分布式系统的时间问题

Author	Taosheng Shi
WeChat Contact	data-lake
Mail Contact	tshshi@126.com
Organization	NOKIA
Document category	Distributed System
Document location	https://github.com/stone-note/articles

Version	Status	Date	Author	Description of changes
0.1	Draft	12/7/2017	Taosheng Shi	Initiate
0.2	Draft	DD-MM-YYYY	YourNameHere	TypeYourCommentsHere
1.0	Approved	DD-MM-YYYY	YourNameHere	TypeYourCommentsHere

Contents

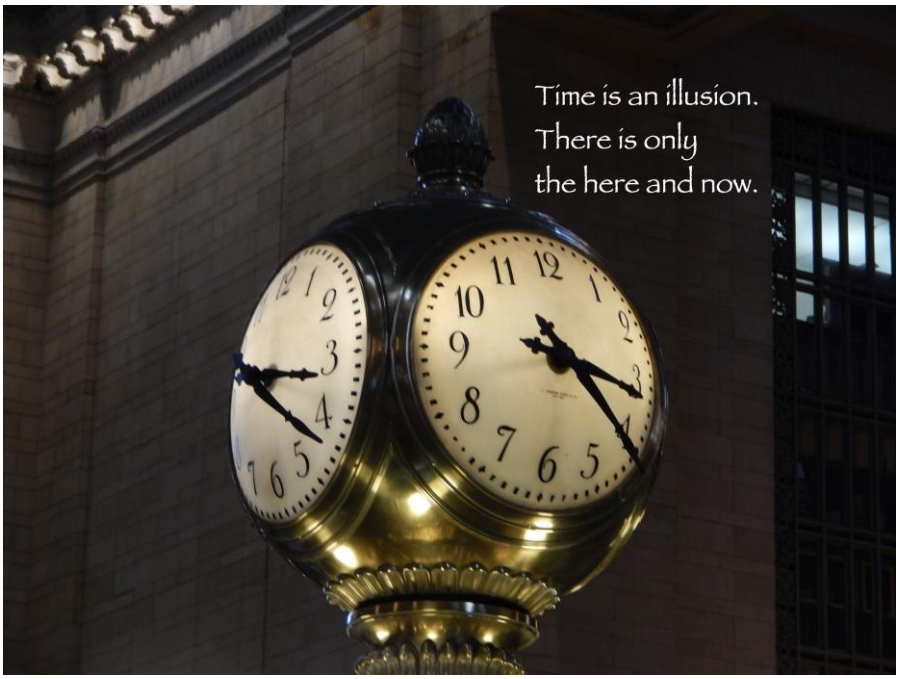
1 什么是时间？

时间是一个很抽象的概念。

牛顿的绝对时空原理——时间在整个宇宙中是一个不变常量：

- 在牛顿的时空观念中，时间和空间是各自独立的，没有关联的两个事物。
- 绝对空间就像一间空房子，它区分物理事件发生的地点，用 3 维坐标来描述。
- 绝对时间就像一个滴答作响的秒表，它区别物理事件发生的先后次序，用不可逆转的 1 维坐标来描述。

“牛顿以为他知道时间是什么”——有人调侃说。在牛顿的绝对时空里，时间的概念是恒定的，在整个宇宙中是一致的，时间是度量事件先后的依据。这非常像我们的一个单一计算机或者紧耦合的计算机集群里，时间是明确的，事件进行的顺序也是明确的。在计算机科学最初的几十年里，我们从来没有想过计算机之间的时间问题。



在爱因斯坦的狭义相对论中，主要有两点：

- 物理定律，包括时间，对所有的观察者来说是相同的。
- 光速不变。

而广义相对论说，整个时空都是一个引力场，时间和空间都不是连续的。

狭义相对论理论的主要意义是没有概念上的同时性，同时的概念相对于观察者，时间的推移也相对于观察者。同时，这一理论也说明了当事情发生在遥远的地方，我们需要时间去发现事情真的发生了。这个等待的时间可能很长。

分布式系统比爱因斯坦的宇宙还要糟糕。在分布式系统中，信息传播所需要的时间范围是不可预知的，可能远超过阳光到达地球的 8 分钟。在这段时间内，无法知道网络另一端的计算机发生了什么。就算你可以通过发送消息来询问或探测，消息的投递和反馈总是要花费时间的。因此，系统延迟时间和超时值的设置是分布式系统的重要设计点之一。

