Why Amazon Chose TLA$^+$

Chris Newcombe

Amazon, Inc.

Abstract. Since 2011, engineers at Amazon have been using TLA$^+$ to help solve difficult design problems in critical systems. This paper describes the reasons why we chose TLA$^+$ instead of other methods, and areas in which we would welcome further progress.

1 Introduction

Why Amazon is using formal methods. Amazon builds many sophisticated distributed systems that store and process data on behalf of our customers. In order to safeguard that data we rely on the correctness of an ever-growing set of algorithms for replication, consistency, concurrency-control, fault tolerance, auto-scaling, and other coordination activities. Achieving correctness in these areas is a major engineering challenge as these algorithms interact in complex ways in order to achieve high-availability on cost-efficient infrastructure whilst also coping with relentless rapid business-growth[1]. We adopted formal methods to help solve this problem.

Usage so far. As of February 2014, we have used TLA$^+$ on 10 large complex real-world systems. In every case TLA$^+$ has added significant value, either preventing subtle serious bugs from reaching production, or giving us enough understanding and confidence to make aggressive performance optimizations without sacrificing correctness. Executive management are now proactively encouraging teams to write TLA$^+$ specs for new features and other significant design changes. In annual planning, managers are now allocating engineering time to use TLA$^+$.

We lack space here to explain the TLA$^+$ language or show examples of our specifications. We refer readers to [18] for a tutorial, and [26] for an example of a TLA$^+$ specification from industry that is similar in size and complexity to some of the larger specifications at Amazon.

What we wanted in a formal method. Our requirements may be roughly grouped as follows. These are not orthogonal dimensions, as business requirements and engineering tradeoffs are rarely crisp. However, these do represent the most important characteristics required for a method to be successful in our industry segment.

1. *Handle very large, complex or subtle problems.* Our main challenge is complexity in concurrent and distributed systems. As shown in Fig. 1, we often need to verify the interactions between algorithms, not just individual algorithms in isolation.

[1] As an example of such growth; in 2006 we launched S3, our Simple Storage Service. In the six years after launch, S3 grew to store 1 trillion objects [6]. Less than one year later it had grown to 2 trillion objects, and was regularly handling 1.1 million requests per second [7].

Y. Ait Ameur and K.-D. Schewe (Eds.): ABZ 2014, LNCS 8477, pp. 25–39, 2014.

System	Components	Line count	Benefit
S3	Fault-tolerant low-level network algorithm	804 PlusCal	Found 2 design bugs. Found further design bugs in proposed optimizations.
	Background redistribution of data	645 PlusCal	Found 1 design bug, and found a bug in the first proposed fix.
DynamoDB	Replication and group-membership systems (which tightly interact)	939 TLA$^+$	Found 3 design bugs, some requiring traces of 35 steps.
EBS	Volume management	102 PlusCal	Found 3 design bugs.
EC2	Change to fault-tolerant replication, including incremental deployment to existing system, with zero downtime	250 TLA$^+$ 460 TLA$^+$ 200 TLA$^+$	Found 1 design bug.
Internal distributed lock manager	Lock-free data structure	223 PlusCal	Improved confidence. Failed to find a liveness bug as we did not check liveness.
	Fault tolerant replication and reconfiguration algorithm	318 TLA$^+$	Found 1 design bug. Verified an aggressive optimization.

Fig. 1. Examples of applying TLA$^+$ to some of our more complex systems

Also, many published algorithms make simplifying assumptions about the operating environment (e.g. fail-stop processes, sequentially consistent memory model) that are not true for real systems, so we often need to modify algorithms to cope with the weaker properties of a more challenging environment. For these reasons we need expressive languages and powerful tools that are equipped to handle high complexity in these problem domains.

2. *Minimize cognitive burden.* Engineers already have their hands full with the complexity of the problem they are trying to solve. To help them rather than hinder, a new engineering method must be relatively easy to learn and easy to apply. We need a method that avoids esoteric concepts, and that has clean simple syntax and semantics. We also need tools that are easy to use. In addition, a method intended for specification and verification of designs must be easy to remember. Engineers might use a design-level method for a few weeks at the start of a project, and then not use it for many months while they implement, test and launch the system. During the long implementation phase, engineers will likely forget any subtle details of a design-level method, which would then cause frustration during the next design phase. (We suspect that verification researchers experience a very different usage pattern of tools, in which this problem might not be apparent.)
3. *High return on investment.* We would like a single method that is effective for the wide-range of problems that we face. We need a method that quickly gives useful results, with minimal training and reasonable effort. Ideally we want a method that also improves time-to-market in addition to ensuring correctness.