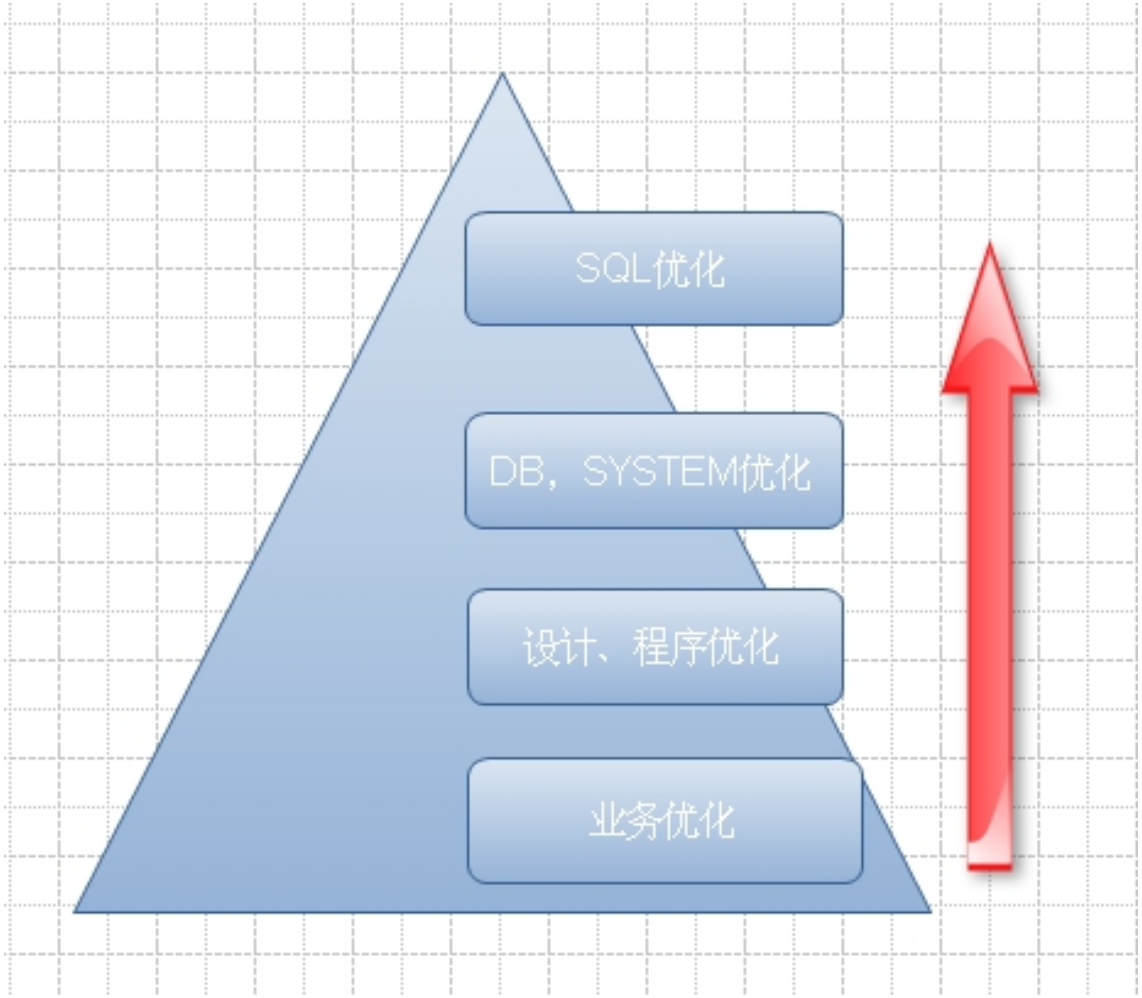
RDS最佳实践

阿里巴巴数据库技术 - 玄惭

2013.06.16

RDS最佳实践-目录

最佳实践-优化顺序



最佳实践-优化顺序

优化顺序--最佳实践

.从下往上优化顺序，更加彻底

1.首先定位系统的瓶颈，判断引起数据库瓶颈的原因
2.定位瓶颈后，判断能否从业务上进行优化，减小对数据库的压力
3.根据应用访问特点，从设计和实现上优化对数据库的访问
4.对数据库处理能力进行升级：io，cpu，mem
5.调整索引，优化sql

.从上往下的优化顺序，见效快

最佳实践-拆分

一.常见误区：将拆分进行到底

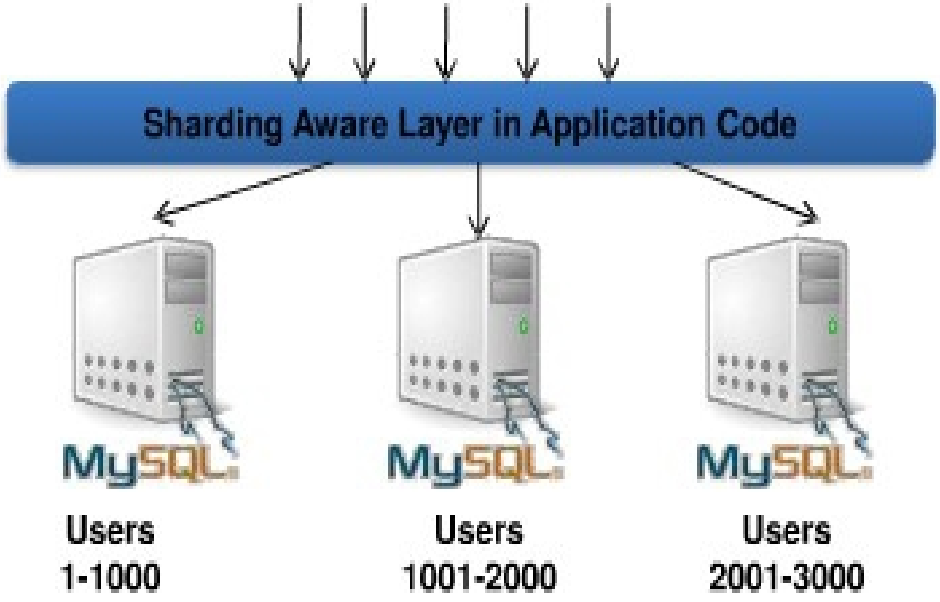
1.我要对一张2百万的表进行拆分
2.只有当主库不能处理写(DML)负载的时候，需要进行拆分
3.拆分对应用增加了复杂度
4.分区是比较安全和简单的拆分
5.读较多的应用可以进行读写分离
6.不是所有的应用都能进行拆分

二.保持简单的架构

1.不是所有的应用都像淘宝一样的
2.过于复杂的架构你是否有能力进行掌控
3.简单的架构给你的优化带来便利

最佳实践-拆分

.单实例RDS容量有限，业务在不断发展，应用压力越来越大。如何解决？
.想设计一个高可扩展的系统，关系数据库的扩展如何解决？
.分布式RDS是什么？



数据访问中间件
MySQL协议，支持各种编程语言
帮助用户做水平拆分

最佳实践-拆分

拆分--最佳实践

绝大多数的应用不需要分库分表
保持简单应用架构
先优化，在考虑拆分
优先考虑垂直拆分，在水平拆分
DRDS帮助你解决水平拆分的难题

最佳实践-性能优化

一.分页优化：

小东 (2013-06-05 18:29:38):
这个可能是什么问题呀。

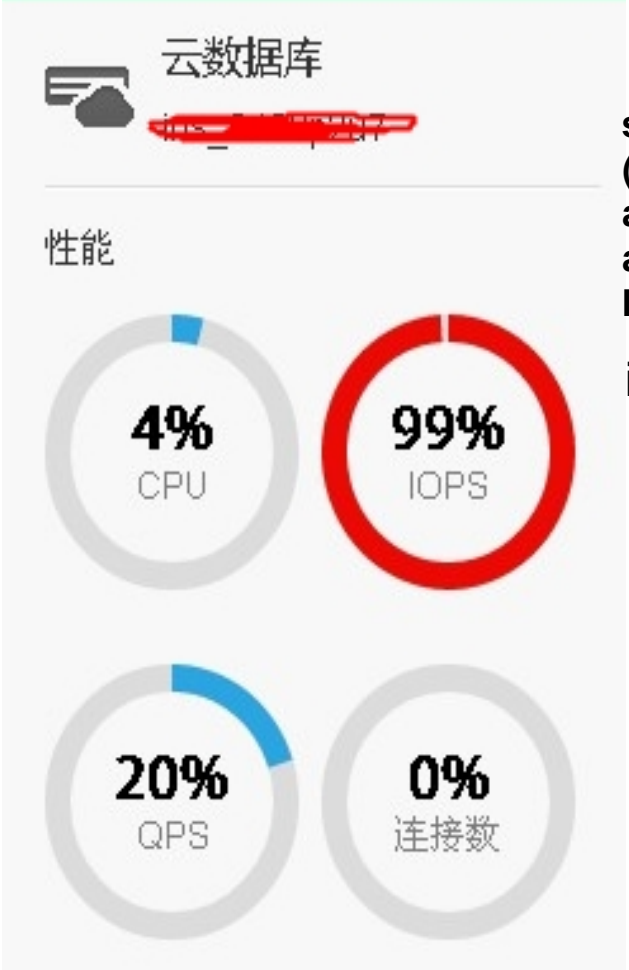
玄惭 (21:42:13):
iops超了

小东 (21:42:31):
嗯嗯 。
这个sql写的看来有问题。

| Query | 51 | Sending data |
select id, ... from t_buyer where sellerId = 765922982
and gmt_modified >= '1970-01-01 08:00:00'
and gmt_modified <= '2013-06-05 17:11:31'
limit 255000, 5000

