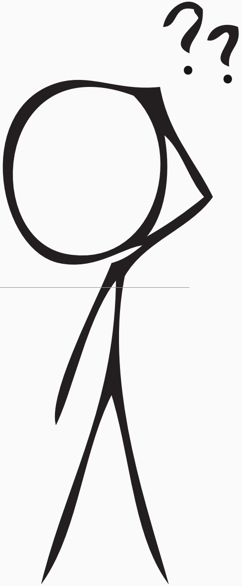
VERIFICATION OF CONCURRENT AND DISTRIBUTED SYSTEMS



Nickolai Novik

March 25, 2018

http://github.com/jettify

ABOUT ME

- **[Software Engineer]** DataRobot
- **[Github]** http://github.com/jettify
- **[Twitter]** https://twitter.com/isinf
- **[aio-libs]** https://github.com/aio-libs
- **[Projects]** *aiomonitor, aiohttp-debugtoolbar, aiobotocore, aiomysql, aioodbc, aiohttp-admin, aiorwlock, aiozipkin, etc*

AGENDA

1. Problem Statement. Motivational Example
2. Model Checking
3. TLA+ Basics
4. Multithreading Queue Spec
5. Aiorwlock Spec
6. Conclusions

POLL

How many of you heard of formal methods?

- I used one on of: Coq, Isable, TLA+, Alloy, Spin.
- I heard about it, but never used.
- I think formal methods are kinda cool.

PROBLEM STATEMENT. MOTIVATIONAL EXAMPLE

BRAVE NEW WORLD OF (MICROSERVISES) DISTRIBUTED SYSTEMS

How to be confidant that critical software works correctly?

1. Processor speed saturated, parallel execution is an answer
2. Concurrent/Parallel program is often requirement
3. Program complexity only raising
4. Simplified (monolith) application out of fashion
5. Due to micro services, everyone should be distributed systems expert

QA APPROACHES

Most of industry uses following techniques for quality assurance:

1. Design review
2. Static code analysis
3. Unit/Integration/Functional testing
4. Code coverage
5. Code review
6. Stress testing
7. Fault-injection testing [1]

DISTRIBUTED ALGORITHMS EXTREMELY HARD

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

Ion Stoica†, Robert Morris‡, David Liben-Nowell‡, David R. Karger‡, M. Frans Kaashoek‡, Frank Dabek‡, Hari Balakrishnan‡

Abstract—
A fundamental problem that confronts peer-to-peer applications is the efficient location of the node that stores a desired 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 implemented on top of Chord by associating a key with each data item, and storing the key/data pair at the node to which the key maps. Chord adapts efficiently as nodes join and leave the system, and can answer queries even if the system is continuously changing. Results from theoretical analysis and simulations show that Chord is scalable: communication cost and the state maintained by each node scale logarithmically with the number of Chord nodes.

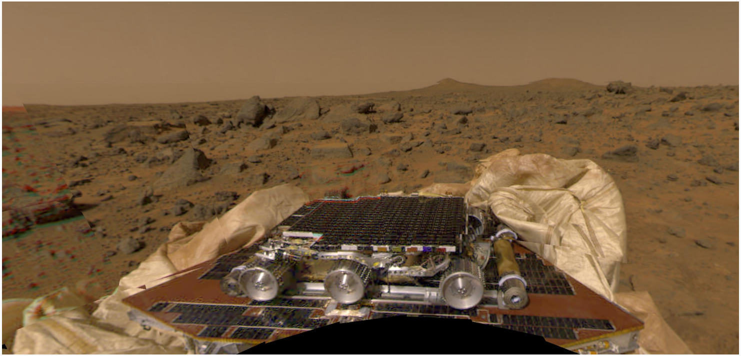
tem.

A Chord node requires information about $O(\log N)$ other nodes for *efficient* routing, but performance degrades gracefully when that information is out of date. This is important in practice because nodes will join and leave arbitrarily, and consistency of even $O(\log N)$ state may be hard to maintain. Only one piece of information per node need be correct in order for Chord to guarantee correct (though possibly slow) routing of queries; Chord has a simple algorithm for maintaining this information in a dynamic environment.

