


jiaxin_12

万物之中，希望至美，至美之物.....

随笔 – 34, 文章 – 0, 评论 – 26, 阅读 – 10万

导航

- 博客园
- 首页
- 新随笔
- 联系
- 订阅 
- 管理

< 2023年2月 >						
日	一	二	三	四	五	六
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	1	2	3	4
5	6	7	8	9	10	11

公告

昵称： Jia-Xin
园龄： 5年8个月
粉丝： 35
关注： 2
[+加关注](#)

搜索

找找看

谷歌搜索

常用链接

- 我的随笔
- 我的评论
- 我的参与
- 最新评论
- 我的标签

最新随笔

- 1.TiDB上百T数据拆分实践
- 2.MySQL replace into那些隐藏的风险
- 3.跨时代的MySQL8.0新特性解读
- 4.MySQL MyISAM和Innodb表生成序列
- 5.pt-archiver归档数据 源库和目标库是否会出现不一致
- 6.MySQL 5.7和8.0性能测试
- 7.MySQL 8.0 information_schema系统的改进
- 8.MySQL 全文索引实现简单版搜索引擎
- 9.MySQL 逻辑备份mysqldump&mysqldump&mydumper原理解析
- 10.MySQL 如何优化cpu消耗

我的标签

- MySQL5.7(17)
- MySQL8.0(9)
- MySQL5.6(4)
- Linux(4)
- 故障处理(3)
- MySQL 优化(3)
- 逻辑备份(2)

MySQL 上亿大表优化实践

目录

- 背景
 - [select xxx_record语句](#)
 - [delete xxx_record语句](#)
- 测试
- 实施
- 索引优化后
- [delete大表优化为小批量删除](#)
- 总结

背景

XX实例（一主一从）xxx告警中每天凌晨在报SLA报警，该报警的意思是存在一定的主从延迟（若在此时发生主从切换，需要长时间才可以完成切换，要追延迟来保证主从数据的一致性）

XX实例的慢查询数量最多（执行时间超过1s的sql会被记录），XX应用那方每天晚上在做删除一个月前数据的任务

分析

使用pt-query-digest工具分析最近一周的mysql-slow.log

pt-query-digest --since=148h mysql-slow.log | less

结果第一部分

# Hostname: MySQL-176-7									
# Files: mysql-slow.log									
# Overall: 4.92k total, 12 unique, 0.01 QPS, 0.05x concurrency									
# Time range: 2019-03-21T12:00:09 to 2019-03-27T15:02:59									
# Attribute	total	min	max	avg	95%	stddev	median		
# =====	=====	=====	=====	=====	=====	=====	=====		
# Exec time	25403s	1s	266s	5s	7s	9s	5s		
# Lock time	502ms	0	223us	102us	159us	31us	98us		
# Rows sent	5.38k	0	10	1.12	0.99	1.11	0.99		
# Rows examine	848.51M	0	111.24M	176.64k	25.99k	3.74M	18.47k		
# Query size	791.64k	6	646	164.80	174.84	32.67	151.03		

最近一个星期内，总共记录的慢查询执行花费时间为25403s，最大的慢sql执行时间为266s，平均每个慢sql执行时间5s，平均扫描的行数为1766万

结果第二部分

# Profile							
# Rank	Query ID	Response time		Calls	R/Call	V/M	I
# =====	=====	=====	=====	=====	=====	=====	=====
# 1	0x538DF872D00055BA7D5D465...	5624.6679	22.1%	1184	4.7506	0.64	SELECT arrival_record
# 2	0xA718D41F73FE40D4CFA40B...	5614.6357	22.1%	1184	4.7421	0.64	SELECT arrival_record
# 3	0x19DA662C5E28F8983CC0595...	5606.9891	22.1%	1184	4.7356	0.64	SELECT arrival_record
# 4	0x1D962E1CF2AB700B3C58826...	5605.5829	22.1%	1184	4.7344	0.64	SELECT arrival_record
# 5	0x374BC8A6D98CD48196127BC...	1549.0816	6.1%	6	258.1803	0.13	DELETE arrival_record
# 6	0xFFFC4A067EA0A788813031B...	1202.5504	4.7%	79	15.2222	24.76	COMMIT
# 7	0xD1A61A368019569A7B5364F...	91.7318	0.4%	74	1.2396	0.00	SELECT mysql_slow_log
# MISC	0xMISC	108.1338	0.4%	24	4.5056	0.0	<5 ITEMS>

select arrival_record操作记录的慢查询数量最多有4万多多次，平均响应时间为4s，delete arrival_record记录了6次，平均响应时间258s

