Fifteen Years of TLA+

Marc Brooker

VP/Distinguished Engineer
mbrooker@amazon.com

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

- The Past
- The Present
- The Future

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

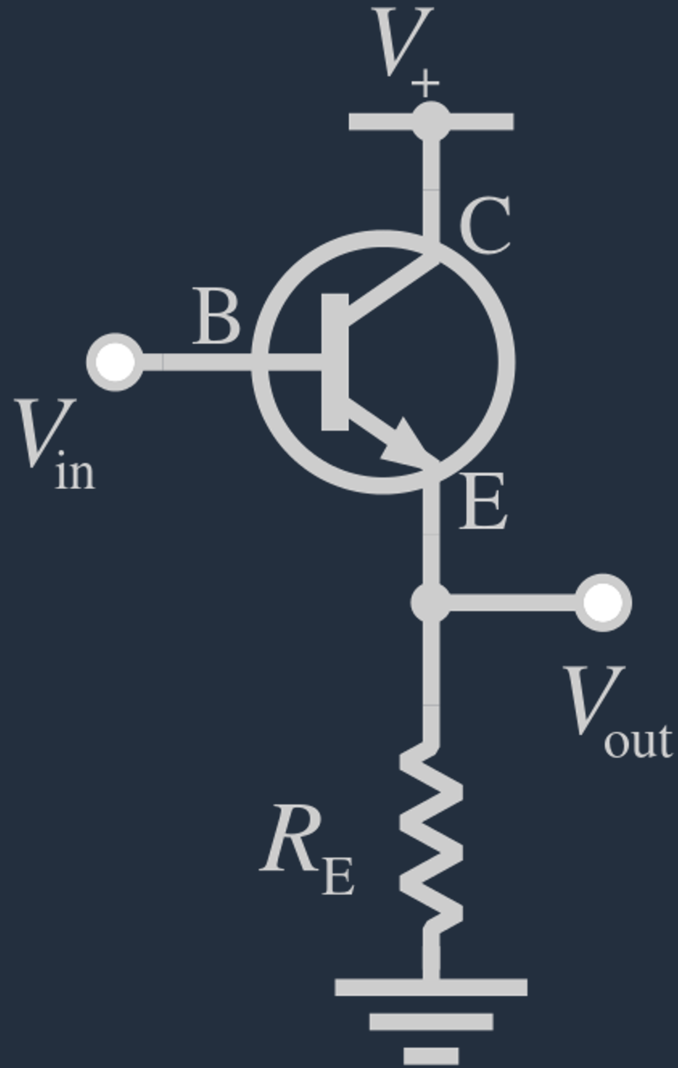
The Past

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

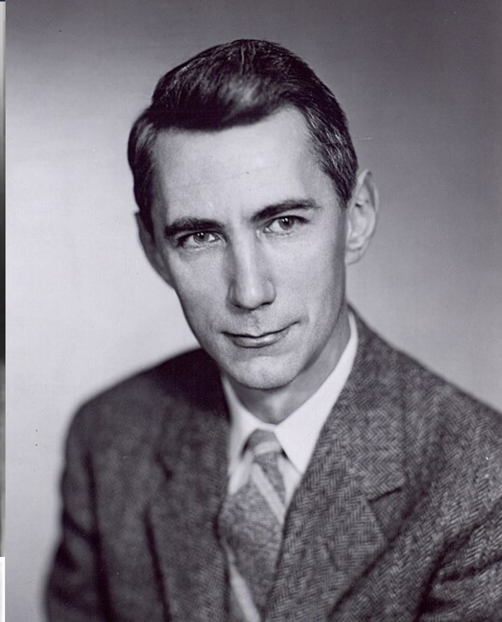
About Me

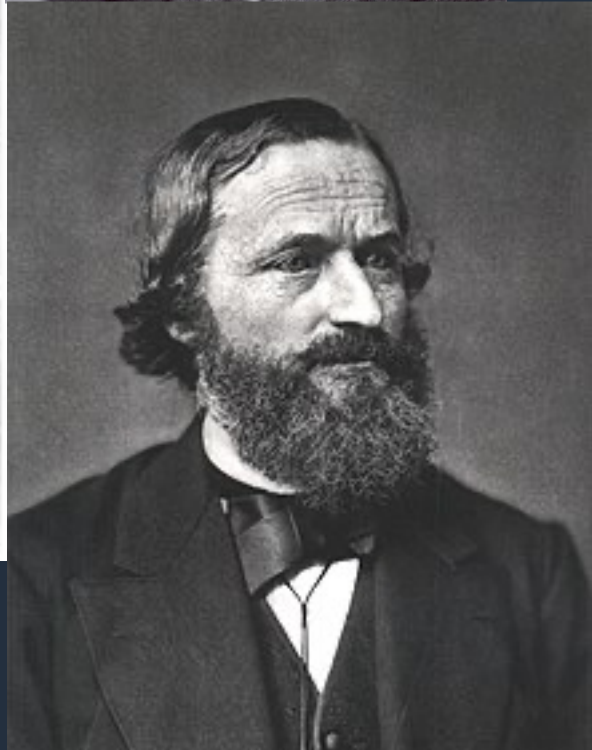
- 15+ years at AWS.
- Oncall for 15 years.
