基于 MySQL 复制进行高可用的避坑详解

场景一、从机故障

还是先看从机故障的场景，上篇文章已经明确：从机故障是不破坏“等效不丢”的。修复从机最简单直接的方法也是：故障过的从机直接放弃，然后根据现有主机中 MySQL 的数据，重新搭建一个新从机。在有自动化管理平台的情况下，这样的工作通常只需要点点鼠标，简单便捷，不易出错。DBA 可能会出于各种不同的想法或者一些客观原因，尝试去修复从机并继续使用，但在生产环境这样做是不理性的。一方面，故障过的服务器，如果没有从根本上解决掉问题，未来可能还会继续故障；另一方面，故障对 MySQL 从机会造成各种千奇百怪的影响，花样繁多，防不胜防，稍不注意，就可能为未来留下隐患。

tips

在使用半同步复制的场景，遇到从机故障时，控制主机行为的关键参数是 rpl_semi_sync_master_timeout 、 rpl_semi_sync_master_wait_no_slave 、 rpl_semi_sync_master_wait_for_slave_count，这几个参数需要根据业务要求慎重决策与配置。

tips

质量可靠的服务器虽然不便宜，但是和硬件不稳定带来的如下等潜在的问题与风险比，确是非常实惠的。

1、不定期不定时的短时业务异常或中断；

2、业务处理不完美时的脏数据、用户投诉、技术团队被迫分心追查跟踪异常业务；

3、反复重建高可用环境；

4、长期在关键业务使用有隐患的服务器的风险、高可用方案长期\反复降级的风险带来的心理压力。

笔者特别推荐在机房中，根据在用服务器数量、故障率、采购周期等经验数据，预留一定数量的“预备服务器”，并且有可能的话，最好是预留少量各方面硬件配置均为最高规格的服务器作为“全通用预备服务器”，这样的服务器才能够随时有效替代任意场景的故障服务器，包括支撑意外的业务上线需求等等。发生过故障的服务器，可以用于非关键场景，比如测试环境等等。

既然谈到了服务器，笔者个人非常不建议在虚拟化或者云环境运行数据库。一方面虚拟化环境存在更多 bug；另一方面虚拟化以后，在数据库的磁盘 IO 多、网络 IO 多的场景下性能损耗很大；以及如果不直接掌握宿主机，则性能、稳定性问题都难以分析和排查。

关于云服务器的成本，千万不能只看单台云主机很便宜，要考虑到：虚拟化、共用、超售带来的云主机性能低于物理服务器的问题。不同的云性能不同，甚至同一个云性能也随时间不断变化；因此只有实测，才能知道需要多少台云主机抵得上一台物理机。云主机数量膨胀所带来的管理复杂性成本、云主机单机性能差导致被迫使用更复杂的整体技术方案的成本，都应当综合考量。

我们从探讨技术的角度出发，研究一下从机的状态。从机经历了崩溃恢复的流程，于是它的状态肯定和 log_slave_updates 、 master_info_repository 、 relay_log_info_repository 、 relay_log_recovery 等几个重要参数和复制是否基于 gtid 的自动位置都有关。

前文已经提到打开 relay_log_recovery 的问题，它在打开 relay_log_purge（默认打开，crash-safe

的复制也要求打开）时，可能导致已经接收到的 relaylog 在没有被 SQL 线程应用的情况下被删除。这样会破坏基于半同步复制做高可用所依赖的基础，考虑到这一点，笔者个人决策是放弃使用 crash-safe 的复制。同时，一般而言 relaylog 本身是不会配置持久化的（sync_relay_log 不等于 1），因此，出现操作系统及更底层层次的故障时，因为 relaylog 损坏导出现一些复制方面的报错也是这个决策下不得不接受的缺陷。当然，这种删除 relaylog 破坏半同步进而导致严重后果的危险，只出现在主从机同时出现问题的场景，它本身已超出我们设想的高可用方案容许的灾难级别；使用 crash-safe 的复制可以减轻一些处理复制故障的工作量，有一定微小的收益；但反过来思考，继续使用这个从机同时也意味着承担服务器再次故障的隐患风险；因此最终是否坚持使用 crash-safe 的复制，还是看 DBA 自己的想法。

