[**理解 OpenStack 高可用(HA) （6）: MySQL HA**](http://www.cnblogs.com/sammyliu/p/4995868.html)

本系列会分析OpenStack 的高可用性（HA）概念和解决方案：

（1）[OpenStack 高可用方案概述](http://www.cnblogs.com/sammyliu/p/4741967.html%20)

（2）[Neutron L3 Agent HA - VRRP （虚拟路由冗余协议）](http://www.cnblogs.com/sammyliu/p/4692081.html%20)

（3）[Neutron L3 Agent HA - DVR （分布式虚机路由器）](http://www.cnblogs.com/sammyliu/p/4713562.html%20)

（4）[Pacemaker 和 OpenStack Resource Agent （RA）](http://www.cnblogs.com/sammyliu/p/5025362.html%20)

（5）[RabbitMQ HA](http://www.cnblogs.com/sammyliu/p/4730517.html)

（6）[MySQL HA](http://www.cnblogs.com/sammyliu/p/4995868.html%20)

**1. MySQL HA 方案**

**1.1 各种方案概述**

Mysql HA 方案有很多种，包括：

* mmm: <http://mysql-mmm.org/>
* mha: <https://code.google.com/p/mysql-master-ha/>
* heartbeat+brdb: <http://lin128.blog.51cto.com/407924/279411> <http://www.centos.bz/2012/03/achieve-drbd-high-availability-with-heartbeat/>
* cluster(使用ndb引擎):<http://database.51cto.com/art/201008/218326.htm>
* 双master+keeplived: <http://database.51cto.com/art/201012/237204.htm>，<http://kb.cnblogs.com/page/83944/>
* 双master: <http://yunnick.iteye.com/blog/1845301>
* Oracel Fabric 方案：<http://www.csdn.net/article/2014-08-20/2821300>

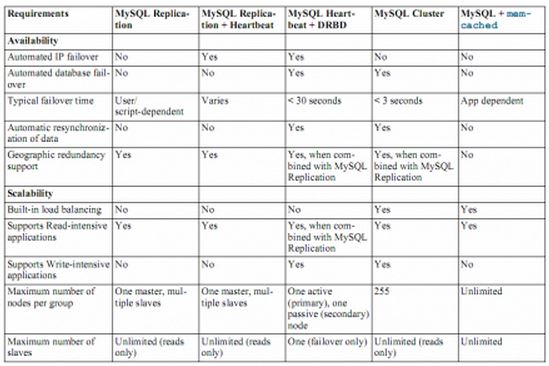
这些高可用方案，大多是基于以下几种基础来部署的：

1. 基于主从复制；
2. 基于Galera协议；
3. 基于NDB引擎；
4. 基于中间件/proxy；
5. 基于共享存储；
6. 基于主机高可用；

从 SLA 的角度看：

* 要达到99.9%：使用MYSQL复制技术
* 要达到99.99%：使用MYSQL NDB 集群和虚拟化技术
* 要达到99.999%：使用shared-nothing架构的GEO-REPLICATION和NDB集群技术

[这里](http://blogstatic.u.qiniudn.com/wp-content/uploads/2010/08/different-ha.png) 有个各种方案的比较：



在这些可选项中，最常见的就是基于主从复制的方案，其次是基于Galera的方案。这篇文章 [分享MYSQL中的各种高可用技术](http://www.cnblogs.com/lyhabc/p/3838764.html) 全面具体地分析了 Mysql 的各种容灾方案。也可见 Mysql 容灾的水很深。

**1.2 MySQL Cluster（NDB Storage Engine）**

MySQL Cluster 是 MySQL 官方也就是 Oracle 主推的一种提供去中心化集群（shared-nothing clustering）和 自动共享（auto-sharding）的MySQL 数据库管理系统。它被设计来提供高可用（99.999%）、高吐吞吐量、低延迟和几乎线性扩展的解决方案。它是基于 MySQL 的 NDB 或者 NDBCLUSTER 存储引擎实现的。（引用自 https://en.wikipedia.org/wiki/MySQL\_Cluster）

