

A Look at Benchmarking the Dependability of SW & OS's

Andréas Johansson Robert Lindström Neeraj Suri
Department of Computer Engineering, Chalmers University, Sweden
{aja, rl, suri}@ce.chalmers.se

1. Introduction

The objective assessment of capabilities, be it as stand-alone or relative evaluations, of a component or a system invariably requires some form of quantification of the desired attributes of interest. For example, the architecture community obtains “performance” benchmarks for micro-processors or systems by measuring the rate of computations or by assessing the time requirements for programs of varied complexity, for example CPU SPECS [1]. The intent is to evaluate and compare the capabilities across systems.

In present day systems, two trends appear. Firstly, the pervasiveness of embedded systems is growing, and consequently our dependence on their correct functionality, i.e., dependable behavior. Second, software is increasingly driving and defining the functionality for most embedded systems. In this context, similar to the efforts undertaken in the performance benchmark community, it follows to be able to benchmark the expected “dependability” of software (SW) and of operating systems (OS) upon which most embedded services are built.

As with other forms of benchmarking (performance, reliability, robustness etc.), the key issue is to be able to quantify the level of trust a user, e.g., end-user or system designer, can expect the SW/OS to provide, in order to design or build a desired service with or from it. Or a user may utilize it as a form of comparative assessment across systems. For both of these issues, the connotation is to (a) ascertain the resilience of the given SW/OS for a specified type and level of perturbations, or (b) ascertain the degradability of services analogous to QoS metrics for communication networks. These are the issues driving the currently running Dependability Benchmarking (DBench) [2] project. This research is sponsored in part by EU DBench IST 2000-25425.

However, two basic issues arise prior to technically developing the benchmarking process for SW and OS's.

SW and OS's by their very nature imply complex interactions across constituent modules, where the interactions are direct (across objects) or indirect (via threads, clients, shared variables, pointers etc.) along with dealing with a large number of functional attributes (memory/variable management, scheduling, consistency etc). Determining the functional boundaries adds another degree of complexity. For an OS, is the kernel the boundary of a “system”? Or does one need to consider drivers, API's and applications as a complete package for benchmarking? This and related issues of what is

measurable and instrumentable in an OS further complicates the issue. Does one require source-code level of details (white box) for both SW/OS's or can one work with fuzzer gray box/ black box SW modules? How does one account for or handle COTS components or third-party application SW?

The second issue is more fundamental in nature. What characterizes “dependability” in SW/OS? Does reliability of SW have a practical connotation if SW is not a single program/application but requires interactions as in the case of an OS? How do the measures of dependability – reliability, availability, security, and safety - translate in the context of SW/OS? Also, the benchmark will be very much influenced by the application area and the operational environment. This includes the choices for workload and the choice of perturbations, e.g., fault-load or stress-load. Similarly, the service expected from the OS determines the nature of faults. For example, in a RT-OS, missing task deadlines on account of high system load translates into error conditions. Experience from the OS area suggests that human interaction, e.g., maintenance and routine work, must be considered as well [3].

On this background, our project interest is to outline both the process of obtaining benchmarks and also defining how to utilize the obtained benchmarks. Currently, the project entails three themes, which are discussed at a high level in this abstract. The primary emphasis is on outlining the benchmark properties. The following sections outline the themes driving the project, namely: (a) what should be representative properties of a benchmark, (b) what processes can be utilized for conducting benchmarks and (c) how does one interpret and utilize benchmark results?

2. Benchmark Properties

The properties presented below are the set of properties a benchmark needs to fulfill in order to be experienced as useful to the user and to produce credible and meaningful results. None of these properties can be viewed as stand-alone. They are to different degrees dependant on each other and trade-offs need to be conducted across them.

Reproducibility: To receive any kind of credibility the benchmark result must be reproducible, at least on a statistical basis, to a third party.

Representativeness: The most important property for a benchmark is, that the benchmark is representative. The complexity of the problem makes this property the most difficult to fulfill. If the benchmark is not representative for the

application area the result tells nothing of the behavior of the system in a real situation.

Portability: Comparisons across the application area requires portability, though factors such as observability, intrusion and interference may be inter-related.

Intrusion: Any intrusion on the target system, be it physical modification or software modification, is costly and reduces portability and should consequently be avoided.