Chord

is popular algorithm for P2P systems, paper published in 2001 by strong team of MIT researchers, 10 years later bug found in specification [10, 11]. Paper won best paper award.

SOMETIMES COST OF ERROR IS VERY HIGH



Mars Pathfinder

rover, the mission was jeopardised by a concurrent software bug in the lander. [9]

SOMETIMES COST OF ERROR IS VERY HIGH 2

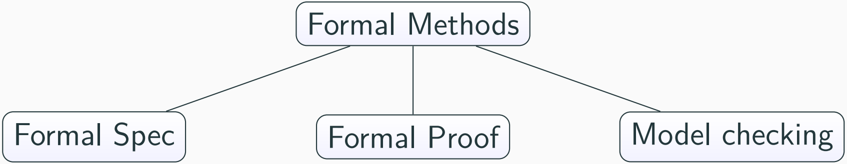
Therac-25

radiation therapy machine, because of concurrent programming errors, it sometimes gave its patients radiation doses that were hundreds of times greater than normal [2]

MODEL CHECKING

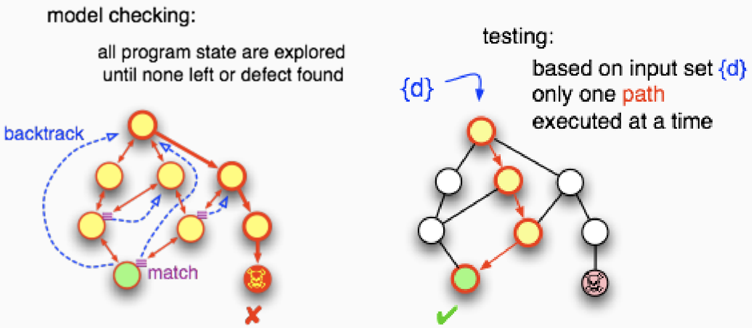
FORMAL METHODS

Formal Methods - techniques and tools based on mathematics and formal logic [7].

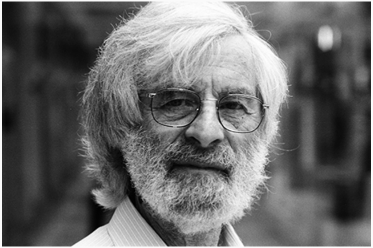


MODEL CHECKING

Model checking is a technique for automatically verifying correctness properties of finite-state systems.



TLA+ – TEMPORAL LOGIC OF ACTIONS



TLA+ language developed by **Leslie Lamport**. It is used to design, model, document, and verify concurrent systems, has been described as exhaustively-testable pseudocode and blueprints for software systems.

TLA+ DESCRIPTION

TLA+ is **specifications** language, it is precise written description of what a system suppose to do. **TLA Toolbox** distribution of TLA related tools. **TLC** - model checker tool to validate invariants stated in spec. **TLAPS** - mechanical proof checker.

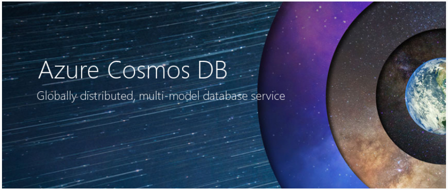
Released	April 23, 1999; 18 years ago
Syntax	math notation, similar to LaTeX
Implem.	Java
License	MIT
IDE	TLA Toolbox (Eclipse based) [5]

TLA+ INDUSTRY USAGE: AWS

TLA+ helped to find design bugs in **S3, Dynamo, EBS, EC2**, etc, some requiring traces of 35 steps. [8]

TLA+ INDUSTRY USAGE: MICROSOFT

MS used TLA+ to define consistency protocol for CosmosDB and memory allocator for XBox [6]



TLA+ INDUSTRY USAGE: OPENS SOURCE

Number open source projects use TLA+ to verify complex algorithms:

- **Elastic** – data replication protocol [3]
- **Mongodb** – data replication protocol [12]
- **Hadoop/YARN** – registry of long lived processes [4]

