Toward Recovery-Oriented Computing

Armando Fox

Stanford University
Stanford, California, USA
fox@cs.stanford.edu

Abstract

Recovery Oriented Computing (*ROC*) is a joint research effort between Stanford University and the University of California, Berkeley. ROC takes the perspective that hardware faults, software bugs, and operator errors are facts to be coped with, not problems to be solved. This perspective is supported both by historical evidence and by recent studies on the main sources of outages in production systems. By concentrating on reducing Mean Time to Repair (MTTR) rather than increasing Mean Time to Failure (MTTF), ROC reduces recovery time and thus offers higher availability. We describe the principles and philosophy behind the joint Stanford/Berkeley ROC effort and outline some of its research areas and current projects.

1 The Case for ROC and "Peres's Law"

If a problem has no solution, it may not be a problem but a fact, not to be solved but to be coped with over time.
—Shimon Peres

Despite marketing campaigns promising 99.999% availability, well-managed servers today achieve 99.9% to 99%, or 8 to 80 hours of downtime per year . Each hour can be costly, from $200,000 per hour for an Internet service like Amazon to $6,000,000 per hour for a stock brokerage firm [Kembel00]. Total cost of ownership ranges from 3 to 18 times the purchase cost of many cluster-based systems, and a third to a half of that money is spent recovering from or preparing for failures [Gillen02]. Despite decades of research that have achieved four orders of magnitude in performance, large cluster-based systems and end-user terminals alike still fail, and we have not made sufficient headway in curbing failures to keep up with the increasing complexity of our systems and our dependence on them. We conclude that such failures are a fact of life: not a problem that will someday be solved once and for all, but a reality that we must live with.

We propose to cope with this reality through fast and graceful recovery. The quantitative rationale for the ROC approach may be summarized as follows. A widely accepted equation for system availability is A=MTTF/(MTTF+MTTR), where MTTF is the mean time to system failure and MTTR the mean time to recovery after a failure. The target is to approach A=1.0, and much historical effort has focused on achieving this by pushing MTTF towards infinity—making hardware ever more reliable, investing more resources in software design and testing, employing redundancy to allow continuous operation in the presence of partial failures, and so on. We argue that an alternate way to approach A=1.0 is to focus on making MTTR<<MTTF. Of course, to some extent this has been embraced in communities such as hardware design and database design; in fact, it is often the case that in those systems, rapid recovery is used within a particular layer of functionality to *prevent* a visible failure in a higher layer, perhaps by making it visible as only a performance blip, so that effectively a sufficiently reduced MTTR in one layer is manifest is increased MTTF in higher layers[1]. What is new in ROC is that we believe it is time to apply emphasis on reducing MTTR at the highest layers—the application and its end users—as a way of improving availability, and that numerous if currently anecdotal successes from the Internet systems community can help illuminate the way to do this.

1.1 Why Focus On Recovery

There are several reasons we have chosen to focus squarely on recovery:

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment

Proceedings of the 28th VLDB Conference,
Hong Kong, China, 2002

[1] Thanks to Lisa Spainhower for this elegant generalization of the observation.

Human error is inevitable. Over 50% of outage incidents, and a comparable fraction of outage minutes, are due to operator error. We conducted two surveys that confirmed this data in both the public switched telephone network [Enriquez02] and for a selection of representative large-scale Internet cluster services [Oppenheimer02].

MTTR can be directly measured. Today's disks have quoted MTTF's of 120 years. Verifying such claims requires many system-years of operation, which is beyond the reach of all but the largest customers. In contrast, the longest MTTR's for commercial database products are on the order of days, and many are on the order of hours, making MTTR claims verifiable.

Lowering application-level MTTR can directly improve the user experience. In April 2002, Ebay had a 280-minute sustained outage affecting most of its services. This and similar previous outages are highly visible, newsworthy, and affect customer loyalty and investor confidence [Dembeck99]. In contrast, had Ebay suffered one 6-minute outage per week, they would have achieved the same availability (according to the formula) but the individual outages are probably not newsworthy because they affect far fewer users. The difference, of course, is that in the latter case the same availability is achieved by having a much shorter MTTR.

