Experience of software engineers using TLA+, PlusCal and TLC

Chris Newcombe

Principal Engineer, Amazon Web Services

Amazon Web Services

- 30+ services [1]; compute, storage, db, queuing, …
- High Scale:
  - S3 (Simple Storage Service) launched in 2006
    now holds over 1 trillion objects
    growing by 3.5 billion per day [2]
- Sophisticated features
  - DynamoDB: strongly-consistent, highly available, scalable, predictable performance, multi-tenant
- "NetFlix is now 100% cloud" (on AWS)[3]

Most services face the same problem

Interacting subtle algorithms

- complex business logic, rules-engines, evolving schemas, …
- interactions with other systems
- dynamic group-membership; live re-sharding / auto-scaling
- concurrency-control
- consistency, workflow
- replication, fail-over
- actions by human operators

Why don't engineers specify their systems?

- Deep ignorance/confusion about benefit, terminology, method, tools
- Deep skepticism of snake-oil

  "The weight of evidence for an extraordinary claim must be proportioned to its strangeness." - Laplace
- No time to look at anything that is not an immediate productivity gain
- Failing to distinguish design flaw vs. implementation bug
- Skills complacence : 'I only need java/python/…'
- Resignation : 'Bugs are inevitable anyway…'
- "Blog-comment culture" - more fun to broadcast/defend an immediate personal opinion than to consider changing it

Moving from 'intuitive correctness' to precision

From *"Java Concurrency In Practice"*, page 17

"Correctness means a class *conforms to its specification*. A good specification defines *invariants* constraining an objects state, and *postconditions* describing the effects of its operations.

Since we often don't write adequate specifications for our classes, how can we possibly know they are correct? We can't, but that doesn't stop us from using them anyway, once we've convinced ourselves that "the code works".

This "code confidence" is about as close as many of us get to correctness, so let's just assume that single-threaded correctness is something that "we know it when we see it". Having optimistically defined correctness as something that can be recognized, we can now define thread safety…"

Terminology difficulties

- Allergic to the word “formal”
- Overloaded terminology:
  “So, ‘specification’ means inputs and outputs, right?”

- Current best solution: call it

“Exhaustively-testable pseudo-code”

The most persuasive example so far

A famous system design (2001)

Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications

Ion Stoica*, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan†
MIT Laboratory for Computer Science
chord@lcs.mit.edu
http://pdos.lcs.mit.edu/chord/

Abstract

A fundamental problem that confronts peer-to-peer applications is to efficiently locate the node that stores a particular data item. This paper presents *Chord*, a distributed lookup protocol that addresses this problem. Chord provides support for just one operation: given a key, it maps the key onto a node. Data location can be easily and involves relatively little movement of keys when nodes join and leave the system.

Previous work on consistent hashing assumed that nodes were aware of most other nodes in the system, making it impractical to scale to large number of nodes. In contrast, each Chord node needs "routing" information about only a few other nodes. Because the

