数据一致性-分区可用性-性能——多副本强同步数据库系统实现之我见

阿里数据库团队：何登成

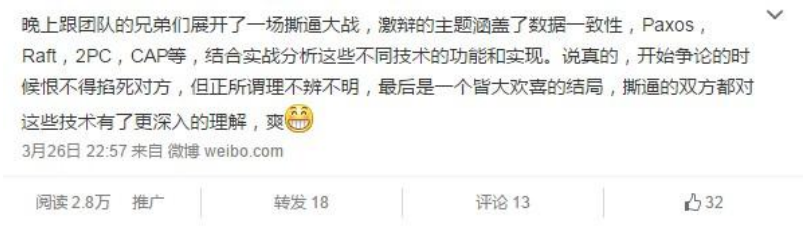
1 背景

最近，@阿里正祥（阳老师）发了上面的一条微博，谁知一石激起千层浪，国内各路数据库领域的朋友在此条微博上发散出无数新的话题，争吵有之，激辩有之，抨击有之，谩骂有之，不一而足。总体来说，大家重点关注其中的一点：

在不使用共享存储的情况下，传统 RDBMS（例如：Oracle/MySQL/PostgreSQL 等），能否做到在主库出问题时的数据零丢失。

这个话题被引爆之后，我们团队内部也经过了激烈的辩论，多方各执一词。辩论的过程中，差点就重现了乌克兰议会时场景...

庆幸的是，在我的铁腕统治之下，同学们还是保持着只关注技术，就事论事的撕逼氛围，没有上升到相互人身攻击的层次。激辩的结果，确实是收获满满，当时我就立即发了一条微博，宣泄一下自己愉悦的心情☺



微博发出之后，也有一些朋友回复是否可以将激辩的内容写出来，独乐乐不如众乐乐。我一想也对，强数据同步，数据一致性，性能，分区可用性，Paxos，Raft，CAP 等一系列知识，我也是第一次能够较好的组织起来，写下来，一来可以加深自己的印象，二来也可以再多混一点虚名，何乐而不为☺

这篇博客文章接下来的部分，将跳出任何一种数据库，从原理的角度上来分析下面的几个问题：

- **问题一：数据一致性。**在不使用共享存储的情况下，传统 RDBMS（例如：Oracle/MySQL/PostgreSQL 等），能否做到在主库出问题时的数据零丢失。
- **问题二：分区可用性。**有多个副本的数据库，怎么在出现各种问题时保证系统的持续可用？
- **问题三：性能。**不使用共享存储的 RDBMS，为了保证多个副本间的数据一致性，是否会损失性能？如何将性能的损失降到最低？
- **问题四：一个极端场景的分析。**

2 问题一：数据一致性

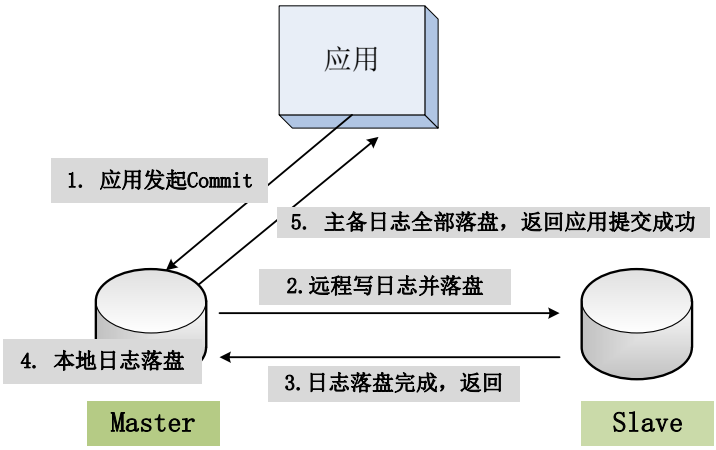
问：脱离了共享存储，传统关系型数据库就无法做到主备强一致吗？

答：我的答案，是 No。哪怕不用共享存储，任何数据库，也都可以做到主备数据的强一致。Oracle 如此，MySQL 如此，PostgreSQL 如此，OceanBase 也如此。

