Tyler McMullen
Fastly
@tbcmcmullen
The Great Velocity EU Incident of 2013
1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.
Vehicle-to-vehicle networks could save over 1,000 lives a year, US says

The federal highway-safety agency wants wireless systems to warn drivers before they crash

By Stephen Lawson

Senior U.S. Correspondent, IDG News Service | AUG 18, 2014 7:34 PM PT
Lineage-driven Fault Injection

Peter Alvaro  
UC Berkeley  
palvaro@cs.berkeley.edu

Joshua Rosen  
UC Berkeley  
rosenville@gmail.com

Joseph M. Hellerstein  
UC Berkeley  
hellerstein@cs.berkeley.edu

ABSTRACT
Failure is always an option; in large-scale data management systems, it is practically a certainty. Fault-tolerant protocols and components are notoriously difficult to implement and debug. Worse still, choosing existing fault-tolerance mechanisms and integrating them correctly into complex systems remains an art form, and programmers have few tools to assist them.

We propose a novel approach for discovering bugs in fault-tolerant data management systems: lineage-driven fault injection. A lineage-driven fault injector reasons backwards from correct system outcomes to determine whether failures in the execution could have prevented the outcome. We present MOLLY, a prototype of lineage-driven fault injection that exploits a novel combination of data lineage techniques from the database literature and state-of-the-art satisfiability testing. If fault-tolerance bugs exist for a particular configuration, MOLLY finds them rapidly, in many cases using an order of magnitude fewer executions than random fault injection. Enriching new system architectures with well-understood fault tolerance mechanisms and henceforth assuming that failures will not affect system outcomes. Unfortunately, fault-tolerance is a global property of entire systems, and guarantees about the behavior of individual components do not necessarily hold under composition. It is difficult to design and reason about the fault-tolerance of individual components, and often equally difficult to assemble a fault-tolerant system even when given fault-tolerant components, as witnessed by recent data management system failures [16, 57] and bugs [36, 49].

Top-down testing approaches—which perturb and observe the behavior of complex systems—are an attractive alternative to verification of individual components. Fault injection [1, 26, 36, 44, 59] is the dominant top-down approach in the software engineering and dependability communities. With minimal programmer investment, fault injection can quickly identify shallow bugs caused by a small number of independent faults. Unfortunately, fault injection is poorly suited to discovering rare, counterexample-in
The Verification of a Distributed System

Leslie Lamport, known for his seminal work in distributed systems, famously said, “A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer useless.”
Edge Compute?
Large-scale
Coordination-free
Distributed
Systems
application
device

edge

origin
... and so on

origin

device

device

edge
Autonomy
Thank You.

@tbmcmullen