高性能MySQL

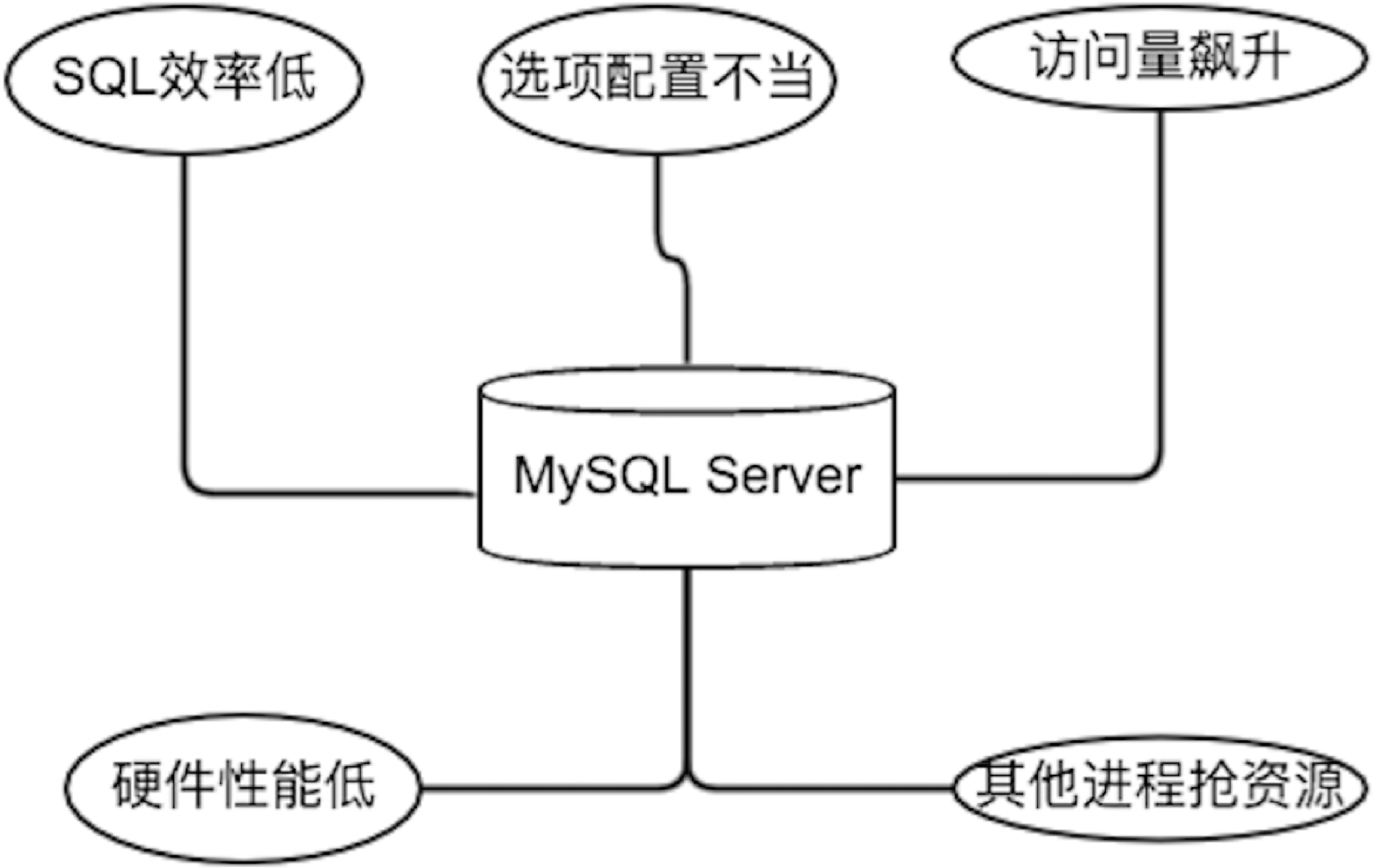
叶金荣 - 知数堂培训联合创始人

2016.12.2

关于我

- 知数堂培训联合创始人
- Oracle MySQL ACE
- ACMUG轮值主席
- MySQL布道师
- 使用MySQL超过16年
- QQ/微信：4700963
- 公众号： 老叶茶馆
- 专注培养互联网MySQL人才

哪些因素导致MySQL无法高性能


SQL效率低
选项配置不当
访问量飙升
MySQL Server
硬件性能低
其他进程抢资源

硬件、系统层瓶颈

- top/free/vmstat/sar/mpstat确认
  - 确认是mysqld进程存在性能瓶颈
  - 确认mysqld进程的CPU消耗占比
  - 确认mysqld进程的CPU消耗是%user，还是%sys高
  - 确认是否物理内存不够用了
  - 确认是否有swap产生