如何实现主备强一致？大家都知道数据库中最重要的一个技术：WAL（Write-Ahead-Logging）。更新操作写日志（Oracle Redo Log，MySQL Binlog 等），事务提交时，保证将事务产生的日志先刷到磁盘上，保证整个事务的更新操作数据不丢失。那实现数据库主备数据强一致的方法也很简单：

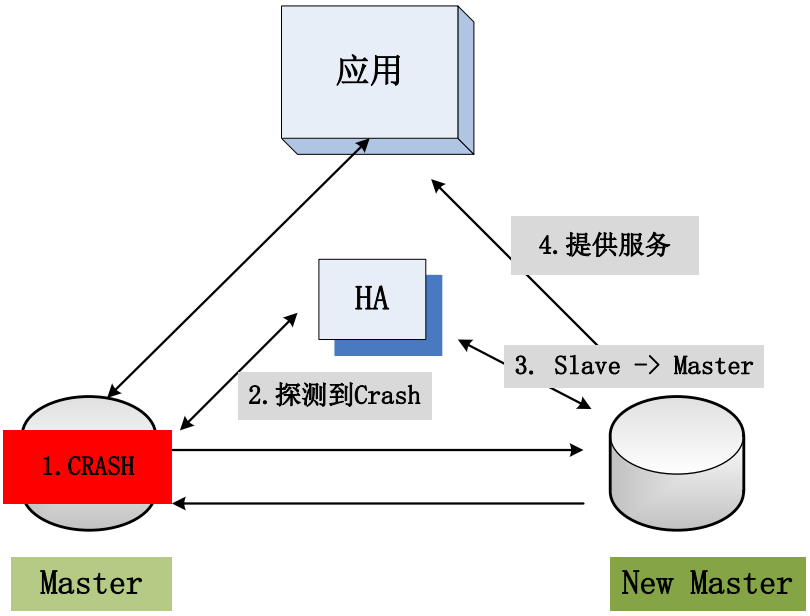
1) 事务提交的时候，同时发起两个写日志操作，一个是将日志写到本地磁盘的操作，另一个是将日志同步到备库并且确保落盘的操作；
2) 主库此时等待两个操作全部成功返回之后，才返回给应用方，事务提交成功；

整个事务提交操作的逻辑，如下图所示：



上图所示，由于事务提交操作返回给应用时，事务产生的日志在主备两个数据库上都已经存在了，强同步。因此，此时主库 Crash 的话，备库提供服务，其数据与主库是一致的，没有任何事务的数据丢失问题。主备数据强一致实现。用过 Oracle 的朋友，应该都知道 Oracle 的 Data Guard，可工作在 最大性能，最大可用，最大保护 三种模式下，其中第三种 最大保护模式，采用的就是上图中的基本思路。

实现数据的强同步实现之后，接下来到了考虑可用性问题。现在已经有主备两个数据完全一致的数据库，备库存在的主要意义，就是在主库出故障时，能够接管应用的请求，确保整个数据库能够持续的提供服务：主库 Crash，备库提升为主库，对外提供服务。此时，又涉及到一个决策的问题，主备切换这个操作谁来做？人当然可以做，接收到主库崩溃的报警，手动将备库切换为主库。但是，手动的效率是低下的，更别提数据库可能会随时崩溃，全部让人来处理，也不够厚道。一个 HA（High Availability）检测工具应运而生：HA 工具一般部署在第三台服务器上，同时连接主备，当其检测到主库无法连接，就切换备库，很简单的处理逻辑，如下图所示：

