Dependability Benchmarking of Computing Systems

Cristian Constantinescu (Intel), Karama Kanoun (LAAS-CNRS), Henrique Madeira (University of Coimbra), Brendan Murphy (Microsoft), Ira Pramanick (Sun Microsystems), Aaron B. Brown (IBM)

The importance of benchmarking is increasing as every aspect of human life is relying on correct operation of computing systems. Although considerable efforts have been made, presently there are no widely accepted dependability benchmarks. The panelists, in the order shown above, address benchmarking of computer hardware, operating systems, applications, and systems.

A Black-Box Approach to Dependability Benchmarking of Computer Hardware

Examples of silent data corruption (SDC), induced by intermittent faults, are given. Two techniques, based on environmental tests, are used to detect SDC: an operating temperature and voltage test, and an electrostatic discharge test. Systems under evaluation execute a Linpack benchmark, which checks correctness of the calculations. Advantages of using environmental tools are two fold. i) systems under evaluation are viewed as “black boxes”, i.e. no proprietary data is required. ii) Linpack benchmark is available at no cost.

Benchmarks for Operating Systems (OSs)

A dependability benchmark for OSs wrt faults induced from the application layer via the OS API is presented. The OS is seen as a black box. System calls are intercepted and corrupted. Three measures are then evaluated: OS robustness (its ability to detect specific application faults), OS reaction time, OS restart time. Several versions of Linux and Windows families are compared using i) Postmark (a realistic workload of a performance benchmark) and ii) Java Virtual Machine (JVM). Sensitivity analyses wrt the faultload used are shown, as a part of the benchmark validation process.

Benchmarking the Dependability of Transactional Systems

Two dependability benchmarks are discussed: a benchmark for database centric systems and another one for web server based systems. The measures collected include system availability and measures of the service (execution of transactions or web-server operations) in the presence of the faultload. The types of faultloads considered include faults, upsets, or stressful situations that might be caused by erroneous human actions, software faults or hardware component failures. The validation of the benchmarks is also discussed.

Ensuring a Benchmark is Relevant

While customers are interested in the reliability of individual components, such as computers, they are far more focused on the reliability of their solutions. The dependability benchmark should be similar to TPC benchmark and not attempt to constrain the system under test, rather it should define a task and let the supplier define the system that best solves that task. The focus of the benchmark should be to measure the dependency, resiliency, performance and cost of the systems to provide a solution under a defined set of fault loads. Impact of human failure should be considered as well.

Availability Benchmarking of Systems

A tractable approach to evaluating overall system availability is to measure its components, some of which can be benchmarked relatively easily while others need to be assessed via alternative methods such as field data analysis. Thus what is needed is first a decomposition of system availability into its components, and then a determination of the methods to measure these components. A hierarchical approach, called the R-cubed (for Rate, Robustness and Recovery) framework, is presented. The R-cubed framework calls for a suite of single attribute metrics, each measuring one aspect of system availability, which together provide a complete picture.

Benchmarking Dependability & Self-Management: IBM’s Autonomic Computing Benchmark

This benchmark extends previous academic work to cover automated system management, including self-optimization, self-configuration, and self-protection. The goal is to measure the impact of a broad range of disturbances injected into the environment of a system under test, focusing in particular on the enterprise middleware domain and J2EE applications. Disturbances include injected errors, unexpected shifts in workload, configuration changes, deployments and upgrades, human errors, and security incidents. Impact is quantified in terms of effect of a disturbance on delivered quality of service, as well as the degree of manual involvement that is required to recover from the disturbance.