搜狐
SOHU.com

Percona xTRAdb cLUSERT
运维实践

Sohu 徐国强

www.sohu.com

XtraDB Cluster 特征

1， 同步复制

复制动作是同步的，实际上数据并不是完全同步的，数据的同步存在一个间隙，只能称为虚同步。

2， 多master

每一个节点都可以作为master，并将改动发送到其他节点。

3，并行复制

复制可以指定多个线程，并且复制是以事务为单位的，多个事务同时并行推送到所有集群节点。

4，新节点自动部署

只需要修改合适的参数，启动新节点的mysqld进程并成功加入集群后，数据完全自动的部署到新节点。

5，数据一致性

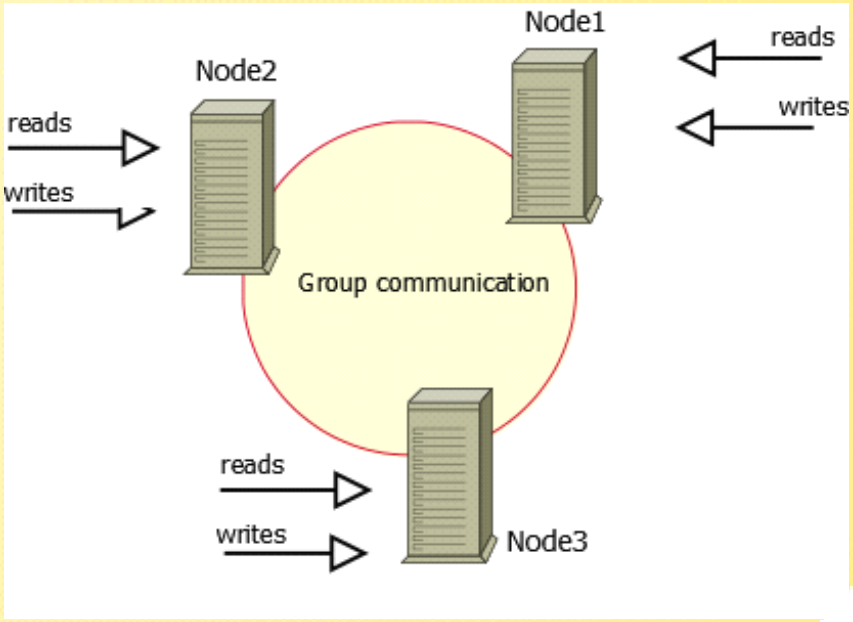
严格的数据一致

6，高可用性

单点故障不影响可用性

7，与传统mysql几乎完全兼容

数据可以直接使用不需要任何转换，程序上也仅仅事务处理机制有变化，并且还可以完全规避



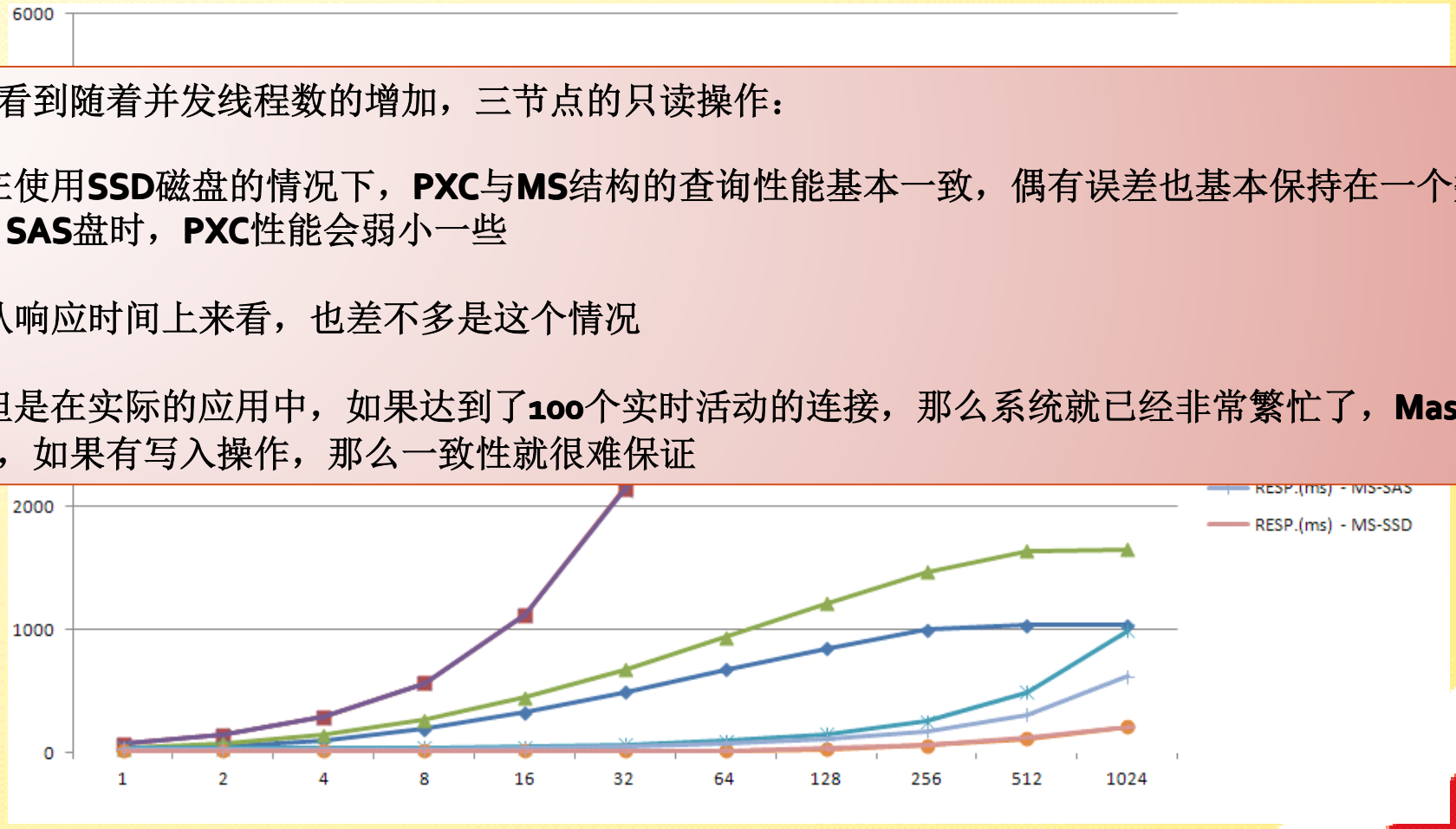
XtraDB Cluster 缺陷

- 1，默认工作在InnoDB引擎表上，因此对其他引擎的表支持的很差，甚至根本不支持，所以不要考虑在PXC上使用MyISAM或者其他的存储引擎
- 2，所有的表都必须要有主键
- 3，不支持的操作： LOCK/UNLOCK TABLES、 lock functions (GET_LOCK(), RELEASE_LOCK()...)
- 4，query log日志不能存放在表里面，必须存放在文件
- 6，由于集群是基于乐观的并发控制（ optimistic concurrency control），事务冲突的情况可能会在commit阶段发生
- 7，不支持XA事务，因为XA事务有可能在commit的时候出现异常发生rollback