官网：<https://www.mysql.com/products/cluster/>

版本：MySQL Cluster 有独立于 MySQL 的版本（7.4版本使用 MySQL 5.6；7.3版本使用 MySQL 5.5）

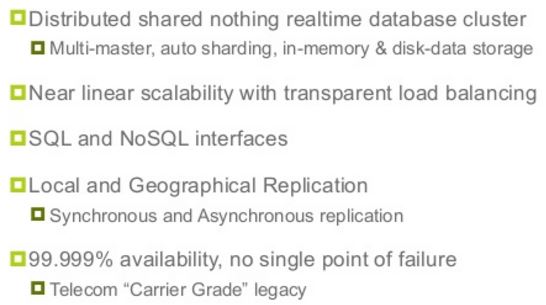
价格：<https://www.mysql.com/products/>



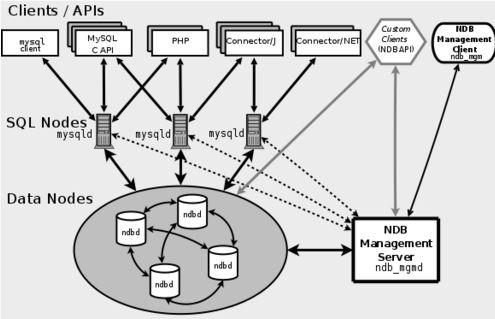
这是 [Choosing the right MySQL High Availability Solution – webinar replay](http://www.clusterdb.com/mysql/choosing-the-right-mysql-high-availability-solution-webinar-replay) MySQL Cluster 和其它几个主要HA方案的 SLA 比较：

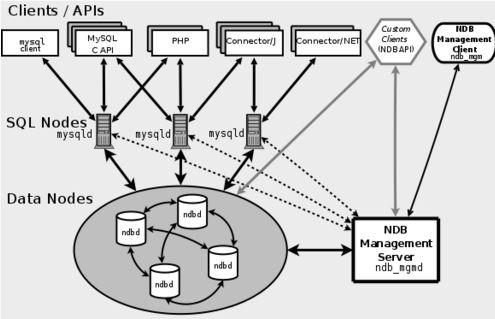


（1）特征



（2）架构



* 1）Sql结点（SQL node--上图对应为MySQLd）:分布式数据库。包括自身数据和查询中心结点数据.
* 2）数据结点(Data node -- ndbd):集群共享数据(内存中).
* 3）管理服务器(Management Server – ndb\_mgmd):集群管理SQL node,Data node.

支持最多 48 个 data nodes；集群最多 255 个节点

**2. MariaDB 和 Galera Cluster**



MySQL 被 Oracle 收购后，基于需求以及对 Oracle 的担心，出现了两个主要的分支。它们都是免费开源的软件。

**2.1 MySQL 的两个主要分支之一之 MariaDB**

    MariaDB由MySQL的创始人麦克尔·维德纽斯主导开发，他早前曾以10亿美元的价格，将自己创建的公司MySQL AB卖给了SUN，此后，随着SUN被甲骨文收购，MySQL的所有权也落入Oracle的手中。MariaDB名称来自麦克尔·维德纽斯的女儿玛丽亚（英语：Maria）的名字。

    MariaDB的目的是完全兼容MySQL，包括API和命令行，使之能轻松成为MySQL的代替品。在存储引擎方面，10.0.9版起使用XtraDB（名称代号为Aria）来代替MySQL的InnoDB。

    版本方面，MariaDB直到5.5版本，均依照MySQL的版本。因此，使用MariaDB5.5的人会从MySQL 5.5中了解到MariaDB的所有功能。从2012年11月12日起发布的10.0.0版开始，不再依照MySQL的版号。10.0.x版以5.5版为基础，加上移植自MySQL 5.6版的功能和自行开发的新功能。

    相对于最新的MySQL5.6，MariaDB在性能、功能、管理、NoSQL扩展方面包含了更丰富的特性。比如微秒的支持、线程池、子查询优化、组提交、进度报告等。

    官网地址：<https://mariadb.org/>

**2.2 MySQL 的两个主要分支之二之 Percona**

  Percona Server就是这样一款产品，由领先的MySQL咨询公司Percona发布。Percona Server是一款独立的数据库产品，为用户提供了换出其MySQL安装并换入Percona Server产品的能力。通过这样做，就可以利用XtraDB存储引擎。Percona Server声称可以完全与MySQL兼容，因此从理论上讲，您无需更改软件中的任何代码。这确实是一个很大的优势，适合在您寻找快速性能改进时控制质量。因此，采用Percona Server的一个很好的理由是，利用XtraDB引擎来尽可能地减少代码更改。

  更多的比较，可以参考网上的大量文章，比如

* [超越MySQL：三个流行MySQL分支的对比](http://www.csdn.net/article/2011-12-29/309890)
* [MySQL分支的选择：Percona还是MariaDB](http://www.biaodianfu.com/mysql-percona-or-mariadb.html)
* [MariaDB or PerconaDB 你选哪种来替代 MySQL?](https://ruby-china.org/topics/9248)

