基于 MySQL InnoDB Cluster 的 MySQL 高可用方案

1、新的业务压力对数据库提出了新的需求

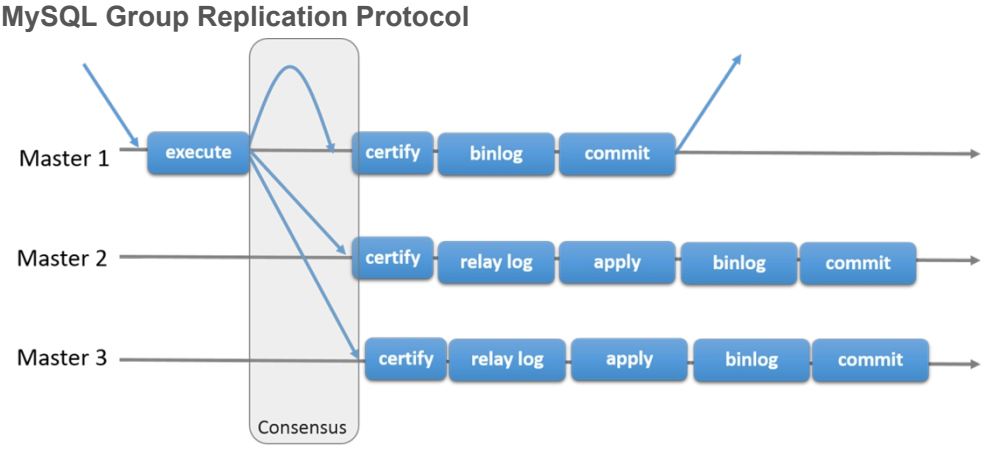
公司之前订单与支付业务相关核心数据库使用的是基于 Alwayson 的 mssql 的高可用架构，单机性能已经优化到了极限，为了应对更高的流量，分库分表势在必行，而出于成本考虑以及将来的扩展性，我们打算趁此机会把核心业务迁移至 MYSQL。由于会做分库分表，这次改造将使 MYSQL 实例数大大增加，上百个 MYSQL 实例的维护，同时需要实现跨机房的高可用、自动 Failover、高一致性、读写分离、负载均衡，我们需要一种更完善的高可用方案同时满足以上的要求。在多方对比后，我们选择了 2016 年底推出的 MYSQL InnoDB Cluster 作为高可用、高扩展性的分布式方案。

2、MySQL 新的高可用架构 Group Replication

MySQL Group Replication，简称组复制，是 MySQL Innodb Cluster 的基础，是官方 2016 年底发布的高可用、高扩展性的集群服务。它基于原生 GTID 复制和 Paxos 协议[1]，自动化的一致性保障，日志顺序分发、多数投票机制确保了主从数据的强一致性（之前的半同步复制极端情况是会出现不一致的情况），同时动态成员关系管理、错误节点监测确保了组内节点对外的可用性，各节点间在组信息上能够实时保持一致。采用插件的方式安装，对原有的数据库表基本没有影响（表必须具有主键），只是需要把 MySQL 升级到 5.7.17 以上，下面的介绍完全基于 5.7 的组复制，8.0 的刚刚推出，还有待观察。

组复制的同步

MySQL 的基础的同步复制一直被诟病的就是糟糕的一致性控制和从库可写导致的冲突问题，由于从库可读写，同步时并不强制 binlog 的连续性，同步异常问题时有发生，组复制很好的解决了这个问题。



组复制的同步仍然是基于原生的复制，不同的是组复制的每个 RW 事务在 commit 时都需要组内超过半数的成员确认该事务的全局顺序，即共同确认一个全局的顺序 ID，顺序 ID 中包含了修改行的唯一标识，最终所有的成员都能够通过相同的顺序执行所有的 RW 事务，从而保证了数据一致性[2]。

---

[1] http://mysqlhighavailability.com/the-king-is-dead-long-live-the-king-our-homegrown-paxos-based-consensus/

[2] http://mysqlhighavailability.com/mysql-group-replication-transaction-life-cycle-explained/

