Formal Methods Solve Only Half My Problems

(and I have a lot of problems)

Marc Brooker

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other coordination tasks. There are many such algorithms in the literature, but combining them into a cohesive system is a challenge, as the algorithms must usually be modified to interact properly in a real-world system. In addition, we have found it necessary to invent algorithms of our own. We work hard to avoid unnecessary complexity, but the essential complexity of the task remains high.

Complexity increases the probability of human error in design, code, and operations. Errors in the core of the system could cause loss or corruption of data, or violate other interface contracts on which our customers de-

aws

Hubris
Humility
Laziness

aws

Kani Rust Verifier Blog

About

Using the Kani Rust Verifier on a Firecracker Example

Jul 13, 2022

In this post we'll apply the Kani Rust Verifier (or Kani for short), our open-source formal verification tool that can prove properties about Rust code, to an example from Firecracker, an open source virtualization project for serverless applications. We will use Kani to get a strong guarantee that Firecracker's block device is correct with respect to a simple virtio property when parsing guest requests, which may be invalid or malicious. In this way, we show how Kani can complement Firecracker's defense in depth investments, such as fuzzing.

aws

Using Lightweight Formal Methods to Validate a Key-Value Storage Node in Amazon S3

James Bornholt
Amazon Web Services
& The University of Texas at Austin

Rajeev Joshi
Amazon Web Services

Vytautas Astrauskas
ETH Zurich

Brendan Cully
Amazon Web Services

Bernhard Kragl
Amazon Web Services

Seth Markle
Amazon Web Services

Kyle Sauri
Amazon Web Services

Drew Schleit
Amazon Web Services

Grant Slatton
Amazon Web Services

Serdar Tasiran
Amazon Web Services

Jacob Van Geffen
University of Washington

Andrew Warfield
Amazon Web Services

Abstract

This paper reports our experience applying lightweight formal methods to validate the correctness of ShardStore, a new key-value storage node implementation for the Amazon S3 cloud object storage service. By "lightweight formal methods"

Using Lightweight Formal Methods to Validate a Key-Value Storage Node in Amazon S3. In *ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21), October 26–28, 2021, Virtual Event, Germany*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3477132.3483540

Semantic-based Automated Reasoning for AWS Access Policies using SMT

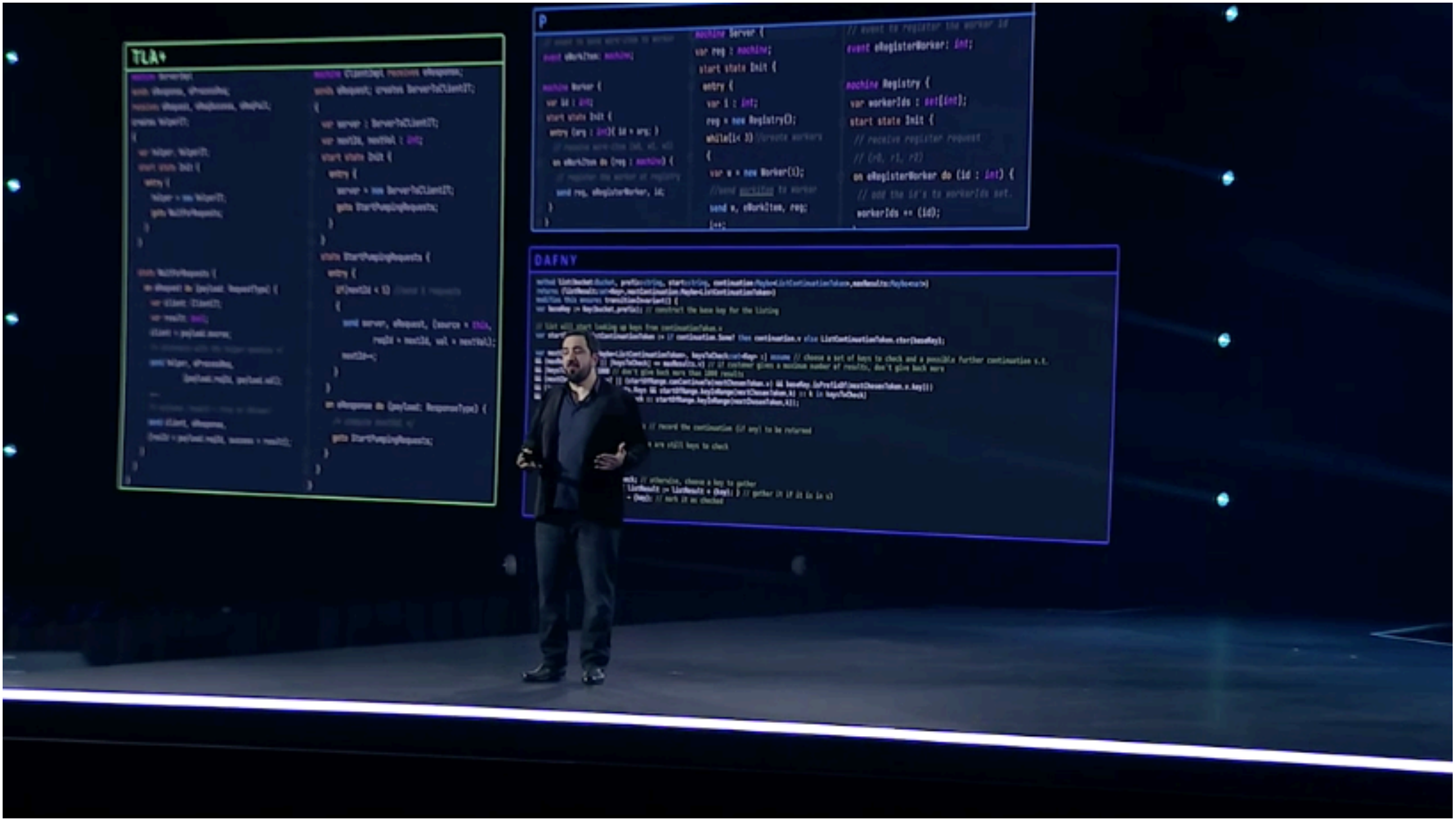
John Backes, Pauline Bolignano, Byron Cook, Catherine Dodge, Andrew Gacek,
Kasper Luckow, Neha Rungta, Oksana Tkachuk, Carsten Varming
Amazon Web Services