- Aurora, Lambda, EC2, EBS, API Gateway, IoT, Bedrock, and others.
- Mostly a practitioner.

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.



aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.





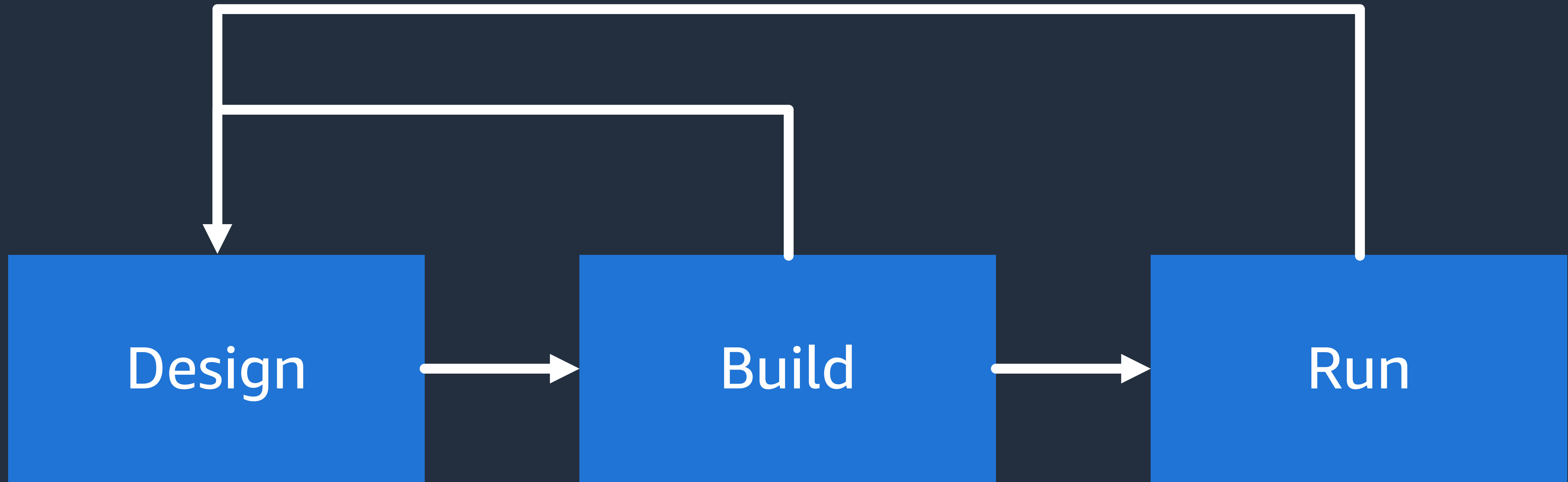
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

With mathematics we can:

- Quantify output behavior
- Predict behavior of systems as design time
- Understand safety margins
- Understand system interactions

(within limits)

