Spanner, 真时和 CAP 理论

Author	Eric Brewer
Translator	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

Spanner 是 Google 的高可用全球规模分布式数据库。Spanner 为所有事务提供强一致性保证。基于 CAP 理论，同时实现全球规模的可用性和一致性组合通常被认为是不可能的。本文将展示 Spanner 是如何实现这一组合以及如何与 CAP 理论保持一致。另外，本文还探讨了 Google 的全球同步时钟“真时”（TrueTime）在读取一致性——特别是在支持一致和可重复分析快照方面发挥的作用。

1 引言

Spanner 是 Google 支持高可用的全球关系型数据库[CDE+12]。Spanner 可以管理大规模的复制数据，这种大规模不仅体现数据量方面，还体现在事务数量方面。 Spanner 为写入其中的每条数据分配全局一致的“真时”时间戳，客户端可以在整个数据库上无需锁定的执行全局一致性读取。

CAP 理论[Bre12]指出，在 C，A，P 三个期望的属性中，你只能选择两个：

C：一致性，本文中我们可以认为是可串行性；

A：可用性，指读取和更新的 100%可用；

P：网络分区，指对网络分片的容忍。

基于 CAP 理论，选择要抛弃的字母，将会得到三种系统：CA，CP 和 AP。需要注意的是，你不是必须选择 3 个属性中的 2 个，许多系统具有零个或一个 CAP 属性。

对于“全球规模”的分布式系统，分区通常被认为是不可避免的——尽管这并不是共识[BK14]。一旦你认为分区是不可避免的，任何分布式系统都必须准备放弃一致性（AP）或放弃可用性（CP），这不是任何人都想要的选择结果。实际上，CAP 理论的出发点是让设计者认真对待这种权衡。这里给出两个重要的警告：首先，在真正发生分区的时候，你只需要丧失系统的部分功能而不是全部——当然你必须付出一些代价[Bre12]。其次，CAP 理论申明的是 100%可用性，而本文讨论的有趣之处是真实生产环境中涉及可用性本身的折中和权衡。

2 Spanner 是 CA 系统

作为一个全球规模分布式系统，Spanner 声称同时具有一致性和高可用性，这意味着 Spanner 系统不会发生分区，这很让人怀疑。这是否意味着 Spanner 就是 CAP 定义的 CA 系统？从技术上来说，直接的答案是“否”，但从效果上来说，答案可以是“是”——即从用户的角度，可以认为 Spanner 是一个 CA 系统。

纯粹主义者从技术上来看，答案当然是“否”。因为分区确实可能发生，并且实际上在谷歌也发生过。特别是在发生分区的时候，Spanner 的策略是选择 C 而放弃 A。所以，技术上 Spanner 是一个 CP 系统。下面我们将具体展开对分区的探讨。

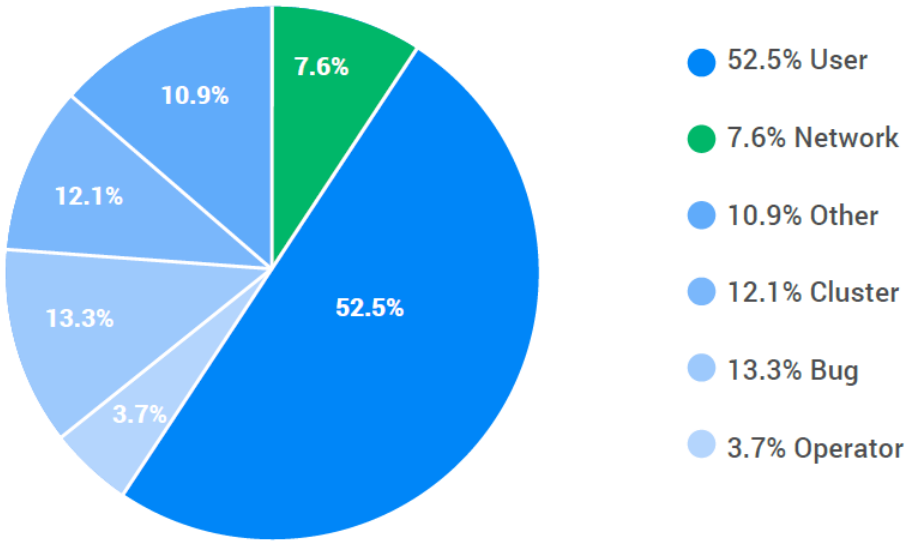
考虑到 Spanner 永远提供一致性，Spanner 宣称是 CA 系统存在的真正问题是用户对可用性的态度——即用户是否会对可用性吹毛求疵。如果 Spanner 的实际可用性很高，宕机事件对用户来说可以忽略，则 Spanner 可以证明“实际上是 CA 系统”声明的正当性。宣称 CA 系统并不意味着 100%的可用性（Spanner 没有提供也不会提供），而是类似 5 个或更多“9”的可用性（1/106 的故障或更少）。反过来，数据库可用性的真正石蕊试纸是用户（希望自己的服务高可用）——即用户是否需要编写额外的代码来处理宕机异常。如果用户没有编写额外代码，也就是说用户是假设数据库高可用的。而基于 Spanner 内部已有的大量用户，我们可以知道用户认为 Spanner 是高可用数据库。