MySQL 主从复制(2)
物理备份(1)
MySQL工具(1)
更多

随笔分类

TiDB(1)

随笔档案

2022年12月(1)
2020年11月(1)
2020年10月(1)
2020年3月(1)
2019年9月(1)
2019年7月(3)
2019年6月(1)
2019年5月(6)
2019年4月(5)
2019年3月(4)
2019年1月(2)
2018年12月(2)
2018年9月(1)
2018年8月(2)
2018年4月(3)

阅读排行榜

- 1. MySQL 5.7和8.0性能测试(27125)
- 2. Linux 查看文件被那个进程写数据(10865)
- 3. MySQL 全文索引实现简单版搜索引擎(9614)
- 4. MySQL 上亿大表优化实践(6119)
- 5. MySQL 字符集utf8和utf-8的关系(5534)

评论排行榜

- 1. MySQL 5.7和8.0性能测试(9)
- 2. MySQL 上亿大表优化实践(8)
- 3. TiDB上百T数据拆分实践(4)
- 4. MySQL 如何优化cpu消耗(2)
- 5. 跨时代的MySQL8.0新特性解读(1)

推荐排行榜

- 1. 跨时代的MySQL8.0新特性解读(12)
- 2. MySQL 上亿大表优化实践(11)
- 3. MySQL 全文索引实现简单版搜索引擎(7)
- 4. MySQL 5.7和8.0性能测试(5)
- 5. TiDB上百T数据拆分实践(2)

最新评论

- 1. Re:TiDB上百T数据拆分实践
@缤纷世界 计算+存储共百台服务器左右...
--Jia-Xin
- 2. Re:TiDB上百T数据拆分实践
问一下好几十T的规模，用了多少台服务器做的集群？
--缤纷世界
- 3. Re:TiDB上百T数据拆分实践
学习了~
--balaoho
- 4. Re:TiDB上百T数据拆分实践
本文首发于公众号渠道
--Jia-Xin
- 5. Re:MySQL 上亿大表优化实践
您好，请教个问题，1k应该=1000吧，所以您在分析Row Examine的平均扫描行

select xxx_record语句

select arrival_record 慢查询语句都类似于如下所示，where语句中的参数字段是一样的，传入的参数值不一样

select count(*) from arrival_record where product_id=26 and receive_time between '2019-03-25 14:00:00' and '2019-03-25 15:00:00' and receive_spend_ms>=0\G

```
# Time range: 2019-03-21T12:00:09 to 2019-03-27T15:02:41
# Attribute      pct      total      min      max      avg      95%      stddev      median
# =====
# Count          24      1184
# Exec time      22      5625s      2s       8s       5s       7s       2s       5s
# Lock time      17      87ms      54us     108us    73us     93us     11us     66us
# Rows sent      21      1.16k      1        1        1        1        0        1
# Rows examine   2       19.89M     560.02k  17.20k   25.99k   21.04k   18.47k
# Query size     22      175.61k    151      152      151.88   151.03   0.50     151.03
# String:
```

select arrival_record 语句在mysql中最多扫描的行数为5600万、平均扫描的行数为172万，推断由于扫描的行数多导致的执行时间长

查看执行计划

explain select count(*) from arrival_record where product_id=26 and receive_time between '2019-03-25 14:00:00' and '2019-03-25 15:00:00' and receive_spend_ms>=0\G;

***** 1. row

id: 1

select_type: SIMPLE

table: arrival_record

partitions: NULL

type: ref

possible_keys: IXFK_arrival_record

key: IXFK_arrival_record

key_len: 8

ref: const

rows: 32261320

filtered: 3.70

Extra: Using index condition; Using where

1 row in set, 1 warning (0.00 sec)