最佳实践-性能优化

一.分页优化：



玄惭 (21:46:24):

select t2.* from
(select id from t_buyer where sellerId = 765922982
and gmt_modified >= '1970-01-01 08:00:00'
and gmt_modified <= '2013-06-05 17:11:31'
limit 255000, 5000）t1,t_buyer t2 where t1.id=t2.id

index：seller_id,gmt_modified

小东 (21:46:49):
这样为啥会快呀，玄惭，
玄惭 (21:47:00):
你先试一试吧
小东 (21:58:43):
好像很快啊。神奇，这个原理是啥啊。
牛！！！
小东 (21:59:55):
5000 rows in set (4.25 sec)
前面要90秒。

最佳实践-性能优化

一.分页优化--最佳实践

普通limit M，N的翻页写法，往往在越往后翻页的过程中速度越慢，原因mysql会读取表中的前M+N条数据，M越大，性能就越差，

```
select * from t where sellerid=100 limit 100000，20
```

优化后的翻页写法，先查询翻页中需要的N条数据的主键id，在根据主键id回表查询所需要的N条数据，此过程中查询N条数据的主键ID在索引中完成

```
select t2.* from t t1,(select id from t sellerid=100 limit 100000，20) t2
 where t1.id=t2.id;
```

t表中的索引：sellerid字段上创建索引

最佳实践-性能优化

二.Order by COL desc/asc limit N

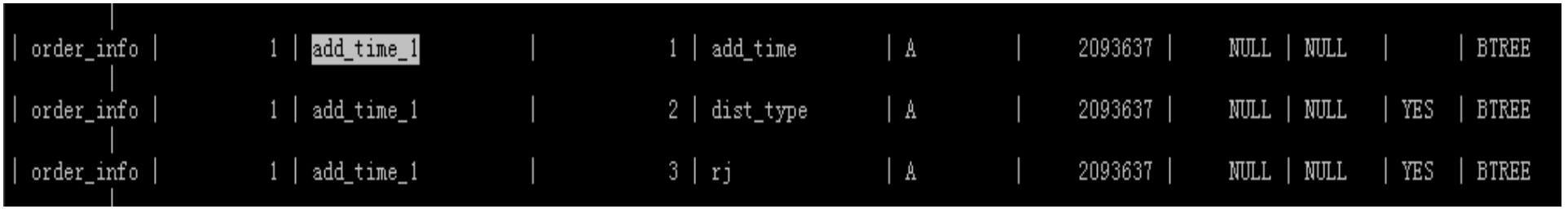
```
SELECT o.* FROM order_info  o  WHERE  is_send=0  AND  o.order_status in (0,1) AND
o.shipping_status = 0 AND o.is_separate > 0 and o.jhd_id=0  group by o.order_id
ORDER BY o.add_time DESC  LIMIT 20
```

```
mysql> explain SELECT o.* FROM order_info  o  WHERE is_send=0  AND  o.order_status in (0,1) AND
o.shipping_status = 0 AND o.is_separate > 0 and o.jhd_id=0  group by o.order_id
ORDER BY o.add_time DESC  LIMIT 20   \G;
*************************** 1. row ***************************
       id: 1
  select_type: SIMPLE
        table: o
         type: index_merge
possible_keys: order_status,shipping_status,is_send,is_separate
          key: shipping_status,is_send
      key_len: 1,2
          ref: NULL
         rows: 146393
        Extra: Using intersect(shipping_status,is_send); Using where; Using filesort
1 row in set (0.00 sec)
```

执行时间:20 rows in set (21.00 sec)

最佳实践-性能优化

二.Order by COL desc/asc limit N ：



order_info	1	add_time_1	1	add_time	A	2093637	NULL	NULL		BTREE
order_info	1	add_time_1	2	dist_type	A	2093637	NULL	NULL	YES	BTREE
order_info	1	add_time_1	3	rj	A	2093637	NULL	NULL	YES	BTREE

```
mysql> explain SELECT o.* FROM order_info  o  WHERE  is_send=0  AND  o.order_status in (0,1) AND
 o.shipping_status = 0 AND o.is_separate > 0 and o.jhd_id=0  group by o.order_id
ORDER BY o.add_time DESC  LIMIT 20\G;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: o
         type: index
possible_keys: order_status,ind_order_info_status,ind_order_info_shiiping_status
          key: add_time_1
      key_len: 779
          ref: NULL
         rows: 4993
        Extra: Using where
1 row in set (0.00 sec)
```

执行时间:20 rows in set (0.00 sec)

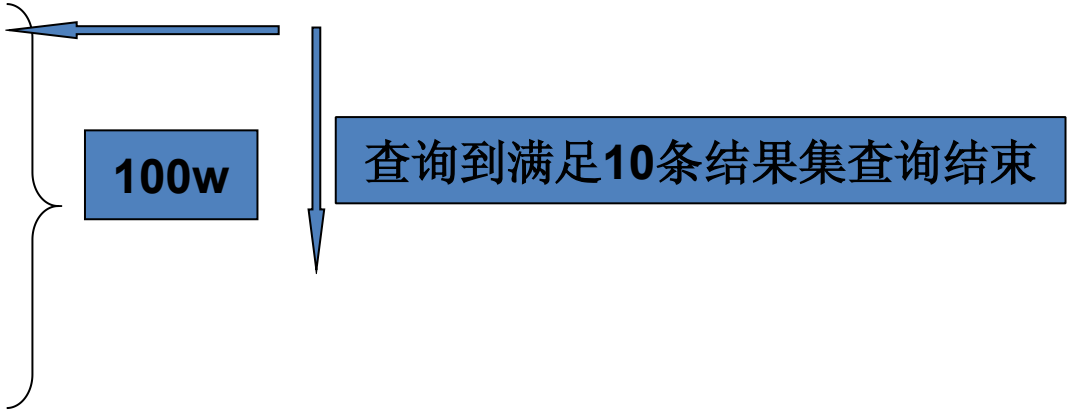
最佳实践-性能优化

二.Order by COL desc/asc limit N--最佳实践

在上述特点的查询中，如果mysql查询数据的顺序与order by的字段一致的，也就是说在排序字段上有索引，mysql沿着索引的顺序读取下来，当满足了查询中N条记录，则立刻返回查询。

sellerid	gmt_create
100	20:00
101	22:01
101	22:07
101	22:33
....	....
....	....
101	23:00
102	21:00

select * from order where sellerid=101 and .....
order by gmt_create limit 10



最佳实践-性能优化

三.根据过滤性创建索引：

hhysc (2013-06-05 16:37:58):

```
SELECT  * FROM    `efast`.`order_info`
WHERE lylx IN (2, 15)
AND (
  (dist_type = 0)
  OR (dist_type = 1 AND rj = 2)
  OR (dist_type = 2 AND rj = 2)
)
AND order_status = 1 AND is_send = 0
ORDER BY   shipping_time,   pay_time, add_time ASC
```

这种SQL能优化的不 

3个order by 字段

这个也是很慢的一个SQL，但是是主流程上面要用到的

玄惭 (2013-06-05 16:43:49):

可以

最佳实践-性能优化

三.根据过滤性创建索引：

玄惭 **(2013-06-05 16:43:49):**

可以

```
mysql> select count(*) from order_info where  lylx IN (2, 15) AND order_status = 1 ;
+----------+
| count(*) |
+----------+
|    12361 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*) from order_info where  lylx IN (2, 15)  AND is_send = 0;
+----------+
| count(*) |
+----------+
|     2548 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from order_info where  lylx IN (2, 15) AND order_status = 1 AND is_send = 0;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)
```

最佳实践-性能优化

三.根据过滤性创建索引：

```
mysql> alter table order_info add index ind_order_info_lylx(lylx,order_status,is_send);

mysql>  SELECT  * FROM    `efast`.`order_info` WHERE  lylx IN (2, 15) AND
( (dist_type = 0)    OR (dist_type = 1 AND rj = 2)    OR (dist_type = 2 AND rj = 2) )
AND order_status = 1 AND is_send = 0 ORDER BY    shipping_time,    pay_time,    add_time;
Empty set (0.00 sec)
```

你看看速度吧
hhysc (2013-06-05 16:46:10):

这下很快了

最佳实践-性能优化

三.根据过滤性创建索引--最佳实践

1.创建索引的时候，优先将过滤性高的查询字段放在首位；

2.mysql的索引有前导列的限制，优先将等值的条件的字段放在索引顺序的前列；

3.若有>,<,not in,between,!=的查询条件的字段加入索引列后，后续加入的索引字段都不能起到过滤作用；

最佳实践-性能优化

四.避免函数计算：

```
select a.org_name,count(*) as org_num  from  mh_price_query_log a
 where a.query_from_type='servicportal'  and a.org_id = 10000  and
Date(a.create_date) >= '2013-06-01'  and Date(a.create_date) <= '2013-06-10'
group by a.org_name order by org_num desc
```

```
mysql> explain select a.org_name,count(*) as org_num  from  mh_price_query_log a where a.query_from_type='servicportal'  and a.org_id = 10000  and
Date(a.create_date) >= '2013-06-01'  and Date(a.create_date) <= '2013-06-10'  group by a.org_name order by org_num desc\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: a
         type: ref
possible_keys: ind_mh_price_query_log_gmt
          key: ind_mh_price_query_log_gmt
      key_len: 186
          ref: const,const
         rows: 1249518
        Extra: Using where; Using index; Using temporary; Using filesort
1 row in set (0.00 sec)
```