Ring of nodes

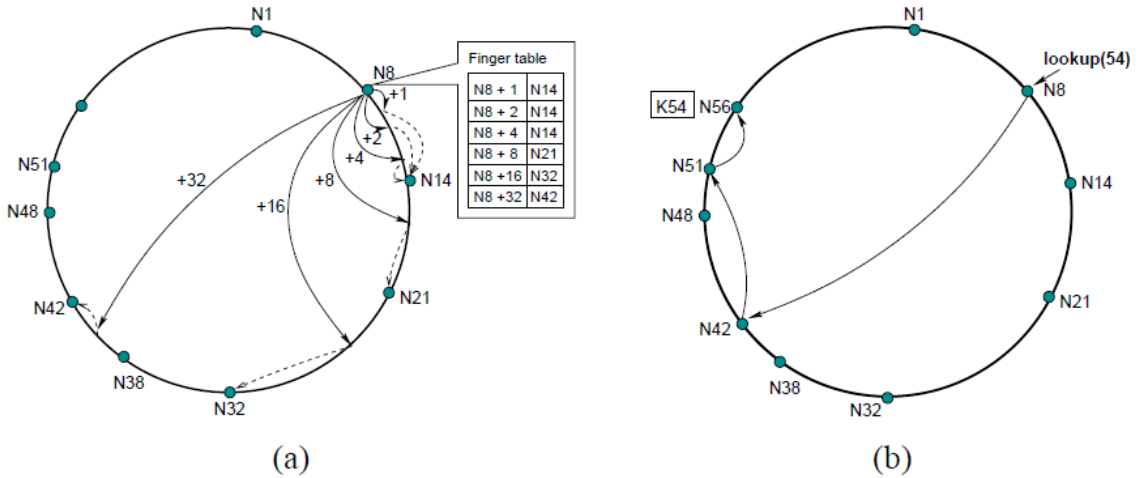


Fig. 4. (a) The finger table entries for node 8. (b) The path a query for key 54 starting at node 8, using the algorithm in Figure 5.

"Three features that distinguish Chord from many other peer-to-peer lookup protocols are its simplicity, ***provable correctness,*** *and provable performance."*

Original pseudo-code for Chord

```
// n joins the network
n.join(n')
  s = n'.find_predecessor(n);
  do
    p = s;
    s = p.successor;
  until n ∈ (p, s]
  successor = s;
  predecessor = p;
  p.notify(n); // tell p to update its state
  s.notify(n); // tell s to update its state
  boot_strap(s);

n.notify(n')
  if (n' ∈ (n, successor))
    finger[1].node = successor = n';
    boot_strap(n');
  if (n' ∈ (predecessor, n))
    predecessor = n';
    boot_strap(n');
```

```
// let node n send queries to fill in its own tables
n.boot_strap(n')
  for i = 1 to m
    p = n'.find_successor(finger[i].start);
    do
      s = p;
      p = p.predecessor;
    until (p < finger[i].start)
    finger[i].start = s;

// verify n's immediate pred/succ
// called periodically
n.stabilize()
  x = predecessor;
  x = x.successor;
  if (x ∈ (predecessor, n)
    predecessor = x;
  x = successor;
  x = x.predecessor;
  if (x ∈ (n, successor))
    finger[1].node = successor = x;
```

Figure 6: Pseudocode for concurrent join. Predecessor functions that parallel their successor equivalents are omitted

Nine years later, another paper …

Lightweight Modeling of Network Protocols in Alloy

Pamela Zave
AT&T Laboratories—Research
pamela@research.att.com

ABSTRACT

"Lightweight modeling" is a design technique in which small, abstract formal models are explored and verified with a push-button tool. Although Alloy is not the obvious choice for process. It makes sense to eliminate conceptual errors before tackling all the additional details that arise in implementation.

Alloy is a formal modeling language analyzed by the

Found **8 major defects** in the Chord ring-membership protocol

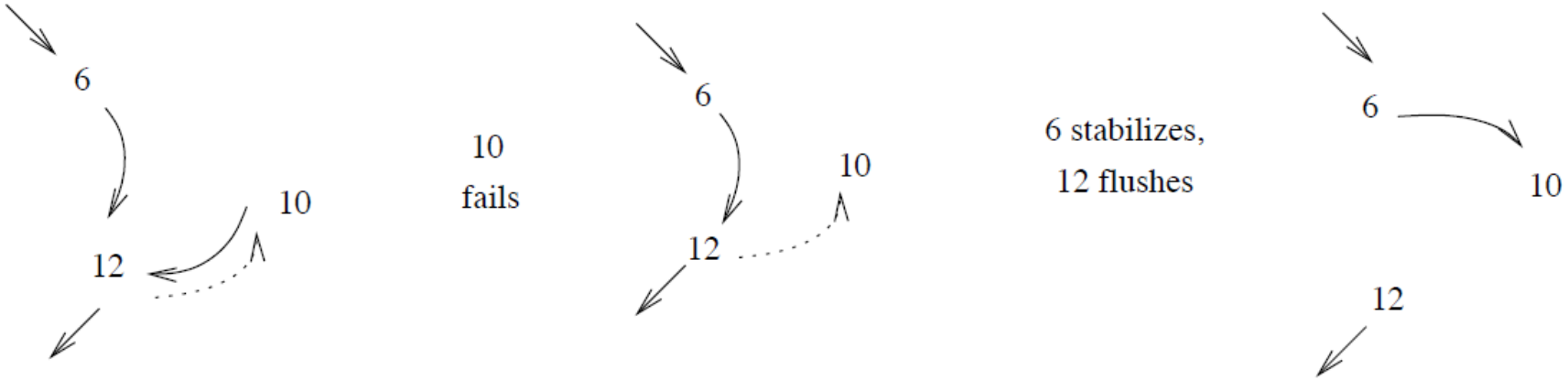


Figure 6: Three stages creating a counterexample in which the cycle is lost.