因为这个特性，组复制也能够支持多主模式，并有一套完善的冲突解决方案。当不同节点同时对同一条记录发起修改事务时，因为 transaction_write_set 有行的唯一标识，基于 Paxos 的冲突检测机制会提交先获得多数选票的事务，而后获得多数选票的事务则会回滚，从而达到最终一致。
那么多主模式是否提升整体写负载呢？　　使用 sysbench 做压力测试，3 个写节点，5000000 条记录随机更新，40 个并发，大约 8000TPS，会产生大量的冲突，而单节点时可以达到 11000TPS，多节点写 TPS 降低了近 30%，平均响应时间也基本翻番。这是因为热点数据冲突的缘故，所以多节点同时写同一个库的设置是不推荐的（跨 WAN 也是不推荐）。但如果多主的模式是应用在写请求按库隔离分布在各个节点的模式下，那么由于日志同步的开销要少于原本的事务执行，此时是可以提升整体写负载能力的
在单主模式下的组复制，除了主节点以外的成员，会自动开启 read_only 和 super_read_only，以确保同一时刻只有一个节点能够产生事务日志
目前一个组复制集群最多支持 9 个节点

组复制的 failover

组复制成员之间会通过一个特定的端口相互进行心跳检测，这个端口在配置文件中配置，该端口不同于对外提供数据库连接的端口。
failover 在不同的场景下会有不同的响应方式，当主节点关闭组复制，或者关闭实例时，组复制会认为该节点是正常退出复制集群，此时退出行为会被广播到所有的剩余成员（error log 会有相应的 view change 记录），并且剩余成员会发起选举，根据权重参数选举出新的主节点，如果权重参数相同，则选择 UUID 最小的实例为主节点。如果主节点是异常连接中断，比如网络断开，服务器断电等，其他成员因为没有收到主节点的退出消息，这时会检查剩余节点是否大于半数，如果大于半数，则认为主节点异常，会从集群组中踢出主节点，并发起选举，选出新的主节点，如果剩余节点小于或等于半数，即只剩最后一个节点，为了避免脑裂，那么该从节点状态不变，没有新的主节点产生，唯一的从节点依然处于只读状态，直到我们手动指定新的主节点或者原主节点恢复连接。根据 Paxos 协议，任何节点成为主节点前都必须获取所有事务日志，并且 relay 完成，该机制保证了 failover 过程中事务不会丢失。整个 failover 过程是完全自动的

组复制的配置与启用

下列是与组复制相关的配置项，基本都是必须配置的项目

```
master-info-repository = TABLE
relay-log-info-repository = TABLE
log_slave_updates = 1
#组复制强制要求使用GTID
gtid_mode=ON
enforce_gtid_consistency=ON
binlog_checksum=NONE
plugin_load ="group_replication=group_replication.so"
transaction_write_set_extraction=XXHASH64
#同组成员使用相同的值，作为组的唯一标识
loose-group_replication_group_name="22378a3d-f74b-11e7-aa82-005056b342e6"
loose-group_replication_start_on_boot=on
#GCS通信同步端口，非对外服务端口
loose-group_replication_local_address= "192.168.0.1:4000"
#该参数只在新加入集群时使用，加入后集群成员变更不受该参数影响
loose-group_replication_group_seeds= "192.168.0.1:4000, 192.168.0.2:4000,192.168.0.3:4000"
#组成员ip白名单（默认本网段所有IP）
loose-group_replication_ip_whitelist = '192.168.0.0/24'
#该参数默认值是QUOTA，在组复制中，默认要求从节点applier queue不能超过相应的阈值
#(group_replication_flow_control_applier_threshold)
#主节点certifier queue也不能超过相应的阈值（group-replication-flow-control-certifier-threshold）
#一旦超过，主节点会主动的降速，这里可以根据各自业务场景决定是否关闭限流功能
loose-group_replication_flow_control_mode=disabled
#使用单主或多主模式
loose-group_replication_single_primary_mode=on
#如果单主则设置成1，默认7，多主可以使用默认值
loose-group_replication_auto_increment_increment=1
loose-group_replication_member_weight=99
slave_preserve_commit_order=ON
```

创建同步账号

```
SET SQL_LOG_BIN=0;
CREATE USER rpluser@'%' IDENTIFIED BY 'xxxxxx';
GRANT REPLICATION SLAVE ON *.* TO rpl_user@'%';
FLUSH PRIVILEGES;
SET SQL_LOG_BIN=1;
CHANGE MASTER TO MASTER_USER='rpluser', MASTER_PASSWORD='xxxxxx' FOR CHANNEL 'GR_recovery'
```

初始化集群

对于第一个加入集群的实例，需要开启该选项(group_replication_bootstrap_group)才能启动组复制，但如果组内已经有 Primary，再加入开启了该选项的新成员，会导致脑裂和同步异常