3 关于可用性的数据

在我们深入研究 Spanner 之前，重新回顾 Chubby 的演进历程是值得的。Chubby 也是一个提供一致性和可用性的全球规模（广域网络）系统。原始的 Chubby 论文[Bur06]提到在 700 天内出现了 30 秒或更长时间的 9 次宕机，其中 6 次与网络相关（如[BK14]中所讨论的）。这样的可用性低于最好的 5 个 9 可用性，现实一点说，如果我们假设每次宕机的平均值为 10 分钟，这样的可用性低于 4 个 9，更有甚者，如果宕机时间是小时级的，则可用性将会低于 3 个 9。

对于锁定和一致的读/写操作，得益于网络，架构和运维的各种改进，现在全球地理分布的 Chubby 单元提供 99.99958%的平均可用性（30s+的宕机时间）。从 2009 年开始，由于“卓越”的可用性，Chubby 的 SRE 工程师开始强制定期停机，以确保我们继续了解 Chubby 对故障的依赖和影响。

在 Google 内部，Spanner 提供与 Chubby 类似的可用性水平。也就是说，好于 5 个 9。云版本的 Spanner 具有相同的基础，但添加了一些新的功能，所以在实际生产环境中可能会稍微低一点。



上面的饼图揭示了 Spanner 事故的内部原因分类。事故可能是一个意外事件，但并非所有事故都是宕机，一些事故可以很容易地掩蔽。图表中的权重是事故出现的频率而不是事故的影响。频率最高的事故类别（User）是用户错误导致的，例如负载过高或者配置错误，并且这些事故大多数只会影响到该特定用户，而其余类别则可能会影响到区域中的所有用户。Cluster 类别的事故反映底层基础架构中的非网络问题，包括服务器和电源的问题。Spanner 通过使用多副本机制来自动规避这些事件。然而，有时 SRE 也需要介入以修复损坏的副本。Operator 类别的事故是由 SRE 引起的事故，例如配置错误。Bug 类别的事故则是导致一些问题的软件错误。这些事故都可能导致或大或小的宕机，曾经两起最大的宕机事故都是软件错误，并且同时影响到特定数据库的所有副本。其他一麻袋的各种问题大多数只发生一次。

Network 类别的事故（8%以下）是由网络分区和网络配置问题造成的。一部分集群节点和另外一部分集群节点分开的情况从来没有发生过，也没有发生过 Spanner 的法定数节点出现在分区中集群数量较少一边的情况。但我们确实遇到过个别数据中心或区域与其他网络断开的情况。还有就是一些配置错误导致临时预留带宽不足和一些与硬件故障相关的延迟。我们曾经遇到一个问题：其中单方向的通信失败导致一个奇怪的分区发生，然后必须通过关闭一些节点来解决。到目前为止，网络事件没有引起大的宕机事故。

总而言之，要声明“实际上是 CA”系统，那么，这个系统必须处于这种相对概率状态：1）系统必须在生产环境中具有非常高的可用性（以便用户可以忽略异常），以及 2）系统由于分区引起的宕机故障次数应该处于一个比较低的份额。Spanner 满足两者。

4 网络才是根本

许多人认为，Spanner 通过使用真时（TrueTime）可以绕过 CAP 理论。真时（TrueTime）是一种能够使用全局同步时钟的服务。虽然真时（TrueTime）很棒，却对于实现 CA 系统没有太大贡献，它的实际价值会在下文讨论。如果说 Spanner 真有什么特别之处，那就是谷歌的广域网。在多年的运营改进的基础上，在生产环境中可以最大程度的减少分区发生，从而实现高可用性。