硬件、系统层瓶颈

- top

```
top - 23:52:22 up 44 days, 13:18,  2 users,  load average: 16.36, 11.54, 1.11
Tasks: 726 total,   1 running, 724 sleeping,   1 stopped,   0 zombie
%Cpu(s): 62.7 us,  5.6 sy,  0.0 ni, 27.8 id,  3.0 wa,  0.0 hi,  1.0 si,  0.0 st
KiB Mem : 98719944 total,   335312 free, 85283712 used, 13100916 buff/cache
KiB Swap: 20971516 total, 20908900 free,    62616 used. 12792824 avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
185880 mysql     20   0 15.784g 2.273g   7496 S 784.1  2.4   2:15.56 mysqld
   568 root      20   0       0      0      0 S   1.0  0.0   1:48.35 kswapd1
193905 mysql     20   0 80.413g 0.076t  12772 S   1.0 82.8 227:08.13 mysqld
   567 root      20   0       0      0      0 S   0.7  0.0   2:58.29 kswapd0
 19924 root      20   0  158224   2856   1536 R   0.7  0.0   0:00.13 top
```

硬件、系统层瓶颈

- 确认瓶颈在CPU、还是内存，或是I/O
- 有无必要升级CPU（更高主频，更多线程）
- 加大物理内存
- 使用更好的I/O设备（SSD、PCIe SSD）
- 如果mysqld进程的CPU很高，绝大多数是因为索引不当

硬件、系统层瓶颈

- free -m

	total	used	free	shared	buffers	cached
Mem:	252	212	39	0	0	18
-/+ buffers/cache:		206	46			
Swap:	7	7	0			
Total:	260	220	39			

硬件、系统层瓶颈

- cached+buffer之和远小于used，疑似内存泄露
- 重启使用内存最多的进程可以解决
- 建议升级/替换版本
- 修复代码，或向开发者报告bug

硬件、系统层瓶颈

- vmstat -S m 1

```
vmstat -S m 1
procs -----------memory---------- ---swap-- -----io---- --system-- -------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs       us sy id wa st
42  0     80    338      0  13400    0    0   536 112400 34657 221678 43  6 50  1  0
 0  1     80    358      0  13379    0    0   372  71421 33717 227494 43  6 50  1  0
 0  1     80    352      0  13383    0    0   524 106847 28957 202112 35  6 58  1  0
33  0     80    354      0  13385    0    0   540  73859 38159 259421 40  7 52  1  0
 0  1     80    347      0  13386    0    0   456 108401 65412 205543 33  6 60  1  0
31  1     81    329      0  13409    0    0   448  76707 37723 240639 45  7 48  1  0
27  0     81    344      0  13392    0    0   360 104701 40005 188579 34  6 59  1  0
35  0     81    344      0  13395    0    0   360  78647 37873 222284 39  7 54  1  0
```

硬件、系统层瓶颈

- mysqld进程使用CPU（%user）很高，绝大多数是因为索引不当导致
  - 加上合适的索引
- 如果是CPU的%sys很高，则可能是swap，或mysql锁并发很严重导致
  - 检查内存分配是否得当
  - 检查是否有内存泄露风险
  - 检查有无严重锁等待事件
- 如果是CPU的%iowait很高，则是I/O设备存在瓶颈
  - 提升I/O设备性能

硬件、系统层瓶颈

- sar -u 1 50

```
Linux 3.10.0-327.el7.x86_64 (imysql)     10/09/2016     _x86_64_(32 CPU)

11:57:57 PM     CPU     %user     %nice   %system   %iowait    %steal     %idle
11:57:58 PM     all     66.49      0.00      8.43      0.38      0.00     24.70
11:57:59 PM     all     72.19      0.00      9.26      0.16      0.00     18.39
11:58:00 PM     all     74.34      0.00      9.08      0.09      0.00     16.49
11:58:01 PM     all     72.09      0.00     10.03      0.13      0.00     17.76
11:58:02 PM     all     67.52      0.00     12.30      0.13      0.00     20.06
11:58:03 PM     all     66.55      0.00     10.54      0.22      0.00     22.69
11:58:04 PM     all     72.03      0.00      9.70      0.09      0.00     18.17
11:58:05 PM     all     72.08      0.00      9.77      0.09      0.00     18.06
11:58:06 PM     all     72.20      0.00      9.87      0.09      0.00     17.83
```