尽管笔者放弃了 crash-safe 的复制，但是还是可以尽量配置好一个从机，来降低遇到故障时的损害程度：如果我们设置 log_slave_updates=on 、 master_info_repository=TABLE 、relay_log_info_repository=TABLE，那么从机在顺利执行崩溃恢复后，gtid 位置信息是正确的，复制执行到的位置信息也是正确的，尽管 relaylog 可能损坏，但是损坏的 relaylog 并不会被执行。

tips

在对性能的影响有限的范围内，通过合理配置参数尽力降低从机遇到故障时受到的损害有一定价值：它可以降低修复从机的难度，在最坏情况发生时多提供一份可能性。同时在损害不是很严重时，恢复速度一般比较快；反观搭建新的从机，需要不少时间，尤其数据量很大时（还可能需要等待维护窗口、新服务器上架等）。

tips

“顺利执行崩溃恢复”，指不使用 innodb_force_recovery、innodb_force_load_corrupted 等强制参数与 tc-heuristic-recover 等人工干涉手段，MySQL 实例可以自行执行完成崩溃恢复流程的情况。

tips

损坏的 relaylog 不会执行的原因是 binlog_checksum、slave_sql_verify_checksum 默认开启。

tips

电池\电容后备的 RAID 卡，可能在数据是否已写入到磁盘上，欺骗操作系统。如果故障和 RAID 卡有关（包括停电太久），可能会出现配置与理论上没问题，但是还是发现数据没有落盘（包括文件系统也可能会出现一些奇怪的问题）。

基于上述配置的前提，在 relaylog 确实损坏的情况下，DBA 可以短暂地取消主机半同步复制的设置，然后在从机上删除复制关系，再使用自动位置的方式，重新完整配置半同步复制关系即可。但是在重新投入生产前，应仔细核对主从数据的一致性。

tips

重新配置复制关系前，应核对主机上 gtid_purged，避免找不到 binlog。

场景二、主机故障

讲完从机故障，我们来说主机故障的场景。

主机故障后，如果可以恢复运行，最大的风险在于，从机的复制会自动重连。从机重连以后会尝试继续开始复制，并且很可能可以获取到少量之前没有接收到的 binlog。表面上看，主机崩溃恢复以后，它的状态和 binlog 状态是一致的，从机接收到这些 binlog 以后，将它应用掉，主从数据一致，单纯从数据库的角度去看，没有什么问题。但是我们回顾一下“等效不丢”和我们的高可用切换方法，再考虑到主机崩溃恢复多少需要一些时间，一般慢于从机追复制，我们又会发现一个坑。这种场景下，特定的“少量之前没有接收到的 binlog”均属于“已

经发出提交操作但没有收到提交成功的事务”，根据我们上一篇文章，这样的事务是须要由应用 app 检查，然后根据数据状态判断事务是提交了还是回滚了，并且对它进行相应补偿处理的。

举一个会发生问题的例子：T1 事务在 10:00 分提交，但是没有收到提交成功的信息，主机就崩溃了；假定 T1 的实际情况为成功写入 binlog，但是没有发给从机（只要同时有很多事务并发处理，或者多次尝试这个场景，肯定会有这样的事务）；然后在 10:05 分，成功切换到从机，从机中没有 T1 的数据；10:10 分，应用 app 对事务的状态进行了判断，发现事务不存在，因此执行了补偿事务，T2；10:15 分，主机恢复运行了，主机崩溃恢复依赖 binlog，因此主机提交了 T1 事务；10:20 分从机重连，获取到 T1 事务的 binlog 并应用。至此，我们在从机上，发现 T1 和它的补偿事务 T2 同时存在；如果是普通主从复制，主机上存在 T1，如果是双主单活复制，则主机上也是 T1 和 T2 同时存在。（还有因为主键冲突等原因导致复制报错的可能性）这个例子中，单纯从数据库的角度看，数据没错，但是业务逻辑角度则是出错了。

我们怎么解决这个问题呢，不进行补偿?