用到了索引IXFK_arrival_record，但预计扫描的行数很多有3000多万行

```
show index from arrival_record;

+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | |
| Collation | Cardinality | Sub_part | Packed | Null | Index_type |
| Comment | Index_comment |
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| arrival_record | 0 | PRIMARY | 1 | id | A | 107990720 | NULL |
```

数时是不是算错了？

--koala0703

```
NULL || BTREE || |
| arrival_record | 1 | IXFK_arrival_record | 1 | product_id | A | 1344
| NULL | NULL || BTREE || |
| arrival_record | 1 | IXFK_arrival_record | 2 | station_no | A |
22161 | NULL | NULL | YES | BTREE || |
| arrival_record | 1 | IXFK_arrival_record | 3 | sequence | A |
77233384 | NULL | NULL || BTREE || |
| arrival_record | 1 | IXFK_arrival_record | 4 | receive_time | A |
65854652 | NULL | NULL | YES | BTREE || |
| arrival_record | 1 | IXFK_arrival_record | 5 | arrival_time | A |
73861904 | NULL | NULL | YES | BTREE || |

+-----+-----+-----+
--+-----+-----+-----+
-----+-----+-----+
-+-----+-----+
```

```
show create table arrival_record;

.....

arrival_spend_ms  bigint(20) DEFAULT NULL,
total_spend_ms   bigint(20) DEFAULT NULL,
PRIMARY KEY ( id ),
KEY IXFK_arrival_record ( product_id , station_no ,
sequence , receive_time , arrival_time ) USING BTREE,
CONSTRAINT FK_arrival_record_product FOREIGN KEY (
product_id ) REFERENCES product ( id ) ON DELETE NO
ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=614538979 DEFAULT
CHARSET=utf8 COLLATE=utf8_bin |
```

- 该表总记录数约1亿多条，表上只有一个复合索引，product_id 字段基数很小，选择性不好
- 传入的过滤条件 where product_id=26 and receive_time between '2019-03-25 14:00:00' and '2019-03-25 15:00:00' and receive_spend_ms>=0 没有station_nu字段，使用不到复合索引 IXFK_arrival_record的 product_id , station_no , sequence , receive_time 这几个字段
- 根据最左前缀原则，select arrival_record只用到了复合索引 IXFK_arrival_record的第一个字段product_id，而该字段选择性很差，导致扫描的行数很多，执行时间长
- receive_time字段的基数大，选择性好，可对该字段单独建立索引，select arrival_record sql就会使用到该索引

现在已经知道了在慢查询中记录的select arrival_record where语句传入的参数字段有 product_id, receive_time, receive_spend_ms，还想知道对该表的访问有没有通过其它字段来过滤了？

神器tcpdump出场的时候了

使用tcpdump抓包一段时间对该表的select语句

```
tcpdump -i bond0 -s 0 -l -w - dst port 3316 | strings | grep select |
```

```
egrep -i 'arrival_record' >/tmp/select_arri.log
```

获取select 语句中from 后面的where条件语句

```
IFS_OLD=$IFS
IFS=$'\n'
for i in `cat /tmp/select_arri.log`;do echo ${i#*'from'}; done |
less
IFS=$IFS_OLD
```

```
arrival_record arrivalrec0_ where arrivalrec0_.sequence='2019-
03-27 08:40' and arrivalrec0_.product_id=17 and
arrivalrec0_.station_no='56742'
arrival_record arrivalrec0_ where arrivalrec0_.sequence='2019-
03-27 08:40' and arrivalrec0_.product_id=22 and
arrivalrec0_.station_no='S7100'
arrival_record arrivalrec0_ where arrivalrec0_.sequence='2019-
03-27 08:40' and arrivalrec0_.product_id=24 and
arrivalrec0_.station_no='V4631'
arrival_record arrivalrec0_ where arrivalrec0_.sequence='2019-
03-27 08:40' and arrivalrec0_.product_id=22 and
arrivalrec0_.station_no='S9466'
arrival_record arrivalrec0_ where arrivalrec0_.sequence='2019-
03-27 08:40' and arrivalrec0_.product_id=24 and
arrivalrec0_.station_no='V4205'
arrival_record arrivalrec0_ where arrivalrec0_.sequence='2019-
03-27 08:40' and arrivalrec0_.product_id=24 and
arrivalrec0_.station_no='V4105'
arrival_record arrivalrec0_ where arrivalrec0_.sequence='2019-
03-27 08:40' and arrivalrec0_.product_id=24 and
arrivalrec0_.station_no='V4506'
arrival_record arrivalrec0_ where arrivalrec0_.sequence='2019-
03-27 08:40' and arrivalrec0_.product_id=24 and
arrivalrec0_.station_no='V4617'
arrival_record arrivalrec0_ where arrivalrec0_.sequence='2019-
03-27 08:40' and arrivalrec0_.product_id=22 and
arrivalrec0_.station_no='S8356'
arrival_record arrivalrec0_ where arrivalrec0_.sequence='2019-
03-27 08:40' and arrivalrec0_.product_id=22 and
arrivalrec0_.station_no='S8356'
```