Interference: Too much uncontrolled interference on the target system means that the measurements are not conducted on the “original” system. Therefore, means to estimate and possibly reduce interference is needed.

Observability: The ability of the instrumentation mechanism to probe the system limits the type and level of detail of measurements that can be conducted.

Cost: The cost associated with the benchmarking activity is always an important issue. The cost of a benchmark should be lower than other techniques to be of any value.

Automation: The level of automation in the benchmark process affects the usage and the cost of the benchmark.

Execution time: The time to conduct the benchmark includes setup, execution and analysis time. The execution time should be short (also good for reproducibility).

Scaling: To make the benchmark usable to systems of different sizes, scaling rules are needed. Scaling affects mostly the workload and the faultload. The two must be scaled together due to their inherent interaction.

3. Benchmark Measurement Techniques

A multitude of academic and real-world techniques exist for dependability assessment such as stress testing, fault injection [4], modeling and robustness testing [5]. However, in our opinion, none of these offers, by themselves, the services that a SW/OS level dependability benchmark requires. A benchmark should satisfy the properties mentioned above. None of the mentioned techniques offer this today. Also, the complex interactions within SW/OS's preclude the use of only one of the mentioned techniques, as each details partial and focused aspects of the systems. However, it is plausible that a combination of these aforementioned techniques, in conjunction with new techniques, can be used for our purpose. Indeed, finding a combination of techniques that meet the properties stated in Section 2 is a primary ongoing research activity.

4. Benchmark Interpretation

Assuming that a viable assessment process, as in Section 3, can indeed be developed, a basic issue when conducting any sort of experimental activity is how to interpret the results. The multi-faceted nature of SW/OS systems makes it difficult to directly interpret the result of the benchmark. We do not currently believe that a single numerical benchmark score will accurately represent an OS characterization. A vector covering a string of attributes may be a better approach.

However, the very nature of attributes of SW/OS's probably precludes the assignment of simple numbers to quantify them. This implies that some kind of grading system must be used in order to put them in “classes”, reflecting the level of service they provide. How this grading is accomplished warrants more research. Also, the multitude of attributes of interest means that they constitute a vector of both numbers and grades. How to quantify these vectors and compare them is an open question. Rules on how to make comparisons between vectors, with respect to the context, will need to be defined prior to the notion of benchmarking having any real or tangible interpretation.

5. Conclusions

In this paper we have outlined the problem of benchmarking SW/OS's. Before a benchmark is developed several unresolved questions must be considered, such as how to represent the complex interactions between modules in SW systems, defining the boundaries of a system in a white/gray/black box SW, the impact of applications on OS's? Other important issues include understanding how to define and interpret the concept of dependability for SW/OS assuming it is measurable per se.

At present, work within the project includes detailing a comprehensive list of properties that a dependability benchmark must fulfill in order to give reliable results and to be useful for the user, and understanding the inter-play across these properties required to obtain trade-offs across them. Also, the research targets determining the enabling techniques for assessment of dependability in SW/OS's, including a perusal of the viability of using combination of existing techniques and development of new techniques.

The multidimensional nature of large SW/OS makes it impossible to use one single number to measure it. A substantial focus of the project is dealing with determining vectors with composite numerical and non-numerical scores to express the grading of attributes of SW and OS's. Development of rules to obtain and relatively compare such attribute grading vectors constitutes ongoing research.

References

- [1] J. L. Henning, “SPEC CPU2000: Measuring CPU performance in the new millennium”, *IEEE Computer*, pp.28-35, July 2000.
- [2] K. Kanoun, et al, “DBench (Dependability Benchmarking)”, in *Workshop on The European Dependability Initiative*, DSN-2001, pp. D12-15, July 2001.
- [3] B. Murphy, “Windows 2000 Dependability”, in *Workshop on Dependable Networks and OS*, DSN-2000, pp. D20-28, 2000.
- [4] J. Arlat, et al., “Fault Injection and Dependability Evaluation of Fault-Tolerant Systems”, *IEEE TOC*, pp.913-23, Aug. 1993.
- [5] P. Koopman and J. DeVale, “Comparing the Robustness of POSIX Operating Systems”, *FTCS-29*, pp.30-37, 1999.