

Submitted to the 2002 DSN Workshop on Dependability Benchmarking

Capturing the Human Component of Dependability in a Dependability Benchmark

Aaron B. Brown, Leonard C. Chung, and David A. Patterson
Computer Science Division, University of California at Berkeley
387 Soda Hall #1776, Berkeley, CA 94720-1776, USA
{abrown,leonardc,pattrsn}@cs.berkeley.edu

Abstract

Motivated by the observation that a system's dependability is significantly influenced by the behavior of its human operators, we describe the construction of a dependability benchmark that captures the impact of the human system operator on the tested system. Our benchmark follows the usual model of injecting faults and perturbations into the tested system; however, our perturbations are generated by the unscripted actions of actual human operators participating in the benchmark procedure in addition to more traditional fault injection. We introduce the issues that arise as we attempt to incorporate human behavior into a dependability benchmark and describe the possible solutions that we have arrived at through preliminary experimentation. Finally, we describe the implementation of our techniques in a dependability benchmark that we are currently developing for Internet and corporate e-mail server systems.

Keywords: Dependability benchmarking, human operators, operator error, e-mail, fault injection

Submission category: Paper submission

Word count: approx. 5200 words

The material included in this paper has been cleared through authors' affiliations.

Contact Author:

Aaron Brown
University of California at Berkeley
477 Soda Hall #1776
Berkeley, CA 94720-1776, USA

Phone: +1-510-642-1845

Fax: +1-510-642-5775

Email: abrown@cs.berkeley.edu

This page intentionally left blank

Capturing the Human Component of Dependability in a Dependability Benchmark

Aaron B. Brown, Leonard C. Chung, and David A. Patterson
Computer Science Division, University of California at Berkeley

1 Introduction

It has been widely acknowledged that dependability benchmarks will play a crucial role in driving progress toward highly reliable, easily maintained computer systems [4] [10]. Well-designed benchmarks provide a yardstick for assessing the current state of the art and provide the framework needed to evaluate and inspire progress in research and development. To achieve these goals, benchmarks must be accurate, realistic, and reproducible; in the case of dependability benchmarks, this means that the benchmarks must evaluate systems against the same set of dependability-influencing factors seen in real-life environments.

One of the most significant of these factors is human behavior. A system's human operators exert a substantial influence on that system's dependability: they can increase dependability via their monitoring, diagnosis, and problem-solving abilities, but they can also decrease dependability by making operational errors during system maintenance. The human error factor is particularly important to dependability: anecdotal data from many sources has suggested that human error on the part of system operators accounts for roughly half of all outages in production server environments [3]. Recent quantitative studies of Internet server sites and of the US telephone network infrastructure numerically confirm the significance of human error as a primary contributor to system failures [5] [13].

Existing work on dependability benchmarks has included little consideration of the effects of human behavior, positive or negative; this is unfortunate, but perhaps not surprising, given that human behavior has typically been under the purview of fields such as human-computer interaction or psychology, not systems benchmarking. In this paper, we present our first steps at bringing consideration of human behavior into the dependability benchmarking world, and describe our work-in-progress toward building a human-aware dependability benchmark. Although our methodology begins with a reasonably traditional dependability benchmarking framework, we deviate from existing work by directly including human operators in the benchmarking process as a source of system perturbation.

Of course, introducing humans complicates the benchmarking process significantly, and much of our

research focus is on how to include humans while keeping the benchmarks efficient and repeatable. A key insight is that we measure the human dependability impact *indirectly*, quantifying the end-to-end human impact on performance and availability metrics rather than trying to deduce the dependability impact of individual human actions. Other techniques that we will discuss for simplifying the benchmark process include approaches for choosing and preparing human operators for our tests, selecting human-dependent metrics that can be automatically collected, developing an appropriate workload for the human operator, and managing the inherent variability introduced by human operators.

Finally, while we have not yet had the opportunity to carry out a full-scale dependability benchmark that implements all of our techniques, preliminary experiments have helped us refine our approach while demonstrating its viability. We hope to have results from a full benchmark by the time of the workshop.

The remainder of this paper is organized as follows. Section 2 describes our dependability benchmarking methodology, including discussion of workload, metrics, and how we adapt existing dependability benchmarking techniques to incorporate humans. Section 3 considers some of the issues that arise in building a reproducible benchmark involving human operators. Section 4 presents a concrete example of how we are implementing our methodology as a dependability benchmark for e-mail server systems. We consider related work in Section 5, and conclude in Section 6.