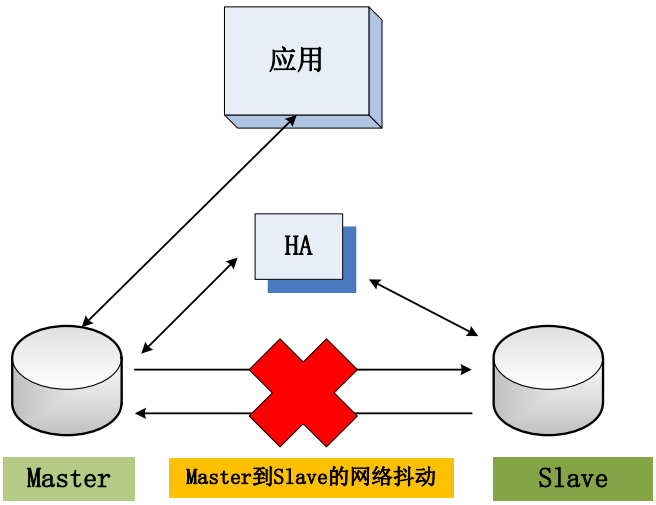


HA 软件与主备同时连接，并且有定时的心跳检测。主库 Crash 后，HA 探测到，发起一个将备库提升为主库的操作（修改备库的 VIP 或者是 DNS，可能还需要将备库激活等一系列操作），新的主库提供对外服务。此时，由于主备的数据是通过日志强同步的，因此并没有数据丢失，数据一致性得到了保障。

有了基于日志的数据强同步，有了主备自动切换的 HA 软件，是不是就一切万事大吉了？我很想说是，确实这个架构已经能够解决 90%以上的问题，但是这个架构在某些情况下，也埋下了几个比较大的问题。

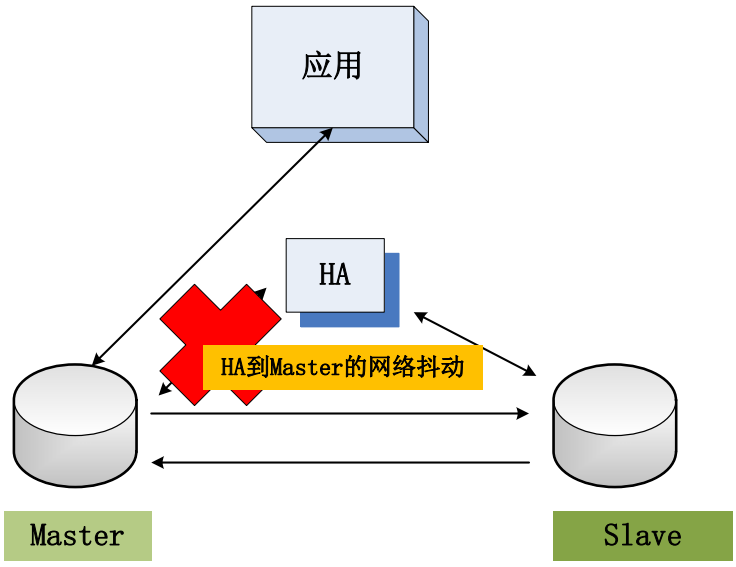
首先，一个一目了然的问题，主库 Crash，备库提升为主库之后，此时的数据库是一个单点，原主库重启的这段时间，单点问题一直存在。如果这个时候，新的存储再次 Crash，整个系统就处于不可用状态。此问题，可以通过增加更多副本，更多备库的方式解决，例如 3 副本（一主两备），此处略过不表。

其次，在主备环境下，处理主库挂的问题，算是比较简单的，决策简单：主库 Crash，切换备库。但是，如果不是主库 Crash，而是网络发生了一些问题，如下图所示：



若 Master 与 Slave 之间的网络出现问题，例如：断网，网络抖动等。此时数据库应该怎么办？Master 继续提供服务？Slave 没有同步日志，会数据丢失。Master 不提供服务？应用不可用。在 Oracle 中，如果设置为 最大可用 模式，则此时仍旧提供服务，允许数据不一致；如果设置为 最大保护 模式，则 Master 不提供服务。因此，在 Oracle 中，如果设置为 最大保护 模式，一般建议设置两个或以上的 Slave，任何一个 Slave 日志同步成功，Master 就继续提供服务，提供系统的可用性。