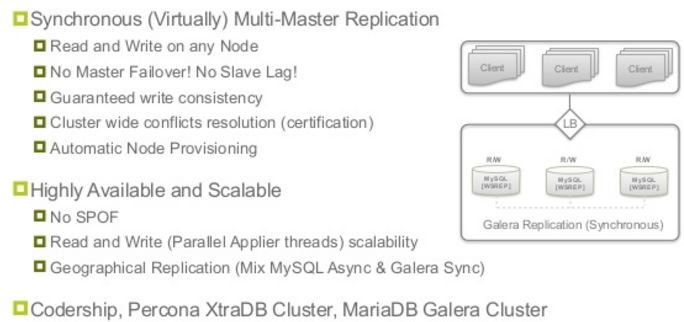
**2.3 Galera Cluster**

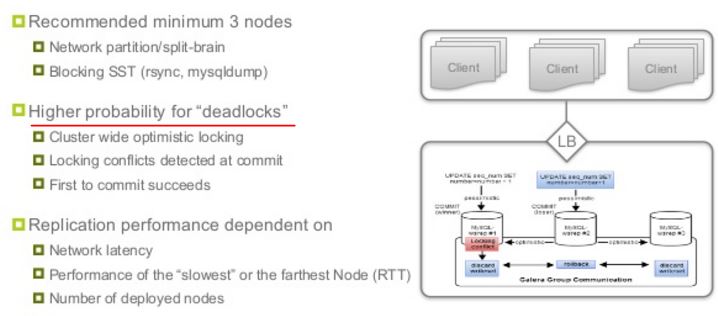
  Galera Cluster 是一套在innodb存储引擎上面实现multi-master及数据实时同步的系统架构，业务层面无需做读写分离工作，数据库读写压力都能按照既定的规则分发到各个节点上去。在数据方面完全兼容 MariaDB 和 MySQL。

官网：<http://galeracluster.com/products/>

使用案例：HP, OpenStack，KPN

特征：





**2.4 MySQL Galera Cluster**

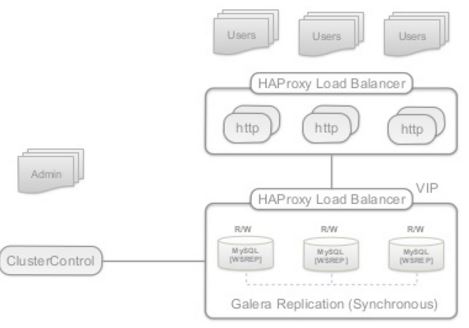
**2.4.1 MySQL Galera Cluster 的特点和局限**

  Galera Cluster 可以同时支持 MySQL 和 MariaDB：

* 安装 MySQL Galera Cluster：需要安装带 wsrep patch 的MySQL版本（比如 MySQL 5.5.29）和 Galera复制插件，详细步骤请参考 [MySQL多主复制-MySQL Galera安装部署](http://www.zrwm.com/?cat=130)
* 安装 MiraDB Galera Cluster：参考 [OpenStack 在 RedHat 平台上的 MariaDB HA 方案](http://docs.openstack.org/ha-guide/controller-ha-galera.html#maria-db-ha)，以及 [MariaDB Galera Cluster 部署](http://code.oneapm.com/database/2015/07/02/mariadb-galera-cluster/)。

  为了支持MySQL，Galera Cluster 中使用了由 Coreship 提供的补丁 （<https://launchpad.net/codership-mysql>）。MySQL Galera 集群：

* 使用通用的 Wsrep replication 来替代 MySQL Cluster 中的 Replication
* 基本 Quorum的集群，最少三个节点，只能奇数个节点
* 使用偶数个节点时，可以使用一个 Galera Arbiter （garbd）
* 较高的死锁可能性：在多主集群中，不支持对表加锁和解锁（LOCK/UNLOCK TABLES cannot be supported in multi-master setups）。因此，在两个transaction 从不同的节点更新同一个 row 的时候，只有一个 transaction 会成功，对另外一个transaction，MySQL 会返回死锁错误（Error: 1213 SQLSTATE: 40001 (ER\_LOCK\_DEADLOCK)）。
* 部署：



（本节信息来自 <http://www.slideshare.net/Severalnines/galera-cluster-for-mysql-vs-mysql-ndb-cluster-a-high-level-comparison-42724783>）

* MySQL Galera 集群有非常多的使用限制（[来源](http://support.severalnines.com/entries/21692388-Limitations-in-Galera-Cluster-for-MySQL)）：

[复制代码](javascript:void(0);)

In MySQL-5.5.x/wsrep-23.x, Galera Replication has some limitations, these are documented in readme-wsrep.

Galera replication originally only worked with InnoDB storage engine, but it now also supports MyISAM storage engine. Any writes to other table types, including system (mysql.\*) tables are not replicated. However, DDL statements are replicated in statement level, and changes to mysql.\* tables will get replicated that way. So, you can safely issue: CREATE USER..., but issuing: INSERT INTO mysql.user..., will not be replicated.

MyISAM replication is recent and should be considered experimental. Non-deterministic functions like NOW() are not supported. The Configurator for Galera enables wsrep\_replicate\_myisam by default.

DELETE operation is unsupported on tables without primary key. Also rows in tables without primary key may appear in different order on different nodes. As a result SELECT...LIMIT... may return slightly different sets.