- 8，整个集群的吞吐量/性能取决于最慢的那个节点（成本）
- 9，最小建议的集群节点数为3，否则很容易产生脑裂（成本）
- 10，加入新节点，开销大，有多少个节点就有多少重复的数据
- 11，不能有效的解决写缩放问题，所有的写操作都将发生在所有节点上

XtraDB Cluster 性能--测试环境

- 1，PXC 和 Master-Slave 均为3个节点
- 2，SAS 、SSD 磁盘
- 3，Percona XtraDB Cluster 5.5.28
- 4，Percona Sever 5.5
- 5，Oracle Linux 6.3
  - 2.6.39-200.24.1.el6uek.x86_64
- 6，DELL R710
  - CPU Xeon 5620 * 2
  - Memory 64GB
- 7，Sysbench
- 8，haproxy 和 LVS

XtraDB Cluster 性能—LVS 只读测试



可以看到随着并发线程数的增加，三节点的只读操作：

1，在使用**SSD**磁盘的情况下，**PXC**与**MS**结构的查询性能基本一致，偶有误差也基本保持在一个数量级上； **SAS**盘时，**PXC**性能会弱小一些

2，从响应时间上来看，也差不多是这个情况

3，但是在实际的应用中，如果达到了**100**个实时活动的连接，那么系统就已经非常繁忙了，**MasterSlav**结构，如果有写入操作，那么一致性就很难保证

搜狐 SOHU.com

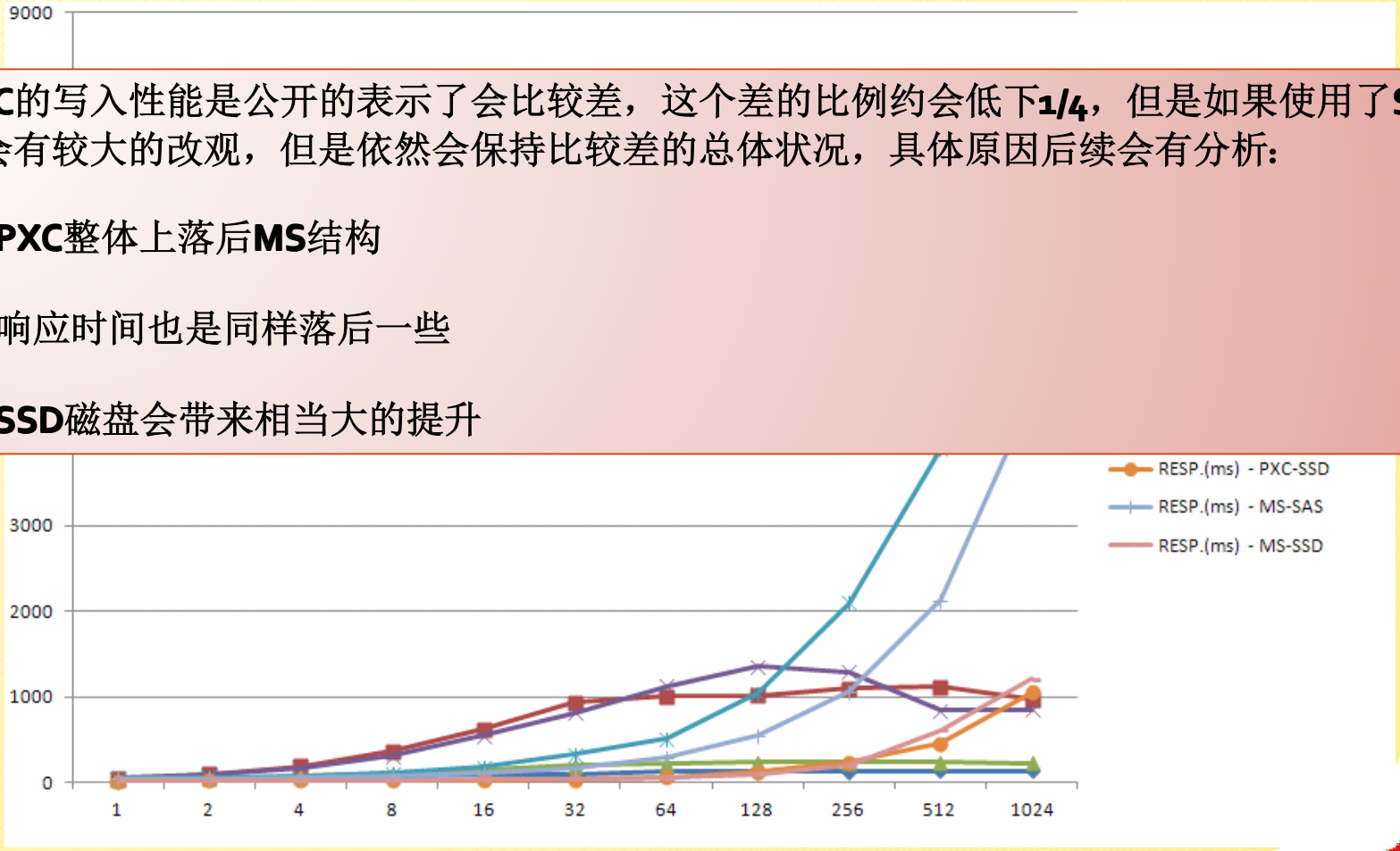
XtraDB Cluster 性能—LVS 单节点写测试

PXC的写入性能是公开的表示了会比较差，这个差的比例约会低下1/4，但是如果使用了SSD磁盘，则会有较大的改观，但是依然会保持比较差的总体状况，具体原因后续会有分析：

