MySQL源码改造之并行复制改进

MySQL源码优化的探寻之路

DTCC

2015中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2015

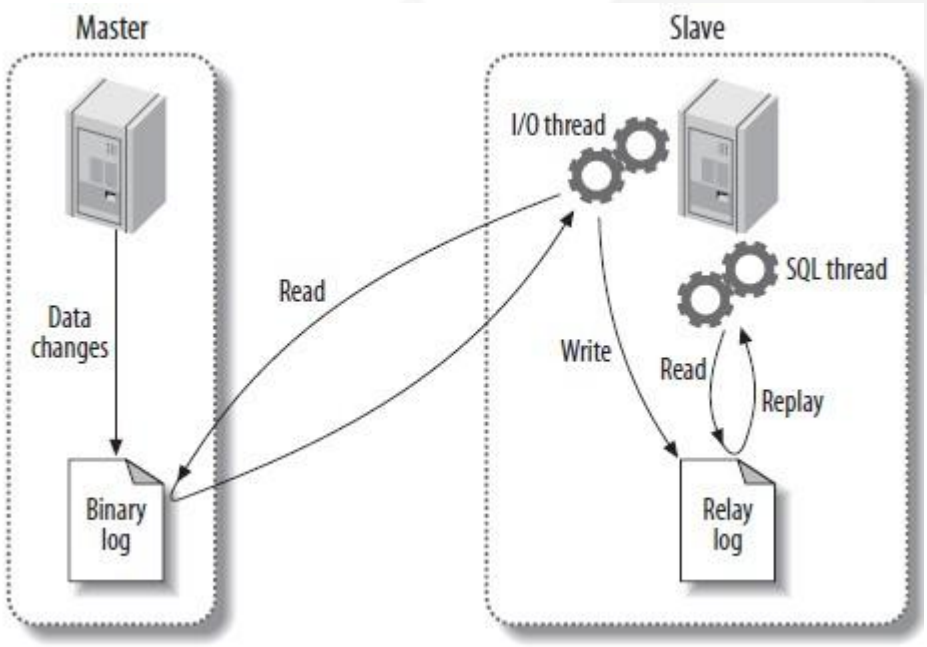
大数据技术探索和价值发现

任赟婷 携程旅行网

renyunting@ctrip.com

背景

- MySQL主从复制的单线程工作机制，常常是制约复制性能的最大瓶颈。
- MySQL 5.6开始支持Per-DB的多线程并行复制，但在单库复制压力场景下仍然无能为力。