Unsupported queries:

\* LOCK/UNLOCK TABLES cannot be supported in multi-master setups. 不支持跨节点的表锁

\* lock functions (GET\_LOCK(), RELEASE\_LOCK()... )

Query log cannot be directed to table. If you enable query logging, you must forward the log to a file:

log\_output = FILE

Use general\_log and general\_log\_file to choose query logging and the log file name.

Maximum allowed transaction size is defined by wsrep\_max\_ws\_rows and wsrep\_max\_ws\_size. Anything bigger (e.g. huge LOAD DATA) will be rejected.

Due to cluster level optimistic concurrency control, transaction issuing COMMIT may still be aborted at that stage. There can be two transactions writing to same rows and committing in separate cluster nodes, and only one of the them can successfully commit. The failing one will be aborted. For cluster level aborts, MySQL/galera cluster gives back deadlock error.

code (Error: 1213 SQLSTATE: 40001 (ER\_LOCK\_DEADLOCK)).

XA transactions can not be supported due to possible rollback on commit.

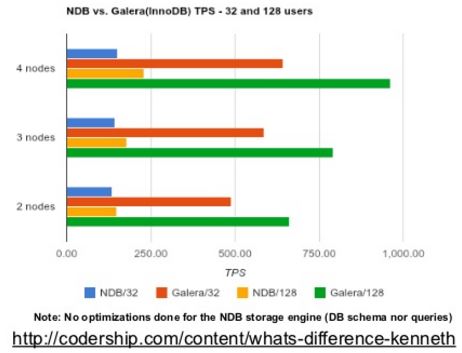
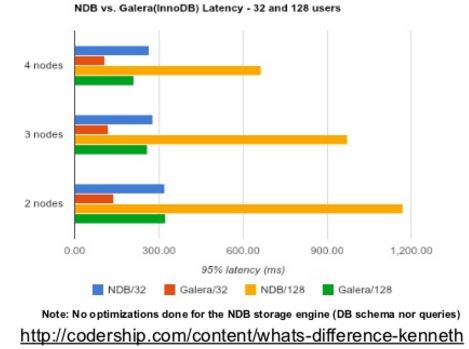
[复制代码](javascript:void(0);)

MySQL 5.6 的同样描述在 [percona/debian-percona-xtradb-cluster-5.6](https://github.com/percona/debian-percona-xtradb-cluster-5.6/blob/master/Docs/README-wsrep)

原因和概率：

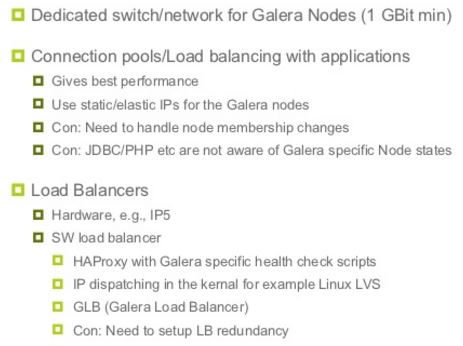
* + MySQL Galera 在本地节点上使用悲观锁（pessimistic locking）
  + MySQL Galera 在其它节点上使用乐观锁（optimistic locking）
  + 在大的负载压力下的发生概率大概为 1/500
  + 在正常的负载压力下的发生概率大概为 1/10000

**2.4.2 NDB （MySQL Cluster 使用的引擎）和 MySQL Galera Cluster 的性能对比**

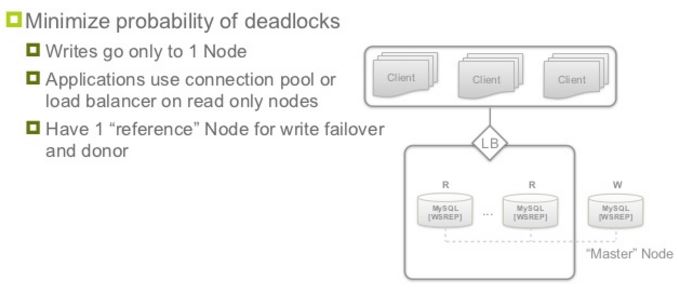
 

**2.4.3 一些 best practice**

常规的推荐做法：



减少死锁的一些推荐做法：

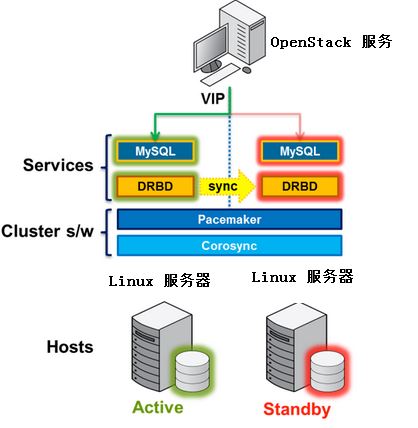


**3. 使用 Pacemaker + DRBD + CoroSync 的 A/P 方案**

与 RabbitMQ HA 方案类似，OpenStack 官方推荐的 Mysql Active/Passive HA 方案也是 Pacemaker + DRBD + CoroSync。具体方案为：

