these systems; also, a more objective, directly-quantified metric is desirable. We are pursuing an approach based on complexity analysis of manual healing processes [2], and hope to ultimately map the quantitative measurement of maturity into dollar cost.

Quantifying healing effectiveness. Our current measure of effectiveness considers only throughput of correctly-handled requests (subject to a response time cutoff). To get a more complete picture of effectiveness, these metrics must be extended to capture broader impacts of disturbances, and in particular the impact of disturbances on SUT capacity and response time distribution. Adding a measure of capacity degradation is the most challenging aspect, since, depending on the workload, capacity, throughput, and response time can be linked.

Accounting for incomplete healing. A complete self-healing cycle includes *bypassing* the component(s) affected by a disturbance, *repairing* those components, and *reintegrating* them into the SUT. A SUT might complete only some of these stages of healing in response to a disturbance, or alternately may simply *tolerate* the disturbance without any active healing process. The degree of healing is not always evident from the effectiveness and maturity scores we have described, yet it can affect the SUT's ability to tolerate future disturbances. An open issue is whether the benchmark can be extended to identify the extent of self-healing, perhaps via additional rounds of disturbance injection.

Accounting for healing-specific resources. A SUT may include resources dedicated to self-healing capability (such as monitors, hot standby cluster nodes, spare disks, etc.). To prevent situations where a system is intentionally over-provisioned in order to get a good self-healing benchmark score, the cost of these extra resources must be factored into the benchmark result. Currently, we simply require that resources and their utilizations be reported along with the benchmark results. We envision a more advanced benchmark that measures these quantities automatically and maps them to cost; combined with the cost-based maturity metric proposed above, this would allow the benchmark to estimate the overall cost (in resource and labor) for the SUT's healing capability.

Unified metrics. Ultimately, we imagine the results of the self-healing benchmark (per-disturbance or in aggregate) being reported on a 2-D space, with one axis measuring effectiveness of self-healing as discussed above, and the other measuring the cost of the healing capability (including the cost of healing-dedicated resources and the cost of human labor to fill in the gaps in autonomic response). These are the natural evolution of our throughput-degradation and autonomic maturity metrics, although significant research work will be required to define and validate these high-level metrics.

5. Conclusion

Our implementation of the first benchmark for autonomic self-healing capability demonstrates the feasibility and utility of autonomic computing benchmarks, and provides users with a quantitative way to measure the resiliency of their IT systems. However, there remains a great deal of work ahead, to extend the existing benchmark to address some of the issues discussed above, and to move beyond self-healing to other autonomic capabilities. We believe that quantitative benchmarks are critical to the success of autonomic computing, and look forward to future progress in this area.

4. References

[1] A.B. Brown, J.L. Hellerstein, et al. Benchmarking Autonomic Capability: Promises and Pitfalls. *ICAC 2004*, NY, NY, June 2004.

[2] A.B. Brown, A. Keller, and J.L. Hellerstein. A Model of Configuration Complexity and Its Application to a Change Management System. *IM 2005*, Nice, France, May 2005.

[3] IBM Corp. *Architectural Blueprint for Autonomic Computing*, www-03.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf, 2004.

[4] K. Kanoun, H. Madiera, and J. Arlat. A Framework for Dependability Benchmarking. *2002 Workshop on Dependability Benchmarking (at DSN 2002)*, Washington, D.C., June 2003.

[5] S. Lightstone, J. Hellerstein, et al. Towards Benchmarking Autonomic Computing Maturity. *IEEE Workshop on Autonomic Computing Principles and Architectures,* Banff, Alberta, Canada, 2003.

[6] *SPECjAppServer2004 Design Document*, Version 1.00, 2004, http://www.spec.org/jAppServer2004/docs/ DesignDocument.html.