形式化验证工具TLA+：程序员视角的入门之道

原创 祥光 阿里技术 今天

收录于话题

#架构 45 #算法 13 #数据技术 3

一 引言

女娲是飞天分布式系统中提供分布式协同的基础服务，支撑着阿里云的计算、网络、存储等几乎所有云产品。在女娲分布式协同服务中，一致性引擎是核心基础模块，支持了Paxos，Raft，EPaxos等多种一致性协议，根据业务需求支撑不同业务状态机。如何保证一致性库的正确性是一个很大挑战，我们引入了TLA+、Jepsen等工具保证一致性库的正确性。本文即从程序员视角介绍形式化验证工具TLA+。

从理论上证明一个程序或者算法的正确性往往是困难的，工程中一般使用测试来发现问题，但再多的测试也无法保证覆盖到了所有的行为，那些没覆盖到的行为就成为潜在的隐患，一旦在线上再暴露出来，往往会带来不可预期的结果。形式化验证正是为了解决这样的问题，它使用计算机强大的计算能力，暴力的搜索所有可能的行为，检查是否满足事先设定的属性，任何不符合预期的行为都能被发现，从根本上保证算法的正确性。

二 TLA+简介

TLA+(Temporal Logic of Actions) 是Leslie Lamport开发的一门形式化验证语言，用于程序的设计、建模、文档和验证等，特别是并发系统和分布式系统。TLA+的设计初衷是用简单的数学理论和公式精准地对系统进行描述。TLA+及其相关工具有助于消除程序中很难找到、纠错成本高的基本错误。

使用TLA+对程序进行形式化验证，首先要用TLA+对程序进行描述，这样的描述称为规范(Specification)。有了Specification以后就可以使用TLC模型检查器来运行它，运行的过程会遍历所有可能的行为，检查Specification中设定的属性，发现非预期的行为。

TLA+基于数学，使用的是数学思维，与任何编程语言都不相似。为了降低TLA+的门槛，Lamport又开发了PlusCal语言，PlusCal与编程语言类似，可以很方便的描述程序逻辑，并且借用TLA+提供的工具可以直接将PlusCal翻译成TLA+。大多数工程师会发现PlusCal是开始使用TLA+的最简单方法，但简单带来的代价就是PlusCal不具备TLA+的一些功能，有时不能像TLA+那样构造复杂的模型，因此PlusCal还不能取代TLA+。先使用PlusCal编程语言完成基本的逻辑，然后进一步基于生成的TLA+代码再修改，可以简化TLA+的开发。

三 TLA+应用

TLA+在学术界和工业界都有着广泛的应用。TLA+ Examples给出了一些使用TLA+验证过的分布式算法和并发算法。在分布式算法和并发算法的研究领域，提出一个新的算法或者改进一个现有的算法，TLA+验证基本是标配。很多分布式算法论文在非形式化的论证介绍之外，会附带TLA+的Specification来证明自己的算法是经过形式化验证的。对TLA+比较熟悉的业内人士来说，直接看TLA+的Specification甚至比看大段的论文理解的更快，对于论文的语言描述没有看明白，或者觉得有歧义的时候，查看TLA+的Specification对照着理解，有时候是阅读论文的一把利器，甚至有时候一些算法细节只能在TLA+的Specification里看到。由于Specification是逻辑严密滴水不漏的，可以更好的作为实现的指导。

Lamport的TLA+主页上列出了一些TLA+在工业界的应用。以Amazon为例，Amazon AWS的一些系统的核心算法就使用了TLA+来做形式化验证，如表1列出了TLA+给AWS的一些系统找出的问题，其中涵盖了一些非常核心的组件，这些核心组件的问题一旦在线上暴露，造成的损失将是不可估量的。正是如此，现在分布式云服务的核心算法使用TLA+来对设计做验证已经成为行业标准了，所以作为云服务的从业者或者对此感兴趣的同学，熟悉TLA+绝对是不可或缺的加分项。

表1：TLA+给AWS的系统找出的问题

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.

四 TLA+入门

在VS Code中安装TLA+插件就可以开始使用TLA+了。这里先以一个简单的示例入门TLA+。

考虑一个单比特位的时钟，由于只有一个比特位，只能取值0或者1，其行为只有如下两种情况：

0 -> 1 -> 0 -> 1 -> 0 -> ...