TLA+ BASICS

TLA+ HELLO WORLDS

```
---------------------- MODULE HourClock ----------------------
EXTENDS Naturals
VARIABLE hr
HCini  ==  hr \in (1 .. 12)
HCnxt  ==  hr' = IF hr # 12 THEN hr + 1 ELSE 1
HC  ==  HCini /\ [][HCnxt]_hr
--------------------------------------------------------------
THEOREM  HC => []HCini
==============================================================
```

MODULE *HourClock*

EXTENDS *Naturals*

VARIABLE *hr*

$HCini \triangleq hr \in (1 \,..\, 12)$

$HCnxt \triangleq hr' = \text{IF } hr \neq 12 \text{ THEN } hr + 1 \text{ ELSE } 1$

$HC \triangleq HCini \land \Box[HCnxt]_{hr}$

THEOREM $HC \Rightarrow \Box HCini$

TLA+ SYNTAX. LOGIC

Basic logical operators:

Symb.	Python	Description
$\lor$	`or`	logical OR
$\land$	`and`	logical AND
$\neg$	`not`	logical NOT
$=$	`==`	boolean operator, checks equality, it is not an assignment operator.
$\triangleq$	`=`	means defined to equal.

EXERCISE

$$FALSE \wedge (TRUE \wedge (FALSE \vee TRUE) \vee (TRUE \vee FALSE)) = \boxed{??}$$

EXERCISE

$$\boxed{FALSE \wedge} (TRUE \wedge (FALSE \vee TRUE) \vee (TRUE \vee FALSE)) = \boxed{FALSE}$$

TLA+ SYNTAX. MORE LOGIC

More logic operators:

Symb.	Python	Description
$\exists$	`any()`	means "there exists", written as `\E` in ASCII
$\forall$	`all()`	means "for all", written as \A in ASCII
:		reads as "such that"

$\exists x \in 1, 2, 3, 4, 5 : x > 3$ - exists x in set of integers $1, 2, 3, 4, 5$ such that $x > 3$ expression evaluates to TRUE.

TLA+ SYNTAX. MORE LOGIC

More logic operators:

Operator	Description
$\Box$	formula is TRUE on each step [] in ASCII
$\Diamond$	eventually TRUE
$\implies$	logical implication `x_imp_y = y if x else True`
$\boxed{'}$	reads as prime, state of variable on next step `x = x + 1`

`Init /\[][Next]_hr` formula true on each steup for temporal variable `hr`

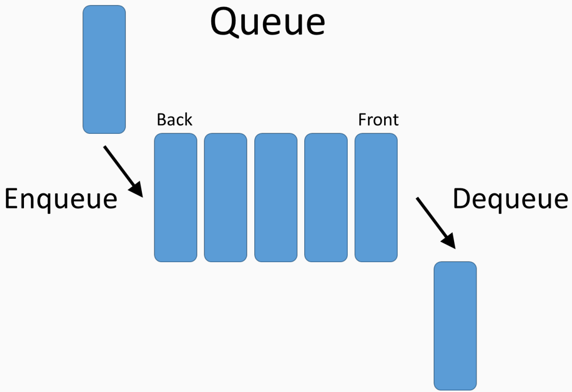
TLA+ SPEC TEMPLATE

```
---------------------- MODULE ModuleName ---------------------
(* Imports and variable declarations*)
EXTENDS Naturals, Sequences, Integers, FiniteSets
--------------------------------------------------------------
(* Initial Conditions *)
Init == ...
TypeOK == ...
--------------------------------------------------------------
(* Body of the spec *)
Next == ...
Invariant == ...
--------------------------------------------------------------
(* Invariant declaration with temporal formula *)
THEOREM  Spec => ...
==============================================================
```

MULTITHREADING QUEUE SPEC

MULTITHREADING QUEUE

Classic *bounded buffer*, attempts to put an element into a full queue or take from empty will block.