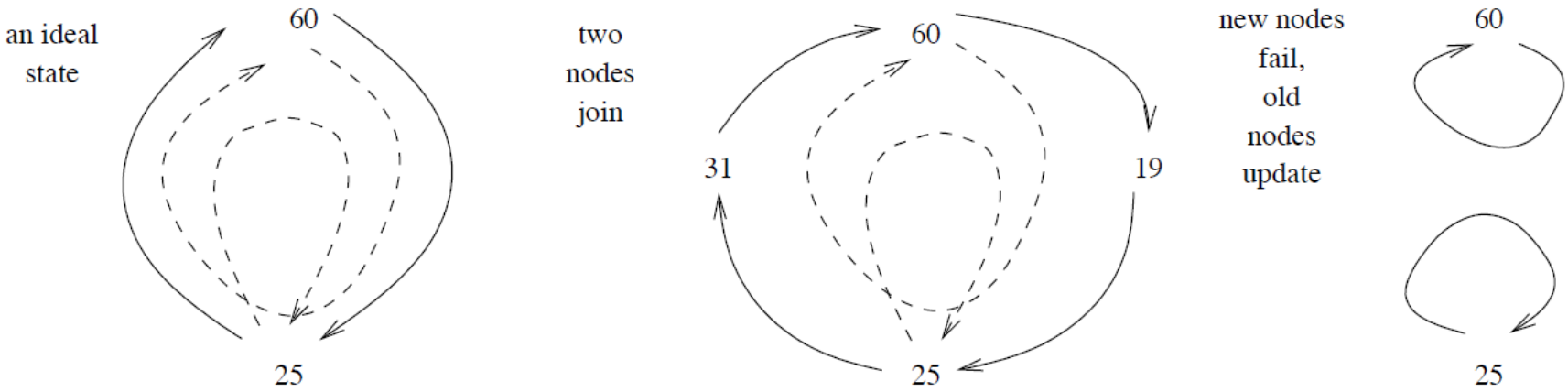


Figure 8: Three stages creating multiple cycles.

Initial conceptual difficulties when engineers are learning TLA+

- Single global state -- “But where is the concurrency?”
- Use of non-determinism
- Interleaving vs. non-interleaving specification
  - “Can ‘either … or …’ allow 2 or more clauses to happen?”
- Unfamiliar kinds of expressions
  - ‘programming’ via recursive operators, instead of using set comprehensions
- Idioms? E.g. for abstracting network
- “What is the difference between [->] and [|->] ?”
- Actions that inadvertently access the state of other processes (e.g. to get data that should have been sent as part of a message payload)

More advanced conceptual difficulties

- Liveness is a property of an infinite behavior, so what does TLC's "liveness checking" actually tell you?
- How can we easily tell if it safe to use a symmetry set for model-checking?

Suggestions to help increase adoption

- Syntax highlighting for PlusCal
- A wiki-style cookbook of specification idioms
- A wiki-style cookbook of tricks for using TLC
- More real-world example specifications and, ideally, their inductive invariants.

Use at Amazon so far

1. "Tim" Fault-tolerant replication (interacting distributed algorithms)
2. "Fan" Fault-tolerant network protocol
3. "Mike" Lock-free data structure
4. "Chris" Concurrent algorithm, involving third-party components
5. "Sam" Fault-tolerant replication (interacting distributed algorithms)
6. "Cahill" Isolation of database transactions

"Tim"
interacting distributed algorithms

- Fault-tolerant replication, group-membership
- Principal Engineer, one of the best at Amazon
- Algorithm was already designed, and coded in java, with months of thorough design analysis (30+ pages of informal prose proofs)
- Approx. 6 weeks, while doing other work
- Pure TLA+. We might have considered PlusCal, but hadn't tried it yet. Not sure how PlusCal works with multiple modules.
- 939 non-comment, non-blank lines split across 7 modules
- 529 pure comment lines
- 11 invariants. No liveness properties.
- Distributed TLC, on cluster of 10 "cc1.4xl" nodes in EC2
- **TLC found 3 important bugs**. Counter-examples had 35+ steps.

