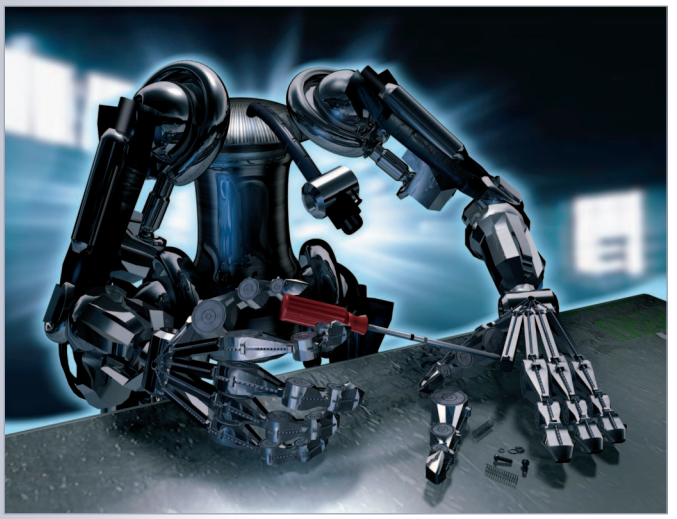

Approaches to Recovery-Oriented Computing

Armando Fox
Stanford University

David Patterson
University of California, Berkeley

Developers once put most of their efforts in improving dependability into air-traffic control, space flight, and other mission-critical systems. With worldwide IT spending now in the trillions of dollars, however, dependability is a problem with much broader implications.

The pace of innovation that fueled the online services revolution depends on the ability to rapidly prototype systems from reusable hardware and software components. This has led to dependence on integrating "commodity" subsystems — hardware and software taken "off the shelf" and used as-is, as opposed to purpose-built subsystems designed to work together. Equally important is our reliance on "commodity programmers" who don't necessarily have formal training in fault-tolerance techniques. Finally, the rapid pace of innovation results in accelerated system evolution, making it difficult to apply some of those formal techniques.

These factors increasingly lead researchers to consider recovery-oriented approaches to dependability: they acknowledge that hardware fails, software has bugs, and human operators make mistakes, despite our efforts to address all three issues. The articles in this issue's theme section describe how a range of techniques based on these perspectives can augment and complement other efforts to improve dependability.

Challenges

To handle the dependability challenges inherent to modern software systems, our industry needs to focus more on minimizing recovery time from inevitable failures. We can even justify this argument quantitatively. Given that we measure availability as

$$\text{MTTF} / (\text{MTTF} + \text{MTTR}), \quad (1)$$

where MTTF is mean time to failure and MTTR is mean time to recovery, a tenfold decrease in MTTR is as valuable as a tenfold increase in MTTF because

$$\text{MTTF} / (\text{MTTF} + \text{MTTR} / 10) = 10 \times \text{MTTF} / (10 \times \text{MTTF} + \text{MTTR}). \quad (2)$$

Although MTTF and MTTR improvements are synergistic (Equation 2), focusing on

 Published by the IEEE Computer Society 1089-7801/05/$20.00

shorter recovery has received less attention than lengthening MTTF, even though it has the potential for greater improvements. For example, a sufficiently fast recovery followed by retrying a failed operation might serve to completely mask the failure, giving users the impression of very high MTTF. Furthermore, even a very long MTTF is a statistical characterization of a system that can't guarantee any particular operational interval will be failure-free; when a failure does occur, its impact on the system's operator can be related directly to MTTR: a one-minute outage might inconvenience a subset of users, but a four-hour outage of a major Internet site can be front-page news.

Recovery–oriented approaches are challenging, in part, because they require us to deal with "cleaning up" after something has already gone wrong, but it's often not obvious what to clean up. If a crash were caused by corrupted system data structures, for example, these should be discarded and rebuilt before continuing, but the application data itself must remain intact (or be reconstructed if it has been damaged). Moreover, we have to be very quick in identifying what to keep, throw away, or rebuild. If a decision is too difficult to automate, we must also determine what kind of visibility into the system and what manual-repair facilities the human operators charged with recovery would find most useful.