1，PXC整体上落后MS结构

2，响应时间也是同样落后一些

3，SSD磁盘会带来相当大的提升

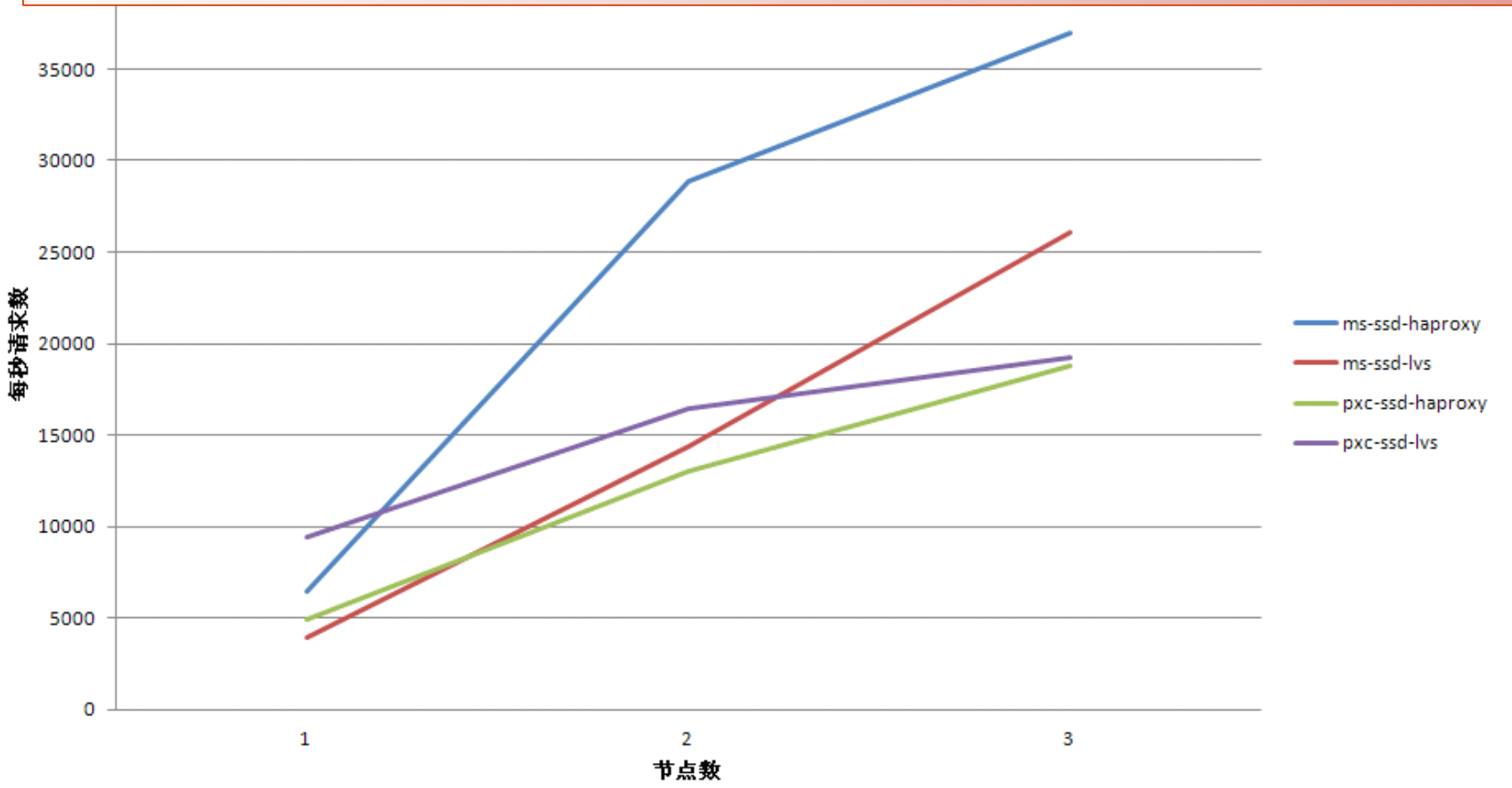


XtraDB Cluster 性能—业务模拟测试

- 1，同时启2个sysbench，一个只读，另一个只写
- 2，写请求只发送到某一个固定节点，读请求负载均衡到集群环境的所有节点
- 3，2个sysbench的读写请求数比例设置约为9：1，保证测试时间约等于或大于1小时
- 4，数据量为1.5亿
- 6，依次对pxc/ms,lvs/haproxy各种环境进行测试，共4项：pxc-ssd-lvs, pxc-ssd-haproxy, ms-ssd-lvs, ms-ssd-haproxy
- 7，ms测试导致主备不同步，暂时忽略其对测试的影响。需等待ms同步后再进行下一项测试
- 8，分别对集群环境中存在单节点、2节点、3节点测试3组
- 9，每组测试5轮。每秒请求数计算：SUM(读请求数+写请求数)/SUM(读执行时间+写执行时间), GROUP BY 测试项