硬件、系统层瓶颈

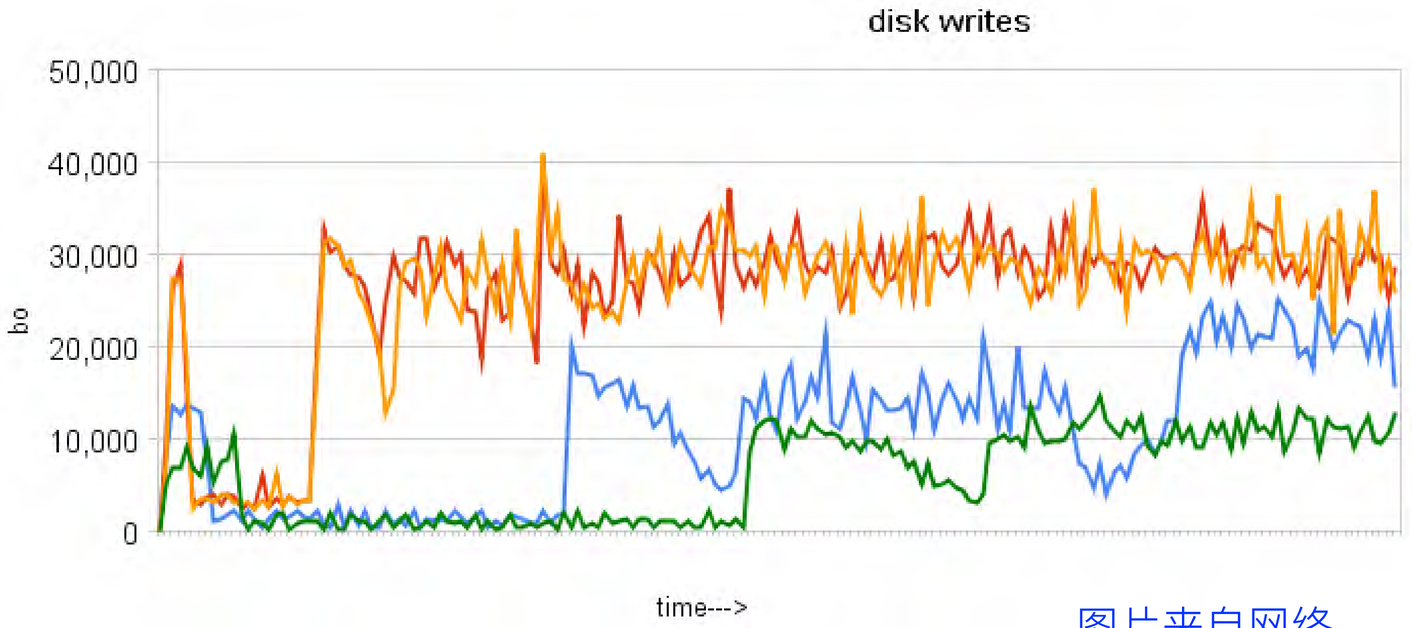
- sar -d 1 50

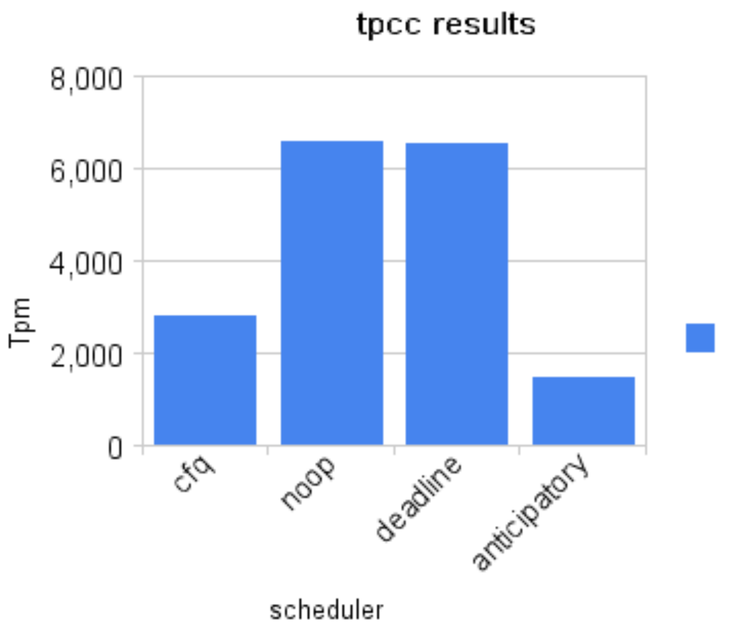
```
Linux 3.10.0-327.el7.x86_64 (imysql)      10/10/2016      _x86_64_ (32 CPU)

12:01:25 AM       DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
12:01:26 AM    dev8-16   2990.00      0.00 267137.00     89.34      0.20      0.07      0.05     14.60
12:01:27 AM    dev8-16   2949.00      0.00 248692.00     84.33      0.18      0.06      0.05     15.00
12:01:28 AM    dev8-16   3043.00      0.00 333020.00    109.44      0.24      0.08      0.07     20.10
12:01:29 AM    dev8-16   3199.00      0.00 285494.00     89.24      0.18      0.06      0.04     13.20
12:01:30 AM    dev8-16   3207.00      0.00 329196.00    102.65      0.17      0.05      0.04     12.80
12:01:31 AM    dev8-16   3125.00      0.00 304370.00     97.40      0.23      0.07      0.05     16.10
12:01:32 AM    dev8-16   3062.00      0.00 232723.00     76.00      0.12      0.04      0.04     11.50
12:01:33 AM    dev8-16   3060.00      0.00 266480.00     87.08      0.15      0.05      0.04     12.50
12:01:34 AM    dev8-16   2913.00      0.00 261099.00     89.63      0.20      0.07      0.06     18.50

Average:       dev8-16   3069.59      0.00 280986.34     91.54      0.19      0.06      0.05     15.02
```