***Abstract*—Cloud computing provides on-demand access to IT resources via the Internet. Permissions for these resources are defined by expressive access control policies. This paper presents a formalization of the Amazon Web Services (AWS) policy language and a corresponding analysis tool, called ZELKOVA, for verifying policy properties. ZELKOVA encodes the semantics of policies into SMT, compares behaviors, and verifies properties. It provides users a sound mechanism to detect misconfigurations of their policies. ZELKOVA solves a PSPACE-complete problem and is invoked many millions of times daily.**

I. INTRODUCTION

In this paper, we present the development and application of ZELKOVA, a policy analysis tool designed to reason about the semantics of AWS access control policies. ZELKOVA translates policies and properties into Satisfiability Modulo Theories (SMT) formulas and uses SMT solvers to check the validity of the properties. We use off-the-shelf solvers and an in-house extension of Z3 called Z3AUTOMATA.

ZELKOVA reasons about all possible permissions allowed by a policy in order to verify properties. For example, ZELKOVA can answer the questions "Is this resource accessible by a


TLA+
P
machine Server {
var reg : machine;
start state Init {
entry {
var i : int;
reg = new Registry();
while(i< 3) //create workers
{
var w = new Worker(i);
//send workitem to worker
send w, eWorkItem, reg;
i++;
// event to register the worker id
event eRegisterWorker: int;
machine Registry {
var workerIds : set[int];
start state Init {
// receive register request
// (r0, r1, r2)
on eRegisterWorker do (id : int) {
// add the id's to workerIds set.
workerIds += (id);
DAFNY

Safety

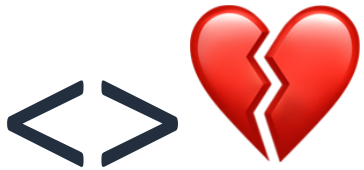
Liveness

[]❤️

aws

Safety

Liveness



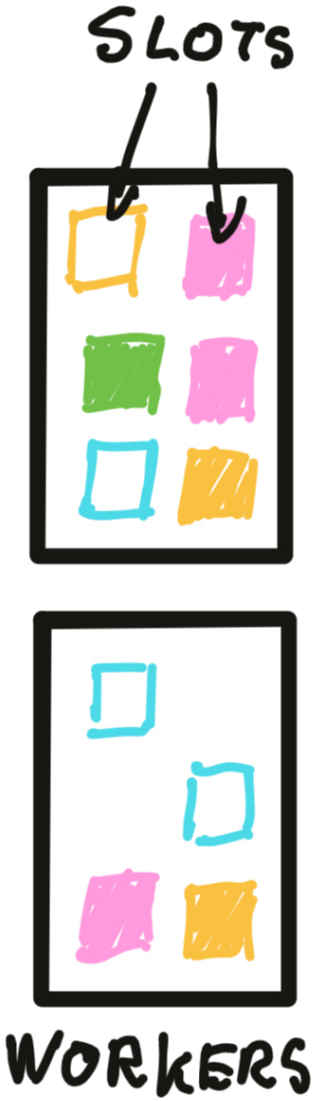
Distributed systems are complex dynamical systems.

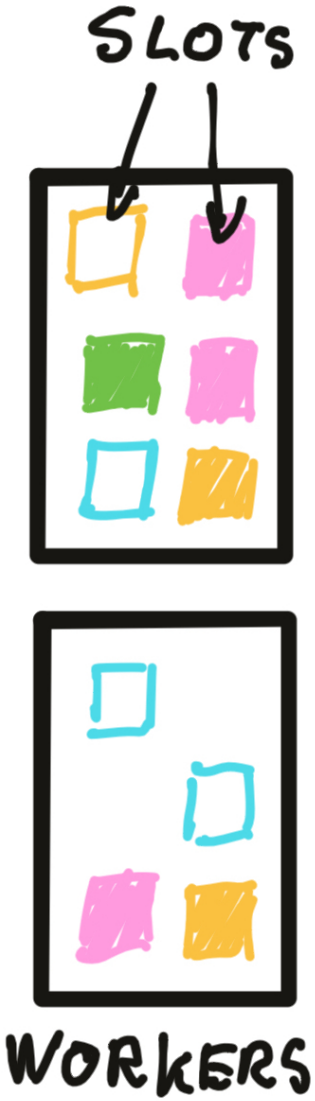
aws

Distributed systems are complex dynamical systems.

