Modeling with TLA+/PlusCal: Motivation

Murat Demirbas

Game plan

Why should you model?

TLA+ specification language

TLA+ use in the industry

My experiences with TLA+

TLA+ workshop objectives

Why should you model?

Why should you model?

There is a spectrum of rigor needed for software we build. For critical parts of the system, modeling gives an effective ROI to the working engineer under ridiculous time pressure

- saves times rather than wasting time
- provides a thinking tool for handling complexity
- gives effective coverage of corner cases

Writing is nature’s way of letting you know how sloppy your thinking is. –Guindon

Math is nature’s way of letting you know how sloppy your writing is. –Lamport

Modeling benefits

Modeling your designs enable you to exhaustively test and debug them against

- corner cases
- failed assumptions
- concurrency
- failures

You can catch errors at design time and fix them before they become costly
https://muratbuffalo.blogspot.com/2018/10/debugging-designs-with-tla.html

Distributed systems are hard!

Fallacies of distributed systems:

- One hop is faster than two hops
- No hop is faster than one hop
- Interactions between multiple protocols seem to be safe

What you thought as an atomic block of execution wasn't, because another process got scheduled to execute and changed the system state in a way you didn't anticipate

TLA+

TLA+ specification language

Formal language for describing and reasoning about distributed and concurrent systems

Based on mathematical logic, set theory, & temporal logic

Supported by a tool set (e.g., TLC model checker)

TLA+ is motivated by the practical problem of reasoning about real algorithms – L. Lamport

TLA+ history

Lamport presented Temporal Logic of Actions in the late ’80s

In 1993, Lamport introduced TLA+ formalism built around TLA

In 1999, we got TLC, an explicit state model checker for TLA+ specs

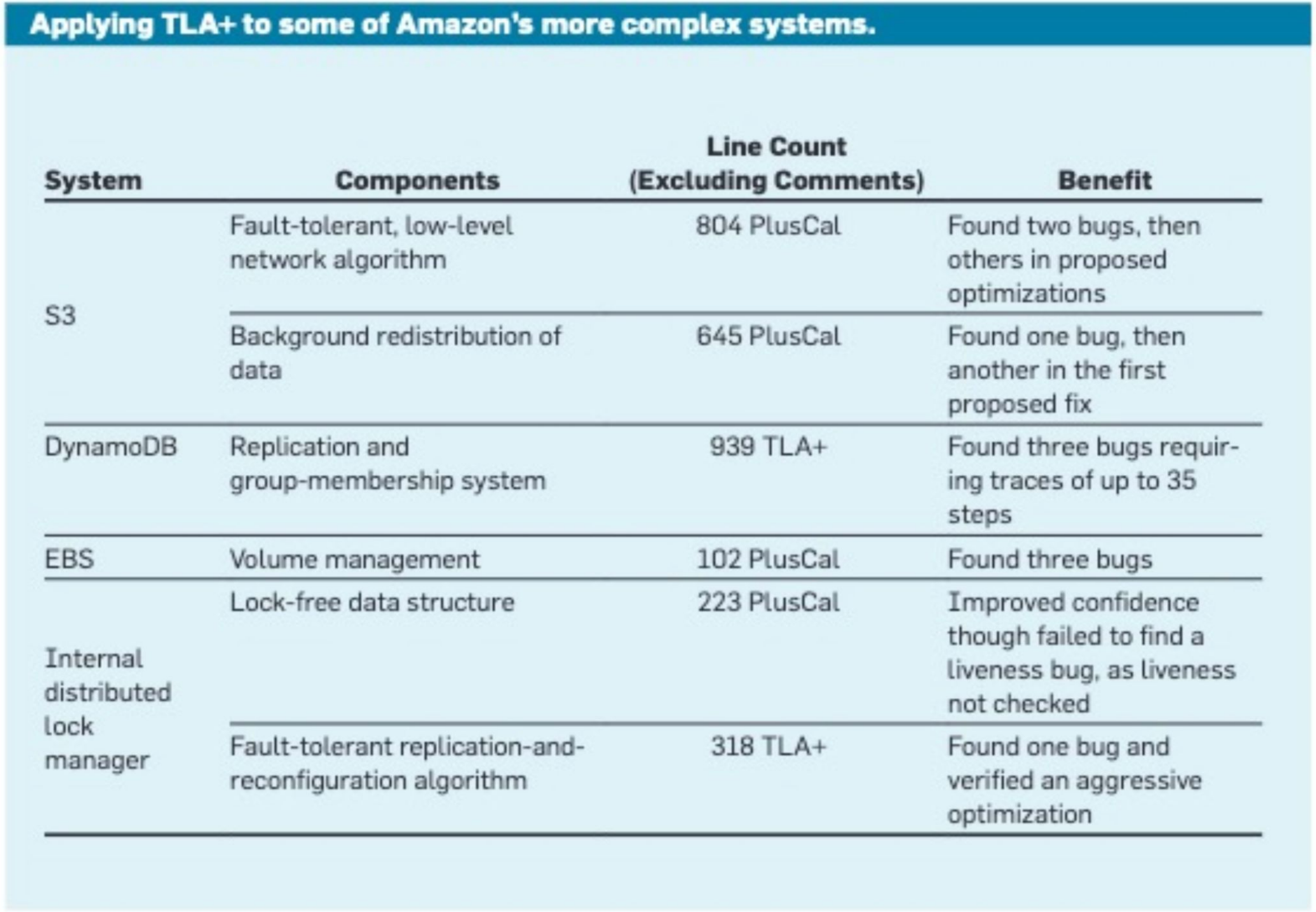
In 2009, Lamport introduced PlusCal, a pseudocode syntax that is automatically translated into TLA+