Frequent "recovery" may lengthen effective MTTF. Software rejuvenation [Garg97] and recursive restartability [Candea01] both exploit the observation that by returning a system periodically to its start state (typically a well understood and heavily tested state), we can reclaim stale resources, clean up corrupted state and other side effects of software aging, and eliminate the corresponding side effects (e.g. performance degradation due to memory leaks), and that we can do these things by relying on well-tested but limited-functionality hardware support such as the virtual memory system.

1.2 The ROC Research Agenda

Anecdotally, we know that some systems support (or at least tolerate) some of the above scenarios well; for example, "rolling reboots" are standard procedure for most cluster-based services [Brewer01]. The ROC research questions may therefore be stated as follows:

?? For the recovery scenarios above, what does it mean for the corresponding system or subsystem to be *designed for recovery*?

?? How can we identify and classify faults so that we can select the most effective strategy when recovery is needed?

?? How will we measure our success?

We now describe a sampling of ROC work in progress that addresses each of these questions.

2 Some Research Areas and Projects

Our initial targets are large Internet-scale and corporate-scale applications: email, portals, wide area storage, and so on. We chose these in part because they have interesting state management requirements, but often do not need transactional guarantees, allowing us to ask whether recovery can be improved by trading some of those guarantees away. Also, the very large scale of these systems brings some tradeoffs into sharp relief: search engines feature a single specialized application running on thousands of nodes, and some guarantees such as consistency must be relaxed in order to meet throughput and latency requirements and provide for incremental scaling [Brewer01]. Finally, these services are trying to build mission-critical functionality from semi-reliable COTS parts in the face of high feature churn; we believe these constraints and the attendant market pressures are indicative of many future mission-critical systems, so our recovery strategies should address these cases.

Measurement and Benchmarking. We are building on our and others' earlier work on availability benchmarking [Brown00,Lambright00] to come up with metrics that capture more than just "up or down" availability, such as graceful performance degradation during recovery. Similarly, we are considering the end-user-visible effects of different forms of unavailability [Merzbacher02] and ways to incorporate human operator behavior in dependability benchmarks [Brown02b]. Given our stated focus on recovery, we are gratified to see initial industrial support for benchmarking recovery from various kinds of failures as well [Zhu02].

Recursive Restartability. A *recursively restartable* system [Cand01] gracefully tolerates successive *partial* restarts at multiple levels, which can be used to recover from transient failures more quickly than a full reboot woudl require. To apply RR to a system, we construct a *restart tree* that captures restart dependencies among components: restart tree nodes are highly fault-isolated and a restart at any node will restart the entire corresponding subtree rooted at that node. To enforce the containment boundaries and the subtree restart behavior, we rely on hardware-level support such as virtual memory, process groups, and physical node boundaries. A policy oracle decides which subtree to restart in response to a particular detected failure; a simple arithmetic model quantifies the cost of the oracle making a mistake. We have successfully applied RR to an amateur satellite ground station controller [CCF+02] to reduce its time to recovery by a factor of 3-4x, and are currently investigating "design for restartability" for stateful components that are required to provide bounded or probabilistic data durability and integrity.

System-level Undo for Operators. We are "wrapping" an off-the-shelf IMAP mail server with system-level Undo functionality, to recover from (e.g.) administrative errors that would otherwise cause data loss

or an unacceptable user-perceived data inconsistency [Brown02]. A system with undo also provides a forgiving environment that promotes ingenuity and exploration: the system operator can try innovative solutions to problems without the fear of permanent disasterous consequences, and an operator-in-training can safely learn by making mistakes and recovering from them. Psychology has shown that this approach of learning by trial-and-error is one of the most effective method of human learning [Reason90], yet only with an undoable system is the cost of mistakes low enough to make it feasible.

Fault injection. FIG (Fault Injection in *glibc*) is a lightweight, low-overhead, extensible tool for triggering and logging errors at the application/system boundary. FIG uses the LD_PRELOAD environment variable to interpose itself between the application and *glibc*, the GNU C library, causes some *libc* calls to fail to simluate a failure in the operating environment. Using FIG to trigger such faults in a variety of applications from desktop applications to transaction servers, we have been able to start classifying the successful recovery techniques that appear in the applications that fare best under fault injection [BST02].