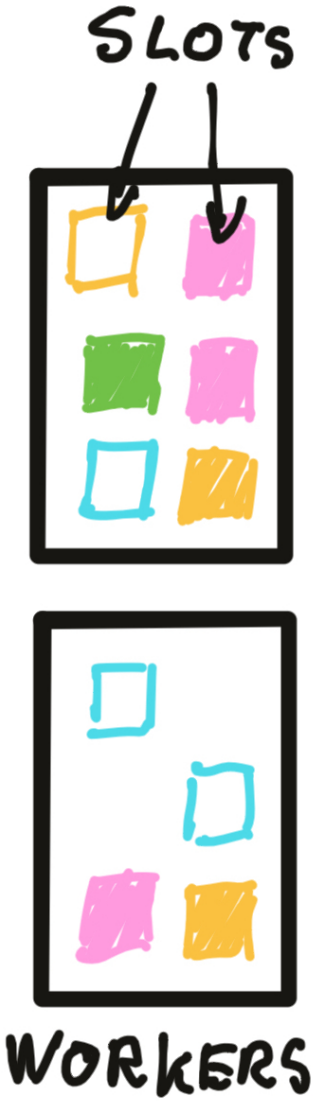
We don’t understand their dynamics!

aws

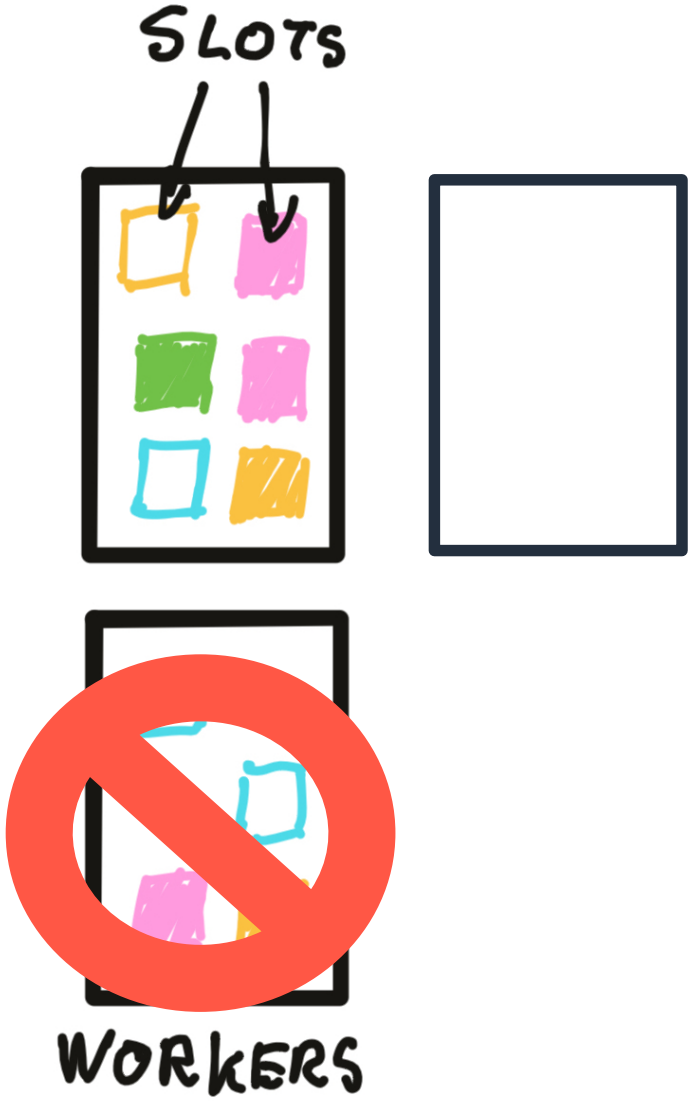




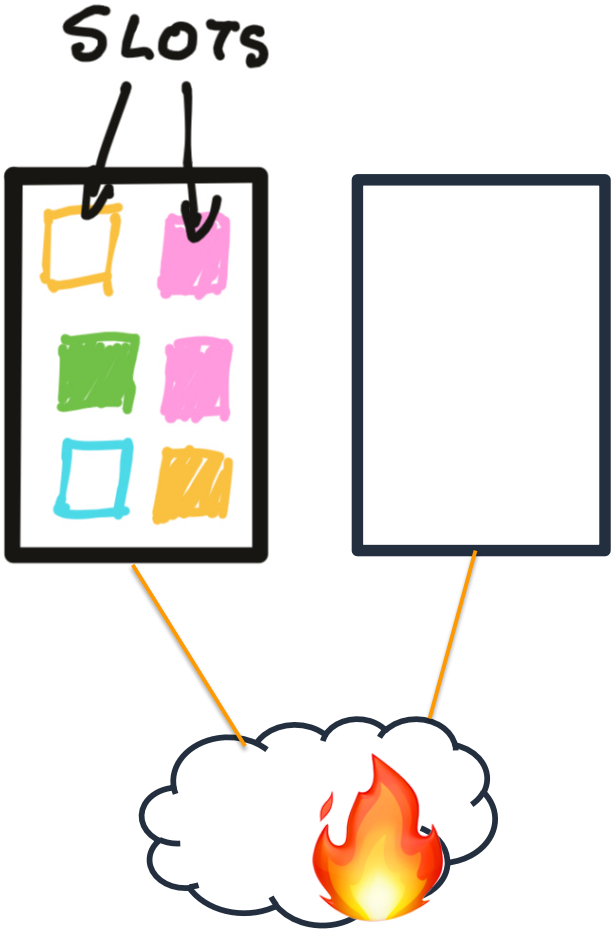
aws



aws



aws

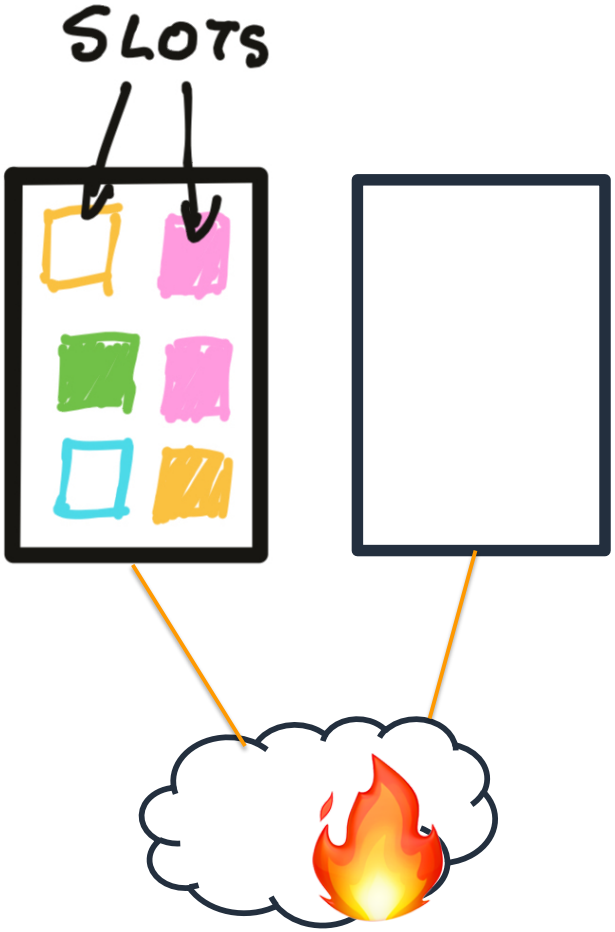