* 配置 DRBD 用于 Mysql
* 配置 Mysql 的 var/lib/mysql 目录位于 DRBD 设备上
* 选择和配置一个 VIP，配置 Mysql 在该 IP 上监听
* 使用 Pacemaker 管理 Mysql 所有的资源，包括其 deamon
* 配置 OpenStack 服务使用基于 VIP 的 Mysql 连接

[OpenStack 官方推荐的 Mysql HA A/P 方案](http://docs.openstack.org/zh_CN/high-availability-guide/content/s-mysql.html) 配置完成后的效果：

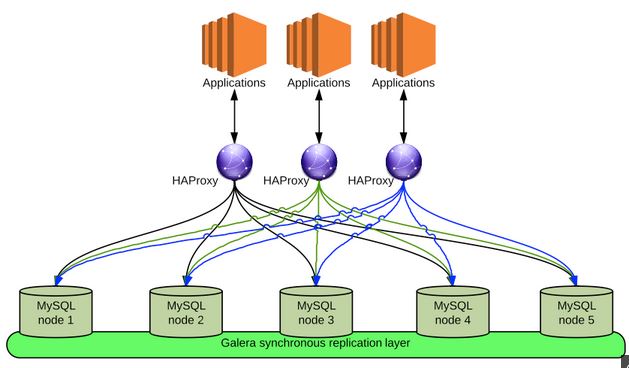


[这个文档](https://github.com/percona/percona-pacemaker-agents/blob/master/doc/PRM-setup-guide.rst) 详细阐述了具体的配置步骤。这个方案的问题是，drbd 容易出现脑裂；而且，两个 mysql 节点只有一个能提供服务，存在资源浪费。

**4. 使用 MySQL Galera 的多主方案**

**4.1 三节点方案**

架构如下：



Galera 主要功能：

* 同步复制
* 真正的multi-master，即所有节点可以同时读写数据库
* 自动的节点成员控制，失效节点自动被清除
* 新节点加入数据自动复制
* 真正的并行复制，行级
* 用户可以直接连接集群，使用感受上与MySQL完全一致

优势:

* 因为是多主，所以不存在延迟
* 不存在丢失交易的情况
* 同时具有读和写的扩展能力
* 更小的客户端延迟
* 节点间数据是同步的,而Master/Slave模式是异步的,不同slave上的binlog可能是不同的

局限：见 4.3.1 章节的详细描述

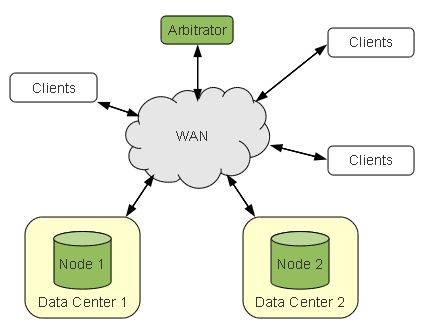
另外，MySQL Galera Cluster 并不是适合所有需要复制的情形，你必须根据自己的需求来决定，比如，

* 如果你是数据一致性考虑的多，而且写操作和更新的东西多，但写入量不是很大，MySQL Galera Cluster就适合你。但是，这种方案中整个集群的写入吞吐量是由最弱的节点限制，如果有一个节点变得缓慢，那么整个集群将是缓慢的。为了稳定的高性能要求，所有的节点应使用统一的硬件，而且集群节点建议最少3个。
* 如果你是查询的多，且读写分离也容易实现，那就用 replication 好，简单易用，用一个 master 保证数据的一致性，可以有多个slave用来读去数据，分担负载，只要能解决好数据一致性和唯一性，replication就更适合你，毕竟 MySQL Galera Cluster集群遵循“木桶”原理，如果写的量很大，数据同步速度是由集群节点中IO最低的节点决定的，整体上，写入的速度会比replication慢许多。

详细配置过程可以参考 OpenStack HA Guide, [这个文章](http://bbs.chinaunix.net/thread-4092993-1-1.html) 和 [MySQL Multi-master Replication With Galera](http://www.sebastien-han.fr/blog/2012/04/01/mysql-multi-master-replication-with-galera/)。

**4.2 两节点 + Galera Arbitrator A/A HA 方案**

 3.1 中的A/A 方案需要三个节点，因此成本比较高。本方案提供使用两个节点情况下的 A/A 方案。相信信息可以参考 [这篇文章](http://galeracluster.com/documentation-webpages/arbitrator.html?highlight=arbitrator)。



该方案使用 Arbitrator 作为第三个节点来使用，它其实是一个守护进程。它有两个作用：

1. 当你使用偶数个节点时，它能当作一个奇数节点使用，来防止脑裂发生。
2. 它能被用作持续的系统状态的快照，用于备份目的。

**4.3 OpenStack 使用 HAProxy + MySQL Galera Cluster 的问题**

[这篇文章](http://lists.openstack.org/pipermail/openstack-dev/2014-May/035264.html)中，作者对200个OpenStack用户/运维人员做过一个关于数据库使用的调查，结果是