首先，Google 的系统运行在自己的私有全球网络之上。Spanner 不在公共互联网上运行——实际上，Spanner 的每个数据包只流过 Google 控制的路由器和链路（不包括到远程客户端的边缘链路）。此外，每个数据中心通常具有至少三个将其连接到私有全球网络的独立光纤，以确保每对数据中心的路径分集（路径多样性）。同样，在数据中心内部存在设备和路径的冗余。因此，通常灾难性事件，例如切断光缆，不会导致分区或宕机。

因此，带来分区的真正风险不是网络路径断开，而是某种类型的广泛配置或软件升级，这些配置或升级会同时切断多个网络路径。这才是真正的风险，Google 一直在努力防止和减轻这种风险。一般的策略是限制任何特定更新的影响范围（“爆炸半径”），以便当我们需要不可避免地推送一个错误的更改时，只影响部分路径或副本，然后我们必须在其他任何更改之前修复这些问题。

虽然 Google 的基础网络可以大大减少分区的风险，但它不能提高光速。跨越广域网的一致操作具有显著的最小往返时间，这个时间可以达到几十毫秒，当跨越大陆时则更长（1000 英里的距离约为 5 百万英尺，每纳秒½英尺，因此最小值为 10 毫秒）。为了平衡区域内延迟和容灾的需求，Google 定义“区域”的范围只具有 2ms 往返时间。Spanner 通过事务流水线管理来缓解延迟，但这并不能减少单事务延迟。对于读操作来说，由于能够使用全局时间戳和本地副本（如下所述），延迟通常较低。

具有弱一致性的模型具有较低的更新延迟。然而，没有长程往返，弱一致性的模型也有一个较低的持久性窗口，因为在数据复制到另一个区域之前，灾难可以同时摧毁本地和远程多个副本。

5 网络分区期间发生什么

为了理解分区，我们需要更多地了解 Spanner 的工作原理。与大多数 ACID 数据库一样，Spanner 使用两阶段提交（2PC）和严格的两阶段锁定，以确保隔离和强一致性。2PC 已经被称为“反可用性”协议[Hel16]，因为 2PC 要求所有成员节点必须参与工作（即必须可用）。为了缓解这个问题，Spanner 为每个 2PC 成员节点建立 Paxos 组，这样就算部分 Paxos 组成员宕机，2PC 的“成员”还是高可用的。同时，数据也被分成组，形成放置和复制的基本单元。

如上所述，一般来说，当发生分区时，Spanner 会选择 C 而放弃 A。在实际生产环境中，这是基于以下考虑：

- 使用 Paxos 组来实现更新的共识。如果领导节点由于分区而不能维持仲裁（法定数），则更新被停止，并且系统不可用（通过 CAP 定义）。最终，在多数节点的分区，会重新选举新的领导节点。
- 对跨组事务使用 2PC 意味着分区成员可以阻止事务提交。

分区在生产环境中最可能的结果是，有法定数节点的分区在选举新的领导节点之后继续工作，即服务继续可用，但是少数节点分区一方的用户则无法访问服务。这是差别可用性的一个案例：少数节点分区一方的用户可能有其他重大问题，比如失去连接，或已经宕机。这意味着构建在 Spanner 之上的多区域服务即使在分区期间也能够良好工作。一些 Paxos 组完全不可用的可能性也是有的，但是不大。

只要所有已经建立联系的组都具有基于法定数选举的领导节点，并都位于分区的一侧，则 Spanner 中的“事务”可以工作。这意味着有些事务会完美工作，有些事务会超时，但系统总是一致的。
Spanner 的实现属性是，任何读取的返回都是一致的，即使事务稍后中止（包括超时等任何原因）。

除了常规事务之外，Spanner 还支持快照读取——从过去的特定时间读取。 Spanner 随时间维护数据的多个版本，每个版本都有一个时间戳，因此可以使用正确版本准确地回答快照读取。特别地，每个副本知道它被写入的时间（必须的），并且任何副本可以在该时间戳之前单方面地回答读取（除非它太旧了，并且已经被垃圾收集）。类似地，很容易在同一时间跨多个组读取（异步）。快照读取根本不需要锁。事实上，只读事务被实现为在当前时间（在任何最新的副本）的快照上读取。

因此，快照读取使分区更加健壮。特别地，快照读取将会在下列条件下起作用：

1. 正在初始化的分区中每个组至少存在一个副本，以及
2. 这些副本的时间戳是过时的。

如果领导节点由于分区而失效，并且持续与分区同时失效，则第二条可能不成立，因为不可能在分区的这一侧上选举新的领导节点。在发生分区期间，读取在分区开始之前时间戳处的数据，很可能在分区的两侧上都成功，因为任何可达副本的数据都满足需求（译者注：看来可用性还要看用户是否需要最新的数据）。