2 Methodology

Traditional dependability benchmarks measure the impact of injected software and hardware faults on the performance and correctness of a test system being subjected to a realistic workload [4] [10]. For example, the system's performance might fall outside its window of normal behavior while it recovers from a hardware fault; the length of the recovery process and the magnitude of the performance drop are measures of the system's dependability in response to that fault. Typically, dependability benchmarks are run without human operator intervention in order to eliminate the possible variability that arises when human behavior is involved. But as dependability emerges from a synergy of system behavior and human response, ignoring either component or their interactions significantly limits the accuracy of the benchmark; both system and operator must be benchmarked together.

Thus we need to extend the traditional methodology to capture the human components of dependabil-

ity. Most basic is the need to measure the performance and correctness impact of hardware and software faults *when the human operator participates in the detection and recovery process*. But there is more—human operators perform maintenance tasks on systems (such as backups and restores, software upgrades, system reconfiguration, and data migration), and the dependability impact of these tasks must be measured as well. Moreover, humans invariably make mistakes and these errors can also impact dependability; we must therefore measure the performance and correctness impact of such errors.

We accomplish these goals by treating the operator as an additional source of perturbation to the system alongside traditional hardware and software fault injection. The “human perturbation” comes in two forms. *Reactive perturbations* arise when the operator reacts to the system’s behavior after a hardware or software failure occurs during the benchmark; the dependability impact of these perturbations can be either negative or positive depending on how well the operator diagnoses and repairs the failure. In contrast, *proactive perturbations* arise as the operator performs system maintenance tasks unrelated to failure occurrences. These too can have a negative or positive dependability impact depending on how well the operator performs the task, how many errors are made, and how the maintenance task itself affects the system.

We have two choices for how to incorporate human-induced perturbations into our dependability benchmarks. One option is to use a model of human operator behavior to perturb the system during the benchmark. While this approach provides reproducibility and has the advantage of not requiring human participation, it unfortunately reduces to an unsolved problem—if we were able to accurately simulate human operator behavior, we would not need human system operators in the first place! So we are left with the alternate approach: including human operators in the benchmarking process. Doing this raises several challenges, notably how to deal with human variability, how to perform valid cross-system comparisons with different operators, and how to structure benchmark trials so that the number of human operators is minimized. Despite these challenges, this approach is the only way to truly capture the full unpredictable complexities of the human operator’s behavior and the resulting impact on a system’s dependability. We will return to the challenges and discuss possible solutions in Section 3.

In the end our methodology for human-aware dependability benchmarking looks very similar to the traditional methodology with two exceptions: first, we allow the human operator to interact with the sys-

tem during the benchmark, and second, we task the operator with keeping the test system running and with carrying out a sequence of maintenance tasks on it. The traditional methodology required dependability metrics, a workload, and a “perturbation workload”; our extended methodology also requires a process for choosing human operators, a maintenance task workload for the chosen operators, and new human-aware dependability metrics. In this section we will focus on the latter two problems, maintenance task workloads and metrics; we will return to the problem of choosing operators in Section 3.1.1.

2.1 Human operator workload

Defining the human operator workload requires selecting a set of maintenance tasks for the operator to perform during the benchmark; these tasks must be representative of the types of maintenance performed in real-world production installations. We do not worry about the reactive perturbations as those will arise naturally through the operator’s response to injected-fault-induced system failures.

The ideal way to obtain a representative set of maintenance tasks is to carry out a “task analysis” study in which the experimenter shadows real system administrators/operators as they run a production system similar to that being benchmarked [9]; recording how these operators spend their time provides a list of tasks ranked by importance or frequency. Unfortunately, while such task analysis studies are the most accurate way to get a task workload for the human operator, they tend to be time-consuming or even impossible, especially when the type of system being benchmarked has never been deployed in production.

For those cases where task analysis is impractical, though, all is not lost. We can instead draw on anecdotal evidence and several published studies of what system administrators/operators do [1] [2] [6] [7] [8] to construct a set of general categories of maintenance tasks that should be included in the operator’s task workload. What we arrive at from such an analysis is the following set of task categories:

Initial configuration: setting up new systems, including hardware, operating system, and application installations. This category also includes deploying additional capacity into an existing system.