XtraDB Cluster 性能—业务模拟测试

如果排除延迟问题，**MS**确实比**PXC**的性能更好（但是实际情况下，**MS**的延迟已经非常严重，测试了一小时，基本上延迟越来越大）

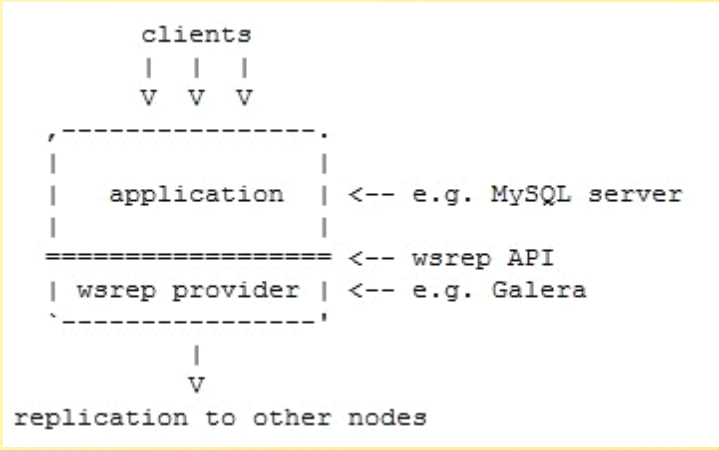


基于Galera实现

Percona的这个Cluster实际上就是基于Galera实现，添加了一些mysql的参数，并调用Galera的接口，这是整体的工作流程：

1，客户端提交MySQL数据库访问请求

2，通过wsrepAPI调用Galera将数据变化复制到集群中其他节点



数据复制流程（一）

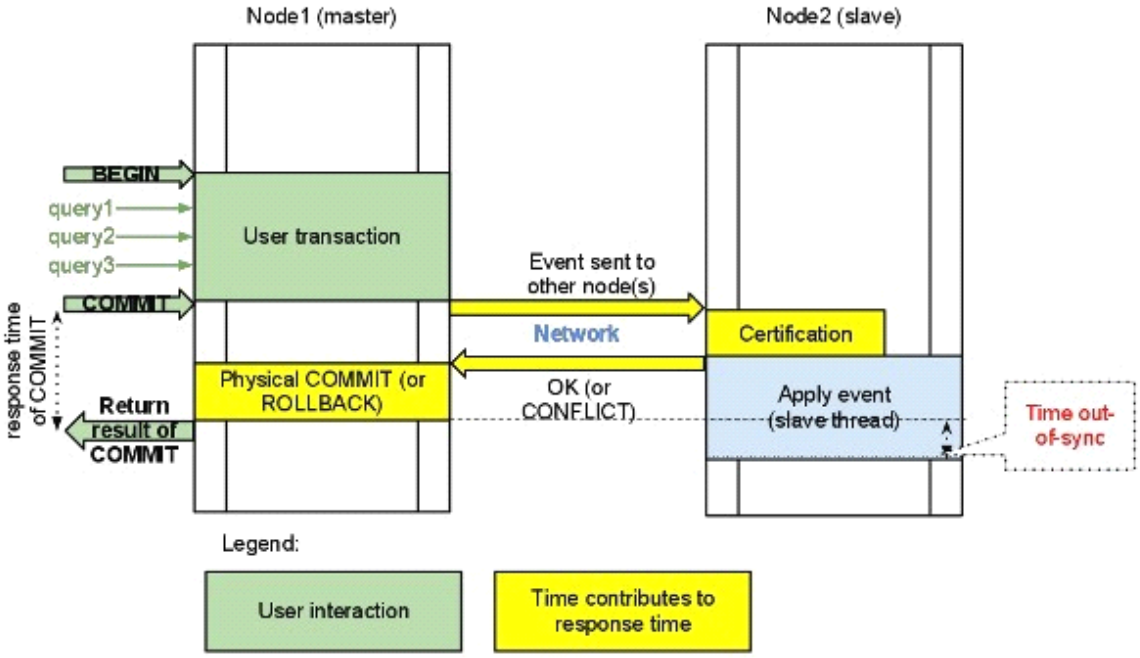
描述两个节点间的数据复制。

要点：

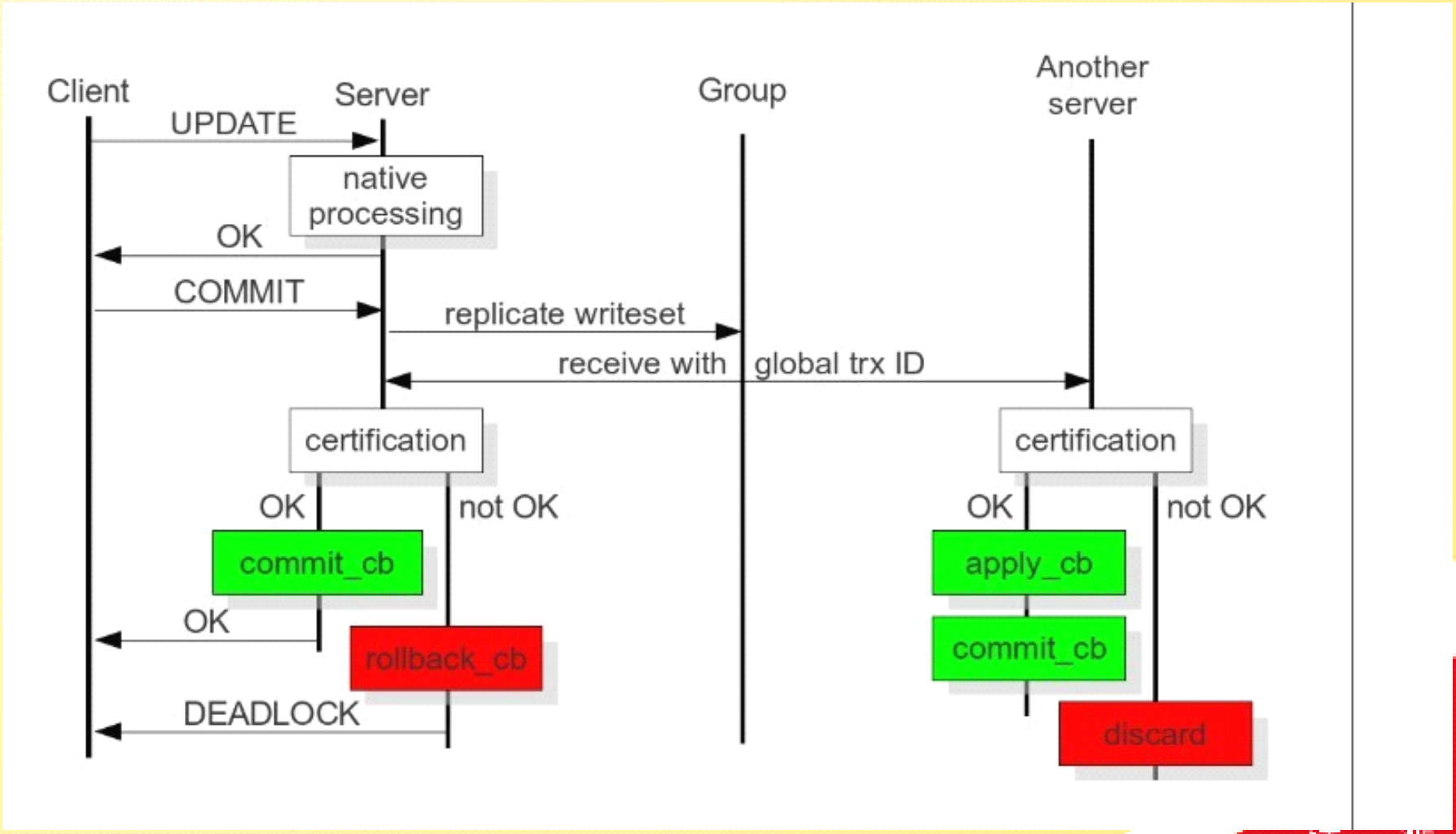
1，只有当发生Commit操作时，才发送数据验证请求到其他节点。

2，传输/返回验证数据时间和其他节点的数据验证时间，将影响到前数据操作节点真正commit的时刻。

3，其他节点只要验证成功了，就会返回成功的信号，即使当前数并没有真正的写入当前节点，这时间内，将造成数据的不一致。



数据复制流程（二）



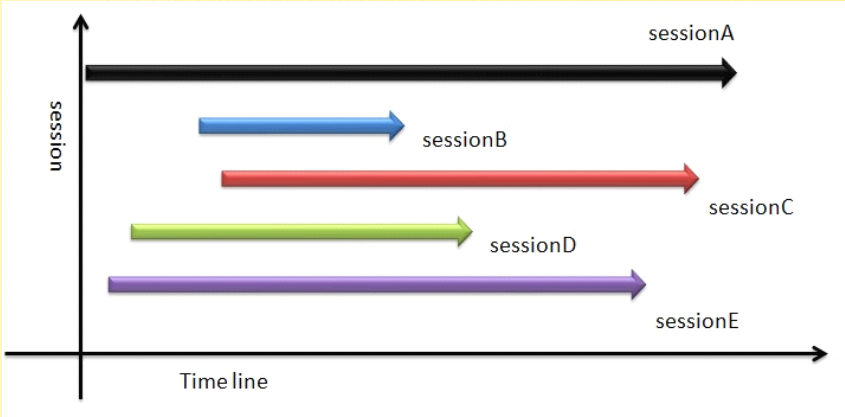