QUEUE IMPLEMENTATION, PART 1

```python
import threading

class BoundedQueue:
    def __init__(self, capacity=3):
        self.capacity = capacity
        self.buffer = [None] * capacity
        self.mutex = threading.Lock()
        self.condition = threading.Condition(self.mutex)
        self.size = 0
        self.head = 0
        self.tail = 0

    def is_full(self):
        return self.size == self.capacity

    def is_empty(self):
        return self.size == 0

    def _next(self, x):
        return (x + 1) % self.capacity
```

QUEUE IMPLEMENTATION, PART 2

```python
    def put(self, item):
        with self.condition:
            while self.is_full():
                self.condition.wait()
            self.buffer[self.tail] = item
            self.tail = self._next(self.tail)
            self.size += 1
            self.condition.notify()

    def get(self):
        with self.condition:
            while self.is_empty():
                self.condition.wait()
            item = self.buffer[self.head]
            self.buffer[self.head] = None
            self.head = self._next(self.head)
            self.size -= 1
            self.condition.notify()
            return item
```

QUEUE TLA SPEC: PART 1

```
--------------------------------- MODULE buffer ---------------------------------
EXTENDS Naturals, Sequences

CONSTANTS Producers,
          Consumers,
          BufCapacity,
          Data

ASSUME /\ Producers # {}
       /\ Consumers # {}
       /\ Producers \intersect Consumers = {}
       /\ BufCapacity > 0
       /\ Data # {}
VARIABLES buffer,
          waitSet
----------------------------------------------------------------------------------
```

QUEUE TLA SPEC: PART 2

```
----------------------------------------------------------------------------
Participants == Producers \union Consumers
RunningThreads == Participants \ waitSet

TypeInv == /\ buffer \in Seq(Data)
           /\ Len(buffer) \in 0..BufCapacity
           /\ waitSet \subseteq Participants

Notify == IF waitSet # {}
          THEN \E x \in waitSet : waitSet' = waitSet \ {x}
          ELSE UNCHANGED waitSet

NotifyAll == waitSet' = {}

Wait(t) == waitSet' = waitSet \union {t}
----------------------------------------------------------------------------
```

QUEUE TLA SPEC: PART 3

```
Init == buffer = <<>> /\ waitSet = {}
Put(t,m) == IF Len(buffer) < BufCapacity
              THEN /\ buffer' = Append(buffer, m)
                   /\ Notify
              ELSE /\ Wait(t)
                   /\ UNCHANGED buffer
Get(t) == IF Len(buffer) > 0
            THEN /\ buffer' = Tail(buffer)
                 /\ Notify
            ELSE /\ Wait(t)
                 /\ UNCHANGED buffer
```

QUEUE TLA SPEC: PART 4

```
Next == \E t \in RunningThreads : \/ t \in Producers /\ \E m \in Data : Put(t,m
                                  \/ t \in Consumers /\ Get(t)

Prog == Init /\ [][Next]_<<buffer, waitSet>>
---------------------------------------------------------------------------
NoDeadlock == [](RunningThreads # {})

THEOREM Prog => []TypeInv /\ NoDeadlock
```

Invariant definition

QUEUE TRACE

TLC shows all steps that leads to *invariant violation*.

Name	Value
waitSet	{"p1", "p2", "p3", "c3", "c4", "c5"}
<Action line 46, col 9 to line 47, col 62 of module b...	State (num = 26)
buffer	<<"m1", "m1">>
waitSet	{"p1", "p2", "p3", "c4", "c5"}
<Action line 46, col 9 to line 47, col 62 of module b...	State (num = 27)
buffer	<<"m1">>
waitSet	{"p1", "p2", "p3", "c5"}
<Action line 46, col 9 to line 47, col 62 of module b...	State (num = 28)
buffer	<< >>
waitSet	{"p1", "p2", "p3"}
<Action line 46, col 9 to line 47, col 62 of module b...	State (num = 29)
buffer	<< >>
waitSet	{"p1", "p2", "p3", "c1"}
<Action line 46, col 9 to line 47, col 62 of module b...	State (num = 30)
buffer	<< >>
waitSet	{"p1", "p2", "p3", "c1", "c2"}
<Action line 46, col 9 to line 47, col 62 of module b...	State (num = 31)
buffer	<< >>
waitSet	{"p1", "p2", "p3", "c1", "c2", "c3"}
<Action line 46, col 9 to line 47, col 62 of module b...	State (num = 32)
buffer	<< >>
waitSet	{"p1", "p2", "p3", "c1", "c2", "c3", "c4"}
<Action line 46, col 9 to line 47, col 62 of module b...	State (num = 33)
buffer	<< >>
waitSet	{"p1", "p2", "p3", "c1", "c2", "c3", "c4", "c5"}