1 -> 0 -> 1 -> 0 -> 1 -> ...

我们如何用TLA+来描述这个时钟呢？为了更容易入门，先用更方便工程师入门的PlusCal来描述：

```
---------------------- MODULE clock----------------------
(*--fair algorithm Clock        \*PlusCal代码写在TLA+的注释中
variable
   clock \in {0, 1};          \*定义clock变量并初始化为0或1
define
   Inv == clock \in {0, 1}     \*定义不变式
end define
begin
   while TRUE do               \*死循环
      if clock = 0 then        \*如果clock的值为0
         clock := 1;           \*将clock的值变为1
      else                     \*否则
         clock := 0;           \*将clock的值变为0
      end if
   end while
end algorithm;*)               \*结束算法
=============================================
```

图1：单比特时钟的PlusCal描述

图1是单比特时钟的PlusCal描述，相信具有编程功底的同学都能轻易看懂。这段PlusCal代码可以直接使用TLA+提供的工具翻译成TLA+代码：

```
---------------------- MODULE clock----------------------
VARIABLE clock                \*声明clock变量
vars == << clock >>           \*声明变量列表
Inv == clock \in {0, 1}       \*定义不变式
Init == clock \in {0, 1}      \*初始化
Tick ==                       \*时钟Tick
    IF clock = 0              \*如果clock的值为0
      THEN clock' = 1         \*将clock的值变为1
      ELSE clock' = 0         \*将clock的值变为0
Spec ==                       \*Specification
    /\ Init                   \*初始化，/\为逻辑与
    /\ [][Tick]_vars          \*Tick总是为真或者保持变量不变
    /\ WF_vars(Tick)          \*防止Tick永远不执行
=============================================
```

图2：单比特时钟的TLA+描述

有了上面的PlusCal的基础，理解这一段TLA+也不难，重点在于Spec的理解。Spec定义了系统的行为，如图3描述了单比特时钟的行为，Init将clock初始化为0或1，Tick让clock在0和1之间来回跳转，Stutter让clock保持不变。TLA+运行的过程其实就是在图上做遍历。