Reconfiguration: a broad category that includes everything from small configuration tweaks to significant reconfigurations like hardware, operating system, or application upgrades.

Monitoring: using monitoring tools or system probes to detect failures, security incidents, and per-

formance problems.

Diagnosis and repair: recovery from problems detected by monitoring tasks. This category covers diagnostic procedures, root-cause analysis, and recovery techniques like hardware repairs, software reinstallation/configuration, security incident response, and performance tuning. Note that the tasks in this category differ from those in the “System reconfiguration” category in that these tasks are unplanned and must be carried out reactively under time pressure, while the system reconfiguration tasks can be carefully planned and scheduled in advance to minimize their dependability impact.

Preventative maintenance: non-urgent tasks that maintain a system’s integrity, redundancy, and performance, or that adapt the system to changes in its workload. Examples include backup and restore, redundancy and replication management, data reorganization or repartitioning (*e.g.*, of database tables or e-mail mailboxes), rejuvenating reboots, and data purging.

Although these categories will of course have to be translated into specific tasks for each benchmarked system (and some may not apply), they provide a common framework for developing the operator’s task workload. For a specific example of how these task categories were specialized for a dependability benchmark for e-mail server systems, see Section 4.

2.2 Metrics

With the human operator workload established, the next challenge is to develop metrics that capture the human impact on dependability. Recall that traditional dependability benchmarks typically use performance and correctness measures to quantify dependability; dependability rankings are extracted from the behavior of these metrics over time as perturbations are injected and recovery takes place. We can use this same approach to indirectly capture the human impact on dependability: as the human operator repairs problems and performs maintenance tasks, any dependability impact (positive or negative) of those actions will be visible in the performance and correctness metrics already being tracked. For example, if the operator needs to shut down part or all of a service in order to carry out a repair or upgrade, that fact will be reflected in a concurrent drop in performance or correctness. Conversely, if the operator is able to expedite recovery from an injected perturbation, that will be reflected in a more rapid return of the system’s perfor-

mance and correctness metrics to their normal levels.

Thus we *indirectly* measure the human component of dependability, quantifying it by its impact on end-user dependability metrics rather than measuring the impact of each operator action. This approach simplifies the benchmark process, since the dependability metrics can still be collected in an automated way. Furthermore, it enables comparison of benchmark results across systems by eliminating the need to match operator actions on one system to equivalent actions on another (often an impossible task).

Note that, despite our indirect approach, it is still possible to perform some correlation between human operator actions and dependability side-effects within a single system, particularly when the dependability events result from human-initiated maintenance tasks. This insight allows us to use our dependability benchmarks to also evaluate a system's *maintainability*: we can record the number of mistakes made by the operator, the severity of those mistakes, and the time taken to recover from them as maintainability metrics.

3 Building Reproducible Benchmarks Involving Human Operators

Benchmarks should be reproducible and their results should be meaningful when compared across systems. The inherent variability and unpredictability of human actions make it a challenge to achieve these qualities when we start including humans in the benchmarking process. Thus a crucial part of our human-centric benchmarking methodology is to manage the variability in our human operators, both within a single benchmarking experiment and across benchmark runs on different systems or over time.

Variability in the behavior of a system operator comes from at least three sources. First, different potential operators will have different backgrounds and different skill levels coming in to the benchmark. In some cases, experienced sysadmins might be available for the benchmark while in others the benchmarker might have to make do with CS students or technical staff. Second, operators may have different levels of experience with the system and the benchmark tasks. This is a particularly acute problem when benchmarks are carried out more than once, for example to compare systems or to evaluate changes in one system: each iteration of the benchmark process increases the operator's experience with the system and can alter his or her behavior on subsequent iterations. Finally, there is a level of inherent variability in human behavior: two operators with identical experience and identical training given identical benchmark

tasks may still behave differently.

3.1 Managing variability for a single benchmark run

We first consider managing variability for a single benchmark run. The challenge here is to produce a benchmark result that represents the dependability of the *system*, without regard to the quirks of any particular human operator. We start by requiring that the final benchmark result be an average across multiple iterations of the benchmark with a different human operator participating in each iteration; this allows us to use statistical techniques to average out the third source of variability: inherent variability across the operators. Our pilot studies suggest that between 5 and 20 operators (iterations) will be needed to gain a statistically-sufficient averaging effect; work from the UI community confirms these estimates and suggests that 4 or 5 operators maximizes the benefit/cost ratio [12].