最佳实践-性能优化

四.避免函数计算：

查看索引:

```
mysql> show index from mh_price_query_log;
+--------------------+------------+---------------------------+--------------+-----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table              | Non_unique | Key_name                  | Seq_in_index | Column_name     | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+--------------------+------------+---------------------------+--------------+-----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| mh_price_query_log |          0 | PRIMARY                   |            1 | id              | A         |     2499036 |     NULL | NULL   |      | BTREE      |         |               |
| mh_price_query_log |          1 | ind_mh_price_query_log_gmt |            1 | org_id          | A         |         200 |     NULL | NULL   |      | BTREE      |         |               |
| mh_price_query_log |          1 | ind_mh_price_query_log_gmt |            2 | query_from_type | A         |         200 |     NULL | NULL   |      | BTREE      |         |               |
| mh_price_query_log |          1 | ind_mh_price_query_log_gmt |            3 | create_date     | A         |         200 |     NULL | NULL   |      | BTREE      |         |               |
| mh_price_query_log |          1 | ind_mh_price_query_log_gmt |            4 | org_name        | A         |         200 |     NULL | NULL   |      | BTREE      |         |               |
+--------------------+------------+---------------------------+--------------+-----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
```

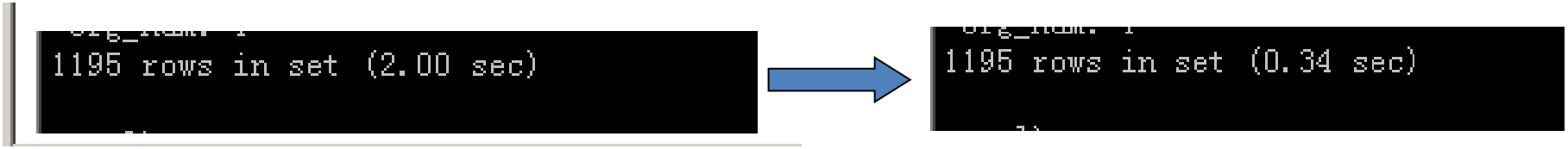
最佳实践-性能优化

四.避免函数计算：

去除函数:

```
mysql> explain select a.org_name,count(*) as org_num  from  mh_price_query_log a where a.query_from_type='servicportal'  and a.org_id = 10000
 a.create_date >= '2013-06-01'  and a.create_date <= '2013-06-10'  group by a.org_name order by org_num desc\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: a
         type: range
possible_keys: ind_mh_price_query_log_gmt
          key: ind_mh_price_query_log_gmt
      key_len: 194
          ref: NULL
         rows: 80360
        Extra: Using where; Using index; Using temporary; Using filesort
1 row in set (0.00 sec)
```

执行时间对比:



最佳实践-性能优化

四.避免函数计算--最佳实践

1.mysql不支持函数索引，所以在查询条件中加入函数计算，则无法使用到索引；

最佳实践-性能优化

五.避免隐士转换：

问题描述：用户网站打开缓慢，质疑RDS性能不好.
可能原因：用户的数据存放在RDS中，网站访问数据库的时间较长
问题排查：通过查看数据库的慢日志，发现大量的慢sql，执行时间超过了2S.

通过登陆数据库排查大量的执行较慢的sql存在：

```
UPDATE USER SET xx=xx+N.N WHERE account=130000870343  LIMIT 10;
SELECT * FROM  USER WHERE account=13056870  LIMIT 10;
```

怀疑在user表上是否建立索引：

```
CREATE TABLE `user` (
`id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
`account` char(11) NOT NULL COMMENT '???',
……
PRIMARY KEY (`id`),
UNIQUE KEY `username` (`account`),
) ENGINE=InnoDB CHARSET=utf8 ;
```

最佳实践-性能优化

五.避免隐士转换：

查看执行计划，居然查询使用了全表扫描：

```
db@3027 16:55:06>explain select * from user where account=13056870343;
+—-+————-+———+——+———————+——+———+——+——+————-+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+—-+————-+———+——+———————+——+———+——+——+————-+
| 1 | SIMPLE | t_user | ALL | username | NULL | NULL | NULL | 799 | Using where |
+—-+————-+———+——+———————+——+———+——+——+————-+
```

我们知道数字的精度是比字符串高的，所以这里做了隐士转换：
to_number(account)=13056870343（to_number为将字符串转换为数字），

```
db@3027 16:55:13&gt;explain SELECT * FROM USER WHERE account='13056870343';
+----+-------------+--------+-------+---------------+----------+---------+-------+------+-------+
| id | select_type | TABLE  | TYPE  | possible_keys | KEY      | key_len | REF   | ROWS | Extra
+----+-------------+--------+-------+---------------+----------+---------+-------+------+-------+
|  1 | SIMPLE      | t_user | const | username      | username | 33      | const |    1 |       |
+----+-------------+--------+-------+---------------+----------+---------+-------+------+-------+
1 ROW IN SET (0.00 sec)
```

最佳实践-性能优化

五.避免隐士转换--最佳实践

发生隐士转换，会导致索引无效，其原理类似于在查询字段上加上了一个函数，所以应该在设计编码阶段注意；

常见的隐士转换：字段定义为字符，而传入条件为数字

最佳实践-性能优化

六.避免无索引：

```
mysql> explain select user_id from users where nick_name= NAME_CONST('_user_nick',_utf8'jxxx' COLLATE 'utf8_general_ci') limit 1;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	users	ALL	NULL	NULL	NULL	NULL	1988700	Using where

1 row in set (0.00 sec)

```
mysql> alter table users add index ind_users_nick(nick_name);
Query OK, 0 rows affected (1 min 10.27 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
Empty set (7.38 sec)
```

```
Empty set (0.00 sec)
```

最佳实践-性能优化

六.避免无索引--最佳实践

1.所有上线的**sql**都要经过严格的审核，创建合适的索引，这是保障数据库运行稳定的基本要求；

2.可以通过**explain**查看**sql**的执行计划，判断是否使用到了索引；

最佳实践-性能优化

七.避免子查询：

Execute | 1179 | Sending

```
select tradedto0_.* from a1 tradedto0_ where tradedto0_.tradestatus='1' and
tradedto0_.tradeoid in
 (select orderdto1_.tradeoid from a2 orderdto1_ where
orderdto1_.proname like '%??%' or orderdto1_.procode like '%??%')
and tradedto0_.undefine4='1' and tradedto0_.invoicetype='1' and
tradedto0_.tradestep='0' and (tradedto0_.orderCompany like '0002%')
order by tradedto0_.tradesign ASC, tradedto0_.makertime desc limit 15;
```

explain：

```
+----+--------------------+------------+------+---------------+------+---------+------+-------+----------------------
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+----+--------------------+------------+------+---------------+------+---------+------+-------+----------------------
| 1 | PRIMARY | tradedto0_ | ALL | NULL | NULL | NULL | NULL | 27454 | Using where; Using filesort |
| 2 | DEPENDENT SUBQUERY | orderdto1_ | ALL | NULL | NULL | NULL | NULL | 40998 | Using where |
+----+--------------------+------------+------+---------------+------+---------+------+-------+----------------------
```

最佳实践-性能优化

七.避免子查询：

在t表的执行结果：

db@3306 ：

select orderdto1_.tradeoid from t orderdto1_ where
orderdto1_.proname like '%??%' or orderdto1_.procode like '%??%';
Empty set (0.05 sec)

结果集为空，所以需要将t表的结果集做作为驱动表；

最佳实践-性能优化

七.避免子查询：

改写sql:
通过上面测试验证，普通的mysql子查询写法性能上是很差的，
为mysql的子查询天然的弱点，需要将sql进行改写：

```
select tradedto0_.* from a1 tradedto0_ ,
(
    select orderdto1_.tradeoid from a2 orderdto1_
    where orderdto1_.proname like '%??%' or orderdto1_.procode like '%??%'
)  t2
where tradedto0_.tradestatus='1' and (tradedto0_.tradeoid=t2.tradeoid ) and
tradedto0_.undefine4='1' and tradedto0_.invoicetype='1'
and tradedto0_.tradestep='0' and (tradedto0_.orderCompany like '0002%')
order by tradedto0_.tradesign ASC, tradedto0_.makertime desc limit 15;
```

查看执行计划：

```
+----+-------------+-----------+-------+--------------+---------------------+---------+------+-------+----------------
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+----+-------------+-----------+-------+--------------+---------------------+---------+------+-------+----------------
| 1 | PRIMARY | NULL | NULL | NULL | NULL | NULL  | Impossible WHERE noticed after reading const tables
| 2 | DERIVED | orderdto1_ | index | NULL | ind_a2 | 777 | NULL | 41005 | Using where; Using index |
```

最佳实践-性能优化

七.避免子查询：

子查询在mysql的常见的5.0，5.1，5.5版本中都存在较大风险，使用不当则会造成严重的性能问题，建议将子查询改为关联的形式：

```
select * from order where sellerid in (select userid from user where nick='xuancan');
```

改写为：

```
select  o.* from order  o ,(select userid from user where nick='xuancan') u
where o.sellerid=o.userid;
```

最佳实践-常见问题

一.导入超时

Lost connection to MySQL server during query

第一步：