aws

TASK
STREAM

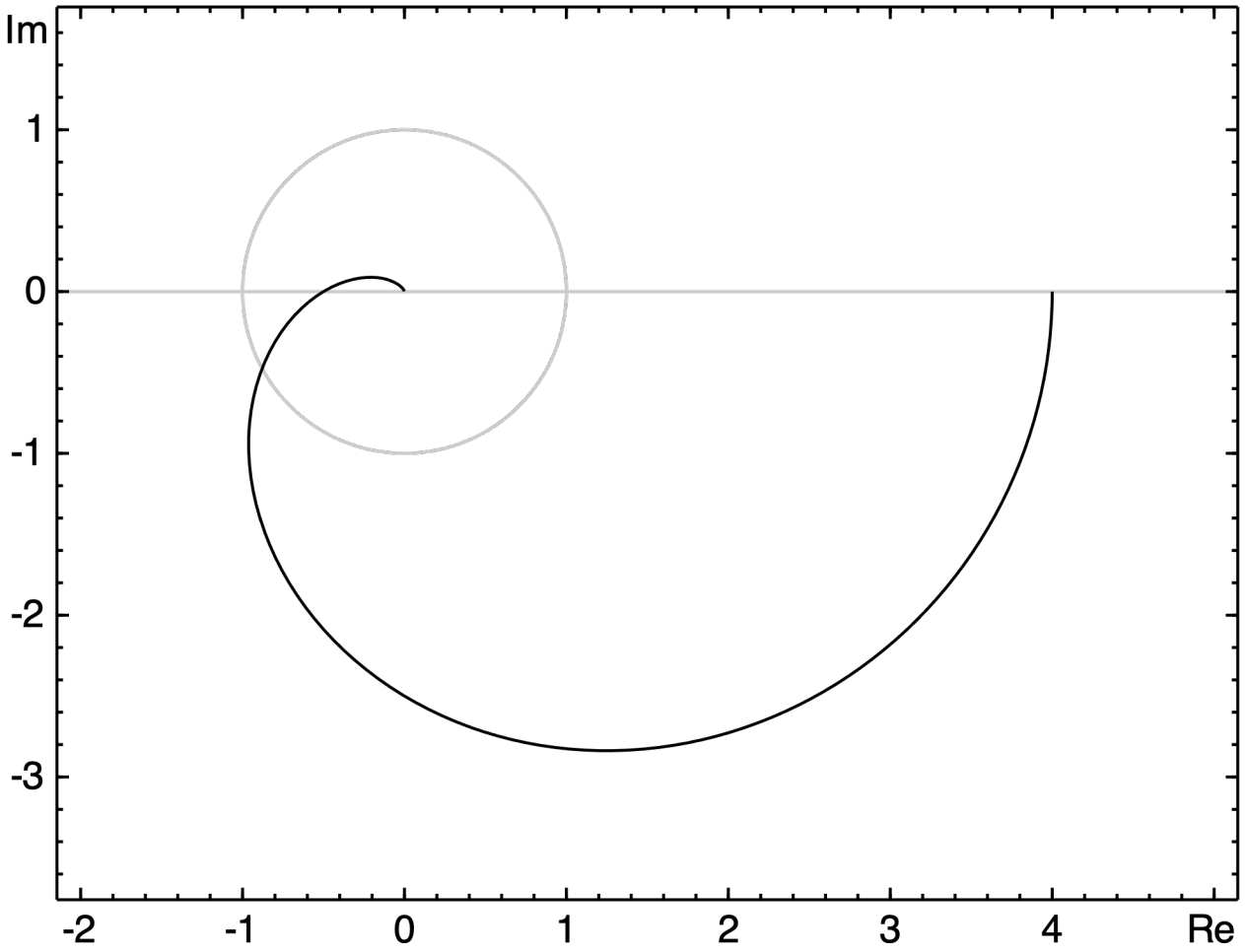
x4

eLSA



aws

aws



"Reasoning purely analytically about the behavior of complex stochastic systems is generally infeasible."

