mysql如何保证数据一致性

个人总结：

- 普通InnoDB
  - 采用日志先行策略，数据变更发生在内存中，达到一定条件再写入硬盘
  - 倘若宕机内存消失，数据库复原会找到redolog日志重做
- 主从分离确保
  - Master事务提交写到binlog，控制参数刷新到磁盘，备库读取，然后记录到自己relay_log，再应用到slave中
- 在复杂环境中出现XA事务---确保Innodb中redo日志和Master中binlog二进制文件的一致。
  - 开启事务，写prepare在redo日志，先写binlog 最后提交信息至redo中
  - 如果innodb和主从日志刷磁盘都非实时，容易造成主从数据不同步

1.MySQL数据库层丢数据场景

本节我们主要介绍一下在存储引擎层上是如何会丢数据的。

1.1. InnoDB丢数据

InnoDB支持事务，同Oracle类似，事务提交需要写redo、undo。**采用日志先行的策略，将数据的变更在内存中完成**，并且将事务记录成redo，**顺序的写入redo日志中**，即表示该事务已经完成，就可以返回给客户已提交的信息。**但是实际上被更改的数据还在内存中**，并没有刷新到磁盘，即还没有落地，**当达到一定的条件，会触发checkpoint**，将内存中的数据（page）合并写入到磁盘，这样就减少了离散写、IOPS，提高性能。

在这个过程中，如果服务器宕机了，内存中的数据丢失，**当重启后，会通过redo日志进行recovery重做。确保不会丢失数据**。因此只要redo能够实时的写入到磁盘，InnoDB就不会丢数据。

```
先来看一下innodb_flush_log_at_trx_commit这个参数：
= 0 ：每秒 write cache & flush disk
= 1 ：每次commit都 write cache & flush disk
= 2 ：每次commit都 write cache，然后根据innodb_flush_log_at_timeout（默认为1s）时间 flush d
```

从这三个配置来看，**显然innodb_flush_log_at_trx_commit=1最为安全**，因为每次commit都保证redo写入了disk。但是这种方式性能对DML（数据库管理语句）性能来说比较低，在我们的测试中发现，**如果设置为2，DML性能要比设置为1高10倍左右。**

在某些DML操作频繁的场景下，库的innodb_flush_log_at_trx_commit需要设置为2，这样就存在丢数据的风险：当服务器出现宕机，重启后进行crash recovery则会丢失innodb_flush_log_at_timeout秒内的数据。

PS:当开启了内部XA事务（默认开启），且开启binlog，情况稍有不一样，后面会进行介绍。

1.2.MyISAM丢数据

MyISAM存储引擎在我们的生产中用的并不多，但是系统的数据字典表元数据等都是存储在MyISAM引擎下。

MyISAM不支持事务，且没有data cache，所有DML操作只写到OS cache中，flush disk操作均由OS来完成，因此如果服务器宕机，则这部分数据肯定会丢失。

2.主从复制不一致

主从复制原理：**MySQL主库在事务提交时写binlog，并通过sync_binlog参数来控制binlog刷新到磁盘“落地”**，而备库通过IO线程从主库读取binlog，并记录到本地的relay log中，由本地的SQL线程再将relay log的数据应用到本地数据库。

在主从环境中，增加了binlog，这就增加了环境的复杂性，因此也增加了丢数据以及数据不一致可能。

在分析这些丢数据的可能性之前，我们先了解一下binlog的刷新机制以及MySQL的内部XA事务是如何保证binlog与redo的一致性的。

2.1.binlog刷新机制

```
master写binlog与innodb引擎写redo类似，也有参数控制：sync_binlog
      = 0 ：表示MySQL不控制binlog的刷新，由文件系统自己控制它的缓存的刷新
      > 0 ：表示每sync_binlog次事务提交，MySQL调用文件系统的刷新操作将缓存刷下去
```

其中**最安全的就是=1，表示每次事务提交，MySQL都会把binlog缓存刷下去**，这样在掉电等情况下，系统才有可能丢失1个事务的数据。当sync_binlog设置为1，对系统的IO消耗也是非常大的。

2.2.内部XA事务原理

MySQL的存储引擎与MySQL服务层之间，或者存储引擎与存储引擎之间的分布式事务，称之为内部XA事务。最为常见的内部XA事务存在与binlog与InnoDB存储引擎之间。**在事务提交时，先写二进制日志，再写InnoDB存储引起的redo日志。**对于这个操作要求必须是原子的，即需要保证两者同时写入，内部XA事务机制就是保证两者的同时写入。

XA事务的大致流程：

1. 事务提交后，InnoDB存储引擎会先做一个PREPARE操作，将事务的UXID写入到redo日志中。
2. 写binlog日志
3. 再将该事务的commit信息写到redo log中

如果在步骤1和步骤2失败的情况下，整个事务会回滚，如果在步骤3失败的情况下，MySQL数据库在重启后会先检查准备的UXID事务是否已经提交，若没有，则在存储引擎层再进行一次提交操作。这样就保证了redo与binlog的一致性，防止丢数据。