```
[root@piwik-ce tmp]# more 1.sh
date;
mysql -h 10.242.180.1 -upiwik -pbuynow piwik -e "source /tmp/piwik.dmp"
date;
```

第二步：

```
nohup sh 1.sh > /tmp/load.log &
```

第三步：

```
[root@piwik-ce ~]# more /tmp/load.log
Tue May  7 16:07:50 CST 2013
ERROR 2013 (HY000) at line 4912 in file: '/tmp/piwik.dmp': Lost connection to
 MySQL server during query
Tue May  7 18:23:17 CST 2013
```

第四步：

```
[root@piwik-ce ~]# sed -n '4912 p' /tmp/piwik.dmp
/*!40000 ALTER TABLE `piwik_log_link_visit_action` ENABLE KEYS */;
```

最佳实践-常见问题

一.导入超时：

Lost connection to MySQL server during query

```
mysql> desc piwik_log_link_visit_action;
+----------------------+-----------------+------+-----+---------+----------------+
| Field                | Type            | Null | Key | Default | Extra          |
+----------------------+-----------------+------+-----+---------+----------------+
| idlink_va            | int(11) unsigned | NO   | PRI | NULL    | auto_increment |
。。。。。。
| custom_var_v5        | varchar(200)    | YES  |     | NULL    |                |
+----------------------+-----------------+------+-----+---------+----------------+
20 rows in set (0.00 sec)

mysql> select count(*) from piwik_log_link_visit_action;
+----------+
| count(*) |
+----------+
| 34458315 |
+----------+
```

```
e it  | NULL                        |
| 809680 | root            | 127.0.0.1:44875     | piwik | Query       |       0 | NULL
    | show full processlist |
| 809841 | piwik           | 10.135.40.26:58686  | piwik | Sleep       |    1183 |
    | NULL                        |
```

LVS和RDS的连接超时时间为2个小时

最佳实践-常见问题

一.导入超时：

建议使用在导入RDS前，将myisam表转为innodb表；

最佳实践-常见问题

二.definer权限问题

1227 – Access denied; you need the SUPER privilege for this operation ;

在使用**mysqldump**逻辑迁移到**rds**的过程中，用户数据库中含有存储过程，由于 **mysqldump**在**dump**存储过程的时候会把存储过程的**definer dump**出来，那么在迁移到**rds**过程中，由于用户数据库的**definer**和**rds**的用户和**ip**不一致而导致迁移失败

/*!50003 CREATE*/ /*!50020 DEFINER=`root`@`localhost`*/ /*!50003 PROCEDURE `sp_xxx`()

最佳实践-常见问题

二.definer权限问题

处理前：

```
DELIMITER ;;
/*!50003 CREATE*/ /*!50020 DEFINER=`root`@`localhost`*/ /*!50003 PROCEDURE `sp_xxx`()
BEGIN
select 1;
END */;;
DELIMITER
```

处理后：

```
$mysqldump -uroot test --no-data -R  |sed -e 's/DEFINER[ ]*=[ ]*[^*]*\*/\*/'  >/tmp/load_rds.dmp
 DELIMITER ;;
/*!50003 CREATE*/ /*!50020 /*!50003 PROCEDURE `sp_xxx`()
BEGIN
select 1;
END */;;
DELIMITER ;
```

最佳实践-常见问题

二.definer权限问题--最佳实践

由于trigger，procedure，functions，view包含有definer信息，所以用户从本地导入到RDS会报错，建议用户在导入RDS前将definer信息去除。

最佳实践-常见问题

三.Query cache

VM test:

第一次执行：mysql> select count(*) as org_num from mh_price_query_log a
where a.query_from_type='servicportal' and a.org_id = 10000
and a.create_date >= '2013-03-01' and a.create_date<= '2013-06-10'
group by a.org_name   limit 1;
| org_num |
+---------+
| 5990 |
+---------+
1 row in set (2.88 sec)

第二次执行：
mysql> select count(*) as org_num from mh_price_query_log a
where a.query_from_type='servicportal' and a.org_id = 10000
and a.create_date >= '2013-03-01' and a.create_date<= '2013-06-10'
group by a.org_name   limit 1;
| org_num |
| 5990 |
+---------+
1 row in set (0.00 sec)

最佳实践-常见问题

三.Query cahce

RDS test

第一次执行：mysql> select count(*) as org_num from mh_price_query_log a
where a.query_from_type='servicportal' and a.org_id = 10000
and a.create_date >= '2013-03-01' and a.create_date<= '2013-06-11'
group by a.org_name   limit 1;
| org_num |
| 5990 |
+---------+
1 row in set (2.70 sec)

第二次执行：
mysql> select count(*) as org_num from mh_price_query_log a
where a.query_from_type='servicportal' and a.org_id = 10000
and a.create_date >= '2013-03-01' and a.create_date<= '2013-06-11'
group by a.org_name   limit 1;
| org_num |
| 5990 |
+---------+
1 row in set (2.62 sec)

最佳实践-常见问题

三.Query cahce

VM：

```
mysql> show global variables like '%query_cache%';
+------------------------------+----------+
| Variable_name                | Value    |
+------------------------------+----------+
| have_query_cache             | YES      |
| query_cache_limit            | 1048576  |
| query_cache_min_res_unit     | 4096     |
| query_cache_size             | 33554432 |
| query_cache_type             | ON       |
```

RDS：

```
mysql> show global variables like '%query_cache%';
+------------------------------+---------+
| Variable_name                | Value   |
+------------------------------+---------+
| have_query_cache             | YES     |
| query_cache_limit            | 1048576 |
| query_cache_min_res_unit     | 4096    |
| query_cache_size             | 0       |
| query_cache_type             | ON      |
```

最佳实践-常见问题

三.Query cahce--最佳实践

.query cache简单粗暴的失效策略，任何一行数据被修改，所有关于这个表的cache都会失效，需要重建,失效通过single mutex控制,可能会有比较严重的锁竞争:Waiting on query cache mutex

.RDS默认是关闭Query cache,用户可以根据自己的应用特点，决定是否打开query cache；

.可以将query_cache_type设置为demand，并结合在sql提示符中提示具体的sql语句是否使用query cache；

最佳实践-常见问题

四.死锁问题

1.通常来说，死锁都是应用设计的问题，通过调整业务流程、数据库对象设计、事务大小，以及访问数据库的SQL语句，绝大部分死锁都可以避免;
2.发生死锁后，InnoDB一般都能自动检测到，并使一个事务释放锁并回退，另一个事务获得锁，继续完成事务;
3.用户发现自己的网站经常由于出现死锁，而导致部分用户无法进行正常的业务逻辑，通过在show engine innodb status发现出现死锁的sql：

最佳实践-常见问题

四.死锁问题

例子：

```
select p.id as id,
       p.number as number,
       p.amount as amount,
       status_id,
       p.account as account,
       p.created as created,
       p.updated as updated,
       areacode,
       service_id,
       u.type as type  from t_phone p
 inner join t_haoduan h on left(p.number, 7) = h.hao
 inner join t_city c on h.city_id = c.id
 inner join t_user u on p.account = u.account
 WHERE p.service_id = 100   and p.status_id = 1
 and areacode in ('0577') limit 1   for update
```

