Using lightweight formal methods to validate a key-value storage node in Amazon S3

James Bornholt\textsuperscript{1,2} Rajeev Joshi\textsuperscript{1} Vytautas Astrauskas\textsuperscript{1} Brendan Cully\textsuperscript{1} Bernhard Kragi\textsuperscript{1} Seth Markle\textsuperscript{1} Kyle Sauri\textsuperscript{1} Drew Schleit\textsuperscript{1} Grant Slatten\textsuperscript{1} Serdar Tasiran\textsuperscript{1} Jacob Van Geffen\textsuperscript{2} Andrew Warfield\textsuperscript{3}

\textsuperscript{1}Amazon Web Services \textsuperscript{2}The University of Texas at Austin \textsuperscript{3}ETH Zurich \textsuperscript{4}University of Washington

Lightweight formal methods to validate ShardStore with low overhead

We developed a lightweight formal methods approach to validate deep correctness properties of ShardStore in an automated fashion. Our approach has three elements:
1. We develop executable reference models as specifications that live alongside the implementation code
2. We apply a suite of automated conformance checking tools to validate that the implementation code respects the specification
3. We implement mechanisms to measure the effectiveness of these checks to ensure future changes to ShardStore are still correct

In return for being lightweight and automated, we accept weaker correctness guarantees than full formal verification—we can still miss bugs. But we gain the ability for future engineers to validate their changes without expensive new verification work by formal methods experts.

Reference models make for simple, maintainable specifications

We choose to write our specifications as executable reference models—small pieces of code that offer the same interface as the component they specify, but without concern for implementation efficiencies. These specifications are easy to update over time because they’re written in Rust like the implementation, and are reused for other purposes (e.g., as mocks for unit testing).

For concurrency we check that the implementation linearizes with respect to the reference model using stateless model checking, which executes a piece of code many times with a different thread interleaving each time.

All of these tools are “pay-as-you-go”—they scale with compute, so we can run them at small scale on engineer laptops to test local changes, or at massive scale in the cloud before deployments to gain confidence in correctness.

The result is that we can validate the correctness of every ShardStore commit, preventing bugs from reaching production, or even reaching code review.

Automated “pay-as-you-go” checkers validate the code on every commit

Rather than a one-size-fits-all tool, we decompose our correctness property into smaller pieces and check each with a different tool.

For crash consistency we check that the implementation refines the reference model using property-based testing to test random traces:

We're hiring!

The S3 Automated Reasoning Group applies formal methods to build correct, secure, durable, and highly available distributed storage systems.
We're hiring for both full-time and intern positions.
Contact us at 53-arg-jobs@amazon.com.