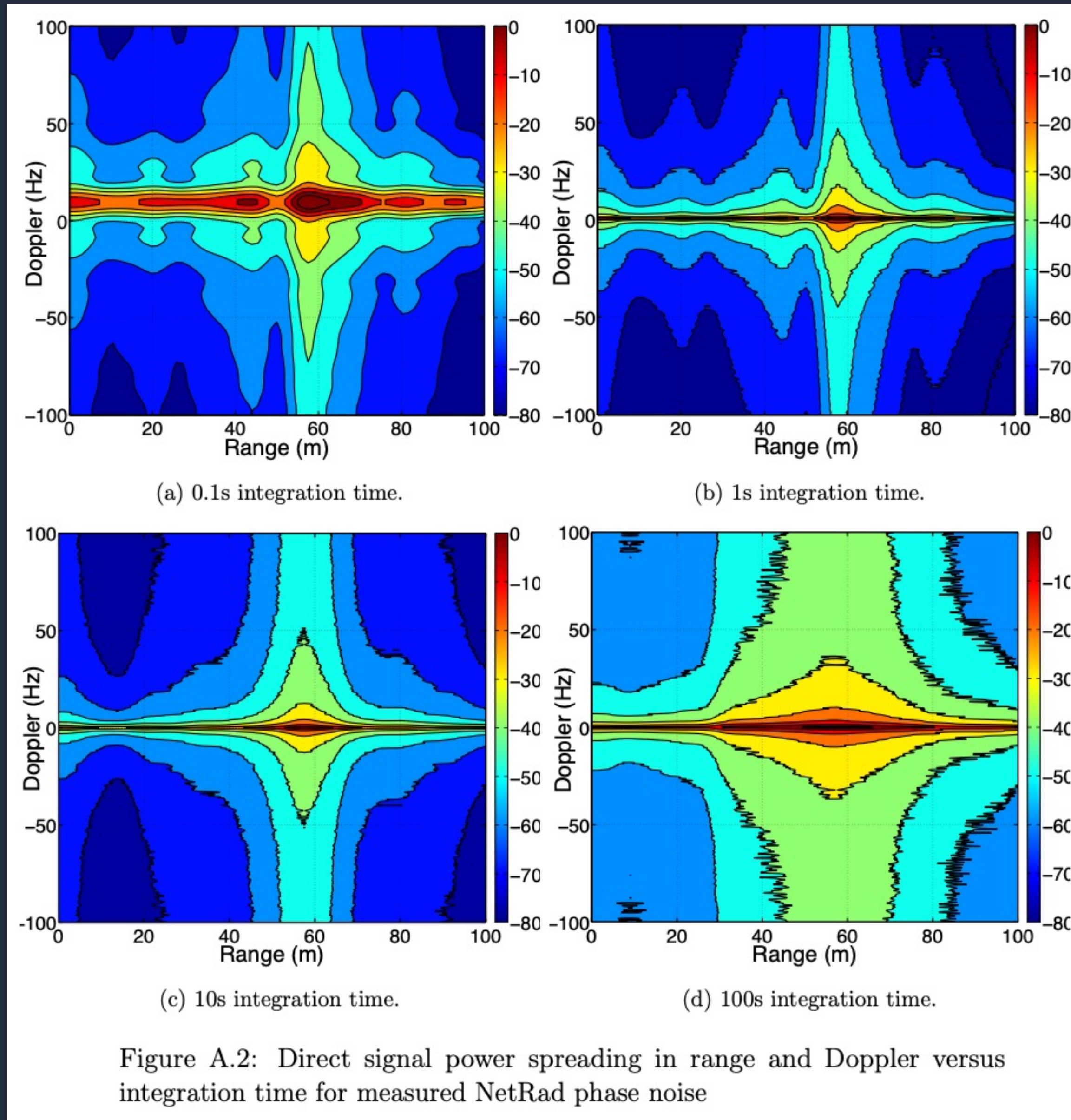
aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

This is awesome!

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.



(a) 0.1s integration time.

(b) 1s integration time.

(c) 10s integration time.

(d) 100s integration time.