Agha and Palmskog, *A Survey of Statistical Model Checking,* TOMACS, January 2018

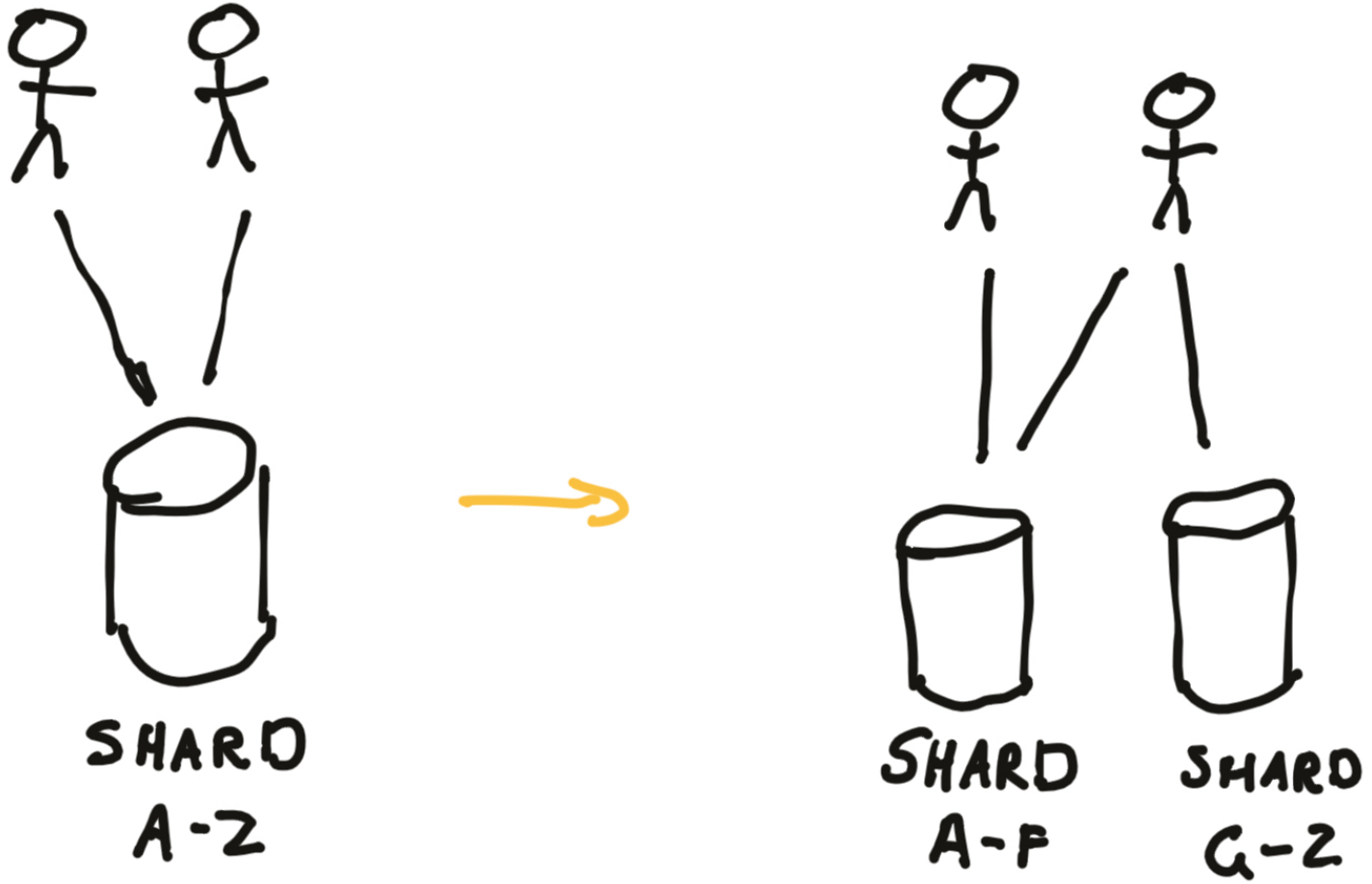
aws

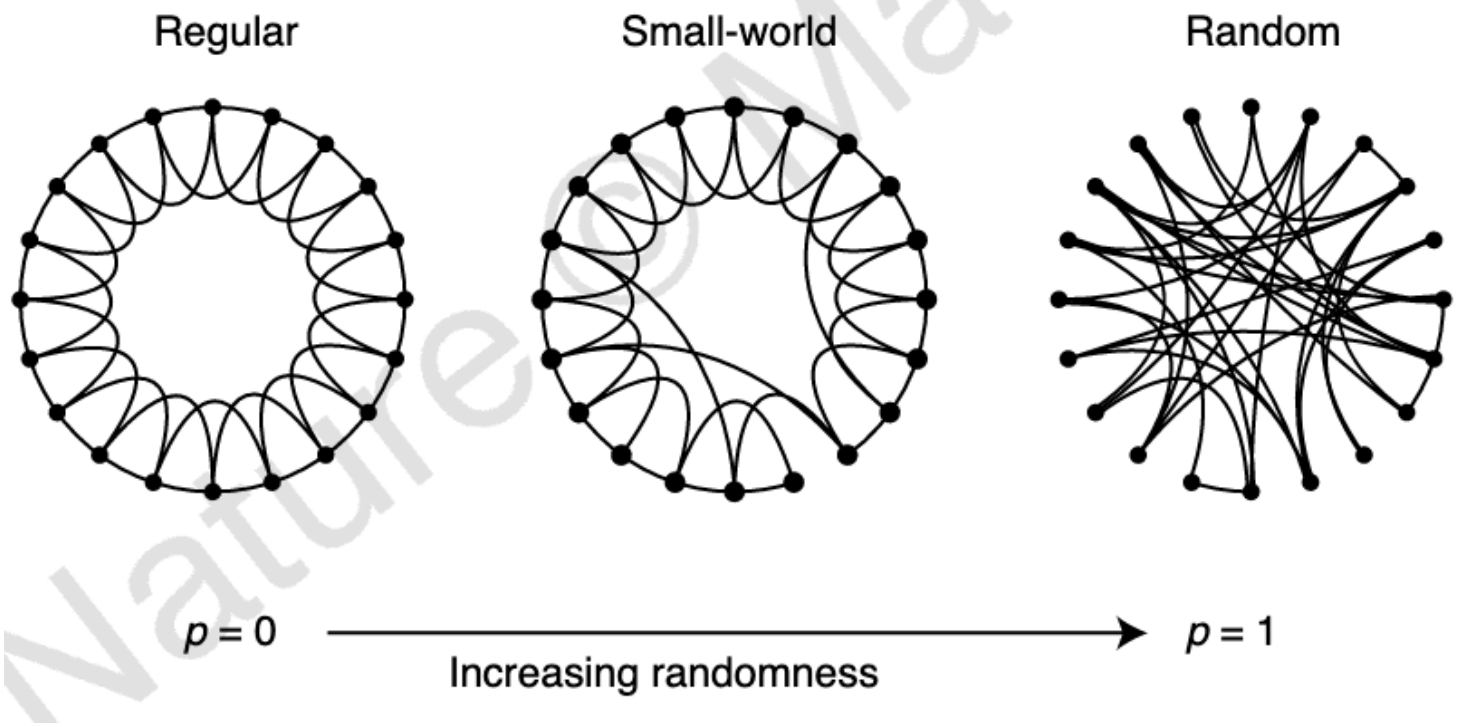
Question #1:

How do we understand system dynamics better?

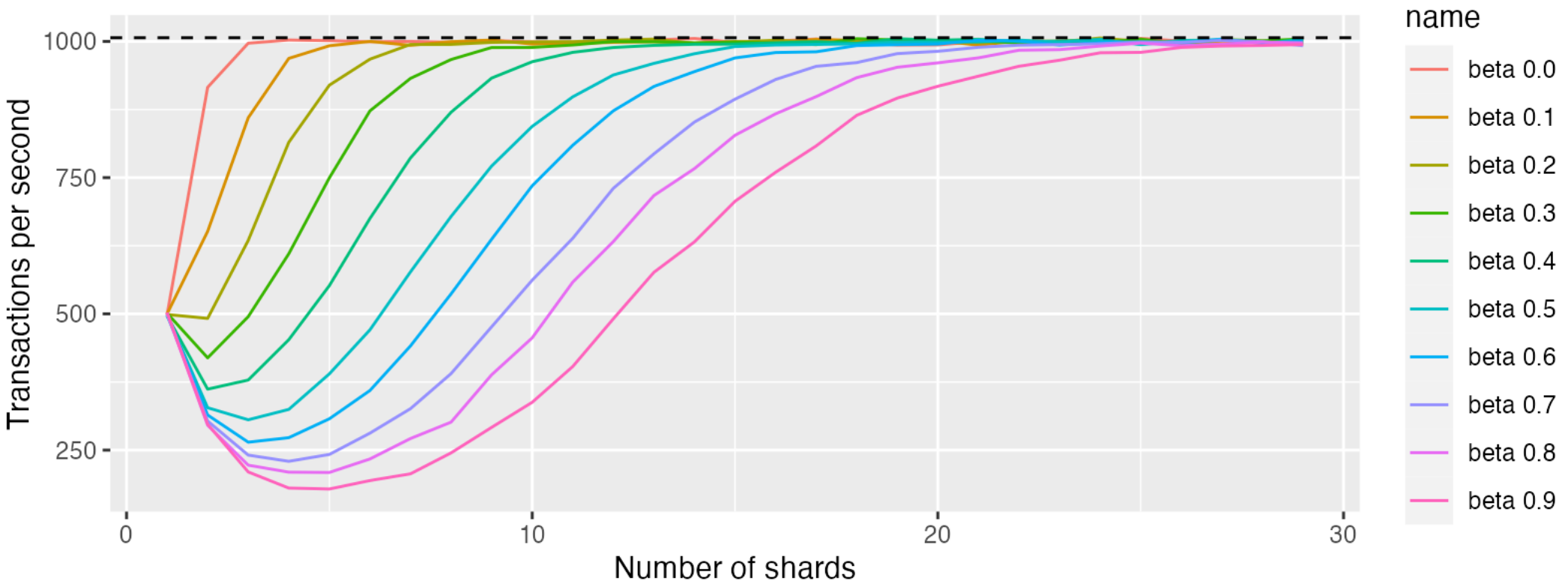
aws

aws





Watts and Strogatz, “Collective dynamics of ‘small-world’ networks”, Nature, 1998



~5,000 lines of Rust

but…

aws

We already have a specification!

We already have a tool that searches the specification's state space!

aws

Question #2:

Can we get more value from specifications?

aws

Distributed systems are complex dynamical systems.

We don’t understand their dynamics.

This is a problem.

aws

Distributed systems are complex dynamical systems.

We don’t understand their dynamics.

This is a problem.