To address the variability arising from different operator backgrounds, the operators should be selected from a set of people with similar background and experience. The chosen operators should be given training on the target system to balance out any remaining variation in their background and skill set. Finally, they should be given access to resources to use to again fill in any gaps in their knowledge that are uncovered as the benchmark proceeds. We consider each of these steps in turn.

3.1.1 Choosing operators

The ideal set of operators for a dependability benchmarking run has a level of skill and experience that is both consistent within the group and similar to what would be seen in real-life operators. This is a challenging problem: real operators vary greatly in their skills and experience, which often depend on the size of the real-life installation and its dependability needs. Our best hope is to define several levels of selection criteria for operators and allow the benchmarker to choose the level that best matches the target environment of the tested system. With this approach, results from one benchmark run should be comparable to results from other benchmarks using the same level of operators; benchmarks using different levels of operators might also be comparable if the operator level is used as a “handicap” on the benchmark results.

We observe at least three levels of qualification for benchmark operators (from highest to lowest):

Expert: The operators have intimate knowledge of the target system, unsurpassed skills, and long-term experience with the system. These are operators who run large production installations of the tar-

get system for their day jobs, or are supplied by the system's vendor. Benchmarks involving these operators will report the best-case dependability for the target system, but may be realistic only for a very small fraction of the system's potential installed base.

Certified: The operators have passed a test that verifies a certain minimum familiarity and experience with the target system; ideally the certification is issued by the system vendor or an independent external agency such as SAGE [14]. Benchmarks involving these operators should report dependability similar to what would be seen in an average corporate installation of the tested system.

Technical: The operators have technical training and a general level of technical skill involving computer systems and the application area of the target system, but do not have significant experience with the target system itself. These operators could be a company's general systems administration or IT staff, or computer science students in an academic setting. Benchmarks involving these operators will report dependability that is on average similar to that measured with certified operators, but there may be more variance amongst operators (hence requiring more operators and benchmark iterations) and more of a learning curve factor (requiring greater training or discounting of tasks performed early in the benchmark iteration).

Should human-aware dependability benchmarks reach widespread commercial use (like the TPC database benchmarks [18]), they will probably use expert operators. Expert operators offer the lowest possible variance, are unlikely to make naive mistakes that could make the system look undeservedly bad, yet still provides a useful indication of the system's dependability and maintainability. Published results from benchmarks like TPC often already involve a major commitment of money and personnel on the part of the vendor, so supplying expert operators should not be a significant barrier.

For non-commercial use of dependability benchmarking where experts are unavailable (as in academic or internal development work), using certified operators is ideal since certification best controls the variance between non-expert operators. As it may be difficult to recruit certified operators, it is likely that technical operators will be often be used in practice; we believe that accurate dependability measurements can still be obtained using these operators by providing suitable resources and training as described below.

3.1.2 Training operators

Picking operators from a given qualification level removes a large amount of human variability, but differences will still remain, especially amongst the lower levels of operator qualification. We can mitigate some of this remaining variance by providing standardized training for the operators before they participate in the benchmark. The goal of the training should be to help the operator build a conceptual model of the system and to provide familiarity with the system's interfaces, rather than teaching the operator how to perform the specific tasks that will appear in the benchmark. Conceptual training allows the operator to apply ingenuity and problem-solving techniques much as would be done in real life, whereas task-specific training simply reduces the human operator's involvement to rote execution of a checklist of operations.

Our initial experiments have suggested that an effective method for conceptual training combines basic instruction on the system's high-level purpose and design with a simple maintenance task that requires exploration of the system's interfaces (for example, changing a configuration parameter that is buried deep in an unspecified configuration file or dialog box). After reading or listening to the basic instruction, the operator performs the introductory maintenance task, gaining familiarity with the system's interfaces and operation while carrying out the task. If the initial task is well-designed, the operator will have built up enough of a mental model of the system upon completion to proceed with the benchmark. With this approach, very little formal training need be given, simplifying the deployment of the benchmark; furthermore, we have found that this approach also helps technical-class operators quickly overcome the learning curve, further reducing variance.