硬件、系统层瓶颈

- 观察tps(iops)、await、svctm、%util等几个指标
- 如果发现IOPS不高，但%util较高，则很可能是文件系统的瓶颈，建议使用xfs或ext4
- 也可能是io scheduler采用cfq等比较低效的策略，建议使用noop或deadline
- 若是机械硬盘，一定要配阵列卡以及CACHE & BBU，并且使用WB策略
- 甚至，也可能是磁盘有损坏，I/O能力异常陡降

硬件、系统层瓶颈





图片来自网络

硬件、系统层瓶颈

- 内核选项调整
  - vm.dirty_ratio ≤ 5
  - vm.dirty_background_ratio ≤ 10
  - 避免因为io压力瞬间飙升导致内核进程卡死，os hung住

```
INFO:task jbd/dm-0-8:389 blocked for more than 120 seconds.
"echo 0 > /proc/sys/kernel/hung_task_timeout_secs" disable this message.

INFO:task flush-253:0:1324 blocked for more than 120 seconds.
echo 0 > /proc/sys/kernel/hung_task_timeout_secs" disable this message.
```

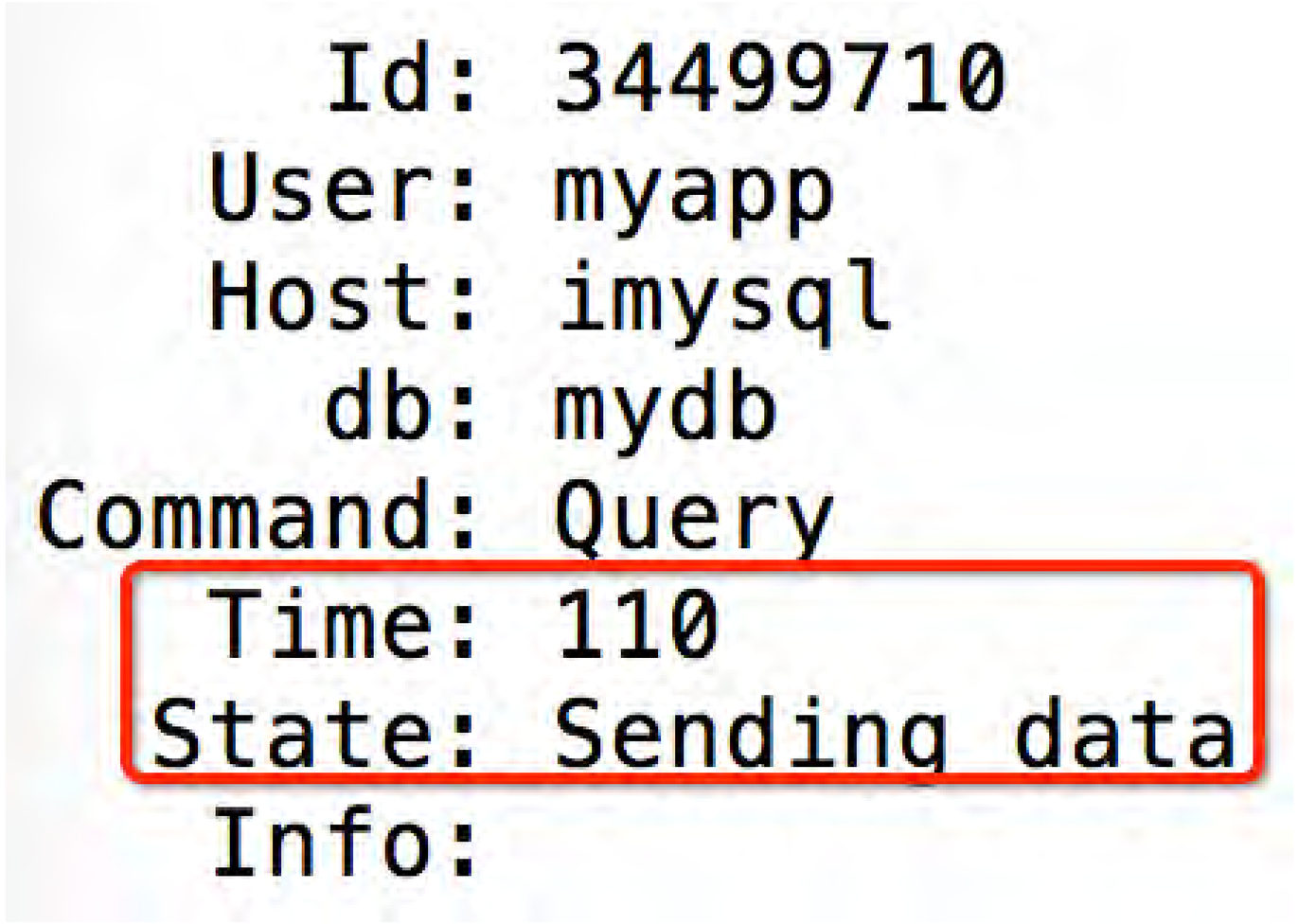
MySQL层瓶颈

- show [full] processlist
- locks & lock waits
  - I_S.innodb_trx、I_S.innodb_locks、I_S.innodb_lock_waits
  - sys.innodb_lock_waits
- show engine innodb status
  - waits、row lock、undo log、redo log

dml很慢

Id	User	Host	db	Command	Time	State	Info
4844891	myapp	10.10.3.79:39437	mydb	Sleep	5052		
4926104	myapp	10.10.3.79:41329	mydb	Query	110	Sending data	SELECT ...
4929351	myapp	10.10.3.79:46930	mydb	Query	6	update	insert into ...
4929352	myapp	10.10.3.79:46934	mydb	Query	4	update	insert into ...
4929357	myapp	10.10.3.79:46941	mydb	Query	5	update	insert into ...
4929364	myapp	10.10.3.79:46955	mydb	Query	9	update	insert into ...

查看MySQL状态


Id: 34499710
User: myapp
Host: imysql
db: mydb
Command: Query
Time: 110
State: Sending data
Info:

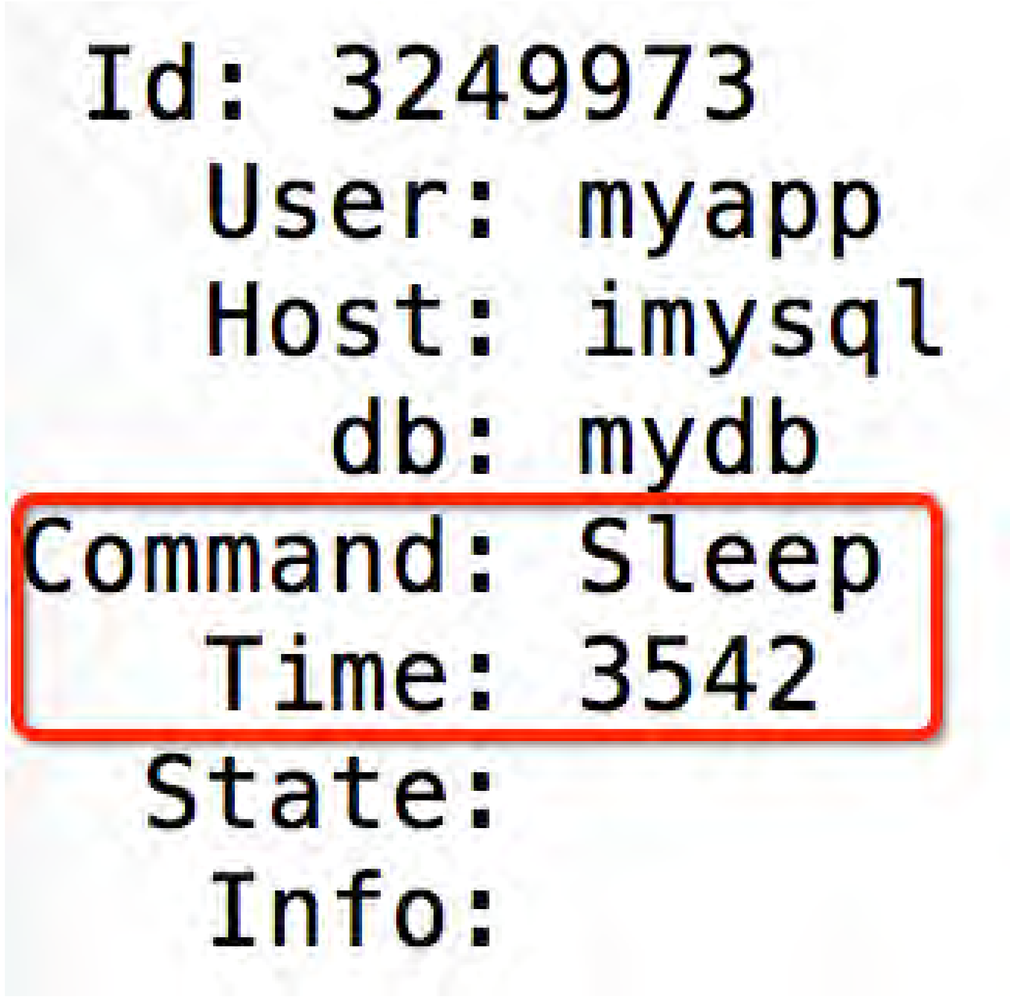
长时间的Sending data

- 从引擎层读取数据返回给Server端的状态
  - 长时间存在的原因
  - 没适当的索引，查询效率低
  - 读取大量数据，读取缓慢
  - 系统负载高，读取缓慢

长时间的Sending data

- 怎么优化
  - 加上合适的索引
  - 或者改写SQL，提高效率
  - 增加LIMIT限制每次读取数据量
  - 检查&升级I/O设备性能

查看MySQL状态


Id: 3249973
User: myapp
Host: imysql
db: mydb
Command: Sleep
Time: 3542
State:
Info:

长时间的Sleep

- 纳尼，这种也要关注吗?

  - 占用连接数

  - 消耗内存未释放

  - 可能有行锁（甚至是表锁）未释放

长时间的Sleep

- 怎么优化
  - 适当调低timeout
  - 主动kill超时不活跃连接
  - 定期检查锁、锁等待
  - 可以利用pt-kill工具

还有哪些状态要关注的呢

- Copy to tmp table

  - 执行alter table修改表结构，需要生成临时表

  - 建议放在夜间低谷执行，或者用pt-osc

还有哪些状态要关注的呢

- Copying to tmp table [on disk]
- Creating tmp table
  - 常见于group by没有索引的情况
  - 需要拷贝数据到临时表[内存/磁盘上]
  - 执行计划中会出现Using temporary关键字
  - 建议创建合适的索引，消除临时表

还有哪些状态要关注的呢

- Creating sort index
  - 常见于order by没有索引的情况
  - 需要进行filesort排序
  - 执行计划中会出现Using filesort关键字
  - 建议创建排序索引

还有什么要关注的呢

```
# Time: 2016-11-26T16:19:13.817889Z
# User@Host: root[root] @ localhost []  Id:    20
# Schema: mydb  Last_errno: 1681  Killed: 0
# Query_time: 3.801728  Lock_time: 0.000831  Rows_sent: 1  Rows_examined: 13719413  Rows_affected: 0