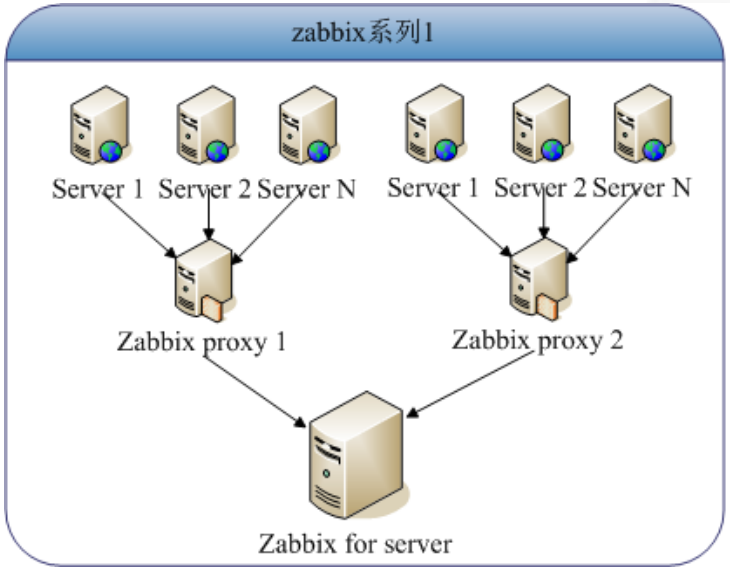


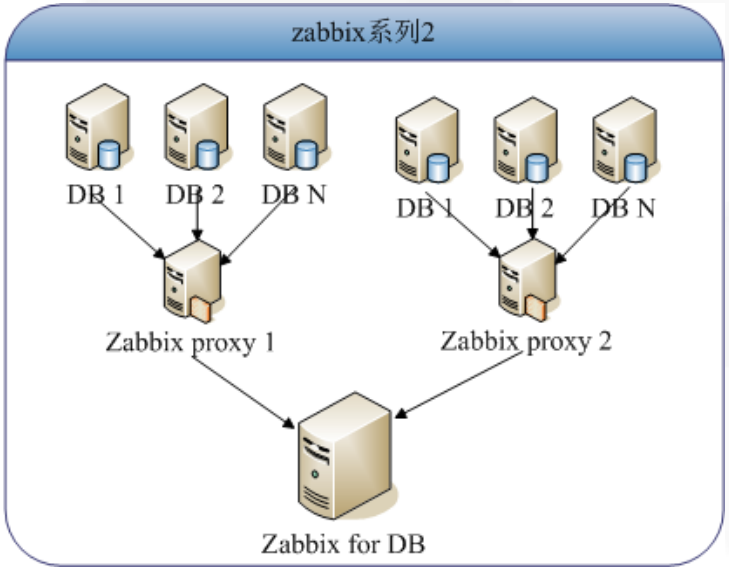
困境

- 首先遭遇到复制瓶颈的是zabbix后台数据库。
- 监控数据越来越多，slave早在master到达负载上限之前成为了性能瓶颈。
- 复制频繁出现延迟问题，当slave有查询压力更加明显。
- 新增slave节点的耗时也越来越长，整个部署周期超过5天。

需求

- 为了缓解复制压力，我们根据监控对象的不同拆分了多组zabbix系统。但是，复制的性能上限仍然严格限制了每组zabbix系统能承担的容量上限。
- 能不能有一种拆分粒度更细化的并行复制，来提高slave上的写入速度？





探索

- 在寻求解决方案的道路上，首先考虑尽可能使用已有的成熟方案。于是在我们面前有两个选择：

1. MySQL主从同步加速transfer （by taobao）

2. MariaDB 10（引入了新的并行复制功能）

探索（续）

- 关于Transfer：

1. 暂没有对应官方5.6.12版本的patch。
2. 最新的发布形式是可执行的mysqld文件。
3. 要求主库的binlog格式必须是row（我们的标准配置是mixed，row模式的replace语句可能出现自增长问题）

```
create table test ( a int(11) default null, id int(11) not null auto_increment, b int(11) default null, primary key (id), unique key d (d) );
insert into test values(5,27,4);
replace into test(a,id,b) values(6,35,4);
commit;
```

show create table时：
主库：auto_increment=36
备库：auto_increment=28

探索（续）

- 关于MariaDB 10：

- 从长远来说这是最佳解决方案，通用且能保证从机事务一致性完全忠实于主机（by google）。
- 问题在于，首个GA版刚发布不久，在正式引入线上环境之前，还有更多事情要做。

方向

- 既然没有马上能用的现成方案，何不尝试下自己造？
- 于是，我们决定以并行复制的需求为契机，作为我们进行MySQL自定制改造的起点，逐步踏入MySQL源码研究和优化的领域。

起步

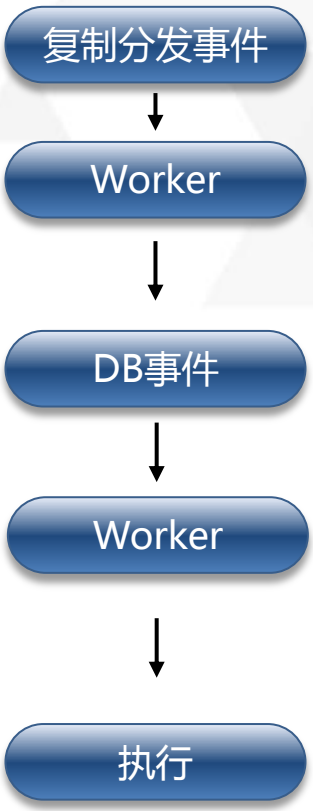
- 基于MySQL 5.6的多线程复制实现，依赖原生参数slave_parallel_workers设置复制的工作线程数。
- 先进行最基本的尝试，调整MySQL的复制SQL进程，改造Log_event类get_slave_worker事件处理线程的获取方式。
- 获取后将事件拆分，在多个slave工作线程中，轮流选择，执行事件。