当整个实例同时重启时，需要指定一个新的 Primary，也需要开启该参数

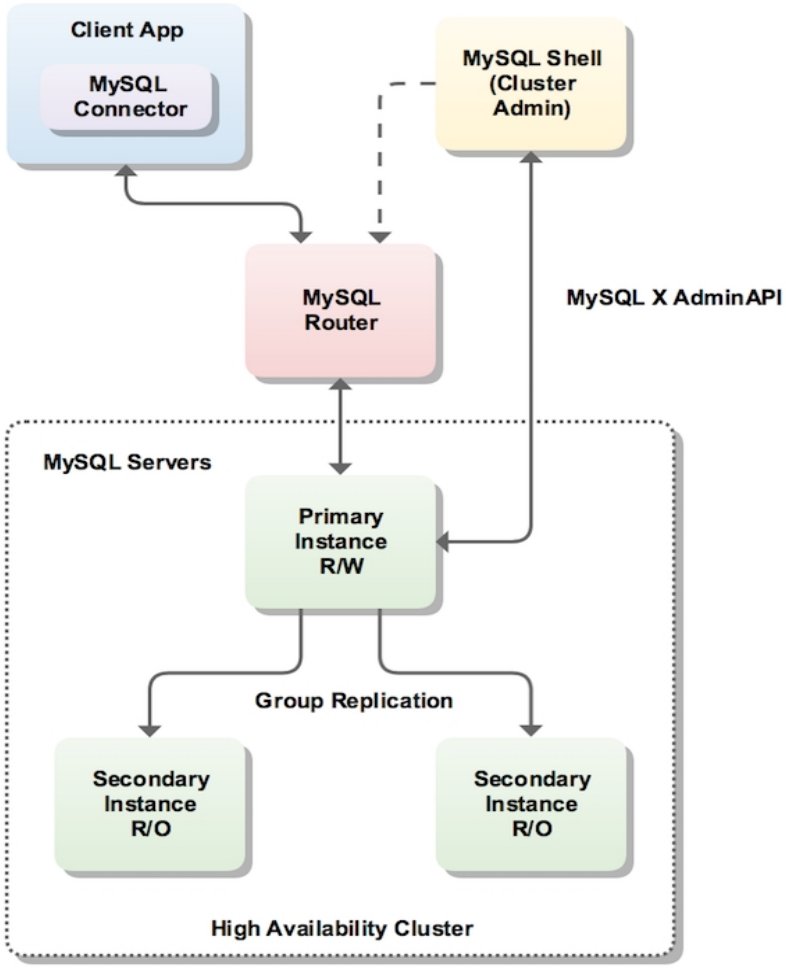
```
#启动组复制
SET GLOBAL group_replication_bootstrap_group=ON;
START GROUP_REPLICATION;
SET GLOBAL group_replication_bootstrap_group=OFF;
#查看组复制状态
select * from performance_schema.replication_group_member_stats;
select * from performance_schema.replication_group_members;
select * from performance_schema.replication_connection_status;
select * from performance_schema.replication_applier_status;
select VARIABLE_VALUE from performance_schema.global_status
       where VARIABLE_NAME = 'group_replication_primary_member';
```

集群成员之间需要能够相互 ping 通 hostname，可以通过 DNS 或者 hosts 文件实现

组复制提供了一套完善的容错功能，保障了数据的一致性，自动 failover 功能则提供了数据库层面的高可用和快速的恢复服务能力，并且具有横向弹性扩展的特性。但是组复制不能实现负载均衡、读写分离以及对应用层的高可用，要实现这些就需要 Cluster、MySQL Router、MySQL Shell。

3、MySQL Router、Cluster 和 MySQLShell 构成完整的 InnoDB Cluster 高可用方案

InnoDB cluster overview



[3]

Cluster 是这个高可用方案中的一个虚拟节点，它会在组复制的所有成员上创建一个名为 MySQL_innodb_cluster_metadata 的数据库，存储集群的元数据信息，包括集群信息、集群成员、关联的组复制、连接的 MySQL Router 等信息，以提供 MySQL Router 查询。它相当于对组复制上的成员做了一层逻辑上的封装，作为一个集群的模式展现出来，各节点的状态与对应实例在组复制中成员的状态实时同步，但是集群的节点与组复制的成员只在创建集群时同步，后期组复制的成员变更并不自动同步到集群中，可以在集群中做手动的节点增减，这样使得面向应用端的具体实例更可控更灵活。