我们可以获得一个信息传播的统计结果，但无法知道每一次消息投递准确时间。从这个角度理解，分布式系统的正确运行都是运气很好的概率上，概率小到我们认为是高可靠的。

关于时间的本质，马赫(Ernst Mach)说：我们根本沒有能力以时间來测量事物的变化，相反的，我们是透过事物的变化因而产生时间流动的抽象概念。

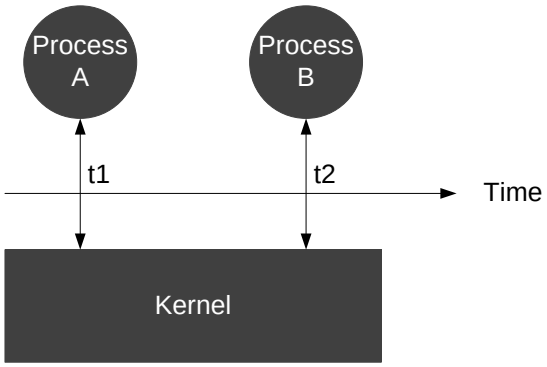
在《七堂极简物理课》中，作者指出：只有存在热量的时候，过去和未来才有区别。能将过去和未来区分开来的基本现象就是热量总是从热的物体跑到冷的物体上。

所以，爱因斯坦说时间是幻像。文学意义上说，不是时间定义了我们的生命，而是我们的生命定义了时间。

这或许是我们放弃物理时钟，使用逻辑时钟的背景吧。

2 物理时间：墙上时钟

中心系统的时间是明确。例如进程 A 和 B 分别在 t1，t2 时刻向内核发起获得时间的系统调用，得到时间必然是 t1<t2。这个简单的机制既是人类常规是时间思维，也是系统中各种功能正常运行所依赖的基础。可以说是常识和共识。



单机系统中的时间依赖于石英钟，并且具有极小的时钟漂移。

物理时间，在英文文献中也称为 wall clock。

3 逻辑时钟：为事件定序

分布式系统中的同步和异步都是对物理时间做的假设。

逻辑时钟第一次摆脱物理时钟的限制。逻辑时钟认为分布式系统中的机器可以对时间无法达成一致，但是对时间发生顺序是认同一致的。一个消息不能在被发送之前收到，这样如果一个进程 A 向进程 B 发送了消息，我们可以认为 A 发生在 B 之前。

这样就定义了分布式系统中不同节点上不同事件之间的因果关系。后来衍生的向量时钟也是同样的道理。

本质上来说，逻辑时钟定义了事件到整数值的映射。所以很多逻辑时钟的实现都采用单调递增的软件计数器，这个计数器的值与任何物理时钟都没有关系。分布式系统中的节点和进程在使用逻辑时钟时，为事件加上逻辑时钟的时间戳，比如文件读写和数据库更新等。

逻辑时钟的贡献来自于 Leslie Lamport 在 1978 年发表的一篇论文《Time, Clocks, and the Ordering of Events in a Distributed System》。

4 Turetime：物理时钟回归

Google 的 Spanner 提出了一种新的思路，在不进行通信的情况下，利用高精度和可观测误差的本地时钟 （TrueTime API）给事件打上时间戳，并且以此比较分布式系统中两个事件的先后顺序。利用这个方法，Spanner 实现了事务之间的外部一致性（external consistency）（如下图所示），也就是说，一个事务结束后另一个事务才开始，Spanner 可以保证第一个事务的时间戳比第二个事务的时间戳要 早，从而两个事务被串行化后也一定能保持正确的顺序。