不行，会有一些事务没有成功写入 binlog。

延迟补偿的时间？

我们不知道主机恢复最久要多久，也不知道多久时间以后，主机一定不会再恢复，调整应用 app 处理逻辑，不能从根本上解决问题。

我们仔细分析上面的例子，不难发现，其实是应用 app 做出正确补偿处理以后，MySQL 数据库的复制机制，后续又“背着应用 app 偷偷篡改了”数据，导致最终结果违背业务逻辑。那解决方案也就有了，在切换到从机以后，我们要停掉从机的复制，才能开放从机给应用 app 访问。

tips

1、极少有高可用方案正确处理了这个问题。

2、DBA 需要掌握业务逻辑，才能知道如何去对待一些特殊情况的数据。单纯的数据库角度的数据正确，并不代表业务逻辑角度的正确。笔者建议，修数据、补数据这样的工作，一定要跟开发一起，共同处理，尤其是业务系统庞大，子系统很多的时候。

关于主机的恢复，建议做法同样是放弃掉现有的内容。将现在的从机作为未来的主机，然后新搭建未来的从机，再组建、配置未来新的高可用关系等等。

从技术探讨角度出发，分析一下主机在执行崩溃恢复流程以后的数据的状态：正常而言，主机执行崩溃恢复流程以后，它的数据和 binlog 保持一致，但是数据可能比从机多，也可能比从机少。数据比从机多，较为容易理解，binlog 没有来得及传输到从机就会发生这样的情况。数据比从机少，则往往发生在 sync_binlog 不等于 1 的场景；binlog 末尾丢失\损坏，就会出现主机数据少于从机的情况。

假设出现数据比从机多的情况，基本只能采取重新搭建的方式处理。出现数据少于等于从机的情况，则需要特别注意，先修改自己的 server-id（建议也修改 server-uuid）以后，再通过复制的方式补全数据。

同时，经过探讨，几个问题的答案就呼之欲出了：

崩溃恢复以后的 MySQL 实例，不考虑故障后续再写入的数据，能否保证故障实例和现存的实例数据一致？

分析 MySQL 执行崩溃恢复的过程，我们可以知道，崩溃恢复处理中，决定一个事务是否存

在的关键在于有没有对应的 binlog。从原理上思考，半同步复制、双 1 等，并没有办法去保证主机上 binlog 有的内容，一定被从机收到；或者是保证，被从机收到的事务，其 binlog 一定不会丢失（即使使用双 1，还有 RAID 卡没电等场景）；以及主机崩溃恢复时，只考虑自己 binlog 中有没有事务，不会考虑从机的情况。因此主机崩溃恢复的场景，必然做不到保证主机崩溃恢复后，数据和从机相同。

从机崩溃恢复的场景，同样只考虑自己 binlog 中的内容，不会考虑主机的情况，单纯的崩溃恢复，同样是做不到保证崩溃恢复后，数据和主机相同。但是从机崩溃的场景，只会缺少事务，如果加上正确处理复制相关事项，则从机可以补足缺少的事务，后续做到一致。

MGR 为什么可以保证一致？

MGR 的崩溃恢复机制实际上分为 MySQL 实例本身在启动时的崩溃恢复，和加入集群时的“Distributed Recovery”。加入集群时的恢复处理，考虑到了集群中的数据情况，遇到真正数据有异常的情况，实例是无法加入集群的（gtid 情况异常的 MySQL 实例可以启动运行，但是无法正常加入 MGR 集群），能成功重新加入集群的都是数据正常，或者缺少数据并且可以补回来的。

tips

MGR 看上去非常美好，但是 MGR 本身和它的配套高可用方案还没有经过足够的考验；稳定性、性能等还略有些不足之处；对网络环境要求也非常高；生产环境使用 MGR，需要非常谨慎小心。笔者建议只使用 single-primary 模式；多检查应用报错信息，MySQL 报错信息；以及多校验各个成员间的数据一致性。