网络问题不仅仅出现在 Master 和 Slave 之间，同样也可能出现在 HA 与 Master，HA 与 Slave 之间。考虑下面的这种情况：



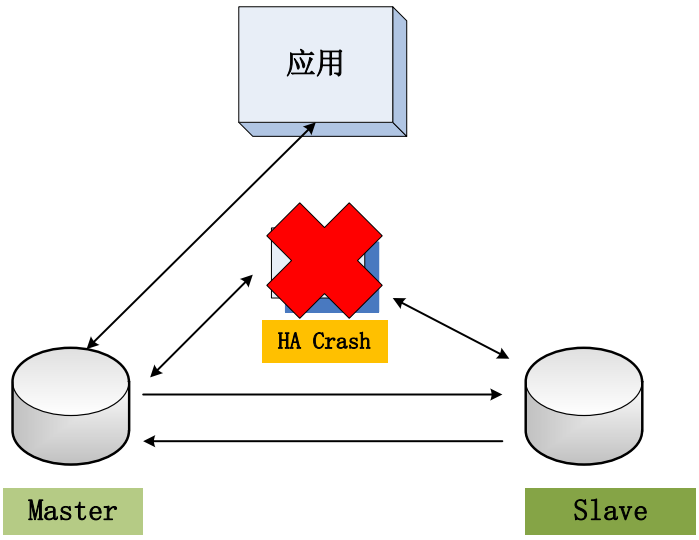
HA 与 Master 之间的网络出现问题，此时 HA 面临两个抉择：

(一) HA 到 Master 之间的连接不通，认为主库 Crash。选择将备库提升为主库。但实际上，只是 HA 到 Master 间的网络有问题，原主库是好的（没有被降级为备库，或者是关闭），仍旧能够对外提供服务。新的主库也可以对外提供服务。两个主库，产生双写问题，最为严重的问题。

(二) HA 到 Master 之间的连接不同，认为是网络问题，主库未 Crash。HA 选择不做任何操作。

但是，如果这时实际上确实是主库 Crash 了，HA 不做操作，数据库不对外提供服务。此时，双写问题避免了，但是应用的可用性受到了影响。

最后，数据库会出现问题，数据库之间的网络会出现问题，那么再考虑一层，HA 软件本身也有可能出现问题。如下图所示：



如果是 HA 软件本身出现了问题，怎么办？我们通过部署 HA，来保证数据库系统在各种场景下的持续可用，但是 HA 本身的持续可用谁来保证？难道我们需要为 HA 做主备，然后再 HA 之上再做另一层 HA？一层层加上去，子子孙孙无穷尽也 ……

其实，上面提到的这些问题，其实就是经典的分布式环境下的一致性问题（Consensus），近几年比较火热的 Lamport 老爷子的 Paxos 协议，Stanford 大学最近发表的 Raft 协议，都是为了解决这一类问题。（对 Raft 协议感兴趣的朋友，可以再看一篇 Raft 的动态演示 PPT：Understandable Distributed Consensus）

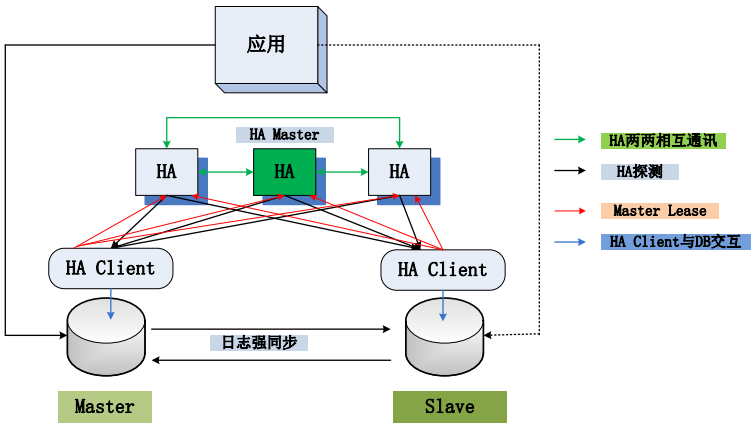
3 问题二：分区可用性

前面，我们回答了第一个问题，数据库如果不使用共享存储，能否保证主备数据的强一致？答案是肯定的：可以。但是，通过前面的分析，我们又引出了第二个问题：如何保证数据库在各种情况下的持续可用？至少前面提到的 HA 机制无法保证。那么是否可以引入类似于 Paxos，Raft 这样的分布式一致性协议，来解决上面提到的各种问题呢？

答案是可以的，我们可以通过引入类 Paxos，Raft 协议，来解决上面提到的各类问题，保证整个数据库系统的持续可用。考虑仍旧是两个数据库组成的主备强一致系统，仍旧使用 HA 进行主备监控和切换，再回顾一下上一节新引入的两个问题：