6 关于真时（TrueTime）

一般来说，同步时钟可以避免分布式系统中的通信。Barbara Liskov 提供了带有许多示例的详细概述[Lis91]。对于我们的目标来说，真时（TrueTime）是一个有界非零误差的全局同步时钟：它返回一个时间间隔，这个时间间隔包含了调用执行的实际时间。因此，如果两个间隔不重叠，则我们知道调用一定是实时排序的。如果间隔重叠，我们将无法知道调用的实际顺序。

Spanner 的一个精妙之处在于它从分布式锁获得串行性，从真时（TrueTime）获得外部一致性（类似于线性化）。Spanner 的外部一致性不变量是指，对任何两个事务，T1 和 T2（即使在地球的两侧）：

如果 T2 在 T1 提交后开始提交，则 T2 的时间戳大于 T1 的时间戳。

借用 Liskov[Lis91，第 7 节]的话：

可以使用同步时钟可以降低违反外部一致性的概率。本质上来说，主服务器保存租约，对象是整个备份副本组。备份节点发送到主服务器的每条消息将授予主服务器租约。如果主存储器保存有来自多数备份节点的未到期租约，则主数据库可以单方面进行读取操作。……

该系统中的不变量是：每当主服务器节点执行读取时，它必须保存有来自大多数备份节点的有效租约。如果时钟不同步，这个不变量将不会成立。

Spanner 使用真时（TrueTime）作为时钟，可以确保不变量成立。特别地，在事务提交期间，领导节点必须等待，直到确认提交时间是过去的（基于误差范围）。这种“提交等待”在生产环境中不是长时间的等待，并且与（内部）事务通信并行地进行。一般来说，外部一致性需要单调增加时间戳，而“等待不确定性”是一种常见的模式。

Spanner 通过更新租约，延长领导节点的选举时间，通常为 10 秒。就像 Liskov 所讨论的，每当法定数节点同意一项决定时，租约被延长，因为参与节点刚刚验证了领导节点是有效的。当领导节点故障时，有两个选项：1）等待租约过期，然后选择新的领导节点，或 2）重启旧领导节点，这可能更快。对于一些故障，我们可以发出“最后一个”UDP 数据包来释放租约，这是用来加速租约的到期。由于计划外的故障在 Google 数据中心中很少见，所以长期租约很有意义（译者注：乐观方法）。租约还确保领导节点之间的时间单调性，并且使得群组参与节点能够在租约时间内提供读取（即使没有领导节点）。

然而，真时（TrueTime）的真正价值在于它在一致性快照方面的能力。回望过去，多版本并发控制系统（MVCC）[Ree78]的历史已经很悠久了——单独保存旧版本，读取操作从旧的版本中读取数据，而不需要考虑当前的事务操作。这是一个非常有用和被低估的属性：特别是，Spanner 快照是一致的

（对于快照时间），因此无论系统保存什么样的不变量，快照也将会保存。这是真的，尽管你不知道什么是不变量！本质上来说，快照来自于连续事务之间，并且反映了快照时间之前的所有内容（更多的就没有了）。如果没有事务一致的快照，系统很难从过去的某个时间点重新启动，因为事务部分提交的内容可能违反了一些不变量或完整性约束。正是缺乏一致性，有时系统很难从备份中还原——冲突的数据需要手动恢复。

例如，考虑使用 MapReduce 在数据库上执行大型分析查询。如果使用 Bigtable 作为数据库，尽管 Bigtable 也存储数据过去的版本，但数据分片的时间标签是“锯齿状”的，这使得结果不可预测，有时还会导致不一致（特别是对于最近的数据）。如果使用 Spanner 作为数据库，同一个 MapReduce 操作可以选择精确的时间戳，并获得可重复和一致的结果。

真时（TrueTime）还使跨多个独立系统记录快照成为可能——只要使用（单调递增）真时（TrueTime）时间戳提交，与快照时间达成一致，并随时间存储多个版本（通常在日志中）。这不仅限于 Spanner：您可以制作自己的事务系统，并且两个系统（甚至 k 系统）上的快照是一致的。一般来说，在这些系统上你需要一个 2PC（同时持有锁）来与快照时间达成一致并确认成功，但系统不需要就其他事项达成一致，可以完全不同。