数据复制流程（三）

根据前面描述的数据复制流程，可以得到这样的结论：

当多个事务同时操作相同的数据资源时，这个资源在集群中是不受任何一个Session影响的，直到有一个Session对这个数据资源进行了成功的Commit操作，这时，其他的Session的所有操作实际上已经不可能成功了，当其他的事务尝试做Commit，会直接返回一个因为deadlock事务失败回滚的信息。

这与mysql默认的机制不同，在mysql innodb默认的情况下，当我们在其他事务中对某个id的数据进行update；此时我们发起一个事务对这个数据进行需要获得排它锁的操作，操作将会进行等待，直到超时失败或者现在持有排它锁的事务提交，当前事务将继续。

(注意：这里的每个Session来源于不同节点，单节点的死锁机制遵循mysql innodb默认的工作模式)



那么上图的session中，那些会成功呢？

只有SessionB

节点状态变化

节点变化过程中用到的部分术语：

SST（State Snapshot Transfer）

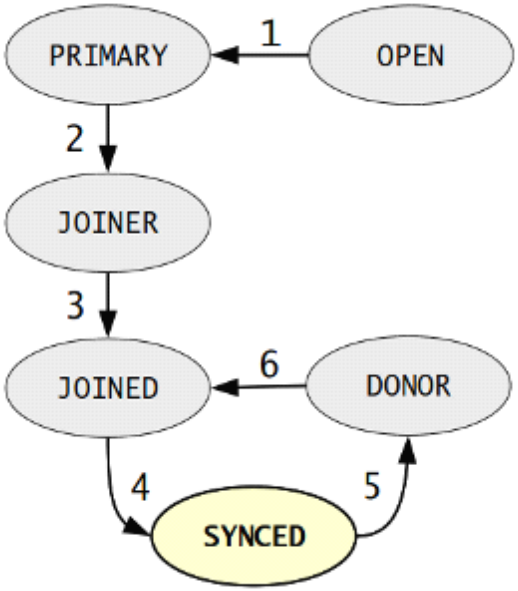
用于节点间传送数据，发生在节点初始化，或者节点故障需要全部重置数据的时候，相当于整个copy一份数据到新节点，这个过程影响非常大，会造成Donor节点的无法访问

