Main Ideas

Failure is inevitable. Humans make mistakes. Systems employ cheap hardware prone to failure and run software that is poorly tested. Systems are composed of many independent subsystems that have unexpected interactions.

When subsystems fail, the complicated interactions make problem diagnosis difficult. Human attempts to fix the failures often worsen the problem because the technician is poorly trained and makes mistakes.

The state of the software industry is this: old software that was designed to be run in a failure-free environment must be patched as new interactions and failure modes appear. New software is not written with enough care to handle many, if any, failure modes. Hence, all software is vulnerable to unexpected failures. Consequently, all systems are vulnerable to unexpected and unexplainable failures.

Since companies lose large amounts of money during every system failure, it would make sense that companies shift from spending money fixing the system to spending money on developing applications that help the existing system recover from failure, or on developing an entirely new system that expects failure. However, the existing system would have to remain operational until a new system could be developed, and companies cannot afford to both develop and new system and maintain the existing system and its failures at the same time.

Technical Challenges

Since existing software fails often, it would be helpful to have a model of a system's interactions between subsystems, a method for testing individual subsystems, and a recovery utility that would either return the broken subsystem to a functioning state, or "reboot" the broken subsystem without disrupting the rest of the system.

The problem is that such a ROC is merely a patch on top of a broken system, and it has no way of preventing failures.

New Ideas and Applications

An ideal ROC would maintain additionally performance metrics for each subsystem, be able to determine how close a subsystem might be to failure, and be able to correct the problem before the failure emerged.