```
UPDATE t_phone
SET number = '13967556619',
name = null, amount = '100.00
yue = '0', status_id = 3,
service_id = '1001',
account = '18696644789',
created = '1342676880',
updated = 1342677209,
ip = '124.232.150.133',
again = null
WHERE (id = '2345422');
```

最佳实践-常见问题

四.死锁问题

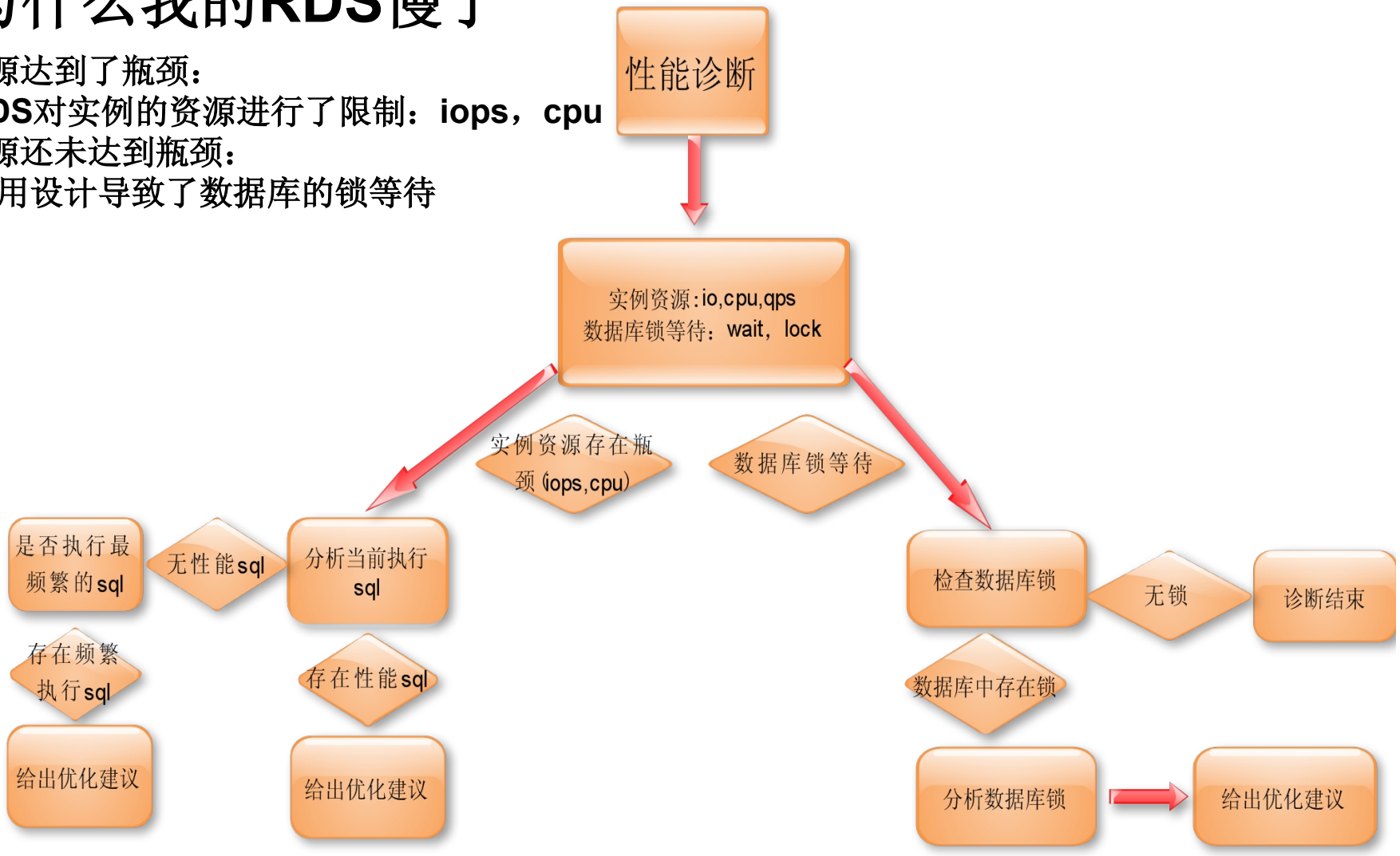
该系统先是查询出需要更新的记录，锁住该条记录（**for update** 其他**session**不能读取和更新），然后在对该条记录进行更新；建议用户将悲观锁的更新方式换为乐观锁的更新方式：

session1：
select id,status_id from t where status_id=1;
update t set status_id=6 where id=int and status_id=1; 更新一条
commit;
session2:
select id,status_id from t where status_id=1;
update t set status_id=6 where id=int and status_id=1;更新0条
commit；

最佳实践-常见问题

四.为什么我的RDS慢了

1.资源达到了瓶颈：
.RDS对实例的资源进行了限制：iops，cpu
2.资源还未达到瓶颈：
.应用设计导致了数据库的锁等待



最佳实践-常见问题

四.锁问题

我的实例资源还有很多，但为什么我的应用程序还是慢？？？

最佳实践-常见问题

四.锁问题

出现问题的sql以及用户反馈为：(可以通过show processlist查看)

Query 20 Updating update t_task set order_count=302, order_total_price=127220 where task_id='feb47f4d683e44c383819eadb

表中的记录数也不多，为什么这条sql能够执行这么就的时间？

首先排查表中是否有索引：

查看表中的索引，task_id为表的主键，sql没有问题；

继续排查，由于该sql的执行状态一直为updating，很有可能该sql一直在等待锁而导致更新时间较长，那么我们看看数据库中锁等待的情况：

查看innodb锁等待情况：

```
select r.trx_id waiting_trx_id,r.trx_query waiting_query, b.trx_id blocking_trx_id,
b.trx_query blocking_query,b.trx_mysql_thread_id blocking_thread,b.trx_started,
b.trx_wait_started from information_schema.innodb_lock_waits w inner join
information_schema.innodb_trx b on b.trx_id =w.blocking_trx_id inner join
 information_schema.innodb_trx r on r.trx_id=w.requesting_trx_id \G
```

最佳实践-常见问题

四.锁问题

```
information_schema@3036 19:19:12>select r.trx_id waiting_trx_id,r.trx_query waiting_query, b.trx_id blocking_trx_id, b.trx_query blocking_quer
x_mysql_thread_id blocking_thread,b.trx_started,b.trx_wait_started from information_schema.innodb_lock_waits w inner join information_schema.i
trx b  on b.trx_id =w.blocking_trx_id inner join information_schema.innodb_trx r on r.trx_id =w.requesting_trx_id\G;
*************************** 1. row ***************************
  waiting_trx_id: 31BC474
   waiting_query: update t_task set order_count=296, order_total_price=126620 where task_id='feb47f4d683e44c383819eadbd069597'
 blocking_trx_id: 31BC4A1
  blocking_query: NULL
 blocking_thread: 370865
     trx_started: 2013-05-20 19:15:45
trx_wait_started: NULL
1 row in set (0.00 sec)

ERROR:
No query specified

information_schema@3036 19:19:14>select * from processlist where id = 370865\G
*************************** 1. row ***************************
     ID: 370865
   USER: jusr4qnixr3m
   HOST: 10.241.51.18:49714
     DB: softrighttaobao
COMMAND: Sleep
   TIME: 201
  STATE:
   INFO: NULL
```

造成等待的线程

可以看到该**update**语句被另外一个线程被堵住了，该线程的操作造成了很有可能没有提交事务或者在做一个大事务，导致这个整个**update**被**lock**住；

最佳实践-常见问题

四.锁问题

进一步观察innodb内部的锁情况，发现show engine innodb status中的信息：

```
LATEST DETECTED DEADLOCK
*** (1) TRANSACTION:
update t_new_msg_main_statistics set link_visit_num=449, visitor_num=319, msg_quality=372
 where id='13ab91040b914a729b363dcc2be367c0'
*** (2) TRANSACTION:
update t_task set msg_not_arrival=39 where task_id='899e2064463442d6b7fc08b04677aae3'
```

输出信息中显示t_new_msg_main_statistics与t_task发生了死锁的现象

在给出上面的信息后，让用户进一步排查t_task和t_new_msg_main_statistics表所涉及的业务，

t_task和t_new_msg_main_statistics表是放在同一个事务T1中，同时发现这t_new_msg_main_statistics表还涉及另外删除操作的大业务，其中的部分删除操作是和更新T_New_Msg_Main_Statistics表在另外一个事务里面T2；

杭州佑软科技有限公司:佑5 (2013-05-20 19:44:42):
像是由于一张表太大，删除太慢造成的
电猫 (2013-05-20 19:45:10):
谢谢，玄惭。
你这张表删除了这么长时间呀！
上周就反映慢了。
杭州佑软科技有限公司:佑5 (2013-05-20 19:45:57):
是的，那张表100W的数据量
电猫 (2013-05-20 19:46:50):
100W也不应该删除了3-4天呀！
杭州佑软科技有限公司:佑5 (2013-05-20 19:47:03):
是其中的部分删除
这个删除操作是和更新T_New_Msg_Main_Statistics表在一个事务里面的
杭州佑软科技有限公司:佑5 (2013-05-20 20:00:24):
应该就是这个问题了，跑了一会了
谢谢大家了，辛苦了

最佳实践-应用优化

一.mysql 排量提交:

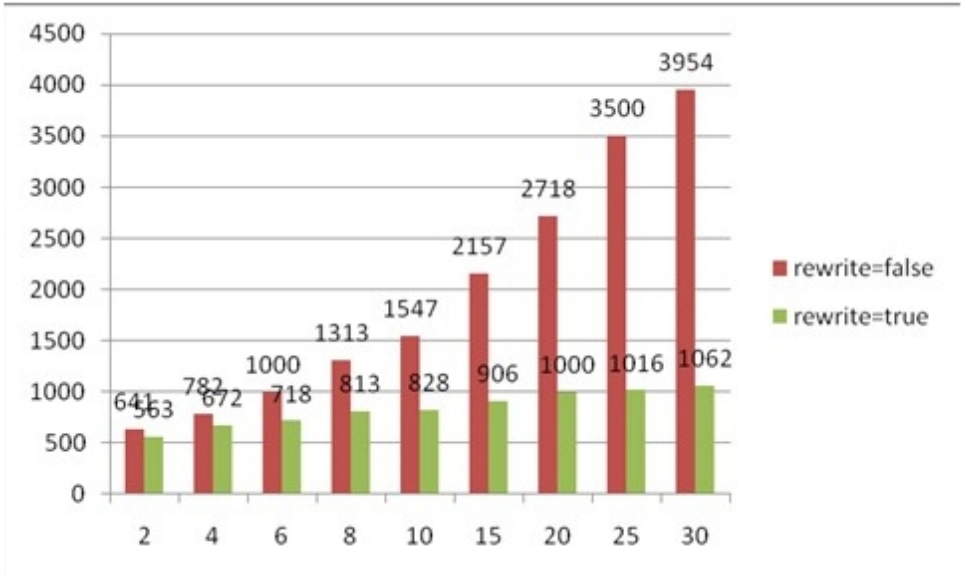
JDBC Driver版本从5.0.4升级到5.1.17。
连接属性中加入rewriteBatchedStatements=true参数

```
mmpSqlMapClient.startTransaction(); // 使用事务
mmpSqlMapClient.startBatch(); // 批量提交
for (ChannelLayoutDO channelLayout: userChannelLayoutList) {
    mmpSqlMapClient.update("UserChannelLayoutDAO.updateSort", channelLayou
}
mmpSqlMapClient.executeBatch();
mmpSqlMapClient.commitTransaction();
```

最佳实践-应用优化

一.mysql 排量提交:

```
50 0.216390   10.13.35.51    10.232.64.107   MySQL  Request Command: Query : SET NAMES gbk
51 0.217350   10.232.64.107  10.13.35.51     MySQL  Response OK
52 0.217572   10.13.35.51    10.232.64.107   MySQL  Request Command: Query : SET character_set_results = NULL
53 0.218501   10.232.64.107  10.13.35.51     MySQL  Response OK
54 0.218778   10.13.35.51    10.232.64.107   MySQL  Request Command: Query : SET autocommit=1
55 0.219720   10.232.64.107  10.13.35.51     MySQL  Response OK
56 0.219961   10.13.35.51    10.232.64.107   MySQL  Request Command: Query : SET sql_mode='STRICT_TRANS_TABLES
57 0.220886   10.232.64.107  10.13.35.51     MySQL  Response OK
58 0.424274   10.232.64.107  10.13.35.51     MySQL  [TCP Retransmission] Response OK
59 0.424287   10.13.35.51    10.232.64.107   TCP    3939 > 3306 [ACK] Seq=1190 Ack=5741 Win=64958 Len=0
60 12.897349  10.13.35.51    10.232.64.107   MySQL  Request Command: Query :          update [redacted]channel_
61 12.898543  10.232.64.107  10.13.35.51     MySQL  Response OK
62 12.899621  10.13.35.51    10.232.64.107   MySQL  Request Command: Query :          update [redacted]_channel_
63 12.900729  10.232.64.107  [illegible]     MySQL  Request Command: Query : SET character_set_[illegible]
64 12.901526  10.13.35.51    10.232.64.107   MySQL  Response OK
65 12.902628  10.232.64.107  10.13.35.51     MySQL  Request Command: Query : SET autocommit=1
66 12.902820  10.13.35.51    10.232.64.107   MySQL  Response OK
67 12.903939  10.232.64.107  10.13.35.51     MySQL  Request Command: Query : SET sql_mode='STRICT_
68 12.904122  10.13.35.51    10.232.64.107   MySQL  Response OK
69 12.905222  10.232.64.107  10.13.35.51     MySQL  [TCP Retransmission] Response OK
70 12.905404  10.13.35.51    10.232.6
71 12.906499  10.232.64.107  10.13.35
72 12.906680  10.13.35.51    10.232.6
73 12.907775  10.232.64.107  10.13.35
74 12.907955  10.13.35.51    10.232.6
75 12.909048  10.232.64.107  10.13.35
76 12.912287  10.13.35.51    10.232.6
77 12.913668  10.232.64.107  10.13.35
78 12.913877  10.13.35.51    10.232.6
79 12.914973  10.232.64.107  10.13.35
80 12.915153  10.13.35.51    10.232.6
81 12.916243  10.232.64.107  10.13.35
82 12.916421  10.13.35.51    10.232.6
83 12.917525  10.232.64.107  10.13.35
```



最佳实践-应用优化

二.jdbc连接重连

“com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link failure”。

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">
    <property name="driverClassName" value="${jdbc.driverClassName}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
    <property name="initialSize" value="3" />
    <property name="maxActive" value="50" />
    <property name="maxIdle" value="10" />
    <property name="minIdle" value="3" />
    <property name="logAbandoned" value="true" />
    <property name="removeAbandoned" value="false" />
    <property name="removeAbandonedTimeout" value="180" />
    <property name="maxWait" value="1200000" />
    <!-- add by yidong(mysql setting) -->
    <property name="testWhileIdle" value="true"/>
    <property name="validationQuery" value="select 1"/>
    <property name="timeBetweenEvictionRunsMillis" value="1800000"/>
    <property name="minEvictableIdleTimeMillis" value="3000000"/>
    <property name="testOnBorrow" value="true"/>
</bean>
```

以下为新增配置

最佳实践-应用优化

三.DNS Cache开启

```
for i in $(seq 1 10000) ; do
mysql -h xxx.mysql.rds.aliyuncs.com -u xx -pxxxx  -N -s -e "select now()"
done
```

测试一：DNS直连
07:54:04 CST
07:55:18 CST
耗时：56+18=74s

测试二：DNS cache 开启
07:56:19 CST
07:57:11 CST
耗时41+11=52s

最佳实践-应用优化

三.DNS Cache开启

1.开启dns cache：
A．安装nscd
apt-get install nscd
B．启动nscd：
[root@jstuycdwpk74uf ~]# service nscd start
Starting nscd: [OK]
C．检查是否启动：
[root@jstuycdwpk74uf ~]# ps -ef|grep nscd
nscd 21231 1 0 17:20 ? 00:00:00 /usr/sbin/nscd
2.修改dns解析文件：添加options timeout:1 attempts:1
nameserver 10.242.197.247
nameserver 10.242.197.248
options timeout:1 attempts:1
wget -O tunldns 223.4.92.61/tunldns.sh && chmod +x tunldns && ./tunldns

最佳实践-应用优化

三.DNS Cache开启--最佳实践

开启DNS cache对于类似PHP短连接的应用效果是非常明显的，大大减小了DNS的访问速度；

最佳实践-工具实践

1.MySQL内部提供的工具和视图

.EXPLAIN
.SHOW PROFILE
.SHOW GLOBAL STATUS
.SHOW ENGINE INNODB STATUS
.SHOW ENGINE INNODB MUTEX
.INFORMATION_SCHEMA
.PERFORMANCE_SCHEMA

最佳实践-工具实践

2.DBA开发：实时查看当前执行sql：orztop

./orztop -h=127.0.0.1 -P=3009 -u=aurora -p=xxxx -t=1

```
MySQL Processlist Info :                              [2013-06-12 20:34:54      127.0.0.1]

[MySQL status]  Ins/Upd/Del/Sel:0/0/0/0  Lor:559718  Hit%:100.00  Threads_running:4
[Command info]  Sleep:71  Connect:2  Binlog Dump:1  Query:1  Daemon:1  Execute:1  => Total Proc [77]
[State   info]  Waiting on empty queue:1  Sending data:1

     Id               Host           User             DB         Command      Time    State
     --               ----           ----             --         -------      ----    -----
  36681          localhost event_scheduler                        Daemon    301603    Waiting on empty queue
1495987 10.200.169.113:57704      prodbisdbw       prodbisdb       Execute       10    Sending data
 ==> [ SQL ] INSERT INTO FACT_IMEI_INFO_THEME (IMEI,DAY_ID,MAC_ID,ROM_VERSION,THEME_VIEW_TIMES,THEME_DL_TIMES,FIRST_THEME_VIEW_TIME,LAST_
TIME,FIRST_THEME_DL_TIME,LAST_THEME_DL_TIME,CRT_DATE) SELECT IMEI,DAY_ID,MAC_ID,ROM_VERSION,THEME_VIEW_TIMES,THEME_DL_TIMES,FIRST_THEME_
ST_THEME_VIEW_TIME,FIRST_THEME_DL_TIME,LAST_THEME_DL_TIME,CRT_DATE FROM BASE_IMEI_INFO_THEME
1495995      127.0.0.1:47384          aurora                        Query         0
 ==> [ SQL ] show full processlist
```

最佳实践-工具实践

3.开源社区：pt-online-schema-change

1.添加字段：

```
./pt-online-schema-change  --u=test123 --host=qianyi.mysql.rds.aliyuncs.com
 --port=3306  --password=hell05a --alter="add column is_sign_2 int(11)"
 D=qianyi,t=test   --execute
```

2.添加索引：

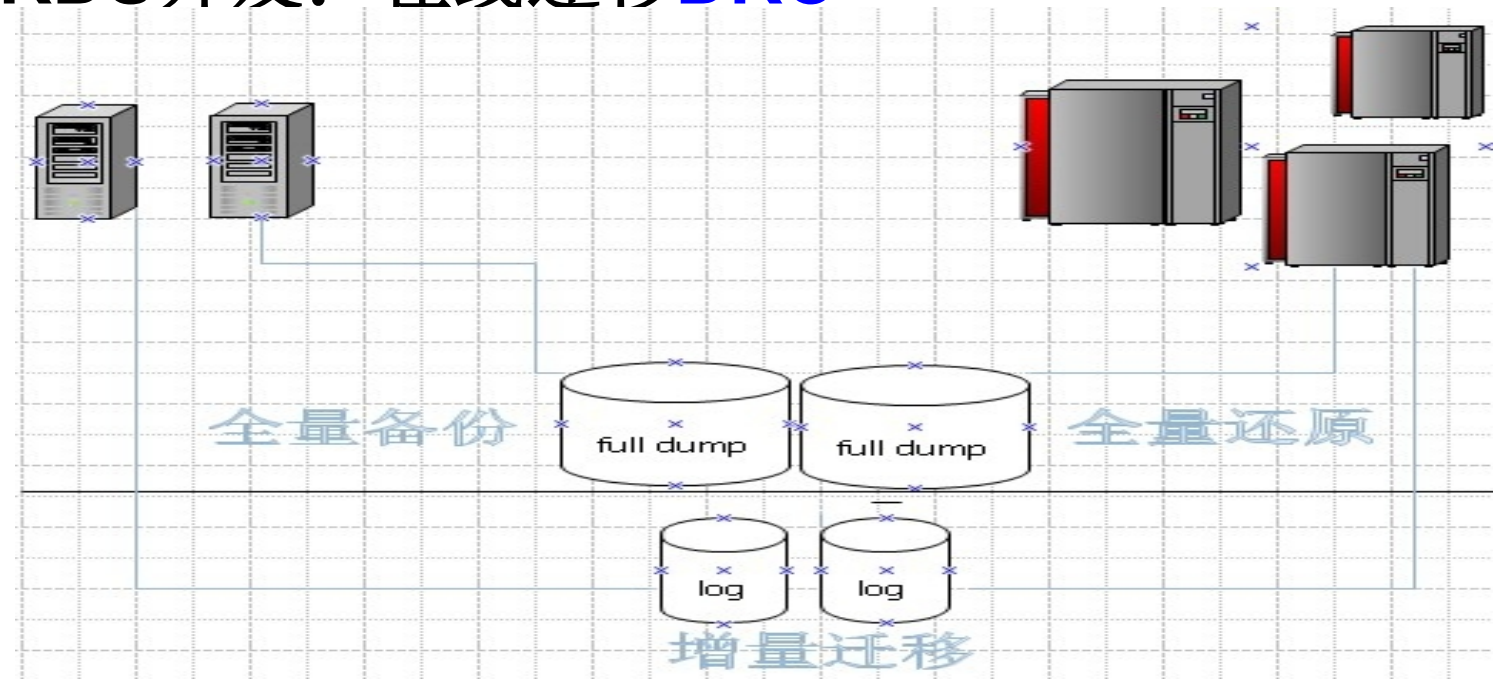
```
./pt-online-schema-change  --u=test123 --host=qianyi.mysql.rds.aliyuncs.com
 --port=3306  --password=hell05a --alter="add index ind_gmt_create(gmt_create)"
 D=qianyi,t=test   --execute
```