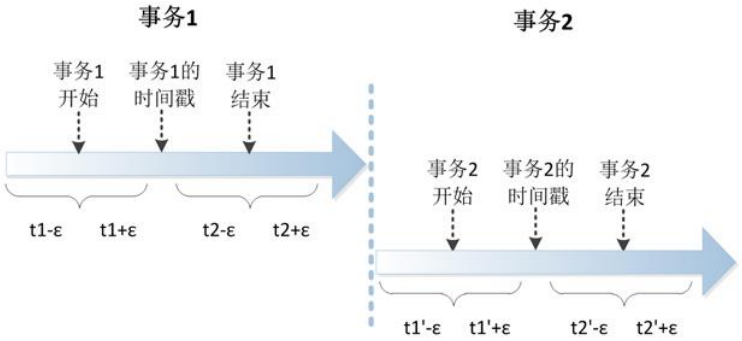


图 事务外部一致性的实现

TrueTime API 是一个提供本地时间的接口，但与 Linux 上 gettimeofday 接口不一样的是，它除了可以返回一个时间戳 t，还会给出一个误差 ε 。例如，返 回的时间戳是 1 分 30 秒 350 毫秒，而误差是 5 毫秒，那么真实的时间在 1 分 30 秒 345 毫秒到 355 毫秒之间。真实的系统中 ε 平均下来是 4 毫秒。

利用 TrueTime API，Spanner 可以保证给出的事务标记的时间戳介于事务开始的真实时间和事务结束的真实时间之间。假如事务开始时 TrueTime API 返回的时间是{t1，ε 1}，此时真实时间在 t1- ε 1 到 t1+ ε 1 之间；事务结束时 TrueTime

API 返回的时间是{t2，ε 2}，此时真实时间在 t2- ε 2 到 t2+ ε 2 之间。Spanner 会在 t1+ ε 1 和 t2- ε 2 之间选择一个时间点作为事务的时间戳，但这需要保证 t1+ ε 1 小于 t2- ε 2，为了保证这点，

Spanner 会在事务执行过程中等待，直到 t2- ε 2 大于 t1+ ε 1 时才提交事务。由此可以推导 出，Spanner 中一个事务至少需要 2 ε 的时间（平均 8 毫秒）才能完成。

由此可见，这种新方法虽然避免了通信开销，却引入了等待时间。为了保证外部一致性，写延迟是不可避免的，这也印证了 CAP 定理所揭示的法则，一致性与延迟之间是需要权衡的。

为什么 Google 要采用这样设计呢？ Truetime 根本上解决了什么问题? Google 的 Spanner 是要解决全球规模分布式系统中关于时间的两大难题：

1. 在数据中心之间同步时间是超级困难和不确定的 synchronizing time within and between datacenters is extremely hard and uncertain
2. 在全球规模范围内串行化请求是不可能的 serialization of requests is impossible at global scale

Google 解决这些问题的办法是接受不确定性，使用 GPS 和原子时钟，把不确定性减低到最小。对于 Spanner 这样全球部署或者跨地域部署的系统，如何来为事务分配 timestamp, 才能保证系统的响应时间在可接受的范围内? 如果整个系统采用一个中心节点来分配 timestamp, 那么系统的响应时间就变得非常不可控，对于离中心节点隔了半圈地球的用户来说， 响应时间估计会是 100ms 级别。

如何理解 turetime 呢？要弄明白 truetime 在事务操作中作用。首先看一下 Spanner 支持的几种事务类型：

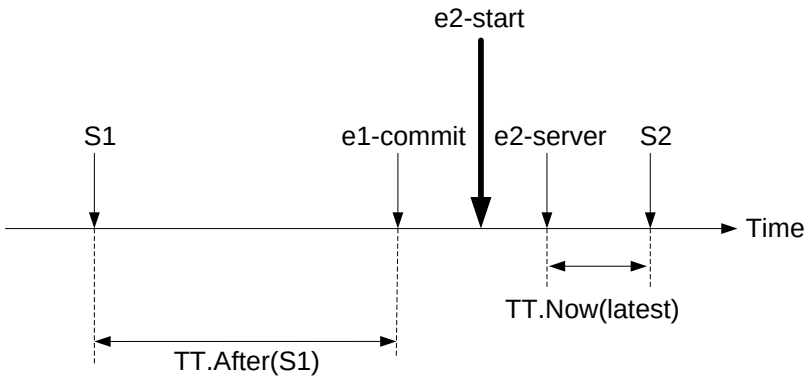
Operation	Timestamp Discussion	Concurrency Control	Replica Required
Read-Write Transaction	§ 4.1.2	pessimistic	leader
Read-Only Transaction	§ 4.1.4	lock-free	leader for timestamp; any for read, subject to § 4.1.3
Snapshot Read, client-provided timestamp	—	lock-free	any, subject to § 4.1.3
Snapshot Read, client-provided bound	§ 4.1.3	lock-free	any, subject to § 4.1.3