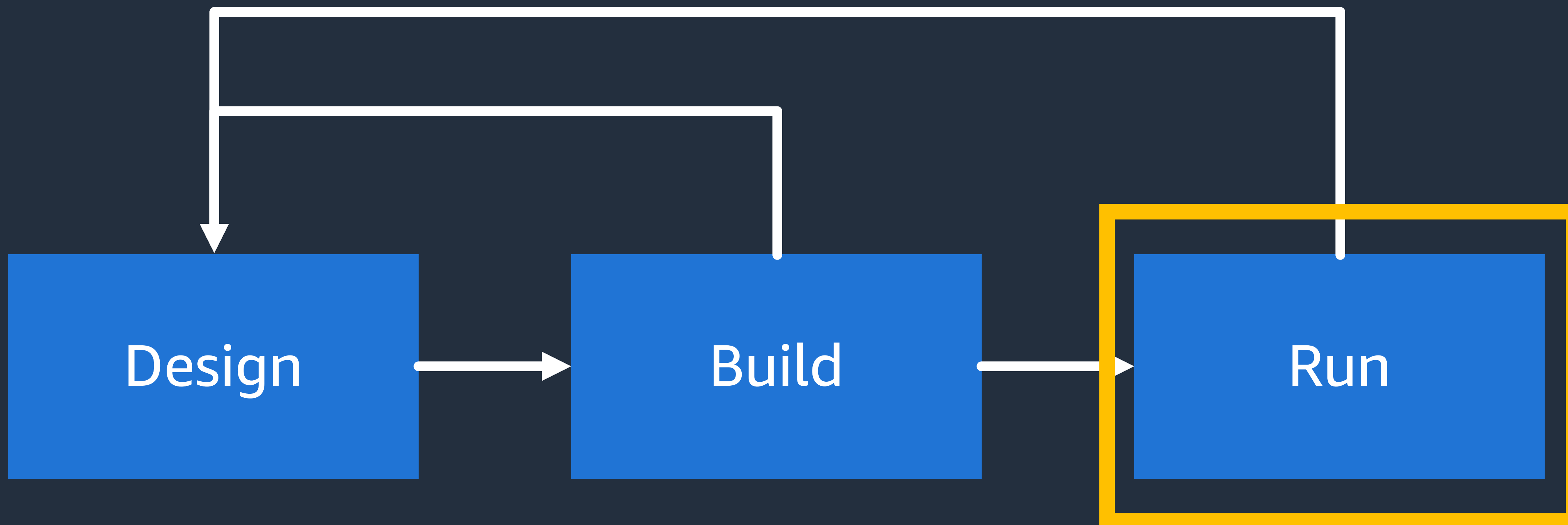
Figure A.2: Direct signal power spreading in range and Doppler versus integration time for measured NetRad phase noise

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

SEVERAL LONG
DAYS LATER

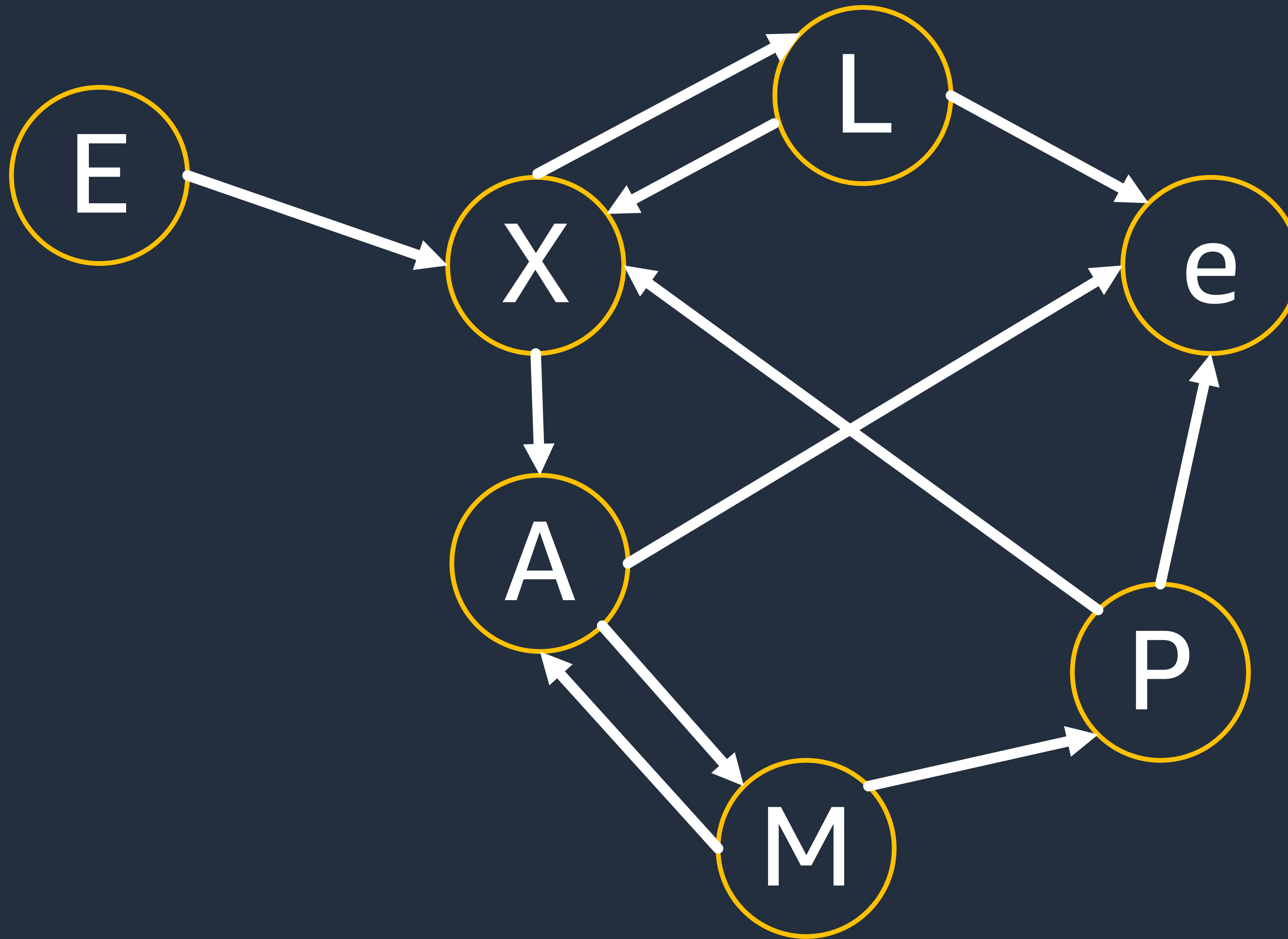
it’s just vibes, man.