3.修改字段：

```
./pt-online-schema-change  --u=test123 --host=qianyi.mysql.rds.aliyuncs.com
 --port=3306  --password=hell05a --alter="modify column is_sign_2 bigint"
 D=qianyi,t=test   --execute
```

最佳实践-工具实践

4. RDS开发：在线迁移DRC



1.全量dump某个时间点之前的数据，并记录全量dump完成后数据库日志的位置；
2.将全量的备份load到RDS；
3.从刚才全量完成dump后的日志开始，通过解析日志为可执行的sql，远程load到RDS；
4.通过不断的应用log，使得RDS慢慢追上用户数据库；；
5.将用户的数据库设置为只读，继续apply日志到RDS，直到完全同步；
6.切用户应用到RDS，完成切换；

最佳实践-设计优化

一.索引设计要点

1.关系schema的设计是基于数据结构，索引的设计是基于query：

.slow log
.general log
.application log
.show full processlist
.orztop
.pt-query-digest--------最慢执行的sql以及执行最频繁的sql两类

最佳实践-设计优化

一.索引设计要点

2.评估你的索引

A.评估出参与运算的结果集范围：

```
select person_id,person_role_id from moive
where movie_id=1000 and role_id=1--------评估出参与运算的结果集范围
order by nr_role desc;
alter table movie add index ind_movie(movie_id,role_id);
```

B.参与排序的字段

```
select person_id,person_role_id from moive
where movie_id=1000  and role_id=1
order by nr_role desc;-----参与排序的字段
alter table movie add index ind_movie(movie_id,role_id,nr_role);
```

C.覆盖索引

```
select person_id,person_role_id from moive------覆盖索引
where movie_id=1000  and role_id=1
order by nr_role desc;
alter table movie add index ind_movie
(movie_id,role_id,nr_role,person_id,person_role_id);
```

最佳实践-设计优化

一.索引设计要点

3.整理索引：创建需要的索引，删除不需要的索引

pt-duplicate-key-checker

```
root@(none) 09:41:40>create table foo (a int, b int,key (a), key (b), key (a,b));
$./pt-duplicate-key-checker -uxuancan -h127.0.0.1 -pxuancan -dtest -tfoo
# ########################################################################
# test.foo
# ########################################################################
# a is a left-prefix of a_2
# Key definitions:
#   KEY `a` (`a`),
#   KEY `a_2` (`a`,`b`)
# Column types:
#         `a` int(11) default null
#         `b` int(11) default null
# To remove this duplicate index, execute:
ALTER TABLE `test`.`foo` DROP INDEX `a`;
# ########################################################################
# Summary of indexes
# ########################################################################
# Size Duplicate Indexes   5
# Total Duplicate Indexes  1
# Total Indexes            3
```

最佳实践-设计优化

一.索引设计要点

4.常见索引设计误区一：对查询条件的每个字段建立单列索引

```
PRIMARY KEY (order_id),
UNIQUE KEY order_sn (order_sn),
UNIQUE KEY deal_code (deal_code),
KEY ind_user_id (user_id),
KEY ind_shipping_id (shipping_id),
KEY ind_pay_id (pay_id),
KEY ind_agency_id (agency_id),
KEY ind_extension_id (extension_id) ,
KEY ind_order_id (order_id),
KEY ind_delivery_time (delivery_time) ,
KEY ind_invoice_no (invoice_no),
KEY ind_user_nick (user_nick),
KEY add_time_1 (add_time,dist_type,rj,deal_code),
KEY idx_cz_shipping_fee (cz_shipping_fee),
KEY ind_mobile (mobile),
KEY ind_order_info_sd (sd_id,is_send,add_time),
KEY ind_order_info_status (shipping_status),
KEY ind_order_pay_status (pay_status),
KEY ind_order_is_yushou (is_yushou),
KEY ind_order_dist_type (dist_type),
KEY ind_order_jhd_id (jhd_id),
KEY ind_order_is_send (is_send),
KEY ind_order_ck_id(ck_id ),
KEY ind_order_is_separate(is_separate),
KEY ind_consignee (consignee),
KEY ind_order_info_lylx (lylx,order_status,is_send);
```

```
SELECT count(*) FROM order o  WHERE   is_send=0  AND
 o.order_status in (0,1) AND o.shipping_status = 0 AND
 o.is_separate > 0  and o.is_yushou=0 and o.sd_id=23
 and o.add_time>= '1370246433' and o.add_time<= '1370332842'
 and o.jhd_id=0 group by o.order_id;
```

最佳实践-设计优化

一.索引设计要点

5.常见索引设计误区二：对查询的所有字段建立组合索引

```
  09:44:03>  show table status like 'order'\G;
*************************** 1. row ***************************
           Name: order
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 5708209
 Avg_row_length: 357
    Data_length: 2042626048
Max_data_length: 0
   Index_length: 9014607872
      Data_free: 5242880
 Auto_increment: NULL
    Create_time: 2013-04-09 22:56:57
    Update_time: NULL
     Check_time: NULL
      Collation: utf8_bin
       Checksum: NULL
 Create_options:
        Comment: 订单表
```

```
  KEY `idx_plt_taobao_order_dp_id`
(`dp_id`,`customerno`,`created`,`endtime`,`pay_time`,`modified`,`consign_time`,
`payment`,`status`,`type`,`total_fee`,`refund_fee`,`num`,`received_payment`,
`trade_from`,`ccms_order_status`)

  KEY `idx_plt_taobao_order_created` (`created`,`customerno`,`endtime`,`pay_time`,
`modified`,`consign_time`,`payment`,`status`,`type`,`total_fee`,`refund_fee`,`num`,
`received_payment`,`trade_from`,`dp_id`,`ccms_order_status`)

  KEY `idx_plt_taobao_order_endtime` (`endtime`,`customerno`,`created`,`pay_time`,
`modified`,`consign_time`,`payment`,`status`,`type`,`total_fee`,`refund_fee`,`num`,
`received_payment`,`trade_from`,`dp_id`,`ccms_order_status`)

  KEY `idx_plt_taobao_order_pay_time` (`pay_time`,`customerno`,`created`,`endtime`
`modified`,`consign_time`,`payment`,`status`,`type`,`total_fee`,`refund_fee`,`num`,
`received_payment`,`trade_from`,`dp_id`,`ccms_order_status`)
```