MySQL Router 可以说是 MySQL Proxy 的升级产品，是介于 Client 与 MySQL 实例之间的代理程序。MySQL Router 会周期性的访问 Cluster 创建的 MySQL_innodb_cluster_metadata 库中的元数据获取集群成员信息,再通过 performance_schema 的系统表获取可连接实例及其状态，启动后会产生 2 个端口，分别对应集群的读写节点和只读节点，它使应用能够透明的连接 Innodb Cluster 下的数据库，即使集群发生 failover 或者增减成员也不用修改应用配置。

MySQL Shell 是新的 MySQL 客户端工具，支持 JavaScript、Python 和 MySQL 脚本，用作搭建 Innodb Cluster。

注意，

Router 需要能够 ping 通集群成员的 hostname，可以通过 DNS 或者 hosts 文件实现。

目前单个 MySQL Router 实例只支持上限 500 个连接数，需要根据实际连接数情况，部署足够数量的 Router 节点。

[3] https://dev.mysql.com/doc/refman/5.7/en/mysql-innodb-cluster-introduction.html

MySQL Shell 搭建 Cluster 的一些常用命令

```
#连接集群
\connect root@MYSQL-000001:4002
#配置检查
dba.checkInstanceConfiguration('root@192.168.0.1:4002')
#连接实例
MySQLsh root@192.168.0.1:4002
#创建cluster
var cluster = dba.createCluster('mycluster')
#添加实例
cluster.addInstance('MYSQL-000001:4002');
#删除实例
cluster。removeInstance('MYSQL-000001:4002');
#查看状态
cluster.status();
#删除整个Cluster集群
cluster.dissolve({force:true})
```

MySQL Router 初始化

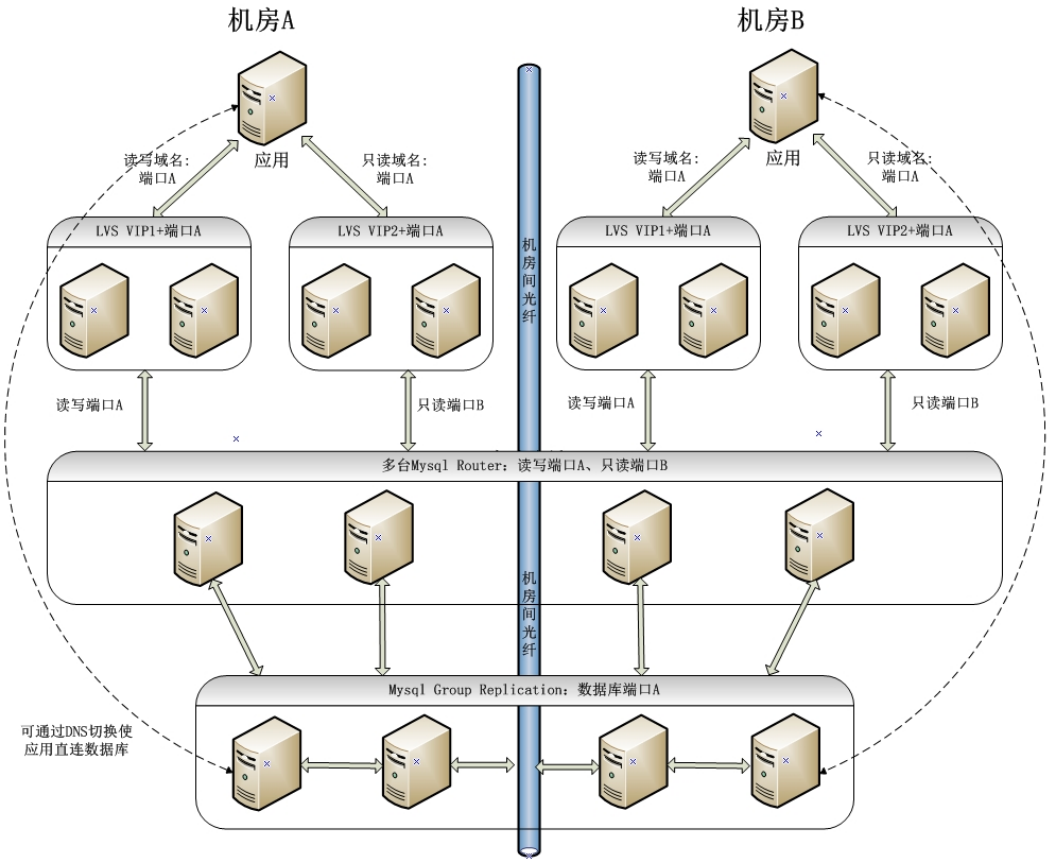
初始化 Router 时如果不用默认端口可以指定写端口，读端口会自动设置为写端口+1，Router 的 name 会记录在 MySQL_innodb_cluster_metadata 中，每台 Router 使用不同的 name。初始化完成后在指定目录下产生一个配置文件,其中包含了初始化时关联集群的成员信息，这些信息只在启动 Router 后做连接用，启动期间集群成员的变化会实时被 Router 接收，但是并不会固化到配置文件中，如果需要修改配置文件可以重新初始化或者手动更新。

```
#初始化Router
MySQLRouter --bootstrap root@ MYSQL-000001:4002 --directory /opt/MySQLRouter4002/ \
-u root --name Router000004_4002 --conf-base-port 4002
#配置文件中，建议把集群状态监测间隔时间调低，默认300秒
sed -i s/ttl=300/ttl=5/g /opt/MySQLRouter4002/MySQLRouter.conf
#停止和启动Router
./stop.sh;./start.sh
```