你还可以使用时间戳作为令牌在工作流中传递。例如，如果对系统进行更新，你可以将更新的时间戳传递到工作流的下一个阶段，以便确定系统是否反映该事件后的时间。在发生分区的情况下，就不一定了。在这种情况下，下一个阶段实际上应该等待保持一致性（或继续工作保持可用性）。如果没有时间戳令牌，很难知道你需要等待什么。当然，传递时间戳不是解决这个问题的唯一方法，但这是一种优雅的，鲁棒的方式，同时确保最终一致性。当不同的阶段不共享代码且具有不同的管理员时，这特别有用——两者可以在没有通信的情况下对时间达成一致。

快照是关于过去的，Spanner 也可以对未来时间达成共识。Spanner 的一项功能是，你可以对未来 schema 变更的时间达成一致。这允许你暂停应用新 schema 的更改，以便能够同时服务两个版本。一旦你准备好了，你可以选择一个时间点，在所有副本上以原子方式同时切换到新的 schema（你也可以选择之前的时间点，但你不可能在目标时间之前准备好）。至少在理论上，你可以进行未来的操作，例如可见性计划的删除或更改。

真时（TrueTime）本身可能受到分区的阻碍。真时（TrueTime）的来源是 GPS 接收器和原子钟的组合，两者都可以通过它们自身的微小漂移来保持精确的时间。由于每个数据中心都有“主时间服务器”（冗余），因此分区的两侧很可能继续享有准确的时间。然而，独立的节点需要到通过网络连接到时间服务器，否则它们的时钟将会漂移。因此，在分区期间，基于本地时钟漂移的速率，独立节点的间隔随着时间缓慢地增长。基于真时（TrueTime）的操作，如 Paxos 领导节点选举或事务提交，因此必须多等待一段时间，但操作仍能够完成（假设 2PC 和法定数仲裁的通信工作良好）。

7 结论

作为一个全球规模的分布式系统，Spanner 只是提供了一致性和大于 5 个 9 的可用性，但 Spanner 宣称是一个“实际上是 CA”系统是合理的。与 Chubby 一样，如果你能够控制整个网络（尽管全球范围内不常见），CA 组合在实际生产环境中是可能的。即使这样，仍旧需要网络路径的大量冗余，处理相关故障的架构规划，以及计划周密的运维，特别是系统升级的时候。在发生宕机的情况时，Spanner 选择一致性而不是可用性。

Spanner 使用 2PC 来实现串行化，使用真时（TrueTime）实现外部一致性，无锁一致性读取，以及一致的快照。

8 致谢

特别感谢 Spanner 和 TrueTime 专家 Andrew Fikes，Wilson Hsieh 和 Peter Hochschild。另外感谢 Brian Cooper，Kurt Rosenfeld，Chris Taylor，Susan Shepard，Sunil Mushran，Steve Middlekauff，Cliff Frey，Cian Cullinan，Robert Kubis，Deepti Srivastava ，Sean Quinlan，Mike Burrows 和 Sebastian Kanthak。

9 参考文献

[BK14] P. Bailis and K. Kingsbury. The Network is Reliable, Communications of the ACM. Vol. 57 No. 9,Pages 48-55. September 2014. Also: https://aphyr.com/posts/288-the-network-is-reliable

[Bre12] E. Brewer. CAP Twelve Years Later: How the “Rules” Have Changed, IEEE Computer, Vol. 45, Issue 2, February 2012. pp. 23--29.

[Bur06] M. Burrows. The Chubby lock service for loosely-coupled distributed systems. Proceedings of OSDI ‘06: Fourth Symposium on Operating System Design and Implementation, Seattle,WA, November 2006.

[CDE+12] J. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, JJ Furman, S. Ghemawat, A. Gubarev, C.Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D.Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google’s Globally-Distributed Database. Proceedings of OSDI ‘12: Tenth Symposium on Operating System Design and Implementation, Hollywood, CA, October, 2012

[Hel16] P. Helland. Standing on Giant Distributed Shoulders: Farsighted Physicists of Yore were Danged Smart! ACM Queue, Vol. 14, Issue 2, March-April 2016.

[Lis91] B. Liskov. Practical Uses of Synchronized Clocks in Distributed Systems. ACM Principles of Distributed Computing (PODC). Montreal, Canada, August 1991.

[MHL+92] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh and P. Schwartz. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging.ACM Transactions on Database Systems, Vol. 17, No. 1, March 1992, pp. 94-162.

[Ree78] D. Reed. NAMING AND SYNCHRONIZATION IN A DECENTRALIZED COMPUTER SYSTEM, PhD Dissertation, MIT Laboratory for Computer Science, Technical Report MIT-LCS-TR-205.October 1978 [See Section 6.3 for list of versions with timestamps]