The Articles

Researchers in both academia and industry are increasing their efforts toward recovery-oriented dependability. A cross-section of ongoing work reveals two clear lessons. First, system availability — from the user's perspective — is the most important factor. As we mentioned, fast recovery can disguise transient failures, such that users either don't notice them or experience only a small performance reduction.

Florin Sultan and colleagues explore this topic in "Recovering Internet Service Sessions from Operating System Failures." They address fast recovery, even when a machine's OS crashes or hangs, through a novel exploitation of backdoor hardware mechanisms intended for system management operations. Their approach is designed to recover session state from a crashed machine and pass it to another node to continue service sessions seamlessly from the user's perspective.

In a similar vein, Zbigniew Kalbarczyk, Ravishankar Iyer, and Long Wang describe their Adaptive Reconfigurable Mobile Objects of Reliability framework in "Application Fault Tolerance with Armor Middleware." Armor lets applications exploit low-cost fine-grained checkpoints run by coordinated multithreaded processes that manage redundant resources across interconnected nodes, detect errors in user applications and infrastructural components, and provide failure recovery. The article discusses several successful case studies using middleware that can take microcheckpoints — encapsulations of just enough application state to back up and recover after a failure, allowing extremely fast recovery while preserving essential system state.

The second lesson is that, although human operators are responsible for more than 50 percent of Internet service failures, they're also the first line of defense when automatic recovery fails — and often know the most about the system. As humans, we quickly recognize our mistakes, but we need tools to help recover from them. In "A New Undo Function for Web-Based Management Information Systems," Nicolás Serrano and colleagues discuss a tool to let multitier Web application operators roll an application's persistent state back to an earlier snapshot. Researchers have given "operator undo" little attention despite the old "to err is human" proverb, but future system designers would do well to include such functionality.

As humans, we quickly recognize our mistakes, but we need tools to help recover from them.

This issue's theme articles don't specifically address autonomic computing or self-healing systems, which are currently hot topics in the press. Although autonomic computing sets forth worthy goals, substantial work remains on more fundamental issues before it can truly succeed. We must fully understand problems and solutions before we can automate them, and the first steps must be to develop new tools that work hand-in-hand with operators, letting them bring their expertise to bear using metaphors and actions that match their view of system operation rather than the system's underlying organization. For example, uninstalling a system upgrade might require changes to many files and settings, but it would be more helpful to present the "undo" as a single

action, as the operator would think of it. If tools can't facilitate recovery when human operators must get involved, we question whether those operators will ever trust fully automatic systems.

A major challenge for recovery–oriented techniques involves knowing when and how frequently to try them. Although techniques such as machine learning and control theory can play useful roles in the future, these articles emphasize ways to empower application writers and service operators today to minimize recovery time in the face of failures. Once we understand the best we can do in empowering these expert users, and what we can learn from their experiences, we can consider automating solutions or deploying them in embedded, space-borne, and other systems in which human-error recovery tools are inappropriate because operators aren't nearby.

Recovery-oriented approaches are only one way to improve system availability. Indeed, decades of fault tolerance literature stimulate and complement these ongoing efforts. Yet, one reason to embrace recovery-oriented approaches is that Internet systems are constructed from off-the-shelf parts and must evolve rapidly to accommodate market pressures and constant growth. In contrast, previous efforts in fault tolerance assumed prespecified systems that rarely changed once deployed. Nonetheless, this issue's theme articles clearly demonstrate the earlier work's impact and show how techniques developed for those systems, such as replication, checkpointing, and logging, have been adapted to recovery-oriented approaches.

Armando Fox is an assistant professor in the computer science department at Stanford University. His research interests include mobile computing and dependability. Fox has a PhD in computer science from the University of California, Berkeley, and was among the "Scientific American 50" research leaders of 2003 for his work on recovery-oriented computing. He is a member of the ACM and the IEEE. Contact him at fox@cs.stanford.edu.

David Patterson is the Pardee Professor of Computer Science at the University of California, Berkeley. In addition to recovery-oriented computing, his research interests include systems in which the new building block is a whole computer rather than a transistor. Patterson received a PhD in electrical engineering and computer science from UCLA. He is a member of the US National Academy of Engineering, a fellow of both the IEEE and the ACM, serves on the US President's Information Technology Advisory Committee, and is President of the ACM. Contact him at patterson@cs.berkeley.edu.