- *select 该表 where条件中有product_id,station_no,sequence 字段, 可以使用到复合索引IXFK_arrival_record的前三个字段*

综上所述, 优化方法为, 删除复合索引IXFK_arrival_record, 建立复合索引idx_sequence_station_no_product_id, 并建立单独索引indx_receive_time

delete xxx_record语句

```

# Attribute      pct      total      min      max      avg      95%      stddev      median
# =====      ===      =====      =====      =====      =====      =====      =====
# Count          0          6
# Exec time      6      1570s      254s      266s      262s      258s      6s      258s
# Lock time      0      699us      100us      128us      116us      125us      9us      113us
# Rows sent      0          0          0          0          0          0          0          0
# Rows examine  77 663.33M 109.21M 111.76M 110.56M 106.64M 0 106.64M
# Query size     0          510        85        85        85        85        0          85
# String:

```

该delete操作平均扫描行数为1.1亿行，平均执行时间是262s

delete语句如下所示，每次记录的慢查询传入的参数值不一样

```

delete from arrival_record where receive_time <
STR_TO_DATE('2019-02-23', '%Y-%m-%d')\G

```

执行计划

```

explain select * from arrival_record where receive_time <
STR_TO_DATE('2019-02-23', '%Y-%m-%d')\G
***** 1. row
*****

id: 1
select_type: SIMPLE
table: arrival_record
partitions: NULL
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 109501508
filtered: 33.33
Extra: Using where
1 row in set, 1 warning (0.00 sec)

```

- 该delete语句没有使用索引（没有合适的索引可用），走的全表扫描，导致执行时间长
- 优化方法也是 建立单独索引`indx_receive_time(receive_time)`

测试

拷贝arrival_record表到测试实例上进行删除重新索引操作

XX实例arrival_record表信息

```

du -sh /datas/mysql/data/3316/cq_new_cimiss/arrival_record*
12K /datas/mysql/data/3316/cq_new_cimiss/arrival_record.frm
48G /datas/mysql/data/3316/cq_new_cimiss/arrival_record.ibd

```

```

select count() from cq_new_cimiss . arrival_record ;
+-----+
| count() |
+-----+
| 112294946 |

```

```
SELECT
table_name,
CONCAT(FORMAT(SUM(data_length) / 1024 / 1024,2),'M') AS
dbdata_size,
CONCAT(FORMAT(SUM(index_length) / 1024 / 1024,2),'M') AS
dbindex_size,
CONCAT(FORMAT(SUM(data_length + index_length) / 1024 /
1024 / 1024,2),'G') AS  table_size(G) ,
AVG_ROW_LENGTH,table_rows,update_time
FROM
information_schema.tables
WHERE table_schema = 'cq_new_cimiss' and
table_name='arrival_record';
```

磁盘占用空间48G, mysql中该表大小为31G, 存在17G左右的碎片, 大多由于删除操作造成的(记录被删除了, 空间没有回收)

mydumper并行压缩备份

并行压缩备份所花时间 (52s) 和占用空间 (1.2G, 实际该表占用磁盘空间为48G, mydumper并行压缩备份压缩比相当高!)