到这里，MySQL Router、Cluster、MySQL Group replication 都配置完成，应用通过连接 Router 的读写端口能够透明连接数据库主从节点，单实例异常或 failover 的影响都可以控制在几十秒的窗口内。但此时 Router 本身还是一个单点，官方文档中推荐将 MySQL Router 安装在应用端来解决 Router 的单点问题，优点是减少网络传输带来的延迟、可使用 Socket 连接、容易扩展，与应用一对一绑定，但实际情况下需要考虑应用部署、发布、健康监测以及集群节点变更后更新 Router 配置文件等问题，这些问题会大大增加运维的复杂度，需要对原有的应用和发布系统做升级改造，成本较高。因此最终我们选择了集中部署加上 LVS 的模式，既方便维护，又具有灵活的扩展性。

4、通过 LVS 使高可用更上一层楼

基于 MySQL Innodb Cluster 和 LVS 的 MySQL 高可用方案双机房网络拓扑图

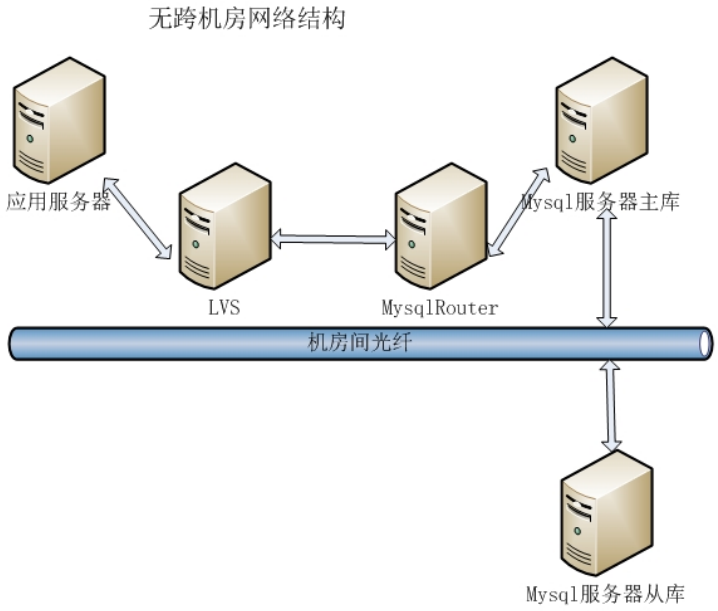


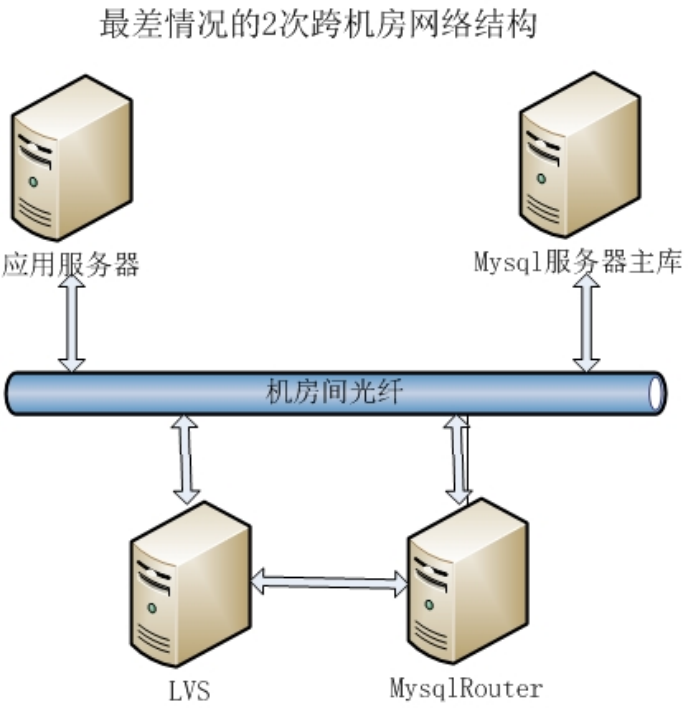
上图是基于双机房的完整性方案，集群一主 3 从，双机房各一组 Router 群集，Router 数量可动态扩展，每组 Router 配置 2 套 LVS，一套映射读写端口，读写端口与数据库端口一致，一套映射只读端口，由于 Router 给出的只读端口不同于读写端口，通过 LVS 端口映射使其与读写端口一致，2 套 LVS 的 2 个 VIP 各绑一个域名向应用提供服务。这样做的目的是，一旦 LVS、Router、Cluster 出现异常或者有大规模调整需要重建中间层，因为应用配置的读写、只读端口都与数据库实例端口相同，可以先把 2 个域名直接指向数据库实例，在中间层配置完成后再指回 LVS。

这套架构因为考虑双机房容灾的因素，所以冗余较多，要求每个机房有独立的 DNS，本机房的应用服务器只访问本机房的 LVS 和 Router，但能够实现双机房高可用，并且应用连 LVS 和 Router 都是本机房访问，进一步缩短响应时间。如果对双机房容灾的要求较低的，可以只在一个机房部署 LVS，从库缩减为 2 台。

5、压测结果

使用 Router+LVS 虽然不会影响数据库服务器本身性能但会影响整体响应时间，因此这次压测的目的是比较各种场景下各节点对响应时间的影响





压测使用的是 sysbench，对原生的测试脚本做了一些调整，耗时指单个语句的平均耗时