# Bytes_sent: 70  Tmp_tables: 7  Tmp_disk_tables: 2  Tmp_table_sizes: 32768
# InnoDB_trx_id: 4A0B89B
# QC_Hit: No  Full_scan: Yes  Full_join: Yes  Tmp_table: Yes  Tmp_table_on_disk: Yes
# Filesort: Yes  Filesort_on_disk: No  Merge_passes: 0
#   InnoDB_IO_r_ops: 0  InnoDB_IO_r_bytes: 0  InnoDB_IO_r_wait: 0.000000
#   InnoDB_rec_lock_wait: 0.000000  InnoDB_queue_wait: 0.000000
#   InnoDB_pages_distinct: 12252
SET timestamp=1480177153;
select ...
```

还有什么可以提前做的?

- 业务上线前，提前消灭垃圾SQL
  - 在开发或压测环境中
  - 调低long_query_time的值，甚至设为0
  - 开启log_queries_not_using_indexes
  - 分析slow query log，并消除潜在隐患SQL

Schema优化

InnoDB表之主键(Primary Key)

- 一定要有主键（Primary Key）
- 没有主键会怎样
  - 数据多次读写后可能更离散，有更多随机I/O
  - MySQL复制环境中，如果选择RBR模式，没有主键的update需要读全表，导致复制延迟
  - 主键最好是有序INT，不要用CHAR/UUID类型

尽量不用TEXT/BLOB类型

- innodb data page默认16KB
- 当每行长度超过8KB时，就需要分裂data page
- 会产生更多离散I/O
- 以及更高碎片率
- 案例：一个100G的表拆分成4个表后，总大小仅25G

索引很重要

- InnoDB的行锁是基于索引实现的。如果没有索引，将会是灾难性的
  - 读取时，全表扫描
  - 修改时，全表记录锁

索引设计

- 基数（cardinality）低的字段一般没必要建立**单列索引**
- 字符型字段上建索引时优先采用部分索引（prefix index）
- 5.6.9后，定义二级索引时无需增加主键列（Index Extensions特性）
- 优先多列联合索引（multi column index），少用单列索引