6/12

```
Finished dump at: 2019-03-26 12:46:56
```

```
du -sh    /datas/dump_arrival_record/  
1.2G     /datas/dump_arrival_record/
```

拷贝dump数据到测试节点

```
scp -rp /datas/dump_arrival_record root@10.230.124.19:/datas
```

多线程导入数据

```
time myloader -u root -S  
/datas/mysql/data/3308/mysql.sock -P 3308 -p root -B test  
-d /datas/dump_arrival_record -t 32
```

```
real 126m42.885s
```

```
user 1m4.543s
```

```
sys 0m4.267s
```

逻辑导入该表后磁盘占用空间

```
du -h -d 1 /datas/mysql/data/3308/test/arrival_record.*  
12K /datas/mysql/data/3308/test/arrival_record.frm  
30G /datas/mysql/data/3308/test/arrival_record.ibd  
没有碎片，和mysql的该表的大小一致
```

```
cp -rp /datas/mysql/data/3308 /datas
```

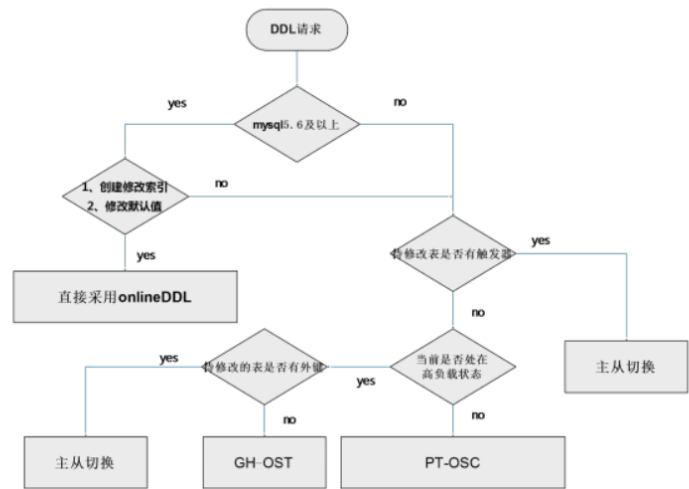
分别使用online DDL和 pt-osc工具来做删除重建索引操作

先删除外键，不删除外键，无法删除复合索引，外键列属于复合索引中第一列

```
nohup bash /tmp/ddl_index.sh &  
2019-04-04-10:41:39 begin stop mysqld_3308  
2019-04-04-10:41:41 begin rm -rf datadir and cp -rp  
datadir_bak  
2019-04-04-10:46:53 start mysqld_3308  
2019-04-04-10:46:59 online ddl begin  
2019-04-04-11:20:34 onlie ddl stop  
2019-04-04-11:20:34 begin stop mysqld_3308  
2019-04-04-11:20:36 begin rm -rf datadir and cp -rp  
datadir_bak  
2019-04-04-11:22:48 start mysqld_3308  
2019-04-04-11:22:53 pt-osc begin  
2019-04-04-12:19:15 pt-osc stop  
online ddl 花费时间为34 分钟，pt-osc花费时间为57 分钟，使用  
online ddl时间约为pt-osc工具时间的一半
```

做DDL 参考

4. 使用建议



实施

由于是一主一从实例，应用是连接的vip，删除重建索引采用online ddl来做。停止主从复制后，先在从实例上做（不记录binlog），主从切换，再在新切换的从实例上做（不记录binlog）

```
function red_echo () {

    local what="$*"
    echo -e "$(date +%F-%T)  ${what}"

}

function check_las_comm(){
    if [ "$1" != "0" ];then
        red_echo "$2"
        echo "exit 1"
        exit 1
    fi
}

red_echo "stop slave"
mysql -uroot -p$password --socket=/datas/mysql/data/${port}/mysqld.sock
-e"stop slave"
check_las_comm "$?" "stop slave failed"

red_echo "online ddl begin"
mysql -uroot -p$password --
socket=/datas/mysql/data/${port}/mysqld.sock -e"set
sql_log_bin=0;select now() as ddl_start;ALTER TABLE
$db_.\`${table_name}\` DROP FOREIGN KEY
FK_arrival_record_product,drop index IXFK_arrival_record,add index
idx_product_id_sequence_station_no(product_id,sequence,station_no),ad
d index idx_receive_time(receive_time);select now() as ddl_stop"
>>${log_file} 2>& 1
red_echo "onlie ddl stop"
red_echo "add foreign key"
mysql -uroot -p$password --
socket=/datas/mysql/data/${port}/mysqld.sock -e"set
sql_log_bin=0;ALTER TABLE $db_.${table_name} ADD CONSTRAINT
_FK_${table_name}_product FOREIGN KEY (product_id) REFERENCES
cq_new_cimiss.product (id) ON DELETE NO ACTION ON UPDATE NO ACTION;"
>>${log_file} 2>& 1
check_las_comm "$?" "add foreign key error"
red_echo "add foreign key stop"

red_echo "start slave"
mysql -uroot -p$password --socket=/datas/mysql/data/${port}/mysqld.sock
```



```
-e"start slave"
check_slave_status "$?" "start slave failed"
```