* 1 人使用 PostgreSQL
* 10几个人使用标准的 MYSQL master/slave replication 方案
* 其他人都是用 MySQL Gelera 集群

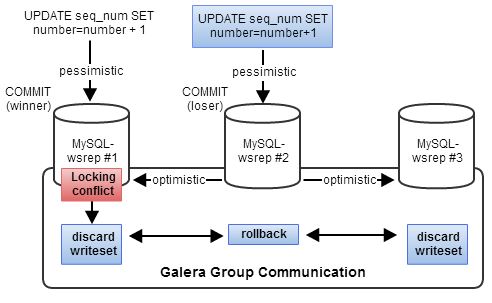
**4.3.1 问题原理**

  OpenStack 官方推荐的A/A HA 方案是使用 Galera 来做三节点HA（<http://docs.openstack.org/ha-guide/controller-ha-galera.html>）。这种模式下，Galera 提供多个 Mysql 节点之间的同步复制，使得多个 Mysql 节点同时对外提供服务，这时候往往需要使用负载均衡软件比如 HAProxy 来提供一个 VIP 给各应用使用。但是，OpenStack 文档回避了这种MySql集群的问题。

  对于 2.4.1 部分描述的 MySQL Galera 的一些局限，如果在 OpenStack 环境中使用 HAProxy 做整个MySQL Galera 的 LB 的话， 因为该集群不支持跨节点对表加锁，也就是说如果OpenStack 某组件有两个会话分布在两个节点上同时写入某一条数据，那么其中一个会话将会遇到死锁的情况。网上这种情况的报告非常多，比如：

* <https://bugzilla.redhat.com/show_bug.cgi?id=1141972> Cause: Unhandled database deadlock conditions triggered with some database configuration edge cases，Consequence: Lost database transactions，Fix: Retry actions on deadlock conditions，Result: Robust database communication in all cases
* <http://www.gossamer-threads.com/lists/openstack/operators/41337> Openstack and mysql galera with haproxy

    文章 [Avoiding Deadlocks in Galera - Set up HAProxy for single-node writes and multi-node reads](http://severalnines.com/blog/avoiding-deadlocks-galera-set-haproxy-single-node-writes-and-multi-node-reads) 对这个问题和解决方法有非常详细的描述。基本原理示意图：



  有一个长长的 OpenStack 邮件列表，[IMPORTANT: MySQL Galera does \*not\* support SELECT ... FOR UPDATE](http://lists.openstack.org/pipermail/openstack-dev/2014-May/035264.html) （写于 2014年五月）中，作者列出了Nova 和 Neutron 中使用的  SELECT ... FOR UPDATE 代码，其中

* Nova 中只有若干地方使用该结构，但是，使用的地方是 quota 代码中
* Neutron 中大量使用该结构，据统计，在11个不同的文件中出现了44次。

  可见，在写并发非常高的情况下，死锁的情况的出现概率是不低的，特别是在 Neutron 中。

  作者还提出了几个选项：

1. 在包含 with\_lockmode('update') 代码的 OpenStack 模块中不使用 MYSQL Galera 作为数据库（问题是目前没更好的选择。。）
2. 在 OpenStack 文档中加入有关问题的注释（这么目前都还没有。。）
3. 修改 Nova 和 Neutron 的代码，替换掉 with\_lockmode('update') 代码 （据说，目前，Nova 代码的修改已经完成，Neutron 还没有。。）
4. 对于 Nova db quota 驱动，做代码优化

**4.3.2 一些 workaround**

（1）上面的邮件回复中提到的一个 workaround，就是使得更新请求只发往一个节点。在使用 HAProxy 的情况下，具体做法是，只设定一个节点为 master，其余的为 backup。HAProxy 会在 master 失效时自动切换到某一个 backup 上。

server 192.168.0.101 192.168.0.101:3306 check

server 192.168.0.102 192.168.0.102:3306 check backup

server 192.168.0.103 192.168.0.103:3306 check backup

    如果还需要进一步优化的话，可以只将写操作放到一个节点，而将读操作在所有节点之间做负载均衡从而提高性能。[Percona XtraDB Cluster reference architecture with HaProxy](https://www.percona.com/blog/2012/06/20/percona-xtradb-cluster-reference-architecture-with-haproxy/) 描述了一个改进的方案，就是提供两个 MYSQL 服务端点，一个（端口 3306）只是使用（包括读写）一个节点，另一个（端口3306）使用三个节点。因此，对 OpenStack 来说，Neutron 使用 3306 端口(如果Nova解决了问题的话)，其它组件使用 3307 端口。