先来看下简单只读事务。Spanner 设计的只读事务有一下要求：

- 客户程序自己实现 retry 动作
- 读操作要显式的声明没有写（例如只读打开一个文件）
- 系统无需获得锁，不阻塞读过程中进来的写操作
- 系统自己选择一个时间戳，用来确定读取副本的时间戳
- 任何一个满足时间戳的副本都可以用作读操作

当然，快照读就更简单了，无锁的读取过去的快照，客户可以指定时间戳，也可以让系统选择一个时间戳。无需赘述。

读写事务则使用两阶段锁。外部一致性保证：



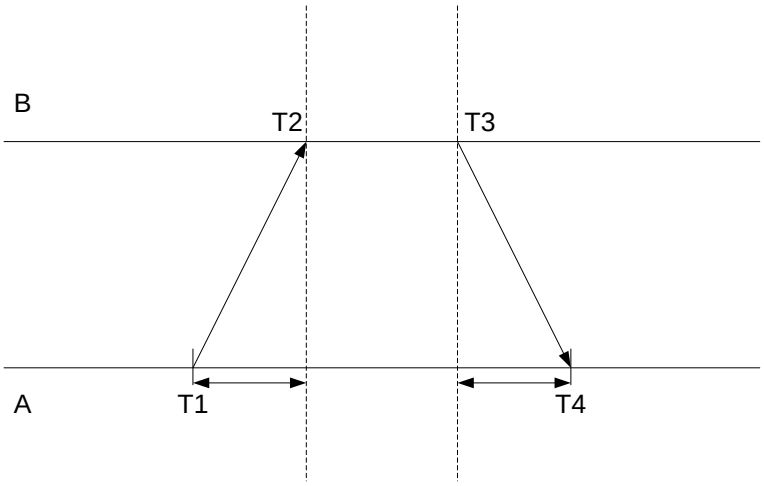
Spanner 利用 truetime 机制，把系统中的操作按照发生的先后顺序，构造一个 Linearizability 的运行记录。所以我们说 Spanner 是实现 Linearizability 的系统。

总结起来，Spanner 是采用全球同步（有一定误差）的物理时间 truetime 戳作为系统的时间戳，并且作为系统内各种操作的版本号。

5 其他影响

5.1 NTP 的时间同步

在分布式系统中，关于时间的共识是非常困难的。不同机器中的石英钟的频率可能不一致，这会导致不同机器中时间并不一致。为了不同机器中的时间，人们提出了 NTP 协议。这样一个机器的时间就依赖于另外一个外部时钟。



NTP 协议基于这样原理：网络通信延迟来回是相等的。基于此可以获得两台机器时间差值θ。在 NTP 服务器（上）获得的时间再加上（时间落后本机）或减去（时间领先本机）这个延迟就可以设置为本地的时间，由此获得时间同步。

$$\theta = T_3 - \frac{(T_2 - T_1) + (T_4 - T_3)}{2}$$

5.2 有限时间内的不可能性

1985 年，Fischer，Lynch 和 Patterson 发布了他们著名 FLP 成果（Impossibility of distributed consensus with one faulty process）：在异步的系统中，如果存在进程故障，系统是不可能达成一致的。FLP 表明，当至少有一个进程可能崩溃时，没有一种确定性地解决异步环境中的共识问题的算法。在工程应用中，这个理论也可以理解为：在不稳定故障的异步系统中，不可能有一个完美的故障检测器。

根本的原因在于时间。FLP 结果并不意味着共识是无法达到的，只是在有限的时间内并不总是可以达到的。同步系统在进程和进程计算之间为消息传递提供了一个已知的上限。异步系统没有固定的上限。

在《分布式系统：概念与设计》一书中，作者针对 FLP 指出：在分布式系统中，并不是说，如果有一个进程出现了故障，进程就永远不可能达到共识。它允许我们达到共识的概率大于 0，这也是与实际情况相符合的。例如现在运行在广域网上的分布式系统，都是异步的，但是事务系统这么多年来一直都能达到共识。