1) HA 软件自身的可用性如何保证？
2) 如果 HA 软件无法访问主库，那么这时到底是主库 Crash 了呢？还是 HA 软件到主库间的网络出现问题了呢？如何确保不会同时出现两个主库，不会出现双写问题？
3) 如何在解决上面两个问题的同时，保证数据库的持续可用？

为了解决这些问题，新的系统如下所示：



相对于之前的系统，可以看到这个系统的复杂性明显增高，而且不止一成。数据库仍旧是一主一备，数据强同步。但是除此之外，多了很多变化，这些变化包括：

1) 数据库上面分别部署了 HA Client；
2) 原来的一台 HA 主机，扩展到了 3 台 HA 主机。一台是 HA Master，其余的为 HA Participant；
3) HA 主机与 HA Client 进行双向通讯。HA 主机需要探测 HA Client 所在的 DB 是否能够提供服务，这个跟原有一致。但是，新增了一条 HA Client 到 HA 主机的 Master Lease 通讯。

这些变化，能够解决上面的两个问题吗？让我们一个一个来分析。首先是：HA 软件自身的可用性如何保证？

从一台 HA 主机，增加到 3 台 HA 主机，正是为了解决这个问题。HA 服务，本身是无状态的，3 台 HA 主机，可以通过 Paxos/Raft 进行自动选主。选主的逻辑，我这里就不做赘述，不是本文的重点，想详细了解其实现的，可以参考互联网上洋洋洒洒的关于 Paxos/Raft 的相关文章。总之，通过部署 3 台 HA 主机，并且引入 Paxos/Raft 协议，HA 服务的高可用可以解决。HA 软件的可用性得到了保障。

第一个问题解决，再来看第二个问题：如何识别出当前是网络故障，还是主库 Crash？如何保证任何情况下，数据库有且只有一个主库提供对外服务？

通过在数据库服务器上部署 HA Client，并且引入 HA Client 到 HA Master 的租约（Lease）机制，这第二个问题同样可以得到完美的解决。所谓 HA Client 到 HA Master 的租约机制，就是说图中的数据库实例，不是永远持有主库（或者是备库）的权利。当前主库，处于主库状态的时间是有限制的，例如：10 秒。每隔 10 秒，HA Client 必须向 HA Master 发起一个新的租约，续租它所在的数据库的主库状态，只要保证每 10 秒收到一个来自 HA Master 同意续租的确认，当前主库一直不会被降级为备库。

第二个问题，可以细分为三个场景：

- 场景一：主库 Crash，但是主库所在的服务器正常运行，HA Client 运行正常

  主库 Crash，HA Client 正常运行。这种场景下，HA Client 向 HA Master 发送一个放弃主库租约的请求，HA Master 收到请求，直接将备库提升为主库即可。原主库起来之后，作为备库运行。
- 场景二：主库所在的主机 Crash。（主库和 HA Client 同时 Crash）

此时，由于 HA Client 和主库同时 Crash，HA Master 到 HA Client 间的通讯失败。这个时候，HA Master 还不能立即将备库提升为主库，因为区分不出场景二和接下来的场景三（网络问题）。因此，HA Master 会等待超过租约的时间（例如：12 秒），如果租约时间之内仍旧没有续租的消息。那么 HA Master 将备库提升为主库，对外提供服务。原主库所在的主机重启之后，以备库的状态运行。