执行时间

2019-04-08-11:17:36 stop slave
mysql: [Warning] Using a password on the command line interface can be insecure.
ddl_start
2019-04-08 11:17:36
ddl_stop
2019-04-08 11:45:13
2019-04-08-11:45:13 online ddl stop
2019-04-08-11:45:13 add foreign key
mysql: [Warning] Using a password on the command line interface can be insecure.
2019-04-08-12:33:48 add foreign key stop
2019-04-08-12:33:48 start slave
删除重建索引花费时间为28分钟，添加外键约束时间为48分钟

再次查看delete 和select语句的执行计划

```
explain select count(*) from arrival_record where receive_time <
STR_TO_DATE('2019-03-10', '%Y-%m-%d')\G
***** 1. row
*****

id: 1
select_type: SIMPLE
table: arrival_record
partitions: NULL
type: range
possible_keys: idx_receive_time
key: idx_receive_time
key_len: 6
ref: NULL
rows: 7540948
filtered: 100.00
Extra: Using where; Using index
```

```
explain select count(*) from arrival_record where product_id=26
and receive_time between '2019-03-25 14:00:00' and '2019-
03-25 15:00:00' and receive_spend_ms>=0\G;
***** 1. row
*****

id: 1
select_type: SIMPLE
table: arrival_record
partitions: NULL
type: range
possible_keys:
idx_product_id_sequence_station_no,idx_receive_time
key: idx_receive_time
key_len: 6
```

ref: NULL

rows: 291448

filtered: 16.66

Extra: Using index condition; Using where

都使用到了idx_receive_time 索引, 扫描的行数大大降低

索引优化后

delete 还是花费了77s时间

```
delete from arrival_record where receive_time < STR_TO_DATE('2019-03-10', '%Y-%m-%d')\G
```

```
# Time range: all events occurred at 2019-04-09T01:01:23
# Attribute      pct   total    min     max     avg     95%   stddev  median
# =====
# Count          33      1
# Exec time      96      77s      77s     77s     77s     77s      0       77s
# Lock time      20      73us     73us    73us    73us    73us      0       73us
# Rows sent       0        0        0        0        0        0        0        0
# Rows examine   95     3.39M    3.39M    3.39M    3.39M    3.39M      0     3.39M
# Query size     33      85      85      85      85      85        0       85
# String:
# Databases:    cd new cmiss
```