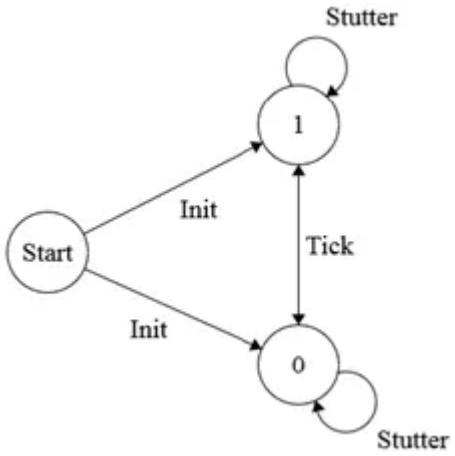


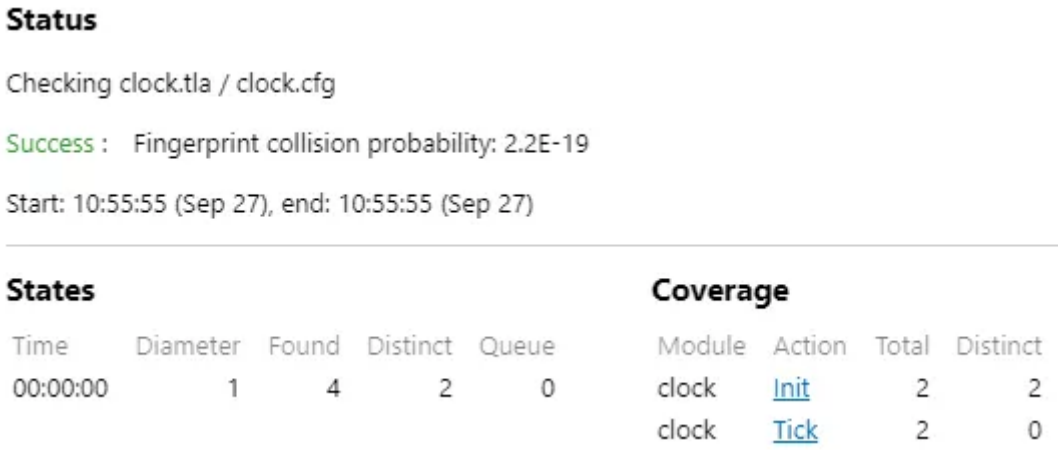
图3：单比特时钟的行为

要让这段TLA+跑起来，上述TLA+代码需保存至clock.tla文件，此外还需要编写一个如图4所示的clock.cfg文件，clock.cfg文件内容很简单，它注明要运行的Specification是哪个，要检查的Invariant是哪个。

```
SPECIFICATION Spec
INVARIANT Inv
```

图4：clock.cfg文件内容

有了这两个文件，就可以用TLC来运行了，运行结束后得到如图5所示的结果，图中展示了一些统计信息。



Status

Checking clock.tla / clock.cfg

Success : Fingerprint collision probability: 2.2E-19

Start: 10:55:55 (Sep 27), end: 10:55:55 (Sep 27)

States

Time	Diameter	Found	Distinct	Queue
00:00:00	1	4	2	0

Coverage

Module	Action	Total	Distinct
clock	Init	2	2
clock	Tick	2	0

图5：运行结果

五 TLA+原理

为了理解TLA+的运行原理，弄清楚它是怎么遍历的，我们可以在运行的时候加上一些参数，让TLC输出状态图。比如我们运行图6所示的一段TLA+代码，图7是运行所需要的cfg文件。这个例子试图找出用面值为1、2和5的钱组合出19块钱的所有组合方式。

```
---------------------- MODULE money ----------------------
EXTENDS Integers                    \*引入Integers模块
CONSTANT MONEYS                     \*声明MONEYS常量
CONSTANT TOTAL                      \*声明TOTAL常量
VARIABLES money                     \*声明money变量
Init == money = 0                   \*初始化
Add(i) == /\ money + i <= TOTAL     \*如果不超过TOTAL
       /\ money' = money + i        \*增加money
Terminating == /\ \A i \in MONEYS: money + i > TOTAL
           /\ UNCHANGED <<money>>   \*保持money不变
Next == \/ \E i \in MONEYS: Add(i)  \*或者增加money
     \/ Terminating                 \*或者终止
Inv == money <= TOTAL               \*不变式
=============================================
```

图6：money.tla

```
CONSTANT MONEYS = {1, 2, 5}
CONSTANT TOTAL = 19
INIT Init
NEXT Next
INVARIANT Inv
```

图7：money.cfg

运行结束后可以得到如图8所示的状态图，图中的顶点为状态，共20种状态，money=0为初始状态，money=19为终止状态，图中的边为动作，共4种动作：Add(1)、Add(2)、Add(5)和