MULTITHREADING QUEUE SPEC RESULTS

- Thread programming is hard.
- More than **30 steps** required to reproduce deadlock!
- Notify all strategy fixes issue with deadlock.
- In number of threads $>$ 2 * buffer capacity algorithm is not deadlock free.

AIORWLOCK SPEC

AIORWLOCK – READ WRITE LOCK FOR ASYNCIO

An RW lock allows concurrent access for read-only operations, while write operations require exclusive access, simple example:

```
import asyncio
import aiorwlock

async def go():
    rwlock = aiorwlock.RWLock()

    async with rwlock.writer:
        print("inside writer: only one writer is possible")

    async with rwlock.reader:
        print("inside reader: multiple reader possible")

loop = asyncio.get_event_loop()
loop.run_until_complete(go())
```

AIORWLOCK – BUG

Failing tests for some corner-case uses #37

Closed **popravich** wants to merge 1 commit into `master` from `corner_case_tests`

Conversation 4 | Commits 1 | Files changed 1

popravich commented on May 19, 2017 — Member

Several tests to show when `RWLock` doesn't work as expected.

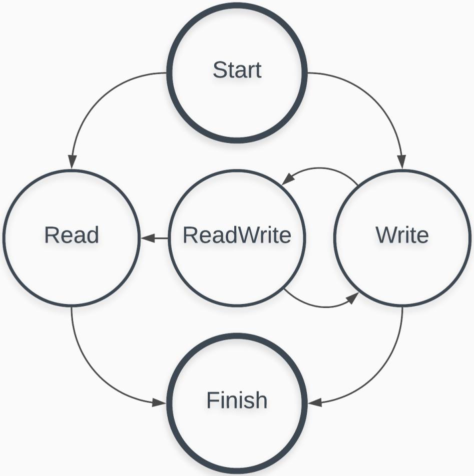
`add failing tests for some corner-case uses` ✗ b00c109

popravich commented on May 19, 2017 — Member

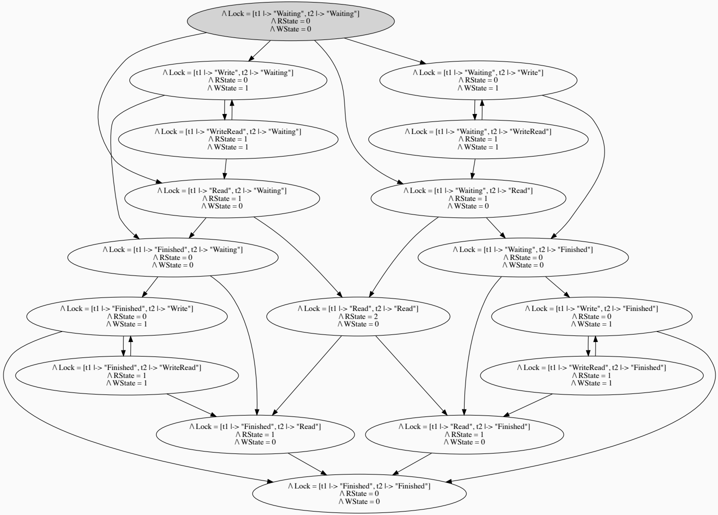
@jettify, please take a look.

AIORWLOCK POSSIBLE STATES

Read Write lock state machine:



AIORWLOCK POSSIBLE STATES



AIORWLOCK TRACE

TLC shows *invariant violation* on step 33!