40 个并发、持续时间 40s、读写 4：1、每个查询平均 100 逻辑读

#数据库 CPU:55%Router CPU:4%　LVS CPU:0%　　net send:50MB/s

同机房应用直连 MySQLd read:44358/s write:11360/s Latency:0.76ms

同机房应用连单台物理 Router read:40301/s write:10324/s Latency:0.83ms

同机房应用连单台虚拟 Router read:25675/s write:6567/s Latency:1.32ms

同机房应用连 LVS 单虚拟台 Router read:23570/s write:6032/s Latency:1.42ms

同机房应用连 LVS4 台虚拟 Router read:29385/s write:7523/s Latency:1.12ms

跨机房应用直连 MySQLd read:32997/s write:8444/s Latency:1.05ms

#2 次跨机房指最糟糕的情况，数据库与应用在同一机房，而 Router 与 LVS 在另一机房

2 次跨机房应用连单台物理 Router read:21723/s write:5559/s Latency:1.53ms

2 次跨机房应用连单台虚拟 Router read:15940/s write:4078/s Latency:2.01ms

2 次跨机房应用连 LVS 单台虚拟 Router read:15362/s write:3928/s Latency:2.07ms

2 次跨机房应用连 LVS4 台虚拟 Router read:17928/s write:4585/s Latency:1.80ms

6、总结

这套架构全面实现了双机房的高可用、自动 Failover、高一致性、读写分离、负载均衡，大大降低了故障恢复时间，使我们能够放心的将支付类核心业务迁移至 MYSQL。

虽然压测中 TPS 有所下降，但中间节点对延迟耗时的影响很少，只要提高并发就可以很轻松的把 TPS 提高到直连数据库时的数值，相对耗时对比如下

跨机房的耗时（往返耗时 0.69ms） >

Router 用物理机与虚拟机的耗时差异（0.48ms） >

多台 Router 与单台 Router 的耗时差异（0.28ms） >

Router 本身的耗时（0.07ms） >

LVS 的耗时（0.05ms）