Terminating。

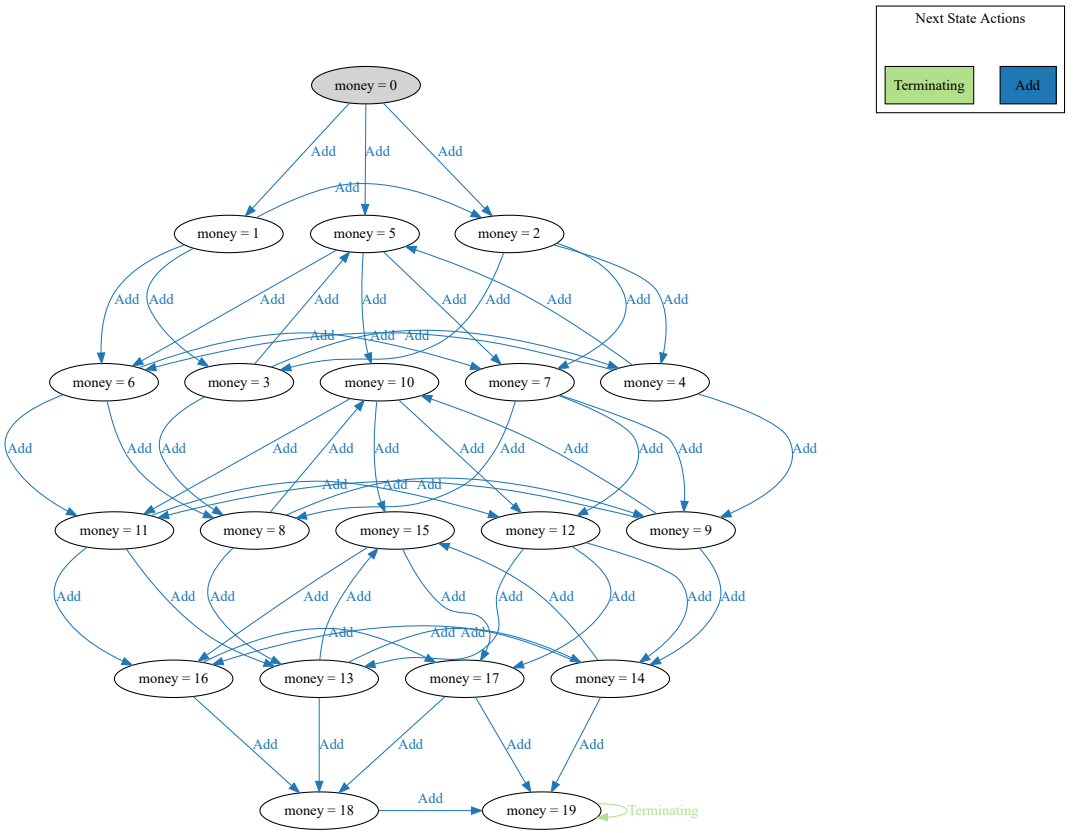


图8：状态图

TLA+的运行是完全串行的，运行的的过程即在状态图上做图的遍历，每遍历到一个状态，就检查一下当前状态是否满足事先设定的不变式，满足则继续遍历，不满足则立即报错。TLA+会尝试所有的遍历路径，不错过任何一种行为。我们知道图的遍历方式有深度优先和广度优先两种，TLA+默认广度优先遍历，也可配置成深度优先模式或者随机行为模式，深度优先模式需要给定一个最大深度。

现在我们知道了TLA+的原理实际上就是状态图的遍历并检查的过程，这样的过程看似简单，却能覆盖到算法所有的路径，不漏掉任何一种行为。实际我们经常使用TLA+检查算法的Safety和Liveness属性。

六 TLA+并发

到这里相信读者对TLA+的原理已经有了初步的了解，但细心的读者可能心中还有一个很大的疑问：TLA+运行过程是完全串行的，那么串行运行的TLA+如何模拟并发算法或者分布式算法呢？

对于串行算法来说，算法中的动作是Totally Ordered，本身就是一个串行的状态机，很容易构造状态图。但并发算法或者分布式算法中的动作是Partially Ordered，不是一个串行的状态机，如何构造出状态图呢？

如果并发算法或者分布式算法中的动作也能变成Totally Ordered，则也可以看作是一个串行的状态机，构造出状态图。

实际上Lamport大师一早就研究了这个问题，在他被引用的最多的论文《Time, Clocks and the Ordering of Events in a Distributed System》中给出了为分布式系统中的事件定序的方法。简单的说就是在保证具有Partially Ordered关系的事件的顺序的前提下，将剩下的无序的事件人为定一个顺序，可以将所有事件排一个序变为Totally Ordered，并且这种定序不会破坏因果关系。

事实上TLA+大放异彩的地方正是在并发算法和分布式算法领域，因为在这些领域算法的行为多种多样，容易疏漏，因此需要TLA+全面检查算法的所有路径，不漏掉任何一种行为。

七 总结

TLA+使用计算机强大的算力搜索算法所有可能的行为，以发现非预期的行为。随着计算机算力的提升，以及软件和硬件系统越来越复杂，TLA+将越来越受到重视，越来越成为工程师的必备技能。

最后如果读者对TLA+感兴趣，这里推荐一本TLA+的入门书籍《Practical TLA+》，比较适合入门，并且网上有免费的电子版可以直接下载。

下一篇 · 流计算引擎数据一致性的本质

阅读原文

喜欢此内容的人还喜欢

阿里排查Java问题工具清单！

顶级架构师

推荐几个开源类库，超好用，远离996！

Hollis

基于Spring Cloud的微服务架构分析

顶级架构师