Failure detection and diagnosis. Pinpoint [CKF+02] is a framework for root-cause analysis in large distributed component applications such as e-commerce systems. Pinpoint tags a subset of client requests as they travel through the system, uses traffic sniffing and middleware instrumentation to detect failed requests, and then applies data mining techniques offline to correlate the failed and successful requests to determine which component(s) were likely to be at fault for the failures. Because it is implemented on the Java 2 Enterprise Edition application server itself, existing J2EE applications can use Pinpoint unmodified; experiments show that it identifies faulty components with high accuracy and a low false-positive rate.

3 Inspiration From Prior Work

Outside of computer science and engineering, we are scanning the literature in disaster handling in emergency systems such as nuclear reactors [Perrow90], human error and "automation irony" [Reason90], and civil engineering failures [Petroski92]. Within computer science and engineering, we look to three large research communities for inspiration and ideas: hardware-level and mission-critical-system fault tolerance, commercial transaction systems, and the Internet systems community.

Hardware and system fault tolerance, whether at the component level or instruction-set architecture level, serves the important function of assuring that the underlying system behaves according to a particular well-defined specification; for example, instruction-level retry in the IBM G5 and other mainframes assures that the hardware behaves according to the ISA specification. Purpose-designed safety-critical systems such as the Space Shuttle software have an excellent reliability record as a full system, but at great cost in both maintenance effort and difficulty of making changes. Although we are most interested in exploring recovery at higher layers and with higher-churn components, we expect to be able to apply some of the ideas from this community in analogous ways.

The Internet and systems community have already begun investigating ways to systematize tradeoffs such as availability vs. consistency [Yu00] and how to substitute soft state for hard state in many kinds of applications [Raman99]. Both approaches have the potential to simplify recovery, and although both require explicit support at application design time, application-transparent recovery has been shown to be inapplicable in a broad range of common failure cases [Lowell00].

The database community has deeply explored failure recovery techniques that provide the strongest guarantees for data integrity; one may say without exaggeration that the sophisticated engineering in such systems today sets the standard for data integrity guarantees after recovery. It has been observed that not all applications require the guarantees such systems provide, and especially at extreme scale, it may be beneficial or necessary to trade some of those guarantees away for improved availability [Fox99]; we ask whether they might instead be traded for faster recovery.

In summary, we expect our work to be complementary to the large body of existing work in fault tolerance, and we hope to gain insights and ideas from real collaborations with those communities.

4 ROC People

ROC includes faculty at Stanford and Berkeley, graduate and undergraduate students at both institutions, and an industrial "advisory panel" that includes representation from Hewlett-Packard Laboratories, IBM Research, Microsoft, Microsoft Research, Intel, VMware, and Yahoo! Inc. Financial support is provided by an NSF ITR grant, an NSF CAREER award, various student fellowships and scholarships, and research grants from Allocity, Hewlett-Packard, IBM, Microsoft, Usenix, and NASA.

The success of ROC is critically dependent on participation from industry: we must develop benchmarks that are realistic and that we can all agree on, base our work on real failure data in real-world environments and at realistic scales, and work on problems that have not already been "solved" by industrial R&D.

Similarly, we expect that great ideas will come from cross-pollination. We hope to add to the excellent work done in fault tolerance to date, and we ask the cooperation of researchers from those areas to help us better inform and guide our own contributions.

References

[Brewer01] Eric A. Brewer. Lessons from giant-scale services. IEEE Internet Computing, vol.5, (no.4), IEEE, July-Aug. 2001. p.46-55.

[Brown00] Aaron Brown and David A. Patterson. Towards availability benchmarking: a case study of software RAID systems. In Proc. USENIX Annual Technical Conference, San Diego, CA, June 2000.

[Brown02] Aaron Brown and David A. Patterson. Rewind, repair, replay: Three R's to Dependability. In proc. SIGOPS European Workshop, Sept. 2002.

[Brown02b] Aaron Brown and David A. Patterson. Including the human factor in dependability benchmarks. Proc. 2002 DSN Workshop on Dependability Benchmarking, Bethesda, MD, June 2002.