1、在函数Log_event::get_slave_worker中建立将分配worker的工作交给新的分配函数

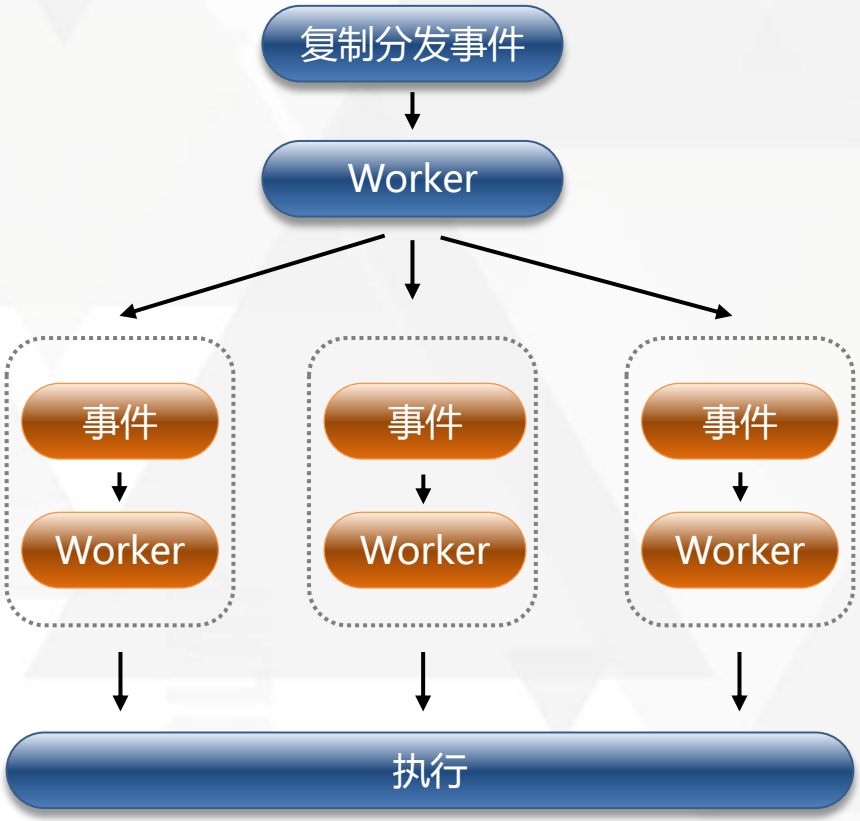
```
  if (simple_replication_mode)
  {
    char **ref_cur_db= it.ref();
     //simple replication
     ret_worker = pick_one_worker(rli, this);
     DBUG_ASSERT(num_dbs == OVER_MAX_DBS_IN_EVENT_MTS
                 || num_dbs == 1);
  }
```

2、构造自己的分配函数

```
Slave_worker *get_worker(DYNAMIC_ARRAY *ws, ulong workeri)
{
  Slave_worker **ptr_current_worker= NULL, *worker= NULL;

  ptr_current_worker= (Slave_worker **) dynamic_array_ptr(ws, workeri);
  worker= *ptr_current_worker;

  //  DBUG_RETURN(worker);
  return worker;
}

Slave_worker *pick_one_worker( Relay_log_info *rli, Log_event *ev)
{
  DYNAMIC_ARRAY *workers = &rli->workers;
  Slave_worker *worker = NULL;
  ulong workeri = rli->picked_worker;
  ulong workers_num = rli->workers.elements;
  bool choose_new = false;

  if (rli->begin_event_started && !rli->is_worker_picked)
  {
    choose_new = true;
  }

  if (choose_new)
  {
    if (workeri == workers_num - 1)
      workeri = 0;
    else
      workeri++;
    rli->is_worker_picked = true;
  }

  worker = get_worker(workers, workeri);

  rli->picked_worker = workeri;

  return worker;
}
```



原复制分发处理方式



改进后的复制分发处理方式

问题1

- 必然问题：由于操作并行后顺序被打乱，引起数据一致性校验问题，例如外键约束等。

- 解决思路：针对类似zabbix的应用，具有两个明显特点
  1. 大量写入集中在几个特定的表；
  2. 这些表的写入对顺序并不敏感。

解决1

- 增加参数变量slave-parallel-simple-tables用来配置需要进行事件拆分的表名。同时增加了变量slave-parallel-simple用来动态开关自定义的表级别并行复制。

- 进一步改造get_slave_worker事件处理线程，针对指定表的DML事件进行拆分，并在多个slave工作线程中，轮流选择执行。不符合拆分规则的，仍然保持串行处理。

```
if (simple_replication_mode)
if (simple_replication_mode
    && (rli->is_worker_picked == true || event_deal_with_rpl_simple_table(this)))
{
   //simple replication
   ret_worker = pick_one_worker(rli, this);
    in_simple_replication_mode = true;
   DBUG_ASSERT(simple_replication_mode || num_dbs == OVER_MAX_DBS_IN_EVENT_MTS
               || num_dbs == 1);
}
```