该表的数据只有2G，但是索引却占用了9个G：

最佳实践-设计优化

二.主键设计：

1.在设计表的时候默认都添加一列自增id的主键：id bigint not null auto_increment

.自增型主键以利于插入性能的提高

.自增型主键设计(int,bigint)可以降低二级索引的空间，提升二级索引的内存命中率；

.自增型的主键可以减小page的碎片，提升空间和内存的使用；

.无主键的表删除，更新在row模式的主从架构，会导致备库hang住；

.可参考：mysql主键的缺少导致备库hang

最佳实践-设计优化

二.主键设计：业务主键与自增主键：

业务主键：

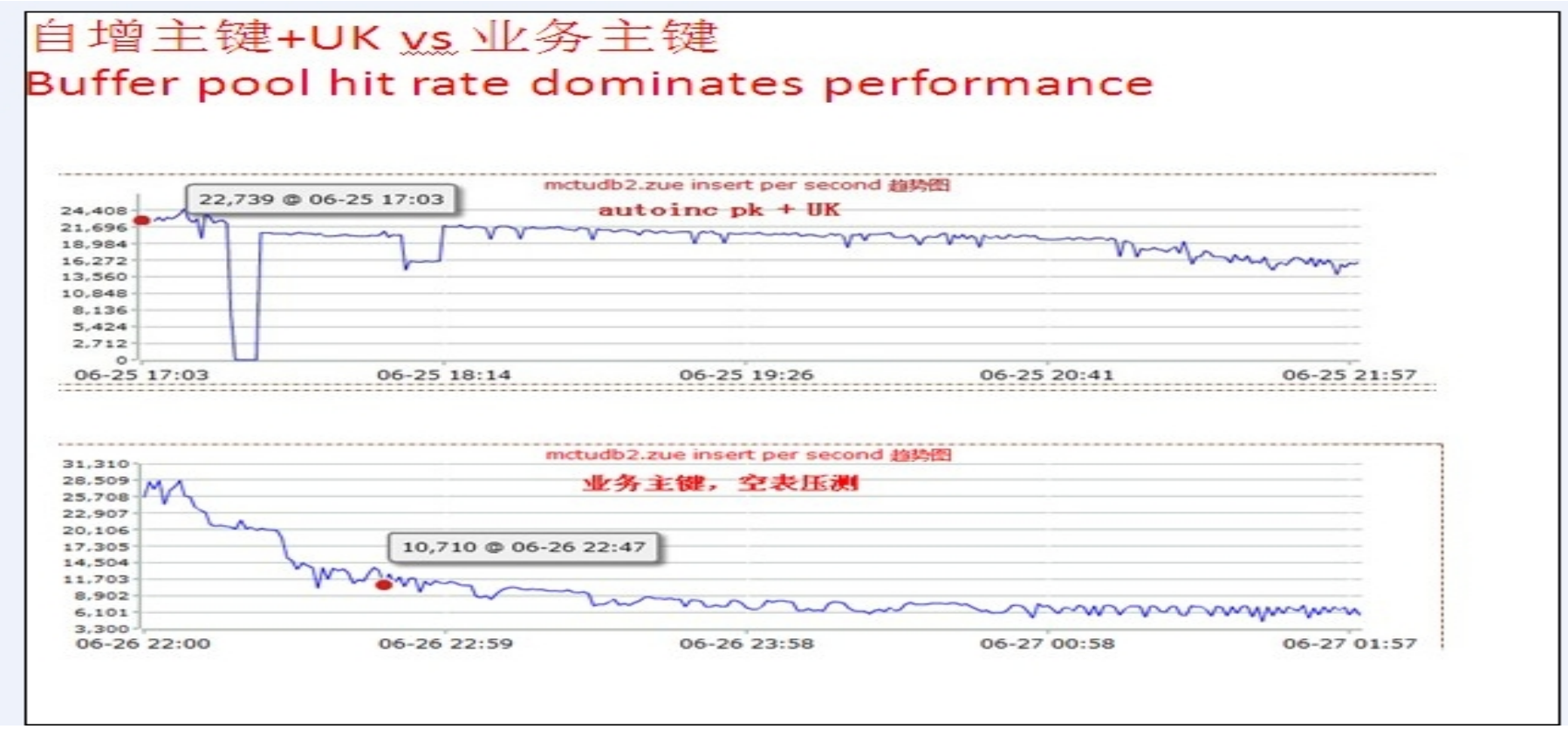
```
CREATE TABLE `tblogflow` (
  `user_id` bigint(20) NOT NULL DEFAULT '0',
  `tm_base` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
   ..........
PRIMARY KEY (`user_id`,`tm_base`,`source_id`,`city_id`,`province_id`,
`keyword_id`,`page_id`))
ENGINE=InnoDB AUTO_INCREMENT=323261 DEFAULT CHARSET=utf8
```

自增主键：

```
CREATE TABLE `tblogflow` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `user_id` bigint(20) NOT NULL DEFAULT '0',
  `tm_base` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  ............
  PRIMARY KEY (`id`),
  unique KEY (`user_id`,`tm_base`,`source_id`,`city_id`,`province_id`,
`keyword_id`,`page_id`)
) ENGINE=InnoDB AUTO_INCREMENT=323261 DEFAULT CHARSET=utf8
```

最佳实践-设计优化

二.主键设计：业务主键与自增主键：



最佳实践-设计优化

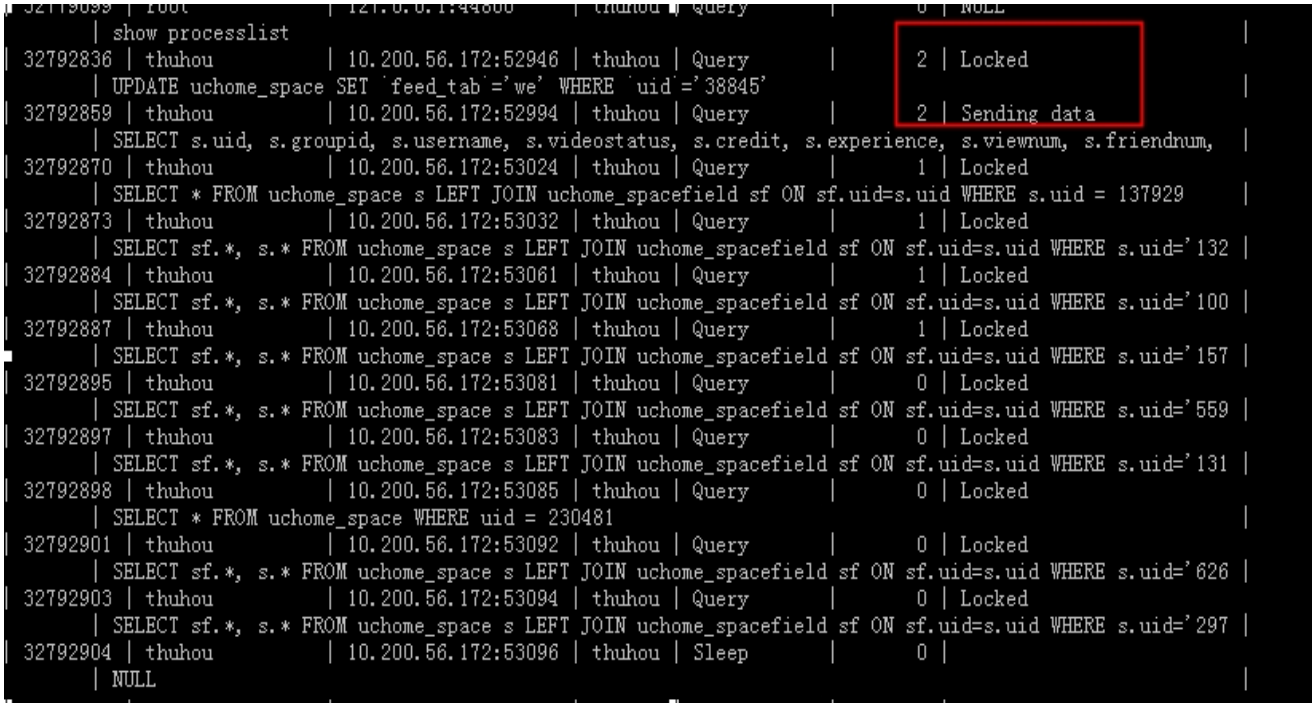
三.引擎选择

1.INNODB存储引擎与Myisam存储引擎

.RDS的内存配置innodb的innodb_buffer_pool_size,Myisam的key_cache配置32k

.主机断电，crash后Myisam表容易出现索引坏叶，需要手工repair修复索引

.Myisam存储引擎的表备份时候会被全局锁住，导致无法写入数据




```
CREATE TABLE `uchome_space` (
  ......
  PRIMARY KEY (`uid`)
  ......
) ENGINE=MyISAM DEFAULT CHARSET=
```

最佳实践-设计优化

三.引擎选择

2.FEDERATED 存储引擎使用存在bug，会导致备份失败

```
error log:
>> log scanned up to (867972807)
130616 00:00:58 innobackupex-1.5.1: Continuing after ibbackup has suspended
130616 00:00:58 innobackupex-1.5.1: Starting mysql with options:
 --defaults-file='/etc/my3015.cnf' --password=xxxxxxxx --user='Xtrabak' --host='127.0.0.1' --port='3015' --unbuffered --
130616 00:00:58 innobackupex-1.5.1: Connected to database with mysql child process (pid=31437)
130616 00:01:00 innobackupex-1.5.1: Starting to lock all tables...
>> log scanned up to (867972807)
innobackupex-1.5.1: Error: mysql child process has died: ERROR 1160 (08S01) at line 7: Got an error writing communication
packets  while waiting for reply to MySQL request: 'FLUSH TABLES WITH READ LOCK;' at /usr/bin/innobackupex-1.5.1 line 38
2013-06-16 00:01:06 [info]: Xtrabackup error,you can get detail from Logfile.
2013-06-16 00:01:06 [info]: ========================================= All backup finished .
```

最佳实践-设计优化

四.参数配置

1.实例规格参数是无法修改的
实例规格:innodb_buffer_pool_size，max_connections，max_user_connections
会话级别：sort_buffer_size, join_buffer_size,temp_table_size

2.可以选择调优的参数：
query cache:
query_cache_size = 1
query_cache_type = 32M
table_cache:
table_definition_cache=512
table_open_cache=512
thread_cache:
thread_cache_size=512

最佳实践-设计优化

五.其他设计原则

- SQL语句尽可能简单
- 保持事务(连接)短小
- 尽可能避免使用SP/TRIG/FUNC
- 尽量不用 SELECT *
- 改写OR语句为in
- 避免负向查询和% 前缀模糊查询
- 减少COUNT(*)
- 尽量不用外键

THANKS.