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.



aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

ACE



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

- Alloy
- Promela + SPIN
- TLA+

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

CACM, 2015

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

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal

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 depend. So, before launching a service, we need to reach extremely high confidence that the core of the system is correct. We have found the standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, code reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that

© 2024, Amazon Web Services, Inc. o

Applying TLA+ to some of Amazon's more complex systems.

System	Components	Line Count (Excluding Comments)	Benefit
S3	Fault-tolerant, low-level network algorithm	804 PlusCal	Found two bugs, then others in proposed optimizations
	Background redistribution of data	645 PlusCal	Found one bug, then another in the first proposed fix
DynamoDB	Replication and group-membership system	939 TLA+	Found three bugs requir-ing traces of up to 35 steps
EBS	Volume management	102 PlusCal	Found three bugs
Internal distributed lock manager	Lock-free data structure	223 PlusCal	Improved confidence though failed to find a liveness bug, as liveness not checked
	Fault-tolerant replication-and-reconfiguration algorithm	318 TLA+	Found one bug and verified an aggressive optimization

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Applying TLA+ to some of Amazon's more complex systems.

System	Components	Line Count (Excluding Comments)	Benefit
S3	Fault-tolerant, low-level network algorithm	804 PlusCal	Found two bugs, then others in proposed optimizations
	Background redistribution of data	645 PlusCal	Found one bug, then another in the first proposed fix
DynamoDB	Replication and group-membership system	939 TLA+	Found three bugs requiring traces of up to 35 steps
EBS	Volume management	102 PlusCal	Found three bugs
Internal distributed lock manager	Lock-free data structure	223 PlusCal	Improved confidence though failed to find a liveness bug, as liveness not checked
	Fault-tolerant replication-and-reconfiguration algorithm	318 TLA+	Found one bug and verified an aggressive optimization

"Found three bugs"

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Applying TLA+ to some of Amazon's more complex systems.

System	Components	Line Count (Excluding Comments)	Benefit
S3	Fault-tolerant, low-level network algorithm	804 PlusCal	Found two bugs, then others in proposed optimizations
	Background redistribution of data	645 PlusCal	Found one bug, then another in the first proposed fix
DynamoDB	Replication and group-membership system	939 TLA+	Found three bugs requiring traces of up to 35 steps
EBS	Volume management	102 PlusCal	Found three bugs
Internal distributed lock manager	Lock-free data structure	223 PlusCal	Improved confidence though failed to find a liveness bug, as liveness not checked
	Fault-tolerant replication-and-reconfiguration algorithm	318 TLA+	Found one bug and verified an aggressive optimization

"…requiring traces of up to 35 steps"

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Applying TLA+ to some of Amazon's more complex systems.