等待执行列表

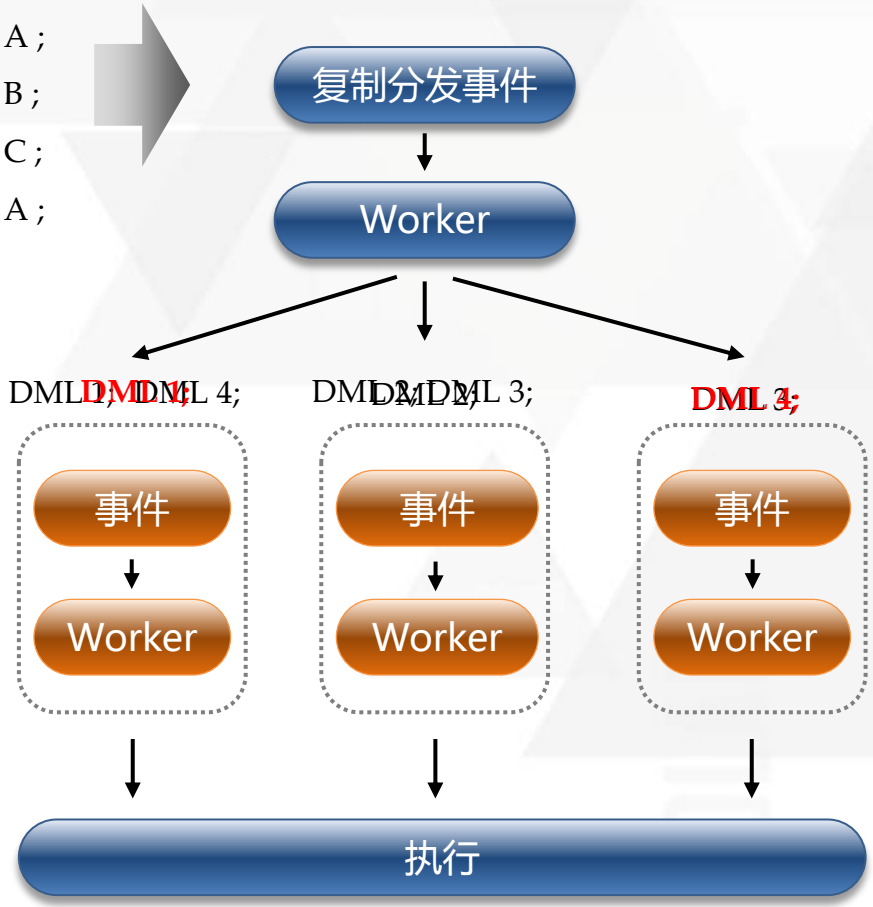
DML 1： of table A ;

DML 2： of table B ;

DML 3： of table C ;

DML 4： of table A ;

……



操作顺序被打乱会引起数据一致性问题

改进后只对指定表进行事件拆分，其他表保持原有顺序

改进后的复制分发处理方式

问题2

- 在成功实现针对指定表进行事件拆分后，我们意外地发现在slave运行不久后就会出现复制线程夯住的现象。并且无法通过stop slave停止，整个实例出现挂死的情况。

- 经过调试排查，分析最可能的问题在于事件拆分后事务被破坏了。多语句事务被拆分后，可能出现事务开始和结束的标识分别在两个工作线程中执行。

解决2

- 找到问题后解决方案也就随之确定，对于多语句的事务，在拆分后必须进行补齐，把事务的开始和结束标志补充完整。

事务开始时所需的信息记录

```
if (simple_replication_mode)
{
   rli->begin_event_started = true;
   rli->is_worker_picked = false;
  rli->normal_event_processed = false;
  rli->normal_event_workerid = 0;
  rli->curr_group_special_event_num = 0;
}
```

事务中事件的处理

```
if (simple_replication_mode && event_deal_with_rpl_simple_table(this))
{
   //simple replication
   ret_worker = pick_one_worker(rli, this);
    in_simple_replication_mode = true;
    return ret_worker;
}
```

事务结束时

```
if (opt_mts_slave_parallel_simple && rli->curr_group_special_event_num > 0
    && !rli->normal_event_processed && get_type_code() == XID_EVENT)
{
  goto repl_simple_end_group;
}
```