We’re not going to grow up till we solve it.

aws

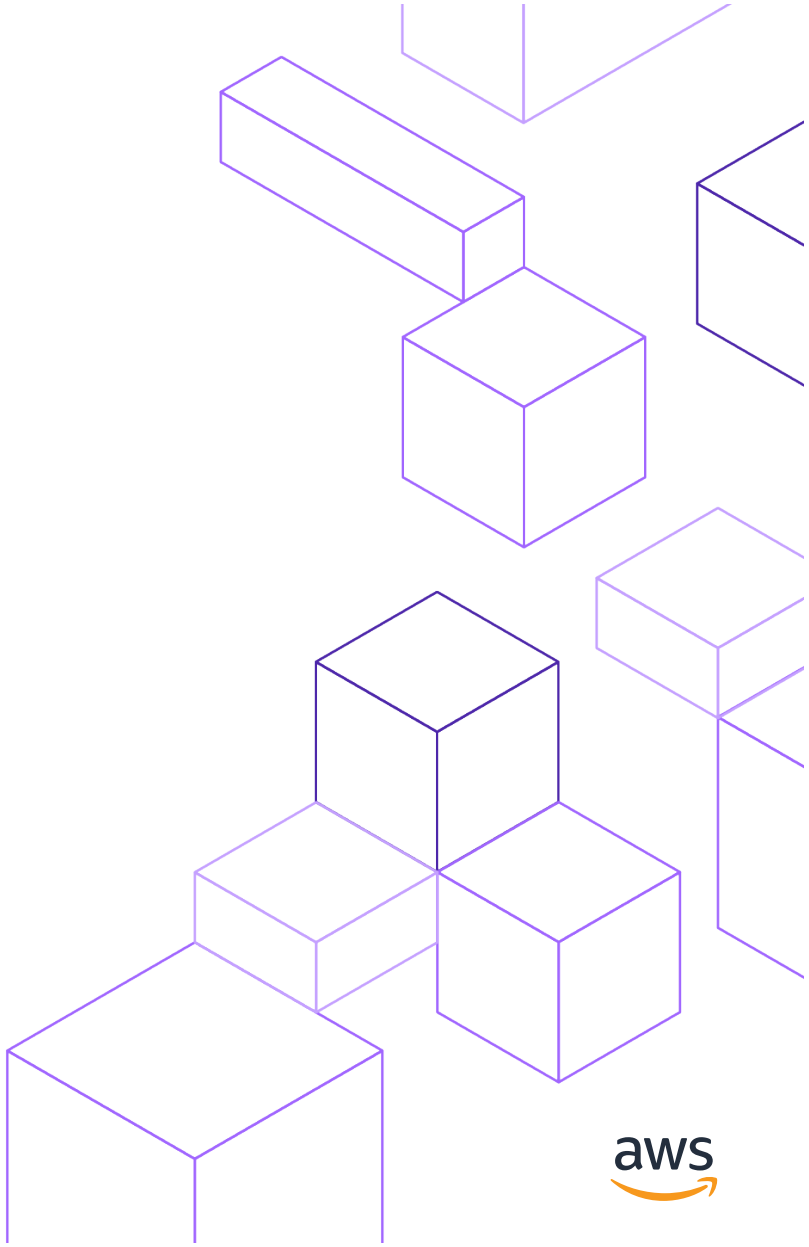
Answers?

Marc Brooker

mbrooker@amazon.com

marcbrooker@gmail.com

@marcjbrooker



Millions of Tiny Databases

Marc Brooker
Amazon Web Services

Tao Chen
Amazon Web Services

Fan Ping
Amazon Web Services

Abstract

Starting in 2013, we set out to build a new database to act as the configuration store for a high-performance cloud block storage system (Amazon EBS).This database needs to be not only highly available, durable, and scalable but also strongly consistent. We quickly realized that the constraints on availability imposed by the CAP theorem, and the realities of operating distributed systems, meant that we didn't want one database. We wanted millions. Physalia is a transactional key-value store, optimized for use in large-scale cloud control planes, which takes advantage of knowledge of transaction patterns and infrastructure design to offer both high availability and strong consistency to millions of clients. Physalia uses its knowledge of datacenter topology to place data where it is most likely to be available. Instead of being highly available for all keys to all clients, Physalia focuses on being extremely available for only the keys it knows each client needs, from the perspective of that client.

This paper describes Physalia in context of Amazon EBS, and some other uses within Amazon Web Services. We believe that the same patterns, and approach to design, are widely applicable to distributed systems problems like control planes, configuration management, and service discovery.