“Fan” Fault-tolerant protocol

- Junior engineer. Very keen to learn, after a presentation on TLA.
- PhD but not in computer science (might be significant)
- 6 weeks. Independent after 3 weeks of approx. 1hr/day help
- First wrote ~700 line TLA+ spec.
  - 93 lines were UNCHANGED statements, very tedious to maintain
- Converted to PlusCal, continued adding to specification
- 995 non-blank, non-comment lines (1728 lines of translated TLA+)
- 4 kinds of processes
- No procedures, just macros
- TLC on single cc2.8xl EC2 instance, and distributed on 10 x cc1.4xl
- Views help, symmetry does not
- So far has found 2 invariant failures, one of which they expected and intend to live with (low probability, and the system ‘fails safe’).

“Mike”
Lock-free algorithm

- Optimization for critical in-memory data-structure
- Mid-level engineer
- Learned & used PlusCal on his own, after a presentation on a paper by Lampson[1]
- Reportedly missed a liveness bug.
  No-one at Amazon has checked liveness properties yet.
  Liveness is less important than safety, but still important.
- 223 non-blank, non-comment lines (excluding translation)
- 9 invariants, as asserts
- 5 macros, 4 procedures, 2 types of process (each 1..2 disjunctions)
- 5..6 labels per procedure

“Chris”: Concurrent algorithm, using third-party components

- Using PlusCal as an ‘analysis/design sketchpad’
- 3400 lines, 80% prose comments
- Wrote prose, then refined sections into TLA+ definitions and PlusCal, to make it precise
- Never got as far as model-checking
- Still major benefits
  - Helped me understand a messy problem
  - Helped me to hugely simplify the problem
- Speculate that I want “Literate specifications”

Literate Specifications?

Donald Knuth wrote[1]

"Some of my major programs … could not have been written with any other methodology that I've ever heard of. The complexity was simply too daunting for my limited brain to handle; without literate programming, the whole enterprise would have flopped miserably."

…

"Literate programming is what you need to rise above the ordinary level of achievement."

“Sam”
interacting distributed algorithms

- Another fault-tolerant replication and group-membership problem
- Got a brief design sketch from a senior engineer, in somewhat hand-wavy prose
- Turning it into a PlusCal ‘shell’, to systematically uncover ambiguity
- 649 lines including prose (but barely started)
- Again, “literate specifications” might help

“Cahill”
Isolation of database transactions

- Spec. for a published algorithm, “Serializable Snapshot Isolation”
- Practical algorithm; now used in postgresql
- Not used at Amazon; spec was written as an example for a conference talk. But I was doing Amazon work in that area at the time.
- 1600 lines of TLA+, inc lots of comments.
- Our first use of Distributed TLC
- Spec available at http://hpts.ws/agenda.html
  (the ‘source bundle’ for the talk called ‘Debugging Designs’)

