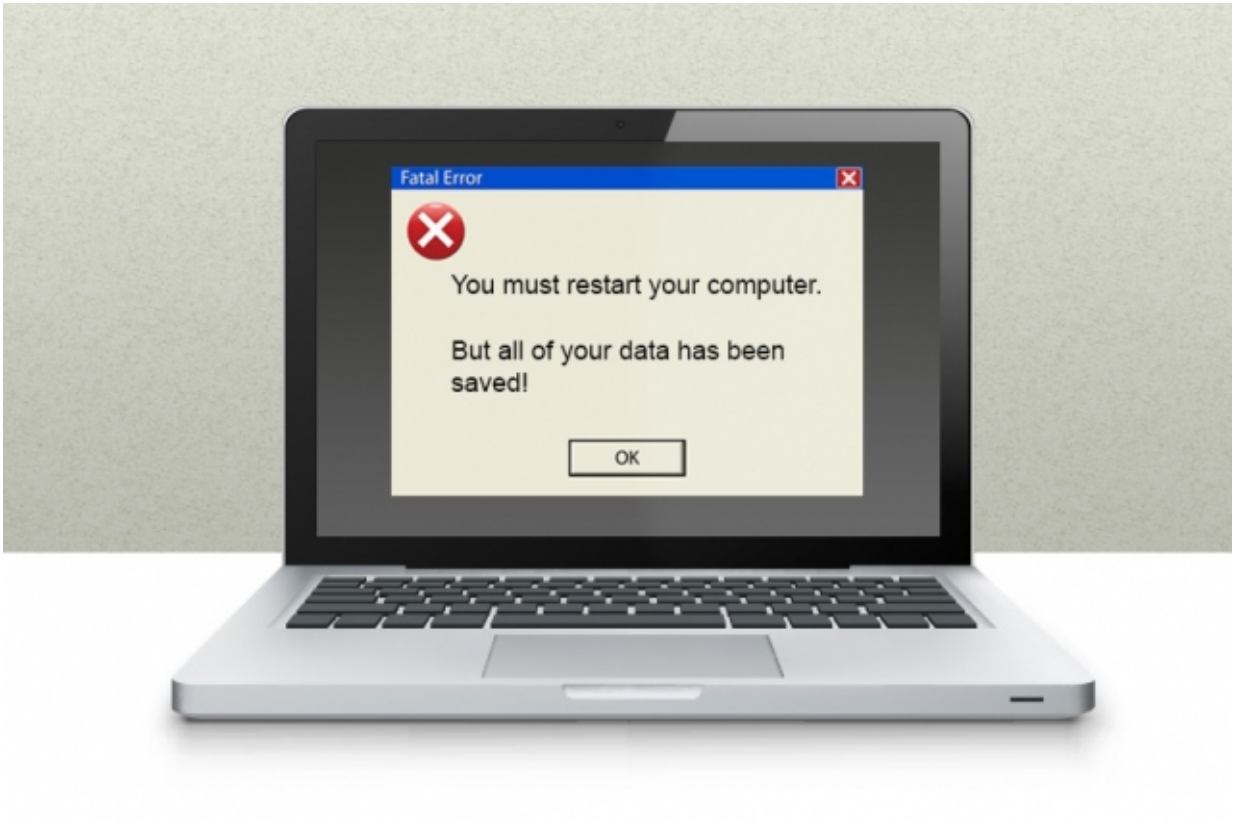
Crash-tolerant data storage: Formally verified working file system could end data loss

August 24 2015, by Larry Hardesty



Credit: Christine Daniloff/MIT

In a computer operating system, the file system is the part that writes data to disk and tracks where the data is stored. If the computer crashes

while it's writing data, the file system's records can become corrupt. Hours of work could be lost, or programs could stop working properly.

At the ACM Symposium on Operating Systems Principles in October, MIT researchers will present the first file system that is mathematically guaranteed not to lose track of data during crashes. Although the file system is slow by today's standards, the techniques the researchers used to verify its performance can be extended to more sophisticated designs. Ultimately, formal verification could make it much easier to develop reliable, efficient file systems.

"What many people worry about is building these file systems to be reliable, both when they're operating normally but also in the case of crashes, power failure, software bugs, hardware errors, what have you," says Nickolai Zeldovich, an associate professor of computer science and engineering and one of three MIT computer-science professors on the new paper. "Making sure that the file system can recover from a crash at any point is tricky because there are so many different places that you could crash. You literally have to consider every instruction or every disk operation and think, 'Well, what if I crash now? What now? What now?' And so empirically, people have found lots of bugs in file systems that have to do with crash recovery, and they keep finding them, even in very well tested file systems, because it's just so hard to do."

Proving ground

Zeldovich and his colleagues—Frans Kaashoek, the Charles A. Piper Professor in MIT's Department of Electrical Engineering and Computer Science (EECS); associate professor of computer science Adam Chlipala; Haogang Chen, a graduate student in EECS; and Daniel Ziegler, an undergraduate in EECS—established the reliability of their file system through a process known as formal verification.

Formal verification involves mathematically describing the acceptable bounds of operation for a computer program and then proving that the program will never exceed them. It's a complicated process, so it's generally applied only to very high-level schematic representations of a program's functionality. Translating those high-level schema into working code, however, can introduce myriad complications that the proofs don't address.

"All these paper proofs about other file systems may actually be correct, but there's no file system that we can be sure represents what the proof is about," Ziegler says.

What distinguishes the MIT researchers' work is that they prove properties of the file system's final code, not a high-level schema. To do that, they took advantage of a tool known as a proof assistant, which provides a formal language for describing aspects of a computer system and the relationships between them.

"This formal proving environment includes a programming language," Chlipala explains. "So we implement the file system in the same language where we're writing the proofs. And the proofs are checked against the actual file system, not some whiteboard idealization that has no formal connection to the code."

The proof assistant, known as Coq, provided the tools, but the MIT researchers still had to do the work. First, they had to describe the components of a file system using Coq's formal language. "You have to define, 'What is a disk?'" Zeldovich says.

"And 'What is a bit?'" Chlipala adds.

Next, they had to formally describe the relationships between the behaviors of these different components under crash conditions. Only

then could they begin to construct a proof that a file system would behave the way it should. Finally, they had to write the corresponding file system. The part of the process that Coq automated was determining that the file system did, in fact, adhere to the logical relationships described in the proof.

Reproducibility

In the course of writing the file system, they repeatedly went back and retooled the system specifications, and vice versa. But even though they rewrote the file system "probably 10 times," Zeldovich says, Kaashoek estimates that they spent 90 percent of their time on the definitions of the system components and the relationships between them and on the proof.

"We've written file systems many times over, so we know exactly what it's going to look like," Zeldovich says. "Whereas with all these logics and proofs, there are so many ways to write them down, and each one of them has subtle implications down the line that we didn't really understand."

"No one had done it," Kaashoek adds. "It's not like you could look up a paper that says, 'This is the way to do it.' But now you can read our paper and presumably do it a lot faster."

"It's not like people haven't proven things in the past," says Ulfar Erlingsson, lead manager for security research at Google, who has observed the new work from a distance. "But usually the methods and technologies, the formalisms that were developed for creating the proofs, were so esoteric and so specific to the problem that there was basically hardly any chance that there would be repeat work that built up on it. But I can say for certain that Adam's stuff with Coq, and separation logic, this is stuff that's going to get built on and applied in

many different domains. That's what's so exciting."

This story is republished courtesy of MIT News (web.mit.edu/newsoffice/), a popular site that covers news about MIT research, innovation and teaching.

Provided by Massachusetts Institute of Technology