写好SQL

- 核心目标
  - 减少CPU计算量
  - 减少内存需求量
  - 减少IO读写量

写好SQL

- 所有WHERE条件都加上引号
  - 避免潜在的类型隐式转换风险
  - 避免个别条件失效时SQL语法错误

写好SQL

- 不SELECT *
  - 减少不必要的I/O
  - 提高可以利用覆盖索引的几率

写好SQL

- 时刻注意安全防范
  - 所有用户输入值都要做过滤
  - 利用PREPARE做预处理
  - 利用SQL_MODE做限制

写好SQL

- 其他
  - LIKE查询时，不要用%通配符最左前导（无法使用索引）
  - 能UNION ALL就不要UNION（UNION需要去重，会产生临时表）
  - SQL中最好不要有运算
  - WHERE子句中，不要有函数

写好SQL

- 关于JOIN
  - 满足业务需求前提下，优先用inner join，让优化器自动选择驱动表
  - 有时候优化器选择的驱动表未必是最优的，可以尝试手动调整
  - 最后的排序字段如果不在驱动表中，则会有filesort

写好SQL

糟糕的SQL

UPDATE t SET c2 = ? **AND** c3 = ? WHERE id = '?'

=>

UPDATE t SET c2 = ? **,** c3 = ? WHERE id = '?'

写好SQL

糟糕的SQL

UPDATE t SET c2 = ?
WHERE id = '?'

=>

UPDATE t SET c2 = ? **WHERE** id = '?'

写好SQL

糟糕的SQL

SELECT * FROM t WHERE id = '?'

=>

SELECT **c2,c3** FROM t WHERE id = '?'

写好SQL

糟糕的SQL

SELECT c2,c3 FROM t WHERE c4 **like '%???%'**

=>

SELECT c2,c3 FROM t WHERE id >= ? AND id <= ? AND c4 **like '%???%'**

或者采用第三方解决方案

写好SQL

糟糕的SQL

SELECT c2,c3 FROM t WHERE **date(c1) = ‘2016-10-15’**

=>

SELECT c2,c3 FROM t WHERE **c1 >= ‘2016-10-15’ AND c1 < ‘2016-10-16’**

或者，用MySQL 5.7的虚拟列（GENERATED COLUMN）

写好SQL

糟糕的SQL

SELECT c2,c3 FROM t WHERE char_col = **int_value**

=>

SELECT c2,c3 FROM t WHERE char_col = **’int_value’**

也可以用MySQL 5.7的Query Rewrite规避

写好SQL

糟糕的SQL

SELECT c1,c2…cN FROM t WHERE … ORDER BY id LIMIT 500000,20;

=>

SELECT c1,c2…cN FROM t INNER JOIN (SELECT id FROM t WHERE … ORDER BY id DESC LIMIT 500000,20) t1 USING (id);

点此看详细优化案例分享，也可以用Query Rewrite自动改写

SQL EXPLAIN

- 关键业务SQL上线前，都要EXPLAIN确认其执行计划
- 或提前分析slow query log，防患未然
- EXPLAIN中如果有Using temporary、Using filesort、或type=ALL/index时，尽量想办法进行优化

SQL EXPLAIN

- type = ALL

```
yejr@imysql.com [(none)]> EXPLAIN SELECT * FROM t WHERE lt > 1475251200\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1781793
     filtered: 33.33
        Extra: Using where
```

SQL EXPLAIN

yejr@imysql.com [(none)]> SELECT * FROM t WHERE lt > 1475251200;

Empty set (0.19 sec)

```
show status like 'handler_read%';
+-----------------------+---------+
| Variable_name         | Value   |
+-----------------------+---------+
| Handler_read_first    | 0       |
| Handler_read_key      | 0       |
| Handler_read_last     | 0       |
| Handler_read_next     | 0       |
| Handler_read_prev     | 0       |
| Handler_read_rnd      | 0       |
| Handler_read_rnd_next | 1781794 |
+-----------------------+---------+
```

```
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| starting             | 0.000104 |
| checking permissions | 0.000016 |
| Opening tables       | 0.000033 |
| init                 | 0.000040 |
| System lock          | 0.000025 |
| optimizing           | 0.000017 |
| statistics           | 0.000030 |
| preparing            | 0.000025 |
| executing            | 0.000007 |
| Sending data         | 0.184295 |
| end                  | 0.000024 |
| query end            | 0.000021 |
| closing tables       | 0.000039 |
| freeing items        | 0.000304 |
| cleaning up          | 0.000039 |
+----------------------+----------+
```

SQL EXPLAIN

- type = index

```
yejr@imysql.com> EXPLAIN SELECT lt FROM t \G
*************** 1. row ***************
           id: 1
  select_type: SIMPLE
        table: t
   partitions: NULL
         type: index
possible_keys: NULL
          key: lastupdate
      key_len: 4
          ref: NULL
         rows: 1781793
     filtered: 100.00
        Extra: Using index
```