- 场景三：主库正常，但是主库到 HA Master 间的网络出现问题

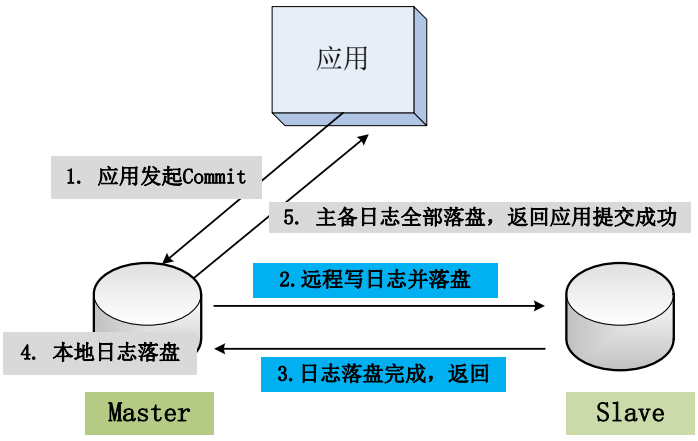
  对于 HA Master 来说，是区分不出场景二和场景三的。因此，HA Master 会以处理场景二同样的逻辑处理场景三。等待超过租约的时间，没有收到续租的消息，提升原备库为主库。但是在提升备库之前，原主库所在的 HA Client 需要做额外的一点事。原主库 HA Client 发送给 HA Master 的续租请求，由于网络问题，一直没有得到响应，超过租约时间，主动将本地的主库降级为备库。如此一来，待 HA Master 将原备库提升为主库时，原来的主库已经被 HA Client 降级为备库。双主的情况被杜绝，应用不可能产生双写。

同过以上三个场景的分析，问题二同样在这个架构下被解决了。而解决问题二的过程中，系统最多需要等待租约设定的时间，如果租约设定为 10 秒，那么出各种问题，数据库停服的时间最多为 10 秒，基本上做到了持续可用。这个停服的时间，完全在于租约的时间设置。

到这儿，基本可以说，要实现一个持续可用（分区可用性保证），并且保证主备数据强一致的数据库系统，是完全没问题的。在现有数据库系统上做改造，也是可以的。但是，如果考虑到实际的实现，这个复杂度是非常高的。数据库的主备切换，是数据库内部实现的，此处通过 HA Master 来提升主库；通过 HA Client 来降级备库；保证数据库崩溃恢复后，恢复为备库；通过 HA Client 实现主库的租约机制；实现 HA 主机的可用性；所有的这些，在现有数据库的基础上实现，都有着相当的难度。能够看到这儿，而且有兴趣的朋友，可以针对此问题进行探讨☺

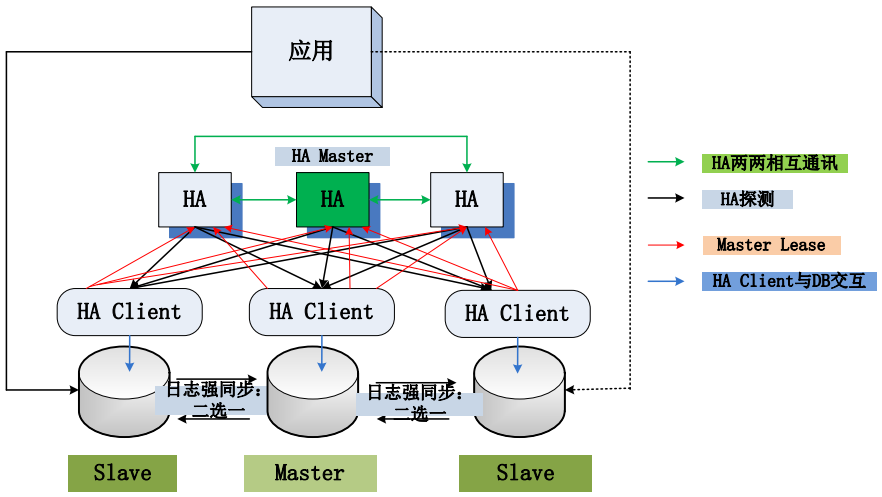
4 问题三：性能

数据一致性，通过日志的强同步，所有数据均可以解决。分区可用性，在出现任何异常情况时仍旧保证系统的持续可用，可用在数据强同步的基础上引入 Paxos/Raft 等分布式一致性协议来解决，虽然这个目前没有成熟的实现。接下来再让我们来看看一个很多朋友都很感兴趣的问题：如何在保证强同步的基础上，同时保证高性能？回到我们本文的第一幅图：



为了保证数据强同步，应用发起提交事务的请求时，必须将事务日志同步到 Slave，并且落盘。相对于异步写 Slave，同步方式多了一次 Master 到 Slave 的网络交互，同时多了一次 Slave 上的磁盘 sync 操作。反应到应用层面，一次 Commit 的时间一定是增加了，具体增加了多少，要看主库到备库的网络延时和备库的磁盘性能。