delete 语句通过receive_time的索引删除300多万的记录花费77s时间*

delete大表优化为小批量删除

应用端已优化成每次删除10分钟的数据（每次执行时间1s左右），xxx中没在出现SLA（主从延迟告警）

```
# Time: 2019-04-17T01:03:31.014895+08:00
# User@Host: action[action] @ [10.230.124.170] Id: 739678
# Query time: 1.173182 Lock time: 0.000072 Rows sent: 0 Rows examined: 26323
SET timestamp=1555434211;
delete from arrival_record where receive_time < STR_TO_DATE('2019-03-18 04:20:00', '%Y-%m-%d %H:%i:%s');
# Time: 2019-04-17T01:03:32.955396+08:00
# User@Host: action[action] @ [10.230.124.170] Id: 739678
# Query time: 1.132365 Lock time: 0.000090 Rows sent: 0 Rows examined: 28270
SET timestamp=1555434212;
delete from arrival_record where receive_time < STR_TO_DATE('2019-03-18 04:30:00', '%Y-%m-%d %H:%i:%s');
# Time: 2019-04-17T01:03:34.826237+08:00
# User@Host: action[action] @ [10.230.124.170] Id: 739678
# Query time: 1.040508 Lock time: 0.000076 Rows sent: 0 Rows examined: 26037
SET timestamp=1555434214;
delete from arrival_record where receive_time < STR_TO_DATE('2019-03-18 04:40:00', '%Y-%m-%d %H:%i:%s');
# Time: 2019-04-17T01:03:38.594063+08:00
# User@Host: action[action] @ [10.230.124.170] Id: 739678
# Query time: 1.174763 Lock time: 0.000081 Rows sent: 0 Rows examined: 20213
SET timestamp=1555434218;
delete from arrival_record where receive_time < STR_TO_DATE('2019-03-18 05:00:00', '%Y-%m-%d %H:%i:%s');
# Time: 2019-04-17T01:03:40.652425+08:00
# User@Host: action[action] @ [10.230.124.170] Id: 739678
# Query time: 1.229694 Lock time: 0.000081 Rows sent: 0 Rows examined: 30609
SET timestamp=1555434220;
delete from arrival_record where receive_time < STR_TO_DATE('2019-03-18 05:10:00', '%Y-%m-%d %H:%i:%s');
# Time: 2019-04-17T01:03:42.585390+08:00
# User@Host: action[action] @ [10.230.124.170] Id: 739678
# Query time: 1.065401 Lock time: 0.000074 Rows sent: 0 Rows examined: 26915
SET timestamp=1555434222;
delete from arrival_record where receive_time < STR_TO_DATE('2019-03-18 05:20:00', '%Y-%m-%d %H:%i:%s');
```

另一个方法是通过主键的顺序每次删除20000条记录

```
#得到满足时间条件的最大主键ID
#通过按照主键的顺序去 顺序扫描小批量删除数据
#先执行一次以下语句
SELECT MAX(id) INTO @need_delete_max_id FROM `arrival_record` WHERE
receive_time<'2019-03-01' ;
DELETE FROM arrival_record WHERE id<@need_delete_max_id LIMIT 20000;
select ROW_COUNT(); #返回20000
```

```
#执行小批量delete后会返回row_count(), 删除的行数
#程序判断返回的row_count() 是否为0, 不为0执行以下循环, 为0退出循环, 删除操作完成
DELETE FROM arrival_record WHERE id<@need_delete_max_id LIMIT 20000;
select ROW_COUNT();
```

#程序睡眠0.5s

总结

- 表数据量太大时，除了关注访问该表的响应时间外，还要关注对该表的维护成本（如做DDL表更新时间太长，delete历史数据）
- 对大表进行DDL操作时，要考虑表的实际情况（如对该表的并发表，是否有外键）来选择合适的DDL变更方式
- 对大数据量表进行delete，用小批量删除的方式，减少对主实例的压力和主从延迟

作者： [jiaxin](#)
出处： <http://www.cnblogs.com/YangJiaXin/>
本文版权归作者和博客园共有，禁止转载，私自转载将追究法律责任

标签: [MySQL 优化](#)

好文要顶

关注我

收藏该文

Jia-Xin

粉丝 - 35 关注 - 2

110

[+加关注](#)

« 上一篇: [MySQL 字符集utf8和utf-8的关系](#)
» 下一篇: [MySQL 几种调式分析利器](#)

posted on 2019-05-07 21:51 Jia-Xin 阅读(6119) 评论(8) 编辑 收藏 举报

[刷新评论](#) [刷新页面](#) [返回顶部](#)

登录后才能查看或发表评论，立即 [登录](#) 或者 [逛逛](#) 博客园首页

【推荐】阿里云新人特惠，爆款云服务器2核4G低至0.46元/天

编辑推荐:

- 现代图片性能优化及体验优化指南
- 探索：优雅地实现异步方法的并行执行
- 如何避免让线程摸鱼，请用异步技术 async await 拿捏他
- 分布式事务 | 使用 DTM 的 Saga 模式
- SQLSERVER 阻塞之 PFS 页到底是什么？

阅读排行:

- ChatGPT：让程序开发更轻松
- 从零开始，打造属于你的 ChatGPT 机器人！
- .Net Core对于`RabbitMQ`封装分布式事件总线
- ChatGPT：好家伙，每个人内心的一块魔镜
- 实现一个简单的在浏览器运行Dotnet编辑器

Powered by:

博客园

Copyright © 2023 Jia-Xin

Powered by .NET 7.0 on Kubernetes