3.1.3 Resources for operators

Even with training, operators may still have gaps in their knowledge that show up during the benchmark; this is again a source of variance because different operators will have different knowledge gaps. To mitigate this variance, we can provide operators with resources that they can use during the benchmark to fill in any knowledge gaps that arise. These resources take two forms: documentation and expert help.

Documentation provides a knowledge base upon which the operator can draw while performing the benchmark tasks. For maximum realism, we believe the operator should be provided with the unedited documentation shipped with the testbed system and be given access to the Internet and its various search

engines. If at all possible, the documentation should be provided electronically so that its usage can be monitored automatically for use in calculating maintainability metrics.

While full documentation may provide an overwhelming amount of information, it is reflective of the information overload that real-life system administrators and operators often encounter when troubleshooting problems or performing maintenance. Preliminary experiments we have carried out in which we provided edited or simplified documentation proved ineffective; the operators involved in those experiments felt that the simplified documentation was in fact too good—it was unrealistic in that it did not require the usual searching, cross-referencing, and interpolation that real documentation required. We further noted that the operators' approaches and error behavior were different using the simplified, task-oriented documentation as opposed to the system's real documentation.

Along with documentation, benchmark operators should also be given a resource of last resort: appealing to an expert for help. In most real-life administration and operation scenarios, the sysadmin/operator has a supervisor or peer network to whom he can appeal when he reaches a dead end. This role can be filled in the benchmark process by providing an "oracle" or expert (typically the benchmarker) who knows the system intimately. The challenge in providing this service to the operator is ensuring that it remains an appeal of last resort, since if it is overused the benchmark measures the oracle rather than the operator. Possibilities include providing only a limited number of "lifelines" (calls to the oracle), imposing an artificial time penalty for using the oracle, or implementing the oracle as an automated "I give up" button that simply completes the current task automatically or restores the system to a known state. While we are still investigating the relative merits of these options, our early experiments have shown that an oracle/expert facility is essential to avoid excessive operator frustration.

3.2 Managing the learning curve effect

Up to now we have neglected an important component of variability: the impact of the *learning curve* on human behavior. The learning curve effect refers to the fact that operators learn something about the target system as they perform the benchmark, so that by the end of the benchmark they have more experience and better skills than when they started. Subsequent benchmark runs will show the effect of this learning, making it difficult to compare different runs directly. The learning curve effect would be irrelevant if we could

assume that a fresh set of operators was available for each benchmark run, but in practice it is almost always necessary to reuse operators. In particular, when comparing several different systems, it is a good idea to use the same set of operators on all of the systems to minimize any remnants of operator variability.

Compensating for the learning curve effect is a challenging problem that we have only begun to address. For the case where the same operators are being used to compare multiple systems, a simple solution is to randomize the order in which each operator uses the different systems. The random ordering should average out variability introduced by the learning curve effect, although it may require a larger pool of operators to achieve this effect. Another complimentary approach is to benchmark each system more than once, using the differences between the benchmark results to estimate and factor out the learning curve effect. Our experiments suggest that, for simple tasks, the learning curve can be conquered within one to two benchmark iterations.

Addressing the learning curve is more difficult if a single system is benchmarked repeatedly over time using the same operators; this scenario might occur in a development environment where dependability benchmarks are being used to guide design. Randomization is not an option here. One approach in this case is to try to force the operator down the learning curve before taking a dependability measurement; this could be done by repeating the benchmark two or more times and only using the result from the last iteration. This approach may not give a true measure of the system's real-life dependability (as it essentially turns the operator into an expert and removes many potential mistakes), but it may be sufficient for certain scenarios.

4 An Example: Benchmarking E-Mail Dependability

E-mail is a service that was originally designed to operate in a best-effort environment but is now expected to operate as a mission critical service. We have chosen e-mail as the first application of our benchmarking methodology because it is a familiar, widely used service with enough complexity and state to make it a worthwhile target.

The approach of our benchmark follows the general methodology of previous dependability benchmarks described in Section 2. The benchmark applies a workload, injects perturbations, and collects metrics while the system is under the supervision of a human operator. The benchmark treats the e-mail service

as a black-box for generality. In addition, as a simplifying assumption, we focus on the “store” component of “store and forward” and treat the e-mail service as a sink.