System	Components	Line Count (Excluding Comments)	Benefit
S3	Fault-tolerant, low-level network algorithm	804 PlusCal	Found two bugs, then others in proposed optimizations
	Background redistribution of data	645 PlusCal	Found one bug, then another in the first proposed fix
DynamoDB	Replication and group-membership system	939 TLA+	Found three bugs requir-ing traces of up to 35 steps
EBS	Volume management	102 PlusCal	Found three bugs
Internal distributed lock manager	Lock-free data structure	223 PlusCal	Improved confidence though failed to find a liveness bug, as liveness not checked
	Fault-tolerant replication-and-reconfiguration algorithm	318 TLA+	Found one bug and verified an aggressive optimization

"… verified an aggressive optimization"

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Faster, safer, and cheaper.

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

"It would be well if engineering were less generally thought of, and even defined, as the art of constructing. In a certain important sense **it is rather the art of not constructing**; or, to define it rudely but not inaptly, it is the art of doing that well with one dollar, which any bungler can do with two after a fashion.”

Arthur Wellington

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

TLA+:

An Engineering Tool

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The Present

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

TLA+ continues to be used:

- Database services (e.g. Aurora, DynamoDB)
- Compute services (e.g. Lambda)
- Storage services (e.g. EBS)
- and many other places

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

NSDI’20

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

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

© 2024, Am

[We] used TLA+ in three ways:

- writing specifications of our protocols to check that we understand them deeply
- model checking specifications against correctness and liveness properties using the TLC model checker, and
- writing extensively commented TLA+ code to serve as the documentation of our distributed protocols.

While all three of these uses added value, TLA+'s role as [an] extremely precise format for protocol documentation was perhaps the most useful.

Brooker et al, *Millions of Tiny Databases,* NSDI'20

aws

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS has a broad automated reasoning practice.

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

OOPSLA'24

Cedar: A New Language for Expressive, Fast, Safe, and Analyzable Authorization

JOSEPH W. CUTLER*, University of Pennsylvania, USA
CRAIG DISSELKOEN, Amazon Web Services, USA
AARON ELINE, Amazon Web Services, USA
SHAOBO HE, Amazon Web Services, USA
KYLE HEADLEY*, Unaffiliated, USA
MICHAEL HICKS, Amazon Web Services, USA
KESHA HIETALA, Amazon Web Services, USA
ELEFTHERIOS IOANNIDIS*, University of Pennsylvania, USA
JOHN KASTNER, Amazon Web Services, USA
ANWAR MAMAT*, University of Maryland, USA
DARIN MCADAMS, Amazon Web Services, USA
MATT MCCUTCHEN*, Unaffiliated, USA
NEHA RUNGTA, Amazon Web Services, USA
EMINA TORLAK, Amazon Web Services, USA
ANDREW M. WELLS, Amazon Web Services, USA

Cedar is a new authorization policy language designed to be ergonomic, fast, safe, and analyzable. Rather than embed authorization logic in an application's code, developers can write that logic as Cedar policies and delegate access decisions to Cedar's evaluation engine. Cedar's simple and intuitive syntax supports common authorization use-cases with readable policies, naturally leveraging concepts from role-based, attribute-based,

© 2024, Amazon Web Servic

OOPSLA'23

Message Chains for Distributed System Verification

FEDERICO MORA, University of California, Berkeley, USA
ANKUSH DESAI, Amazon Web Services, USA
ELIZABETH POLGREEN, University of Edinburgh, UK
SANJIT A. SESHIA, University of California, Berkeley, USA

Verification of asynchronous distributed programs is challenging due to the need to reason about numerous control paths resulting from the myriad interleaving of messages and failures. In this paper, we propose an automated bookkeeping method based on *message chains*. Message chains reveal structure in asynchronous distributed system executions and can help programmers verify their systems at the message passing level of abstraction. To evaluate our contributions empirically we build a verification prototype for the P programming language that integrates message chains. We use it to verify 16 benchmarks from related work, one new benchmark that exemplifies the kinds of systems our method focuses on, and two industrial benchmarks. We find that message chains are able to simplify existing proofs and our prototype performs comparably to existing work in terms of runtime. We extend our work with support for specification mining and find that message chains provide enough structure to allow existing learning and program synthesis tools to automatically infer meaningful specifications using only execution examples.