为了提高性能，第一个很简单的想法，就是部署多个 Slave，只要有一个 Slave 的日志同步完成返回，加上本地的 Master 日志也已经落盘，提交操作就可以返回了。多个 Slave 的部署，对于消除瞬时的网络抖动，非常有效果。在 Oracle 的官方建议中，如果使用最大保护模式，也建议部署多个 Slave，来最大限度的消除网络抖动带来的影响。如果部署两个 Slave，新的部署架构图如下所示：



新增一个 Slave，数据三副本。两个 Slave，只要有一个 Slave 日志同步完成，事务就可以提交，极大地减少了某一个网络抖动造成的影响。增加了一个副本之后，还能够解决当主库 Crash 之后的数据安全性问题，哪怕主库 Crash，仍旧有两个副本可以提供服务，不会形成单点。

但是，在引入数据三副本之后，也新引入了一个问题：主库 Crash 的时候，到底选择哪一个备库作为新的主库？当然，选主的权利仍旧是 HA Master 来行使，但是 HA Master 该如何选择？这个问题的简单解决可以使用下面的几个判断标准：

1. 日志优先。两个 Slave，哪个 Slave 拥有最新的日志，则选择这个 Slave 作为新的主库。
2. 主机层面排定优先级。如果两个 Slave 同时拥有最新的日志，那么该如何选择？此时，选择任何一个都是可以的。例如：可以根据 Slave 主机 IP 的大小进行选择，选择 IP 小的 Slave 作为新的主库。同样能够解决问题。

新的主库选择出来之后，第一件需要做的事，就是将新的 Master 和剩余的一个 Slave，进行日志的同步，保证二者日志达到一致状态后，对应用提供服务。此时，三副本问题就退化为了两副本问题，三副本带来的防止网络抖动的红利消失，但是由于两副本强同步，数据的可靠性以及一致性仍旧能够得到保障。

当然，除了这一个简单的三副本优化之外，还可以做其他更多的优化。优化的思路一般就是同步转异步处理，例如事务提交写日志操作；使用更细粒度的锁；关键路径可以采用无锁编程等。

多副本强同步，做到极致，并不一定会导致系统的性能损失。当然，极致应该是什么样子的？我的想法是：

- 对于单个事务来说，RT 增加。其响应延时一定会增加（至少多一个网络 RT，多一次磁盘 Sync）；
- 对整个数据库系统来说，吞吐量不变。远程的网络 RT 和磁盘 Sync 并不会消耗本地的 CPU 资源，本地 CPU 的开销并未增大。只要是异步化做得好，整个系统的吞吐量，并不会由于引入强同步而降低。

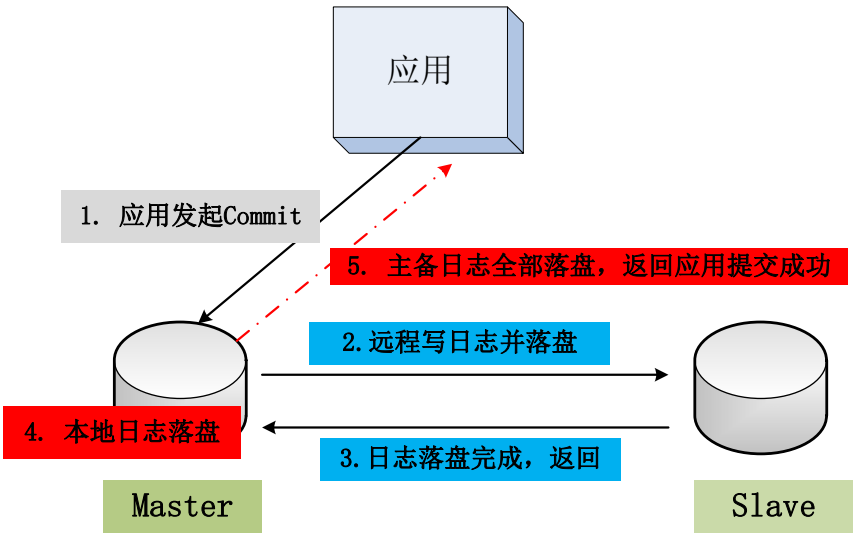
5 总结

洋洋洒洒写了一堆废话，最后做一个小小的总结：

- 能够看到这里的朋友，绝逼都是真爱，谢谢你们!!
- 各种主流关系型数据库系统是否可以实现主备的强一致，是否可以保证不依赖于存储的数据一致性？
  可以。Oracle 有，MySQL 5.7，阿里云 RDS，网易 RDS 都有类似的功能。
- 目前各种关系型数据库系统，能否在保证主备数据强一致的基础上，提供系统的持续可用和高性能？
  可以做，但是难度较大，目前主流关系型数据库缺乏这个能力。