IST（Increment Snapshot Transfer）

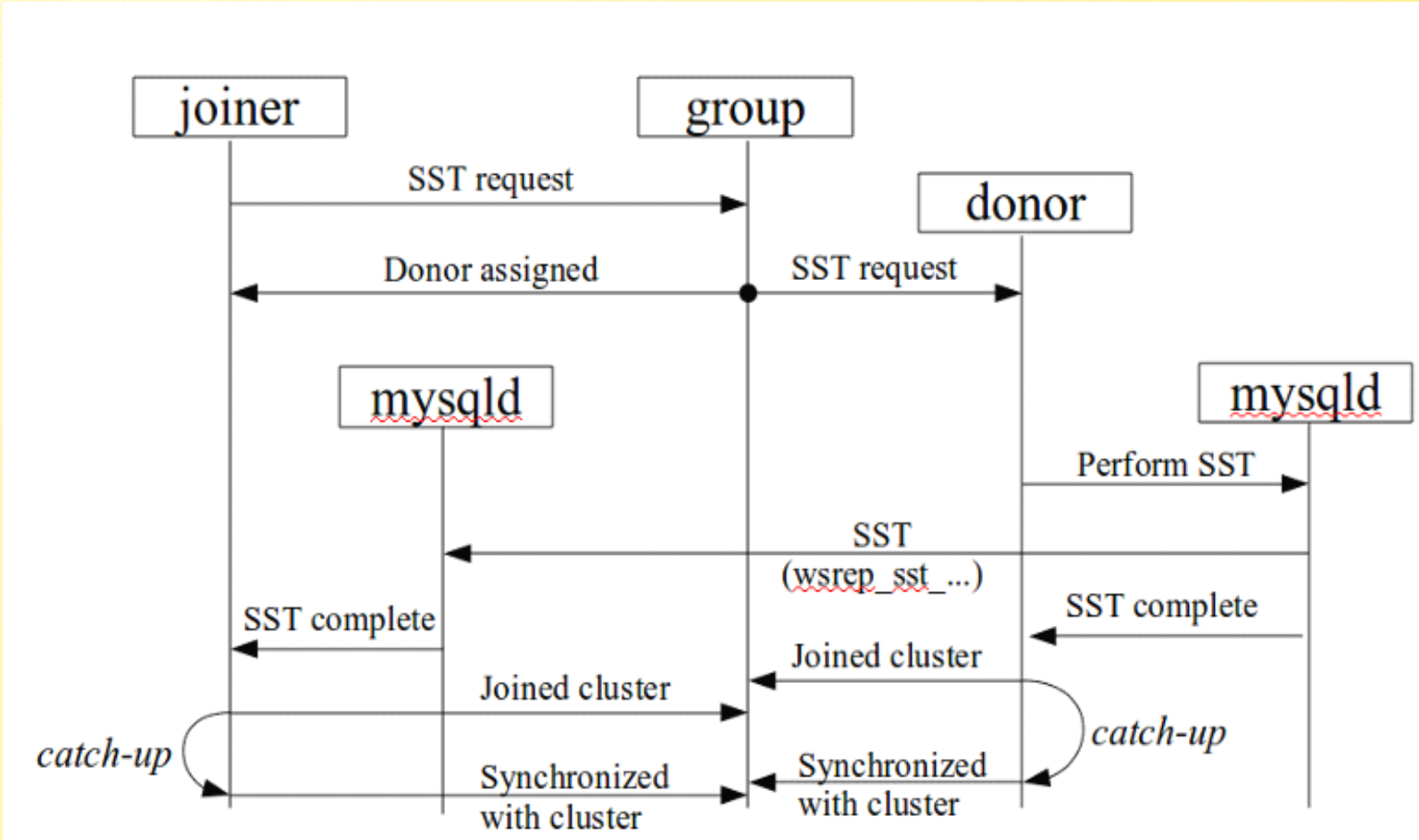
用于节点间传送数据，发生在节点初始化，或者节点故障，但是能够从galera.dat中获得增量同步点的情况，仅仅做必要的增量同步，是最理想的数据恢复方法。

Donor

当发生SST时，集群中被选中作为数据源的节点，可以手动指定也可以自动选择，被选中为Donor的节点可以进行Select，但是新的数据变化情况无法被应用，会被缓存在当前节点的cache文件中。当SST过程结束，Donor节点将变为JOINED状态，并应用这些缓存的内容，从而返回SYNCED状态。



新节点加入流程



名词解释

- Global Transcation ID（GTID）
  - 当通过客户端对数据库进行有顺序的一系列修改时（不一定仅仅是数据库），集群中所有的节点都对这个修改动作进行同步。在wsrep API中，通过GTID对这个修改进行标识，从而在所有节点达到一致。GTID由俩部分组成，一个标识对象的UUID，和一个标识这次修改状态的sequence（对应每一个动作），例如：45eec521-2f34-11e0-0800-2a36050b826b:94530586304

- State Snapshot Transfer（SST）
  - 节点初始化/重做方法，使用指定的sst策略，来做数据的全量同步。

- Incremental State Transfer(IST)
  - Galera 2.x 开始支持。当一个节点加入，他当前的GUID与现集群相同，且缺失的数据能够在donor的Writeset的cache中找到，则可以进行IST，否则只能全部初始化数据，进行SST(State Snapshot Transfer)。

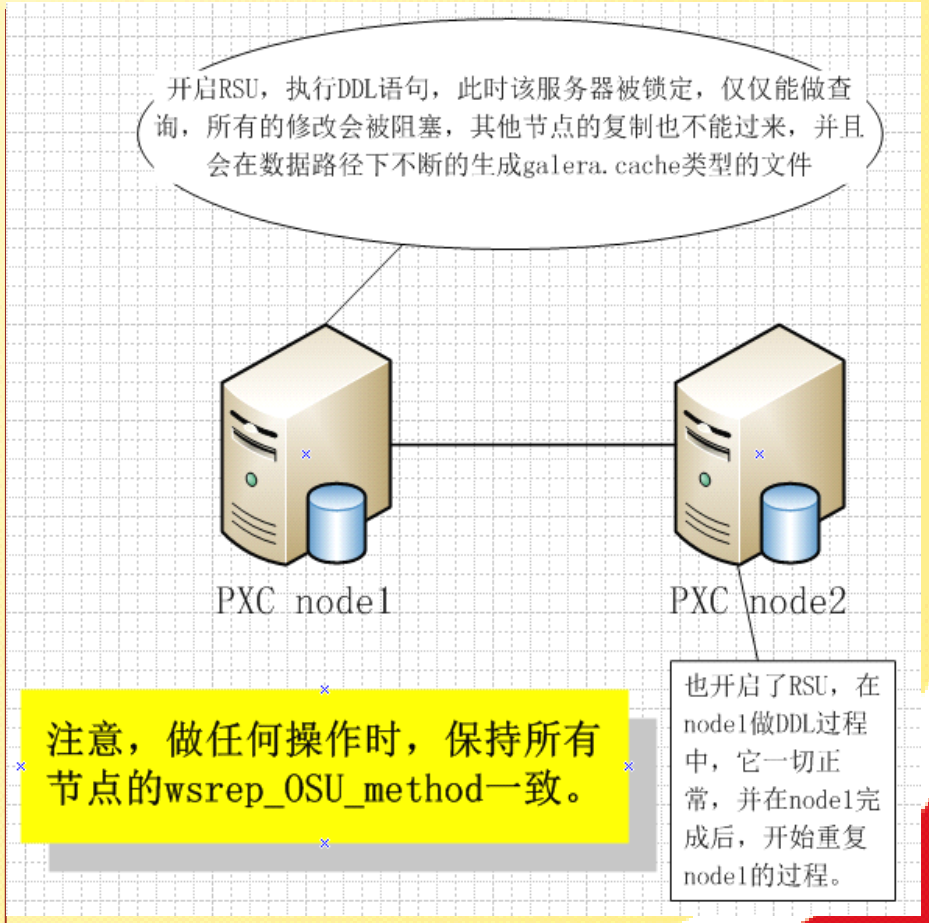
SST方式的选择

- MySQL Dump
  - 需要锁定Donor节点，无法提供访问
  - 速度慢
- Rsync
  - 需要锁定Donor节点，无法提供访问
  - 速度快
- Xtrabackup
  - 只需要短时间锁定Donor，基本不影响访问
  - 速度较快