© 2024, Amazon Web Se

CAV'22

A Billion SMT Queries a Day (Invited Paper)

Neha Rungta(✉)

Amazon Web Services, Seattle, USA
rungta@amazon.com

Abstract. Amazon Web Services (AWS) is a cloud computing services provider that has made significant investments in applying formal methods to proving correctness of its internal systems and providing assurance of correctness to their end-users. In this paper, we focus on how we built abstractions and eliminated specifications to scale a verification engine for AWS access policies, ZELKOVA, to be usable by all AWS users. We present milestones from our journey from a thousand SMT invocations daily to an unprecedented billion SMT calls in a span of five years. In this paper, we talk about how the cloud is enabling application of formal methods, key insights into what made this scale of a billion SMT queries daily possible, and present some open scientific challenges for the formal methods community.

aws

© 2024, Amazon Web Services, Inc. or its affi

SOSP'21

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" we mean a pragmatic approach to verifying the correctness of a production storage node that is under ongoing feature development by a full-time engineering team. We do not aim to achieve full formal verification, but instead emphasize automation, usability, and the ability to continually ensure correctness as both software and its specification evolve over time. Our approach decomposes correctness into independent properties, each checked by the most appropriate tool, and develops executable reference models as specifications to be checked against the implementation. Our work has prevented 16 issues from reaching production, including subtle crash consistency and concurrency problems, and has been extended by non-formal-methods experts to check new features and properties as ShardStore has evolved.

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

1 Introduction

Amazon S3 is a cloud object storage service that offers customers elastic storage with extremely high durability and availability. At the core of S3 are *storage node* servers that persist object data on hard disks. These storage nodes are key-value stores that hold *shards* of object data, replicated by the control plane across multiple nodes for durability. S3 is building a new key-value storage node called *ShardStore* that is being gradually deployed within our current service.

Production storage systems such as ShardStore are notoriously difficult to get right [25]. To achieve high performance, ShardStore combines a soft-updates crash consistency protocol [16], extensive concurrency, append-only IO and garbage

© 2024, Amazon Web Services, Inc. or its affil

Keeping this personal…

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Abstract

Turmoil

Code

Design

Kani

Concrete

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Find bugs earlier, move faster.

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Optimize designs and protocols without risk.

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Crisply communicate exact reasoning.

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Exactly state system properties.

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Formal methods are just good engineering practice.

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The Future

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Prediction 1:

Systems will continue to become more complex.

Certainty level: high

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Prediction 2:

Systems will continue to become more critical.

Certainty level: high

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Prediction 3:

Cost, efficiency, sustainability, and productivity will become more important.

Certainty level: medium

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Prediction 4:
AI will be a big deal

Certainty level: medium

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The Future, More Concretely

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Quantitative Understanding of System Behavior

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Beyond *Safety* and *Liveness:*

- Failure probabilities.
- Latency and throughput distributions.
- System performance under different workloads.
- Automatic optimization of system designs.

aws

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Faster Exploration of the Design Space

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Example: *2f+1* vs *f+1* replication designs

- 30 years of debate and data.
- Still no clear winner, because its failure model and workload dependent.
- But how?
- Given workload properties, matching design to workload should be trivial.

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Example: *OCC* vs *locking*

- 45 years of debate and data!
- Still no clear winner, because its failure model and workload dependent.
- But how?
- Given workload properties, matching design to workload should be trivial.

Kung and Robinson, *On Optimistic Methods for Concurrency Control*, ACM TOCS, 1981
Kung and Papadimitriou, *An Optimality Theory of Concurrency Control for Databases*, ACM TOCS, 1979

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Reaching Working Engineers

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

How?

- I don’t know.
- Should tools be narrower and more specific?
- Is packaging and UI holding us back?
- Are we speaking enough to decision makers about moving faster and with less risk?

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The Code vs Model Gap

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The Model Checking vs Proof Gap

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

Marc Brooker

mbrooker@amazon.com

https://brooker.co.za/blog/

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.