The workload of the benchmark can be broken down into three distinct types: Performance, Perturbation, and Human workloads. The performance workload is injected into an e-mail service using standard protocols. SMTP is used to insert e-mails into the service and POP3 is used to retrieve e-mails from the service. The workload generator uses the user workload parameters in the SPECmail2001 load generator [16] and is fully parameterizable to allow the user to simulate load spikes and explore system behavior under different types of load scenarios. The perturbation workload has not yet been finalized, but we will likely start with two main types of fault injection: coarse-grained hardware and software faults. These two categories include a variety of faults, from failures in both storage and network hardware, to terminated processes and file corruption.

Finally, the human workload is based around system operator maintenance tasks chosen to match the categories defined in Section 2.1. The workload is comprised of three separate steps. Each of these steps entails tasks of varying difficulty to measure the varying levels of operator error and system forgiveness for tasks of different complexity. The first step is a warm-up task consisting of a simple software reconfiguration such as changing the default domain of unqualified e-mail addresses; this step also serves as a “training” step, allowing the operator to become familiar with the system. The second step is a moderately difficult task such as installing and configuring a server-side e-mail virus filter. The third step is a challenging task such as moving a group of users from one server to another.

During each of these tasks, the benchmark measures the overall service dependability to measure the effect such tasks have on the end user experience. The measured metrics include e-mail delivery delays and errors, the number of dropped and corrupted e-mails, and service performance in fault-free, induced-fault, recovery, and service overload scenarios. The overload metric is designed to simulate load spikes that Internet services are commonly subjected to today. Although the long term solution to service dependability in the face of overload scenarios is to avoid such scenarios by scaling the service with more resources, this is not possible when the service is experiencing a sudden increase in load in a short period of time. Thus, we also seek to measure how well the service degrades under load.

Turning to benchmark logistics, we intend to use technical-level operators in our benchmark experiments as described in Section 3.1.1; while we would prefer to use certified operators, these are difficult to find in an academic institution. We plan to address the problem of increased demand on researcher time by fully automating the benchmark, including the workload generator and instrumentation. We will use VMWare [19] to create virtual machines that can easily be re-initialized after each experiment, and Camtasia [17] to capture an audio/video trace of the computer screen and operator for later review by the benchmarker. Through these techniques, we hope to be able run operators through the benchmark without a benchmarker present, except perhaps to serve as the on-call benchmark oracle.

5 Related Work

Our perturbation-based benchmark methodology follows in the footsteps of most of the existing work on dependability benchmarking and extends our earlier work on availability benchmarking, which measured the availability of RAID systems by perturbing them with simulated disk failures [4]. Our methodology also fits into the dependability benchmarking framework defined by Madeira and Koopman [10], with our human-operator-induced perturbations making up the “upsetload” in their terminology. Where our methodology is unique is in its inclusion of the human operator as a perturbation source: we are not aware of any dependability benchmarks to date that include the human component in their dependability measurements.

Many of the techniques, issues, and proposed solutions in this paper are adaptations of traditional behavioral research techniques for human-computer interaction, such as those described in Landauer’s excellent survey [9]. However, unlike the HCI approaches, it is our goal to measure the *system*’s behavior rather than the human’s—in our benchmarks, the human operator is not there to be directly observed or measured, but to provide realistic perturbation and stimulus to the system. In that sense our work is most similar to work in the security community on the effectiveness of security-related UIs, such as Whitten and Tygar’s study of PGP [20]. While we can (and do) borrow advice on topics like selection of operators, task analysis, and experiment logistics from the HCI community, their standard experimental designs and metrics do not directly apply to our dependability benchmarking task.

Finally, our proposed specific implementation of our methodology in an email dependability benchmark is also a novelty. Neither of the two major email benchmarks in production use today (SPEC’s

SPECmail2001 [16] and Netscape's Mailstone [11]) make any attempt at measuring dependability beyond a simple count of dropped client connections; they focus primarily on performance measurement and do not include any facility for injecting perturbations into the system. In the research domain there has been more attention on email dependability, for example in Saito's measurements of the Porcupine mail system's performance under simple perturbations [15], but while this work uses a similar methodology to ours, it again does not include consideration of the human operator.