6 问题四：一个极端场景的分析

意犹未尽，给仍旧在坚持看的朋友预留一个小小的作业。考虑下面这幅图：如果用户的提交操作，在图中的第 4 步完成前，或者是第 4 步完成后第 5 步完成前，主库崩溃。此时，备库有最新的事务提交记录，崩溃的主库，可能有最新的提交记录（第 4 步完成，第 5 步前崩溃），也可能没有最新的记录（第 4 步前崩溃），系统应该如何处理？

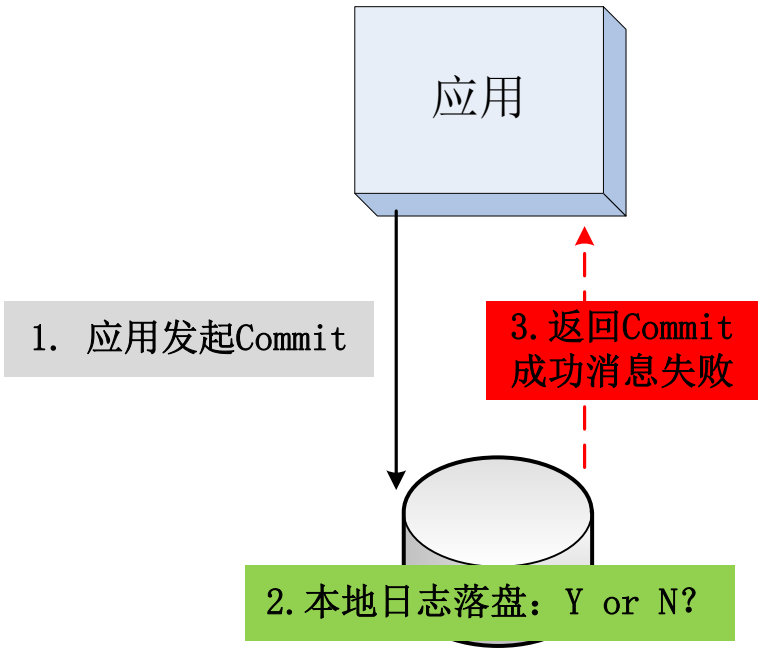


文章在博客上放出来之后，发现大家尤其对这最后一个问题最感兴趣。我选择了一些朋友针对这个问题发表的意见，仅供参考。

@淘宝丁奇

最后那个问题其实本质上跟主备无关。简化一下是，在单库场景下，db 本地事务提交完成了，回复 ack 前 crash，或者 ack 包到达前客户端已经判定超时…所以客户端只要没有收到明确成功或失败，临界事务两种状态都是可以接受的。主备环境下只需要保证系统本身一致。

将丁奇意见用图形化的方式表示出来，就是下面这幅图：



此图，相对于问题四简化了很多，数据库没有主备，只有一个单库。应用发起 Commit，在数据库上执行日志落盘操作，但是在返回应用消息时失败（网络原因？超时？）。虽然架构简化了，但是问题大同小异，此时应用并不能判断出本次 Commit 是成功还是失败，这个状态，需要应用程序的出错处理逻辑处理。

@ArthurHG

最后一个问题，关键是解决服务器端一致性的问题，可以让 master 从 slave 同步，也可以让 slave 回滚，因为客户端没有收到成功消息，所以怎么处理都行。服务器端达成一致后，客户端可以重新提交，为了实现幂等，每个 transaction 都分配唯一的 ID；或者客户端先查询，然后根据结果再决定是否重新提交。

其实，最终的这个问题，更应该由应用的同学来帮助解答：

如果应用程序在提交 Commit 操作，但是最后 Catch 到网络或者是超时的异常时，是怎么处理的？