Rolling Schema Upgrade

Schema Upgrade指的是任何修改数据库结构的DDL语句，这种语句不具有事务性。 Total Order Isolation(TOI)默认的工作方式，在这种模式下，DDL语句的表现将和在一个单机数据库上一样，将会锁定整个集群库。并且这个语句将会同时发 送到所有的集群节点去执行。

Rolling Schema Upgrade(RSU) wsrep是通过设置wsrep_OSU_method参数，DDL语句将会在当前节点执行，并且在执行过程中不锁 定其他节点，当这个节点的DDL操作完成，将会应用操作过程中延迟的replication，然后你在手工的一个一个节点的去做DDL操作。这个滚动 升级的过程中，集群中会有部分服务器有新的表结构，部分有旧的表结构。



节点故障

- 任何的硬件故障、软件崩溃、网络连接异常，都将造成节点故障。判断故障节点的依据，是根据他是否还能够连接PC（Primary component），而PC是从当前集群中随机选择的，因此不能简单的根据服务器是否能够ping通等外部的方式来判断，需要根据各个节点的wsrep_local_state来进行投票。

- 故障侦测 节点每隔evs.inactive_check_period接收一次数据包。假如在evs.keepalive_period间隔都没有任何信息发出，则发出一个heartbeat信号。假如某一个节点在evs.suspect_timeout都没有任何信息发出，则这个节点被标识为suspected，当所有的cluster成员节点都认为这个节点是suspected，则认为这个节点failed。同理，假如某个节点认为另一个节点有问题，则也必须所有的节点都保持一致的想法，才能够定性。

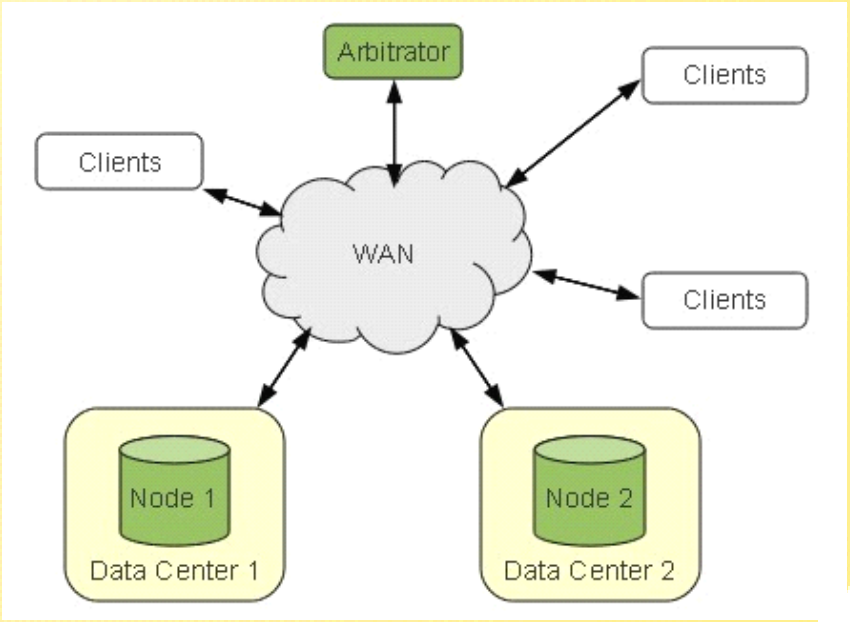
- 综上，这些节点状态参数的设置应该遵循：

evs.keepalive_period <= evs.inactive_check_period <= evs.suspect_timeout <= evs.inactive_timeout <= evs.consensus_timeout

Galera Arbiter

0.8.2的Galera开始支持一个Arbitrator节点，来预防脑裂(split-brain)；在如下图的情况下，如果有其中一个node无法正常连接WAN，那么另外一个node仍然可以通过和Arbitrator的联系来确认状态正常，并且继续提供服务。

需要注意的是，Arbitrator需要能够看到所有的网络流量，尽管它不用这些数据做任何事，因此，它会给网络带来额外的压力。



部分参数调整

- 当网络状况不好时考虑调整的参数设置：
  - wsrep_provider_options = "evs.keepalive_period = PT3S; evs.inactive_check_period = PT10S; evs.suspect_timeout = PT30S; evs.inactive_timeout = PT1M; evs.consensus_timeout = PT1M"
    - evs.keepalive_period 参数控制多久发送一次keepalive请求信号
    - evs.inactive_check_period 参数控制多久检测一次节点活动/静止状态
    - evs.suspect_timeout 参数控制某个节点是否被标识为suspected状态的时间间隔
    - evs.inactive_timeout 参数控制节点不活动时检测周期
    - evs.consensus_timeout 参数控制多久检测一次节点一致性 通过上面的设置，可以使节点超时时间为30秒
    - evs.inactive_timeout参数必须不小于evs.suspect_timeout，evs.consensus_timeout必须不小于evs.inactive_timeout。
- wsrep_slave_threads
  - 建议每个core启动4个复制线程，这个参数很大程度上受到I/O能力的影响，官方甚至在ThinkPad R51一个4200转硬盘、单核心的机器上设置32个线程，并且执行良好。可以通过观察wsrep_cert_deps_distance这个状态变量来获得当前最佳的线程数，这个参数实际上表示单位时间平均多少个writesets能被执行掉。