所以，FLP 表述为：在有限的时间内，不可能达成一致。系统可能或者极有可能达成一致，但这不是保证的。也就是说, 在异步的系统中，有限时间内达到一致性是不可能的。在分布式计算上，试图在异步系统和不可靠的通道上达到一致性是不可能的。

所说，一般的一致性前提都是要求保证：在这里我们不考虑拜占庭类型的错误，同时假设消息系统是可靠的。因此对一致性的研究一般假设信道是可靠的，或不存在异步系统上而行。

概率，深刻的影响着分布式系统可靠性。世界的概率本质虽然不能让我们准确的推断未来（气球在口松开时的运动轨迹），但是对于预测消息是否准确传递却绰绰有余。想一想，我们知道在有限时间内达到绝对的一致性是不可能的，那为什么程序员自信的设置超时值为 5 秒？

就像这个世界的概率本质一样，分布式系统构建在概率的基础之上。

5.3 延迟

程序员必须知道的时间数字：

```
L1 cache reference ......................... 0.5 ns
Branch mispredict ............................ 5 ns
L2 cache reference ........................... 7 ns
Mutex lock/unlock ........................... 25 ns
Main memory reference ...................... 100 ns
Compress 1K bytes with Zippy ............. 3,000 ns  =   3 µs
Send 2K bytes over 1 Gbps network ....... 20,000 ns  =  20 µs
SSD random read ........................ 150,000 ns  = 150 µs
Read 1 MB sequentially from memory ..... 250,000 ns  = 250 µs
Round trip within same datacenter ...... 500,000 ns  = 0.5 ms
Read 1 MB sequentially from SSD* ..... 1,000,000 ns  =   1 ms
Disk seek ........................... 10,000,000 ns  =  10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns  =  20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns  = 150 ms
```

5.4 租约

租约可以说是分布式系统的心跳。在分布式系统中，像锁，集群 leader 这样角色，可能随时变化。为了避免死锁，或者错误的 leader 认知，一般都要设计租约机制。

6 总结

本文主要回顾了计算机系统，特别是分布式系统的时间问题，以及由时间带来的顺序问题。随着全球规模分布式系统逐渐成为现实，分布式系统的研究也开始从理论转向工程实践。Truetime 的极致做法，

7 参考文献

http://physics.ucr.edu/~wudka/Physics7/Notes_www/node55.html

https://queue.acm.org/detail.cfm?id=2655736

http://history.programmer.com.cn/14015/

https://www.geekhub.cn/a/1681.html

http://muratbuffalo.blogspot.fi/2013/07/spanner-googles-globally-distributed_4.html

http://blog.sina.com.cn/s/blog_c35ad45b0102wa53.html

http://duanple.blog.163.com/blog/static/709717672012920101343237/

https://en.wikipedia.org/wiki/Lease_(computer_science)

Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency

SEDA: An Architecture for Well-Conditioned, Scalable Internet Services

Time, Clocks, and the Ordering of Events in a Distributed System

Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems

Probabilistic Accuracy Bounds for Fault-Tolerant Computations that Discard Tasks

https://quizlet.com/blog/quizlet-cloud-spanner

http://baike.baidu.com/link?url=upN85rlJXR69lDiYxYy_aFEecj1t3D5vTwIuUXPMN2rz9xKkYlPK-UEOMjA13PWwuuKXNPnE8YXlMJzluqHHpq

http://duanple.blog.163.com/blog/static/709717672011220118587 75/

https://zh.wikipedia.org/wiki/%E6%8B%9C%E5%8D%A0%E5%BA%AD%E5%B0%86%E5%86%9B%E9%97%AE%E9%A2%98

https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

http://highscalability.com/blog/2013/1/15/more-numbers-every-awesome-programmer-must-know.html

https://wondernetwork.com/pings/

https://gist.github.com/jboner/2841832

https://gist.github.com/hellerbarde/2843375

https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

http://link.springer.com/article/10.1023/A:1019121024410

https://www.infoq.com/articles/pritchett-latency

http://highscalability.com/blog/2012/3/12/google-taming-the-long-latency-tail-when-more-machines-equal.html

https://github.com/colin-scott/interactive_latencies

http://bravenewgeek.com/everything-you-know-about-latency-is-wrong/