<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td><Initial predicate></td><td>State (num = 1)</td></tr>
<tr><td>Lock</td><td>[t1 |-> "Waiting", t2 |-> "Waiting", t3 |-> "Waiting"]</td></tr>
<tr><td>State</td><td>0</td></tr>
<tr><td><Action line 30, col 15 to line 33, col 40 of mod...</td><td>State (num = 2)</td></tr>
<tr><td>Lock</td><td>[t1 |-> "Write", t2 |-> "Waiting", t3 |-> "Waiting"]</td></tr>
<tr><td>t1</td><td>"Write"</td></tr>
<tr><td>t2</td><td>"Waiting"</td></tr>
<tr><td>t3</td><td>"Waiting"</td></tr>
<tr><td>State</td><td>-1</td></tr>
<tr><td><Action line 26, col 18 to line 28, col 38 of mod...</td><td>State (num = 3)</td></tr>
<tr><td>Lock</td><td>[t1 |-> "WriteRead", t2 |-> "Waiting", t3 |-> "Waiting"]</td></tr>
<tr><td>t1</td><td>"WriteRead"</td></tr>
<tr><td>t2</td><td>"Waiting"</td></tr>
<tr><td>t3</td><td>"Waiting"</td></tr>
<tr><td>State</td><td>0</td></tr>
<tr><td><Action line 22, col 18 to line 25, col 43 of mod...</td><td>State (num = 4)</td></tr>
<tr><td>Lock</td><td>[t1 |-> "WriteRead", t2 |-> "Read", t3 |-> "Waiting"]</td></tr>
<tr><td>t1</td><td>"WriteRead"</td></tr>
<tr><td>t2</td><td>"Read"</td></tr>
<tr><td>t3</td><td>"Waiting"</td></tr>
<tr><td>State</td><td>1</td></tr>
</table>

AIORWLOCK SPEC RESULTS

- In fact bug reveled itself on specification phase, before running any models.
- Only three steps required to reproduce issue.
- First aio-libs project with formal verification!

CONCLUSIONS

LIMITATIONS OF MODEL-CHECKING

- State space explosion number of states reachable by a system can quickly become huge, or even infinite
- Used as an adjunct to, not a replacement for, standard quality assurance methods
- Formal methods are not a panacea, but can increase confidence in a product's reliability if applied with care and skill
- Very useful for consistency checks, but can not assure completeness

Questions?

http://github.com/jettify

REFERENCES i

azurewebsites.org.
Principles of chaos engineering, 2018.
http://principlesofchaos.org.

W. contributors.
Therac-25 — wikipedia, the free encyclopedia, 2018.
https://en.wikipedia.org/w/index.php?title=Therac-25&oldid=820552180.

Elasticsearch.
Formal models of core elasticsearch algorithms, 2017.
https://github.com/elastic/elasticsearch-formal-models.

A. Hadoop.
Introduce coordination engine interface, 2017.
https://issues.apache.org/jira/browse/HADOOP-10641.

lamport.
The tla toolbox home page, 2018.
http://lamport.azurewebsites.net/tla/toolbox.html.

Microsoft.
A technical overview of azure cosmos db, 2017.
https://azure.microsoft.com/en-us/blog/a-technical-overview-of-azure-cosmos-db/.

REFERENCES ii

mit.edu.
Mit 16.35 aerospace software engineering, 2002.
http://web.mit.edu/16.35/www/lecturenotes/FormalMethods.pdf.

C. Newcombe.
Why amazon chose tla+.
In Y. Ait Ameur and K.-D. Schewe, editors, *Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 25–39, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

Rapitasystems.
What really happened to the software on the mars pathfinder spacecraft?, 2013.
https://www.rapitasystems.com/blog/what-really-happened-to-the-software-on-the-mars-pathfinder-spacecraft.

I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan.
Chord: A scalable peer to peer lookup service for internet applications.
ACM SIGCOMM Computer Communication Review, 31(4):149–160, 2001.

P. Zave.
How to make chord correct (using a stable base).
CoRR, abs/1502.06461, 2015.

S. Zhou.
Tla+ spec of a simplified part of mongodb replication system, 2017.
https://github.com/visualzhou/mongo-repl-tla.