监控要点

- **wsrep_flow_control_paused**
  - 这个变量标识当前节点落后于集群的程度，0.0-1.0,0.0为没有落后，1.0为flow control已经停止了。应该尽量保证这个变量值为0.0。如果落后实在太厉害，则应该适当增加复制线程数wsrep_slave_threads，如果还没有改善，则应该从集群中删除最慢的节点。
- **wsrep_cert_deps_distance**
  - 这个变量标识当前节点平均时间内并行执行的事务数，这个数据可以用来作为wsrep_slave_threads的参考，同时当并发量很高的时候，这个数值也会很大。
- **wsrep_flow_control_sent**
  - 表示flow_control发出FC_PAUSE事件的次数，暂停的次数越多，表示接受到的请求频繁堵满slave队列，这种情况下不是队列长度不足，就是机器性能太差。（如果集群中多个机器都有这个情况，则考虑调整slave队列长度的相关参数，如gcs.fc_limit等）
- **wsrep_local_recv_queue_avg**
  - 这个参数表示本地节点平均的接收队列长度，如果这个参数不为0.0，则表示接收来的数据不能被及时应用（立即应用了则不会进入队列）。
- **wsrep_local_send_q_avg**
  - 这个参数表示本地节点发送数据的队列长度，如果这个参数不为0.0，则表示向外发送数据的速度比较慢，有堆积。

高负载下的宕机风险

- 利用sysbench通过Haproxy对三节点的Cluster进行并发读写，并发线程数300。
- 此时，整体每个节点的IO负载都比较低，没有造成瓶颈，但是CPU资源消耗很大

```
top - 16:17:25 up 7 days, 23:13,  3 users,  load average: 34.75, 36.71, 36.10
Tasks: 204 total,   1 running, 203 sleeping,   0 stopped,   0 zombie
Cpu(s): 23.7%us,  7.4%sy,  0.0%ni, 67.9%id,  0.2%wa,  0.0%hi,  0.7%si,  0.0%st
Mem:  99191560k total, 80784640k used, 18406920k free,   245752k buffers
Swap: 16777212k total,   749988k used, 16027224k free, 28815004k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
24807 mysql     20   0 37.9g  21g 137m S 1158.6 22.7 526:35.37 mysqld
```

高负载下的宕机风险

- 此时三个节点的日志中均频繁出现如下内容。
- 此时发生过一次三个节点均无法通信，导致脑裂，三个节点 均变为只读状态的故障。
- 同时观察wsrep_*的相关状态参数，发现了wsrep_cert_deps_distance的值，与集群的稳定性息息相关。在当时的测试情况下，当该值超过1300，则集群崩溃的可能性非常大。
- 这种宕机一般不会引起数据丢失和不一致，但是将会终止数据库的可用性。

```
120914 15:57:12 [Note] WSREP: (ee0487b2-fe18-11e1-0800-e7ec3335ef5d, 'tcp://10.10.59.229:5030') reconnecting to 6d0f9357-fe1f-11e1-080
120914 15:57:12 [Note] WSREP: (ee0487b2-fe18-11e1-0800-e7ec3335ef5d, 'tcp://10.10.59.229:5030') turning message relay requesting off
120914 15:59:13 [Note] WSREP: (ee0487b2-fe18-11e1-0800-e7ec3335ef5d, 'tcp://10.10.59.229:5030') turning message relay requesting on, n
120914 15:59:13 [Note] WSREP: (ee0487b2-fe18-11e1-0800-e7ec3335ef5d, 'tcp://10.10.59.229:5030') turning message relay requesting off
120914 16:03:05 [Note] WSREP: (ee0487b2-fe18-11e1-0800-e7ec3335ef5d, 'tcp://10.10.59.229:5030') turning message relay requesting on, n
120914 16:03:06 [Note] WSREP: (ee0487b2-fe18-11e1-0800-e7ec3335ef5d, 'tcp://10.10.59.229:5030') turning message relay requesting off
120914 16:03:32 [Note] WSREP: (ee0487b2-fe18-11e1-0800-e7ec3335ef5d, 'tcp://10.10.59.229:5030') turning message relay requesting on, n
120914 16:03:33 [Note] WSREP: (ee0487b2-fe18-11e1-0800-e7ec3335ef5d, 'tcp://10.10.59.229:5030') reconnecting to 6d0f9357-fe1f-11e1-080
```

高负载引起宕机风险的防治

- 参考建议的监控数据，严格控制
- 对应用的入口SQL详细审核，避免不正常的SQL运行
- 使用一些策略，例如Iptables限制单位时间的并发量、包个数，来避免宕机

节点故障与恢复

- 任何的硬件故障、软件崩溃、网络连接异常，都将造成节点故障。判断故障节点的依据，是根据他是否还能够连接PC（Primary component），而PC是从当前集群中随机选择的，因此不能简单的根据服务器是否能够ping通等外部的方式来判断，需要根据各个节点的wsrep_local_state。

- 前面说过，故障侦测节点每隔evs.inactive_check_period接收一次数据包。假如在evs.keepalive_period间隔都没有任何信息发出，则发出一个heartbeat信号。假如某一个节点在evs.suspect_timeout都没有任何信息发出，则这个节点被标识为suspected，当所有的cluster成员节点都认为这个节点是suspected，则认为这个节点failed。同理，假如某个节点认为另一个节点有问题，则也必须所有的节点都保持一致的想法，才能够定性。

- 综上，这些节点状态参数的设置应该遵循： evs.keepalive_period <= evs.inactive_check_period <= evs.suspect_timeout <= evs.inactive_timeout <= evs.consensus_timeout

节点故障与恢复

- 一旦节点发生故障，分不同的情况，如果无法进行IST，则需要做SST，即需要进行完全的state snapshot transfer，代价非常巨大。

- Donor选择： 有自动选择和手动指定Donor两种方式，通过wsrep_sst_donor= DONOR_NAME来指定，不指定则在当前集群中随机选择。

- 需要注意的是，当某个node被选为Donor时，如果使用Rsync或者Dump，这个节点是不能提供客户端服务的，即使使用Xtrabackup这个节点的性能和稳定性也会受到很大影响，基于这种情况，建议手动选择一个Donor，避免高负载的机器被用来做Donor。

节点故障与恢复

- 节点故障后获取GTID的方法
  - 执行mysqld加--wsrep-recover参数，可获得GTID，如
    - mysqld --defaults-file=/DATA/my6000/my6000.cnf --log_error=/dev/stdout --wsrep-recover
    - 再调用--wsrep_start_position=设置GTID，启动mysqld

- 默认情况下，目前的版本似乎会自动这样做，看源码是从innodb redolog记录的checkpoint里获得的GTID

搜狐
SOHU.com

谢　　谢

完