TLA+ use in the industry

How Amazon Web Services Uses Formal Methods (CACM 2015)

> Before launching any complex service, we need to reach extremely high confidence that the core of the system is correct. We have found that the standard verification techniques in industry (deep design reviews, code reviews, static code analysis, stress testing, fault-injection testing, etc.) are **necessary but not sufficient**.
>
> Human fallibility means that some of the more subtle, dangerous bugs turn out to be **errors in design**; the code faithfully implements the intended design, but the design fails to correctly handle a particular ‘rare’ scenario. We have found that testing the code is inadequate as a method to find subtle errors in design.

If you don’t like formal methods call it "exhaustively-testable design".



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

TLA+ use at MongoDB

Extreme modeling in practice (VLDB'20)

Fault-Tolerant Replication with Pull-Based Consensus in MongoDB (NSDI'21)

Design and Analysis of a Logless Dynamic Reconfiguration Protocol (OPODIS'21)

TLA+ use at other companies

Microsoft, Oracle, LinkedIn, DataDog, Nike, Intel, Confluent, PingCap among others

https://conf.tlapl.us/2024/

More than 700 repos come when searching TLA+ on github:
https://github.com/search?q=tla%2B&type=repositories

My experiences with TLA+

How it started; how it is going?

https://muratbuffalo.blogspot.com/2014/08/using-tla-for-teaching-distributed.html

In 2018, I used it at Azure Cosmos DB as part of my sabbatical

In 2021-24, I used it at AWS for different services and subcomponents

In 2024, I am using it at MongoDB Research, and working with engineers at MongoDB

TLA+ use pattern

Organic and grassroots rather than systematic and by prescription

- engineer faced with tricky problem learns TLA+ and applies it

Sporadic use

- Less than a handful of opportunities a year; make them count
- *Luck favors the prepared mind! –Pasteur*

TLA+ is easy to pick up

Formal methods have a reputation of requiring a huge amount of training and effort to verify a tiny piece of relatively straightforward code, so people think ROI is not justified except for most critical systems.

Our experience with TLA+ has shown that perception to be quite wrong.

Engineers from entry level to senior have been able to learn TLA+ from scratch and get useful results in a week. . . with little help and self-training.

TLA+ workshop objectives

What should you get out of learning TLA+

Mental models

- specification versus implementation separation
- *I used this refinement concept to write documents for executives* – C. Newcombe

Modeling and design skills

- Learning to model protocols/systems
- Learning the art of abstraction
- Syntax and PL style less important

How we’ll go about this?

State-transition computation model

Basic concepts in TLA+/PlusCal

Hands on examples:

- Toy puzzles
- Two-phase commit
- Snapshot-isolated database
- Reading larger specs

Resources

Reference material

https://learntla.com

http://lamport.azurewebsites.net/video/videos.html

https://lamport.azurewebsites.net/tla/book.html

https://github.com/tlaplus/Examples

https://groups.google.com/g/tlaplus

https://github.com/muratdem/TLA-seminar