[BST02] Pete Broadwell, Naveen Sastry Jonathan Traupman. FIG: Fault Injection in glibc. In Workshop on Self-Healing, Adaptive, and Self-Managed Systems (SHAMAN), New York, June 2002.

[Candea01] George Candea and Armando Fox. Recursive restartability: Turning the reboot sledgehammer into a scalpel. Proc. Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII), Elmau, Germany, May 2001.

[CCF+02] George Candea, James Cutler, Armando Fox, et al. Minimizing time to recover in a small recursively restartable system. Proc. Intl. Symp. on Dependable Sys. and Networks (DSN) 2002, Bethesda, MD, June 2002.

[CKF+02] M. Chen, E. Kiciman, E. Fratkin, A. Fox, E. Brewer. Pinpoint: Problem determination in large, dynamic Internet services. . Proc. Intl. Symp. on Dependable Sys. and Networks (DSN) 2002, Bethesda, MD, June 2002.

[Dembeck99] Chet Dembeck. Yahoo cashes in on Ebay's outage. E-commerce Times, June 18, 1999. http://www.ecommercetimes.com/perl/story/545.html

[Enriquez02] P. Enriquez, A. Brown, and D.A. Patterson. Lessons from the PSTN for dependable computing. In Workshop on Self-Healing, Adaptive, and Self-Managed Systems (SHAMAN), New York, June 2002.

[Fox99] A. Fox and E.A. Brewer. Harvest, yield, and scalable tolerant systems. In Proc. Seventh Workshop on Hot Topics in Operating Systems (HotOS-VII), Tucson, AZ, 1999.

[Garg97] S. Garg, A. Puliafito, M. Telek, K.S. Trivedi. On the analysis of software rejuvenation policies. Proc. of the 12th Annual Conf. on Computer Assurance, June 1997, pp. 88–96.

[Gillen02] Gillen, Al, Dan Kusnetzky, and Scott McLaron "The Role of Linux in Reducing the Cost of Enterprise Computing", IDC white paper, Jan. 2002, available at.

[Kembel00] R. Kembel. Fibre Channel: A Comprehensive Introduction, p.8, 2000.

[Lambright00] D. Lambright. Experiences in measuring the reliability of a cache-based storage system. Proc. First Workshop on Industrial Experiences with System Software (WIESS 2000), San Diego, CA, Oct. 2000.

[Lowell00] D. E. Lowell, S. Chandra, and P. Chen. Exploring Failure Transparency and the Limits of Generic Recovery. Proc. 4th Symp.on Operating System Design and Implementation (OSDI 2000), San Diego, CA, October 2000.

[Merzbacher02] Matthew Merzbacher and Dan Patterson. Measuring end-user availability on the Web: Practical experience. In Proc. Intl. Conf. on Dependable Sys. and Networks (DSN) 2002, Bethesda, MD, June 2002.

[Oppenheimer02] David Oppenheimer and David A. Patterson. Studying and using failure data from large-scale Internet services. In Proc. SIGOPS European Workshop, Sept. 2002

[Perrow90] Perrow, Charles. Normal Accidents: Living with High Risk Technologies, Perseus Books.

[Petroski92]. Petroski , H. To engineer is human : the role of failure in successful design, Vintage Books,. NewYork, 1992

[Raman99] Suchitra Raman and Steven McCanne. A model, analysis, and protocol framework for soft state based communication. Proc. ACM SIGCOMM Conference, Cambridge, MA, Sep 1999.

[Reason90], Reason J. T. Human error. New York : Cambridge University Press, 1990.

[ROC02] David Patterson et al. Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies. UC Berkeley Technical Report CSD-02-1175, March 2002

[Yu00] Haifeng Yu and Amin Vahdat. Design and evaluation of a continuous consistency model for replicated services. In Proc. Fourth Intl. Symp. on Oper. Sys. Design and Implementation (OSDI 2000), San Diego, CA, Oct. 2000.

[Zhu02] Ji Zhu, James Mauro, Ira Pramanick. System recovery benchmarking. Proc. DSN Workshop on Dependability Benchmarking, Bethesda, MD, June 2002.