SQL EXPLAIN

yejr@imysql.com> SELECT lt FROM t;

1781793 rows in set (2.31 sec)

Variable_name	Value
Handler_read_first	1
Handler_read_key	0
Handler_read_last	0
Handler_read_next	1781793
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	0

Status	Duration
starting	0.000115
checking permissions	0.000022
Opening tables	0.000035
init	0.000028
System lock	0.000029
optimizing	0.000014
statistics	0.000031
preparing	0.000028
executing	0.000007
Sending data	2.315104
end	0.000026
query end	0.000019
closing tables	0.000023
freeing items	0.008464
logging slow query	0.000019
logging slow query	0.000171
cleaning up	0.000041

SQL EXPLAIN

- type = Using filesort

```
yejr@imysql.com> desc select * from t order by total\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1781793
     filtered: 100.00
        Extra: Using filesort
```

SQL EXPLAIN

yejr@imysql.com> SELECT * FROM t ordery total;

1781793 rows in set (2.98 sec)

Variable_name	Value
Handler_read_first	0
Handler_read_key	0
Handler_read_last	0
Handler_read_next	0
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	1781794

Status	Duration
starting	0.000119
checking permissions	0.000023
Opening tables	0.000041
init	0.000041
System lock	0.000029
optimizing	0.000013
statistics	0.000036
preparing	0.000026
Sorting result	0.000013
executing	0.000009
Sending data	0.000022
Creating sort index	2.983628
end	0.000024
query end	0.000020
closing tables	0.000025
freeing items	0.005723
logging slow query	0.000021
logging slow query	0.000234
cleaning up	0.000052

SQL EXPLAIN

- type = Using temporary

```
yejr@imysql.com> desc select distinct tm from t group by tm\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1781793
     filtered: 100.00
        Extra: Using temporary; Using filesort
```

SQL EXPLAIN

- type = Using temporary

```
yejr@imysql.com> desc select distinct tm from t group by tm order by null\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1781793
     filtered: 100.00
        Extra: Using temporary
```

SQL EXPLAIN

yejr@imysql.com> select distinct thismonth from t group by thismonth order by null;

1 row in set (0.77 sec)

yejr@imysql.com> select distinct thismonth from t group by thismonth;

1 row in set (1.13 sec)

SQL EXPLAIN

group by tm order by null

Variable_name	Value
Handler_read_first	0
Handler_read_key	0
Handler_read_last	0
Handler_read_next	0
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	1781797

group by tm

Variable_name	Value
Handler_read_first	0
Handler_read_key	0
Handler_read_last	0
Handler_read_next	0
Handler_read_prev	0
Handler_read_rnd	1
Handler_read_rnd_next	1781797

SQL EXPLAIN

group by tm order by null

Creating tmp table	0.000061
executing	0.000011
Sending data	0.748907
end	0.002407
query end	0.000023
removing tmp table	0.000340

group by tm

Creating tmp table	0.000041
Sorting result	0.000008
executing	0.000006
Sending data	1.048907
Creating sort index	0.000094
end	0.000014
query end	0.000021
removing tmp table	0.000331

存储引擎选择

- 抛弃MyISAM，全面使用InnoDB
- 适当的场景下可以采用TokuDB
- 误区：MEMORY可不见得就快

关闭Query Cache

- 绝大多数情况下鸡肋，最好关闭
- QC锁是全局锁，每次更新QC的内存块锁代价高，出现Waiting for query cache lock状态的频率很高
- 实例启动前设置query_cache_type = 0 & query_cache_size =0
- 参考：http://t.cn/RAF4d7z http://t.cn/RAF4d7Z

thread pool

- 启用thread pool
  - 应对突发短连接
  - extra port
- 没thread pool怎么办
  - 想办法启用连接池或其他替代方案
  - 适当调低超时阈值，减少空闲连接

关键配置选项

- innodb_buffer_pool_size，约物理内存的50% ~ 70%
- innodb_log_file_size，5.5及以上2G+，5.5以下建议不超512M
- innodb_flush_log_at_trx_commit，0=>最快数据最不安全，1=>最慢最安全，2=>折中，和sync_binlog组合称为双1
- innodb_max_dirty_pages_pct，25%~50%为宜
- max_connections，突发最大连接数的80%为宜，过大容易导致全部卡死

关闭autocommit

- 避免某些行锁被长时间持有，影响tps
- 更严重时，可能连接数暴涨，导致整个实例挂掉
- 采用gui客户端连接时，记得及时关闭连接，或设置超时阈值以及自动提交，否则容易发生行锁等待问题

项目不想意外倒闭就要

- 备份!
- 备份！！
- 备份！！！
- 定期执行物理/逻辑备份，并开启binlog

谢谢，希望有所帮助