1 Introduction

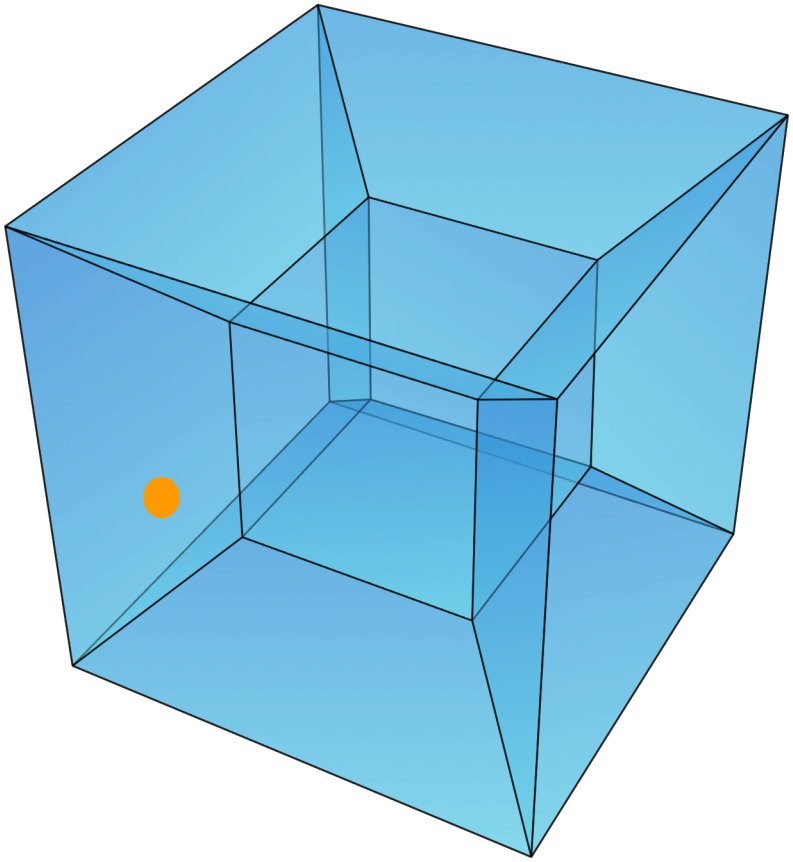
Traditional architectures for highly-available systems assume that infrastructure failures are statistically independent, and that it is extremely unlikely for a large number of servers to fail at the same time. Most modern system designs are aware of broad failure domains (data centers or availability zones), but still assume two modes of failure: a complete failure of a datacenter, or a random uncorrelated failure of a server, disk or other infrastructure. These assumptions are reasonable for most kinds of systems. Schroder and Gibson found [51] that (in traditional datacenter environments), while the probability of a second disk failure in a week was up to 9x higher when a first failure had already occurred, this correlation drops off to less than 1.5x as systems age. While a 9x higher failure rate within the following week indicates some correlation, it is still very rare for two disks to fail at the same time. This is just as well, because systems like RAID [43] and primary-backup failover perform well when failures are independent, but poorly when failures occur in bursts.

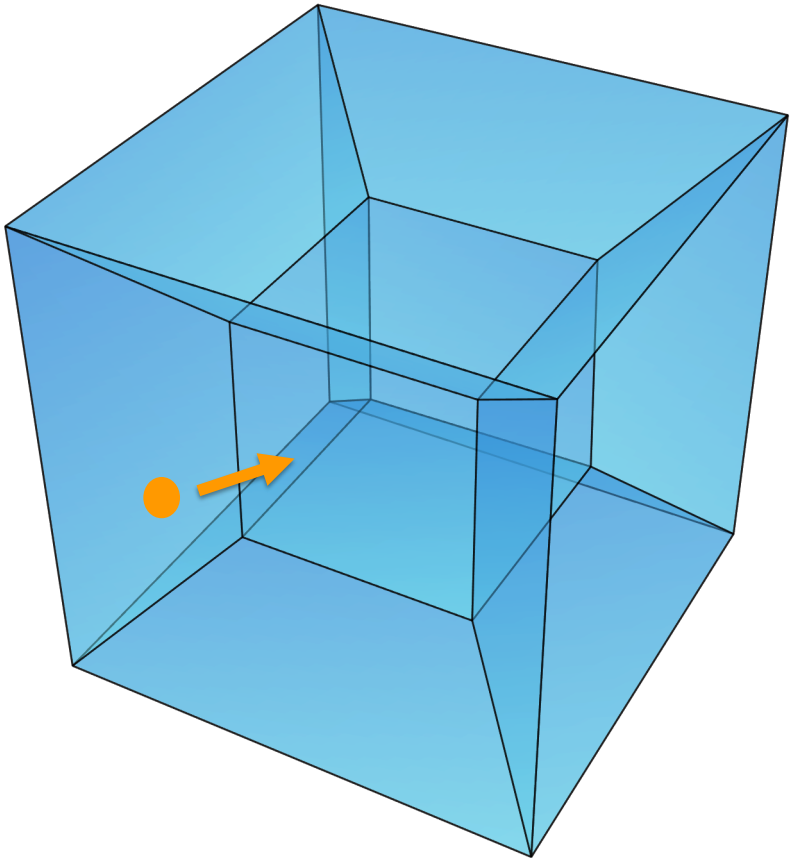
When we started building AWS in 2006, we measured the availability of systems as a simple percentage of the time that the system is available (such as 99.95%), and set Service Level Agreements (SLAs) and internal goals around this percentage. In 2008, we introduced AWS EC2 Availability Zones: named units of capacity with clear expectations and SLAs around correlated failure, corresponding to the datacenters that customers were already familiar with. Over the decade since, our thinking on failure and availability has continued to evolve, and we paid increasing attention to *blast radius* and correlation of failure. Not only do we work to make outages rare and short, we work to reduce the number of resources and customers that they affect [55], an approach we call *blast radius reduction*. This philosophy is reflected in everything from the size of our datacenters [30], to the design of our services, to operational practices.

Amazon Elastic Block Storage (EBS) is a block storage service for use with AWS EC2, allowing customers to create block devices on demand and attach them to their AWS EC2 instances. volumes are designed for an annual failure rate (AFR) of between 0.1% and 0.2%, where failure refers to a complete or partial loss of the volume. This is significantly lower than the AFR of typical disk drives [44]. EBS achieves this higher durability through replication, implementing a chain replication scheme (similar to the one described by van Renesse, et al [54]). Figure 1 shows an abstracted, simplified, architecture of EBS in context of AWS EC2. In normal operation (of this simplified model), replicated data flows through the chain from client, to primary, to replica, with no need for coordination. When failures occur, such as the failure of the primary server, this scheme requires the services of a *configuration master*, which ensures that updates to the order and membership of the replication group occur atomically, are

USENIX Association 17th USENIX Symposium on Networked Systems Design and Implementation 463

USENIX Association 17th USENIX Symposium on Networked Systems Design and Implementation 463





Question #3:

Can we mechanize exploration of the tradeoff space?

aws