等待执行列表

Begin

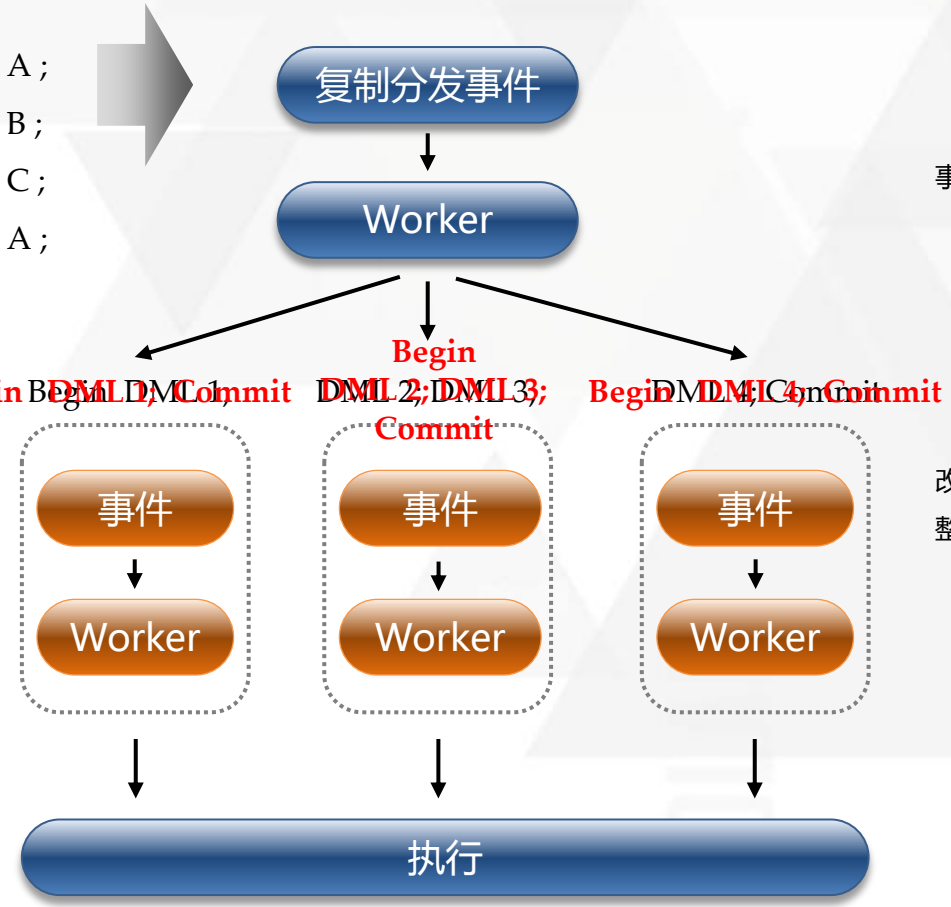
DML 1： of table A ;

DML 2： of table B ;

DML 3： of table C ;

DML 4： of table A ;

Commit



事务完整性被破坏，运行不正常

改进后，拆分前后的事务保持各自完整性

改进后的复制分发处理方式

实现

- 在修正了以上两个大问题和其余一堆小问题后，初步的可用成品终于诞生了。总共完成11个文件的修改，一千多行的变化。

sql/log_event.cc
sql/log_event.h
sql/mysqld.cc
sql/mysqld.h
sql/rpl_info.h
sql/rpl_rli.cc
sql/rpl_rli.h
sql/rpl_rli_pdb.cc
sql/rpl_rli_pdb.h
sql/rpl_slave.cc
sql/sys_vars.cc

- 配置实例：

slave_parallel_workers=5（并行工作线程数）
slave_parallel_simple=1（是否开启自定义并行复制）
slave_parallel_simple_tables=
history;history_uint;trends;trends_uint
（对这4张表的操作进行拆分）