6 Conclusions and Future Directions

As dependability increasingly supplants performance as the essential metric for computer systems, dependability benchmarks are becoming essential tools for system designers and evaluators. Yet to date, dependability benchmarks have ignored the behavior of a computer system's human operators and administrators, a key piece of the dependability puzzle. In this paper we have presented a first attempt at addressing this deficiency: our human-centric benchmarking methodology should provide an effective means of incorporating the effects of human operator behavior into dependability measurements.

What we have presented here is just a first step, however. Our methodology will need to be proven and refined through extensive experimental verification; experimentation will also help explore the extent of cross-operator variability and the tradeoffs involved in issues such as selecting and training operators. We are pursuing this follow-on work in the context of our dependability benchmark for e-mail. Other issues that remain to be explored include the development of techniques for pre-evaluating the skill level of participating operators, more advanced dependability metrics that are parameterized by the operator's skill level, and extensions that allow for a direct measure of a system's maintainability and scalability along with the indirect measurements extracted through the dependability metrics. These are all fruitful and important directions for future research, and we look forward to seeing them addressed by the community.

References

- [1] Anderson, E. "Results of the 1995 SANS Survey." ;*login*., *the USENIX Association Newsletter*, 20(5), 1995.
- [2] Anderson, E. and D. A. Patterson. "A Retrospective on Twelve Years of LISA Proceedings." *Proceedings of the Thirteenth Systems Administration Conference (LISA XIII)*, Seattle, WA, 1999.
- [3] Brown, A. and D. A. Patterson. "To Err is Human." *Proceedings of the First Workshop on Evaluating and Architecting System dependability (EASY '01)*, Göteborg, Sweden, July 2001.

- [4] Brown, A. and D.A. Patterson. "Towards Availability Benchmarks: A Case Study of Software RAID Systems." *Proceedings of the 2000 USENIX Annual Technical Conference*, San Diego, CA, June 2000.
- [5] Enriquez, P. "Failure Analysis of the PSTN." Unpublished talk available at http://roc.cs.berkeley.edu/retreats/spring_02/d1_slides/RocTalk.ppt, January 2002.
- [6] Evard, R. "An Analysis of UNIX System Configuration." *Proceedings of the 11th Systems Administration Conference (LISA '97)*, San Diego, CA, October 1997.
- [7] Kolstad, R. "1992 LISA Time Expenditure Survey." ;login.; *the USENIX Association Newsletter*, 1992.
- [8] Kolstad, R. "Sysadmin Book of Knowledge." <http://ace.delos.com/taxongate>.
- [9] Landauer, T. K. "Research Methods in Human-Computer Interaction." In *Handbook of Human-Computer Interaction*, 2e, M Helander et al. (ed), Elsevier, 1997, 203–227.
- [10] Madeira, H. and P. Koopman. "Dependability Benchmarking: making choices in an n-dimensional problem space." *Proceedings of the First Workshop on Evaluating and Architecting System dependability (EASY '01)*, Göteborg, Sweden, July 2001.
- [11] Netscape, Inc. *Mailstone Utility*. <http://docs.iplanet.com/docs/manuals/messaging/nms41/mailston/stone.htm>.
- [12] Nielsen, J., and Landauer, T. K. "A mathematical model of the finding of usability problems." *Proceedings of the ACM INTERCHI '93 Conference*, Amsterdam, The Netherlands, April 1993, 206–213.
- [13] Oppenheimer, D. and D. A. Patterson. "Architecture, operation, and dependability of large-scale Internet services." Submission to *IEEE Internet Computing*, February, 2002.
- [14] SAGE Certification Program, <http://www.usenix.org/sage/cert/>, 2002.
- [15] Saito, Y., B. Bershad, and H. Levy. "Manageability, Availability, and Performance in Porcupine: A Highly Scalable Internet Mail Service." *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, Kiawah Island, SC, 1999.
- [16] Standard Performance Evaluation Corporation. *SPECmail2001*, <http://www.spec.org/osg/mail2001/>.
- [17] TechSmith, Inc. *Camtasia Recorder*, <http://www.techsmith.com/products/camtasia/camtasia.asp>.
- [18] Transaction Processing Performance Council Benchmarks. <http://www.tpc.org>.
- [19] VMWare, Inc. <http://www.vmware.com>.
- [20] Whitten, A. and J. D. Tygar. "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0." *Proceedings of the 9th USENIX Security Symposium*, August 1999.