[复制代码](javascript:void(0);)

global

log 127.0.0.1 local0

log 127.0.0.1 local1 notice

maxconn 4096

chroot /usr/share/haproxy

user haproxy

group haproxy

daemon

defaults

log global

mode http

option tcplog

option dontlognull

retries 3

option redispatch

maxconn 2000

contimeout 5000

clitimeout 50000

srvtimeout 50000

frontend pxc-front

bind \*:3307

mode tcp

default\_backend pxc-back

frontend stats-front

bind \*:80

mode http

default\_backend stats-back

frontend pxc-onenode-front

bind \*:3306

mode tcp

default\_backend pxc-onenode-back

backend pxc-back #master-master，适用于没有使用 SELECT...UPDATE 语句的应用

mode tcp

balance leastconn

option httpchk

server c1 10.116.39.76:3306 check port 9200 inter 12000 rise 3 fall 3

server c2 10.195.206.117:3306 check port 9200 inter 12000 rise 3 fall 3

server c3 10.202.23.92:3306 check port 9200 inter 12000 rise 3 fall 3

backend stats-back

mode http

balance roundrobin

stats uri /haproxy/stats

stats auth pxcstats:secret

backend pxc-onenode-back #一个master，其它是backup，用来避免deadlock

mode tcp

balance leastconn

option httpchk

server c1 10.116.39.76:3306 check port 9200 inter 12000 rise 3 fall 3

server c2 10.195.206.117:3306 check port 9200 inter 12000 rise 3 fall 3 backup

server c3 10.202.23.92:3306 check port 9200 inter 12000 rise 3 fall 3 backup

[复制代码](javascript:void(0);)

  （2）另外一个方案是，将 OpenStack 所有的MySQL 操作按照读和写做分离（read write splitting），写只在那个主节点上，读在所有节点上做负载均衡。但是，目前 OpenStack 应该还没有原生的支持。一个可选的方案是使用开源软件 maxscale：<https://www.percona.com/blog/2015/06/08/maxscale-a-new-tool-to-solve-your-mysql-scalability-problems/>

（3）关于 openstack 里面的读写分离，其 db 库 oslo 倒是有了接口支持：

def get\_session(self, use\_slave=False, \*\*kwargs):  
"""Get a Session instance.

:param use\_slave: if possible, use 'slave' database connection for this session. If the connection string for the slave database wasn't provided, a session bound to the 'master' engine will be returned. (defaults to False)  
:type use\_slave: bool

但是从代码（Kilo版本）来看，只有 Nova 支持这种 slave\_connection 参数（\nova\nova\db\sqlalchemy\api.py）：

[复制代码](javascript:void(0);)

def model\_query(context, model,

args=None,

session=None,

use\_slave=False,

read\_deleted=None,

project\_only=False):

if session is None:

if CONF.database.slave\_connection == '':

use\_slave = False

session = get\_session(use\_slave=use\_slave)

[复制代码](javascript:void(0);)

而 Neutron 模块至少在 Kilo 版本中还没有实现。下面的代码中，调用 oslo 创建 db session 的时候，根本就没有传入 CONF.database.slave\_connection 的值：

def get\_session(autocommit=True, expire\_on\_commit=False):

"""Helper method to grab session."""

facade = \_create\_facade\_lazily()

return facade.get\_session(autocommit=autocommit,

expire\_on\_commit=expire\_on\_commit)

在 [这个 openstack 邮件列表](http://lists.openstack.org/pipermail/openstack-dev/2014-March/029059.html) 中也能确认这个状态：

Nova is the only project that uses slave\_connection option and it was

kind of broken: nova bare metal driver uses a separate database and

there was no way to use a slave db connection for it.

但是，在 Juno 版本中，唯一没有解决 dead lock 问题的就是 Neutorn，因此，该配置项对解决 Neutron MySQL Galera 死锁没有实质性意义。它只对Nova提高DB性能有帮助。关于该配置项，可以参考官方文档 <https://wiki.openstack.org/wiki/Slave_usage>。

**4.3.3 对该问题的终极处理**

    OpenStack 的 Nova 和 Neutorn 模块都在不少地方使用了 SELECT... FOR UPDAT 语句，可以参考 [A lock-free quota implementation](http://www.cnblogs.com/A%20lock-free%20quota%20implementation) 文章中的描述。因此，如果不想使用上面的 Workaround（它降低了对扩展性的支持）而要做终极处理的话，就需要修改Nova 和 Neutron 的代码将这些语句替换掉了。

    关于代码修改，有如下文档：

[documented]:

<https://github.com/openstack/nova/blob/da59d3228125d7e7427c0ba70180db17c597e8fb/nova/openstack/common/db/sqlalchemy/session.py#L180-196>

[Nova]:

<http://specs.openstack.org/openstack/nova-specs/specs/kilo/approved/lock-free-quota-management.html>