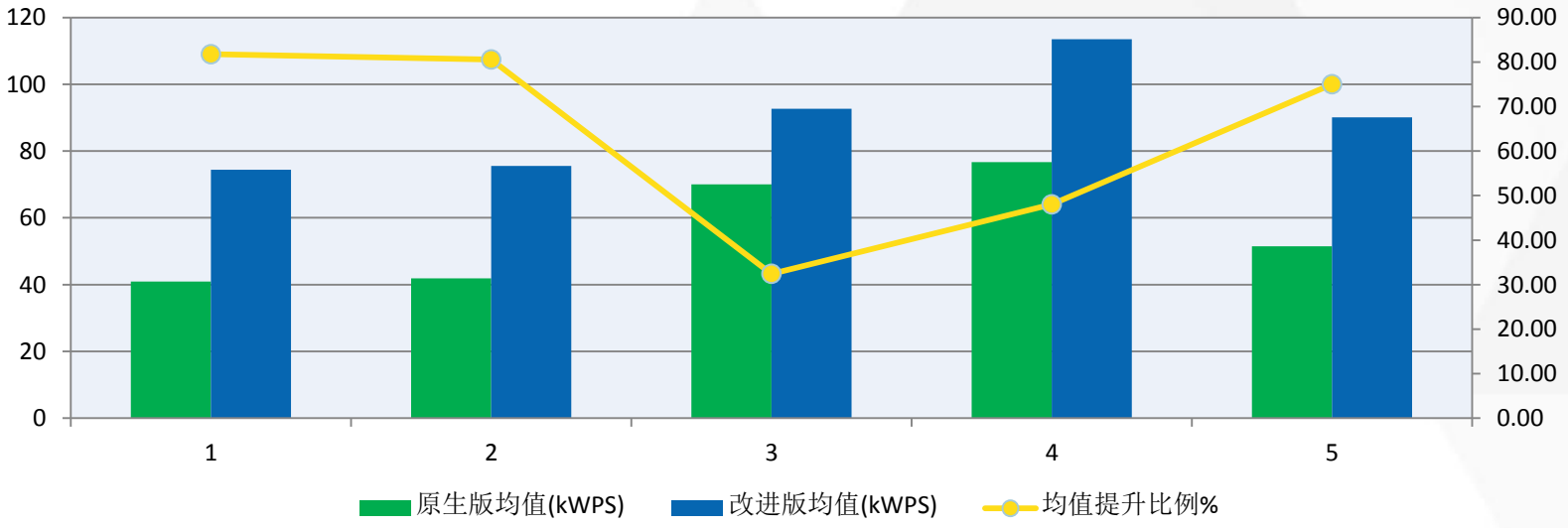
- 线上zabbix运行超过3个月，功能性和稳定性都已经得到验证。

VS

- 测试方法：在同一台服务器上分别开关并行复制，停用复制SQL进程一段时间后开启，对比测试复制容量上限。
- 测试环境：16Core CPU + 64G内存 + PCIE SSD 磁盘，zabbix数据库大小超过1T。
- 测试场景：在MySQL不同buffer size和redolog个数的配置组合下对比。并在其中一个场景增加自定义查询增加压力。

结果

测试场景	[1] 16Gbuf+2log	[2] 16Gbuf+4log	[3] 32Gbuf+2log	[4] 32Gbuf+4log	[5] 32Gbuf+4log+query
原生版均值(kWPS)	40.92	41.84	70	76.7	51.47
改进版均值(kWPS)	74.38	75.56	92.71	113.55	90.09
均值提升比例%	81.77%	80.59%	32.44%	48.04%	75.03%



结论

- 根据测试结果来看，在服务器性能负载较高的情况下（例如内存交换频繁、刷磁盘动作频繁），并行复制的提升更加明显，最高可达到80%的提升。

- 在最接近现实情况的场景5（增加了额外的数据查询压力），并行复制的性能提升可以达到75%。

不足

- 自增长表的支持问题：对SET INSERT_ID的事件没有特殊处理，当做普通事件进行了拆分，导致主从数据的自增长值不一致。
- 配置只能指定表名，多库存在同名表时无法区分。
- 复制是MySQL的核心功能，当进行版本升级时（即使是小版本），功能合并到新版本的成本很高。

收获

- 显式收获：
  1. 已应用到各zabbix系统，提升了复制的容量上限。
  2. 新增slave节点用于迁移或扩容时，大大缩短了部署时间。

- 隐式收获：

  成功的第一步，获得了成长和经验，为后续的源码优化工作奠定了基础。

  例如，在这之后顺利而快速地完成了另一个源码改进版， slow log记录机制优化。增加了根据完整执行时间（包含锁等待）来进行慢查询判断的机制。并且已经成功在大范围生产业务环境稳定运行。

未来

- 短期计划：进一步进行功能性改进，解决现有问题。
- 长期计划：复制功能改进插件化，便于不同版本或分支产品之间迁移。


ChinaUnix
ITPUB
THANKS