2.3.master库写redo、binlog不实时丢数据的场景

上面我们介绍了MySQL的内部XA事务流程，但是这个流程并不是天衣无缝的，redo的ib_logfile与binlog日志如果被设置非实时flush，就有可能存在丢数据的情况。

1.redo的trx_prepare未写入，但binlog写入，造成从库数据量比主库多。

2.redo的trx_prepare与commit都写入了，但是binlog未写入，造成从库数据量比主库少。

从目前来看，**只能牺牲性能去换取数据的安全性，必须要设置redo和binlog为实时刷盘**，如果对性能要求很高，则考虑使用SSD

2.4.slave库写redo、binlog不实时丢数据的场景

master正常，但是slave出现异常的情况下宕机，这个时候会出现什么样的情况呢？如果数据丢失，slave的SQL线程还会重新应用吗？这个我们需要先了解SQL线程的机制。

slave读取master的binlog日志后，需要落地3个文件：relay log、relay log info、master info：

relay log：即读取过来的master的binlog，内容与格式与master的binlog一致

relay log info：记录SQL Thread应用的relay log的位置、文件号等信息

master info：记录IO Thread读取master的binlog的位置、文件号、延迟等信息

因此如果当这3个文件如果不及时落地，则主机crash后会导致数据的不一致。

在MySQL 5.6.2之前，slave记录的master信息以及slave应用binlog的信息存放在文件中，即master.info与relay-log.info。在5.6.2版本之后，允许记录到table中，参数设置如下：

```
master-info-repository  = TABLE
relay-log-info-repository = TABLE
```

对应的表分别为mysql.slave_master_info与mysql.slave_relay_log_info，且这两个表均为innodb引擎表。

master info与relay info还有3个参数控制刷新：

- sync_relay_log：默认为10000，即每10000次sync_relay_log事件会刷新到磁盘。为0则表示不刷新，交由OS的cache控制。
- sync_master_info:若master-info-repository为FILE，当设置为0，则每次sync_master_info事件都会刷新到磁盘，默认为10000次刷新到磁盘；若master-info-repository为TABLE，当设置为0，则表不做任何更新，设置为1，则每次事件会更新表 #默认为10000
- sync_relay_log_info：若relay_log_info_repository为FILE，当设置为0，交由OS刷新磁盘，默认为10000次刷新到磁盘；若relay_log_info_repository为TABLE，且为INNODB存储，则无论为任何值，则都每次evnet都会更新表。

建议参数设置如下：

```
sync_relay_log = 1
sync_master_info = 1
sync_relay_log_info = 1
master-info-repository  = TABLE
relay-log-info-repository = TABLE
```

当这样设置，导致调用fsync()/fdatasync()随着master的事务的增加而增加，且若slave的binlog和redo也实时刷新的话，会带来很严重的IO性能瓶颈。

2.5.master宕机后无法及时恢复造成的数据丢失

当master出现故障后，binlog未及时传到slave，或者各个slave收到的binlog不一致。且master无法在第一时间恢复，这个时候怎么办?

如果master不切换，则整个数据库只能只读，影响应用的运行。

如果将别的slave提升为新的master，那么原master未来得及传到slave的binlog的数据则会丢失，并且还涉及到下面2个问题。

1.各个slave之间接收到的binlog不一致，如果强制拉起一个slave，则slave之间数据会不一致。

2.原master恢复正常后，由于新的master日志丢弃了部分原master的binlog日志，这些多出来的binlog日志怎么处理，重新搭建环境?

对于上面出现的问题，一种方法是确保binlog传到从库，或者说保证主库的binlog有多个拷贝。第二种方法就是允许数据丢失，制定一定的策略，保证最小化丢失数据。

1.确保binlog全部传到从库

方案一：使用semi sync（半同步）方式，事务提交后，必须要传到slave，事务才能算结束。对性能影响很大，依赖网络适合小tps系统。

方案二：双写binlog，通过DBDR OS层的文件系统复制到备机，或者使用共享盘保存binlog日志。

方案三：在数据层做文章，比如保证数据库写成功后，再异步队列的方式写一份，部分业务可以借助设计和数据流解决。

2.保证数据最小化丢失

上面的方案设计及架构比较复杂，如果能容忍数据的丢失，可以考虑使用淘宝的TMHA复制管理工具。

当master宕机后，TMHA会选择一个binlog接收最大的slave作为master。当原master宕机恢复后，通过binlog的逆向应用，把原master上多执行的事务回退掉。

3. 总结

通过上面的总结分析，MySQL丢数据的场景是五花八门，涉及到单库的丢数据场景、主从的丢数据场景以及MySQL内部XA事务原理等，相对还比较复杂，有点难以理解。

只有当我们了解了这些丢数据的场景，才能更好的去学习， 并解决这些问题。

根据分布式领域的CAP理论（Consistency、Availability、Partition tolerance），任何的分布式系统只能同时满足2点，没办法三者兼顾。MySQL的主从环境满足Availability，且主从互不干扰，因此满足Partition tolerance，但是不满足Consistency，如果需要满足Consistency，则肯定会失去Partition tolerance，因此实现100%高可用性的MySQL主从架构还是非常困难的。只能通过一些设计去牺牲部分特性去满足另外的特性。