<https://bugs.launchpad.net/oslo.db/+bug/1394298> Galera deadlock on SELECT FOR UPDATE is not handled。这个针对 Nova 的 fix 已经进了 Kilo 版本。

[Neutron]:

<https://bugs.launchpad.net/neutron/+bug/1364358><https://bugs.launchpad.net/neutron/+bug/1331564>

<https://bugs.launchpad.net/neutron/+bug/1364358> Remove SELECT FOR UPDATE usage。该 ticket 目前还是 incomplete 状态，而最新的注释 “the current design disallows to remove all SELECT FOR UPDATE so the right bug would to ensure all SELECT FOR UPDATE are Galera multi-writers compliant” 更是说明Neutron 这部分的修改还没有完成。因此，对于 Neutron 来说，还得继续使用 workaround。

根据 Mirantis 和 Percona 的 [这个报告](https://www.percona.com/live/mysql-conference-2015/sites/default/files/slides/MySQL%20and%20OpenStack%20Deep%20Dive%20v2%20(Juno).pdf)，Juno 版本中，“All components except neutron are good with using multiple writers”。

**4.3.4 数据最终一致性问题**

这个 [OpenStack 邮件列表](http://lists.openstack.org/pipermail/openstack-dev/2015-February/056007.html) 描述了该问题：

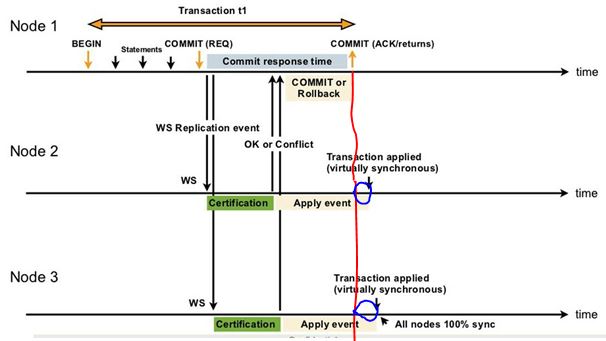
A: start transaction;

A: insert into foo values(1)

A: commit;

B: select \* from foo; <-- May not contain the value we inserted above[3]

这说明，虽然 Galera 是生成同步的，但是作为分布式数据库，本质上还是需要一些时间，即使非常短，完成写入的数据同步到整个集群的。因此，在某些情况下，特别是MySQL 负载很大导致同步压力很大的情况下，这种读写不一致性的问题可能会更加突出。



注意最下面右侧的Node1 和 node2 上的箭头和红线直接的差距（蓝色圆圈内），这也是为什么是 “virtually”同步，而不是直接同步。如果正好在gap时间段内读的话，是无法读到写入的数据的。

好在 MySQL Galera 提供了一个配置项 wsrep\_sync\_wait，它的含义是 “Defines whether the node enforces strict cluster-wide causality checks.” ，可以有如下值：

| **Bitmask** | **Checks** |
| --- | --- |
| 0 | Disabled. |
| 1 | Checks on READ statements, including SELECT, SHOW, and BEGIN / STARTTRANSACTION. |
| 2 | Checks made on UPDATE and DELETE statements. |
| 3 | Checks made on READ, UPDATE and DELETE statements. |
| 4 | Checks made on INSERT and REPLACE statements. |

它的默认值是0，如果需要保证读写一致性可以设置为1。但是需要注意的是，该设置会带来相应的延迟性，因此，它是一把双刃剑，到底对性能有多大的影响，需要经过测试才能使用。关于该配置项和其它 wresp 配置项的具体说明，可以参考 <http://galeracluster.com/documentation-webpages/mysqlwsrepoptions.html>。注意到 Mirantis 和 Percona 的 [这个报告](https://www.percona.com/live/mysql-conference-2015/sites/default/files/slides/MySQL%20and%20OpenStack%20Deep%20Dive%20v2%20(Juno).pdf) 所使用的测试环境中该值被设为1了。

**4.3.5 小结**

看起来，目前至少Neutron 还没有完成代码修改，Nova 中的代码修改看似已经完成，但是需要通过测试来验证。因此，目前情况下，我们还是需要使用 4.3.1 中描述的 workaround。

参考链接：

* <http://leejia.blog.51cto.com/4356849/841084>
* <http://drbd.linbit.com/>
* <http://kafecho.github.io/presentations/introduction-to-pacemaker/#/8>
* <http://chuansong.me/n/412792>
* <http://www.gpfeng.com/?p=603>
* <http://fengchj.com/?p=2273>
* <http://imysql.com/2015/09/14/solutions-of-mysql-ha.shtml>
* <http://www.joinfu.com/2015/01/understanding-reservations-concurrency-locking-in-nova/>
* <http://www.clusterdb.com/mysql/choosing-the-right-mysql-high-availability-solution-webinar-replay>
* <http://galeracluster.com/documentation-webpages/mysqlwsrepoptions.html>