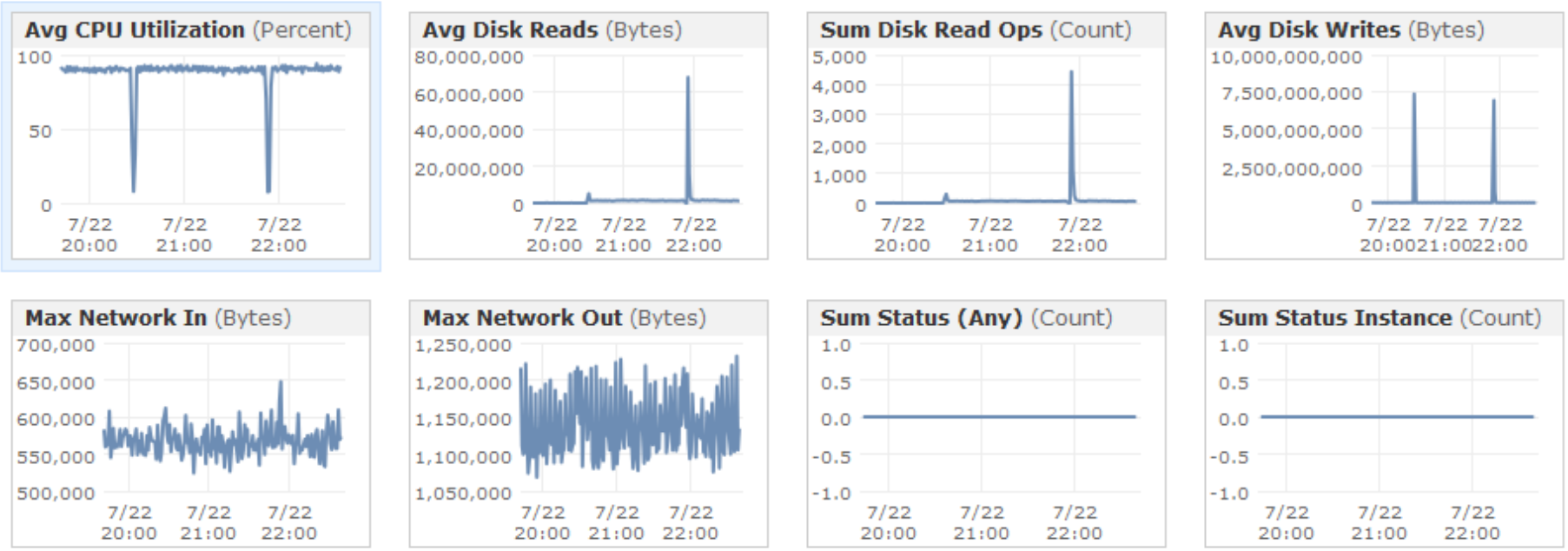
Future challenges

- Refinement
- Model-checking real-time specifications.
- Proofs … perhaps one day
- Adoption by other divisions

TLC performance

- Disk-IO bound for all practical specs we've seen, including distributed TLC.
- Huge improvement using new EC2 "hi1.4xlarge"
  - 16 cores
  - 60.5 GB RAM
  - 120,000 x 4KB random read IOPS
    + 85,000 x 4KB random write IOPS
    across 2 * 1TB SSD volumes
  - 10 Gigabit Ethernet, with cluster placement groups
- May help further to have an option to disable checkpoints

CloudWatch metrics: Graphs are for 1 instance with detailed monitoring enabled. Times are displayed in UTC.



Support Material

Specs for Cahill's algorithm

Download link: http://hpts.ws/sessions/amazonbundle.tar.gz

TLA+

serializableSnapshotIsolation.tla

textbookSnapshotIsolation.tla

Alloy

serializableSnapshotIsolation HPTS.als

textbookSnapshotIsolation HPTS.als

An interesting execution

A Read-Only Transaction Anomaly Under Snapshot Isolation

By Alan Fekete, Elizabeth O'Neil, and Patrick O'Neil
fekete@it.usyd.edu.au, {eoneil, poneil}@cs.umb.edu
Research of all authors was supported by NSF Grant IRI 97-11374.

Abstract. Snapshot Isolation (SI), is a multi-version concurrency control algorithm introduced in [BBGMO095] and later implemented by Oracle. SI avoids many concurrency errors, and it never delays read-only transactions. However

The interval in time from the start to the commit of a transaction, represented [Start(T_i), Commit(T_i)], is called its *transactional lifetime*. We say two transactions T_1 and T_2 are *concurrent* if their transactional

The example from the paper:

R2(X0,0) R2(Y0,0) R1(Y0,0) W1(Y1,20) C1 R3(X0,0) R3(Y1,20) C3 W2(X2,-11) C2

The simplest example found by TLC (note: 'begin' can be a separate operation)

R1(X0) W1(Y1) W2(X2) C2 B3 C1 R3(X2) R3(Y0) C3