MySQL与CPU

@淘宝丁奇

讲师介绍

讲师简介：

丁奇：
08年至10年在百度贴吧，作服务端开发，开始接触MySQL。之后由于业务需要开始看MySQL代码，囫囵吞枣不求甚解。10年得机会进入淘宝核心系统数据库组，主要是MySQL优化和提升可维护性。参与IC、TC读库调优；写了一些插件，打了几个patch到官方；实现MySQL主从同步工具、设计MySQL异构数据同步方案、MySQL中间层。一直游离在了解需求、设计方案、推广方案的三点一线上 。

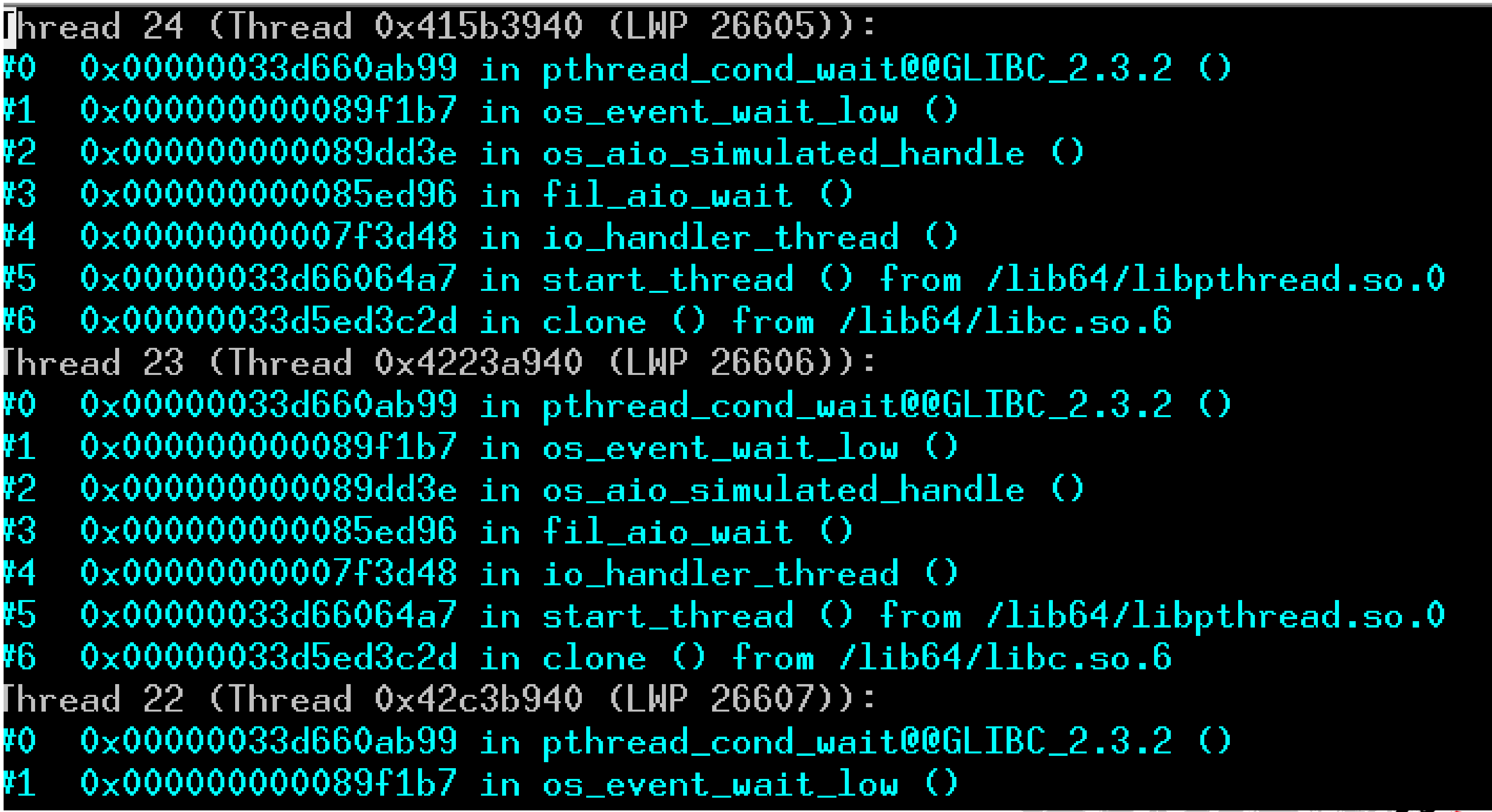
课程目标与目标学员页

- 目标学员：对MySQL、系统优化有兴趣的同学。
- 课程目的：介绍MySQL的线程状况和cpu
- 学员能够获得的收获： 了解MySQL的各个线程工作流程、CPU压力的应用下的瓶颈分析、一些追查问题的方法。

目录

MySQL里有哪些线程，各司何职？

1. 怎么看程序线程？



```
Thread 24 (Thread 0x415b3940 (LWP 26605)):
#0  0x00000033d660ab99 in pthread_cond_wait@@GLIBC_2.3.2 ()
#1  0x000000000089f1b7 in os_event_wait_low ()
#2  0x000000000089dd3e in os_aio_simulated_handle ()
#3  0x000000000085ed96 in fil_aio_wait ()
#4  0x00000000007f3d48 in io_handler_thread ()
#5  0x00000033d66064a7 in start_thread () from /lib64/libpthread.so.0
#6  0x00000033d5ed3c2d in clone () from /lib64/libc.so.6
Thread 23 (Thread 0x4223a940 (LWP 26606)):
#0  0x00000033d660ab99 in pthread_cond_wait@@GLIBC_2.3.2 ()
#1  0x000000000089f1b7 in os_event_wait_low ()
#2  0x000000000089dd3e in os_aio_simulated_handle ()
#3  0x000000000085ed96 in fil_aio_wait ()
#4  0x00000000007f3d48 in io_handler_thread ()
#5  0x00000033d66064a7 in start_thread () from /lib64/libpthread.so.0
#6  0x00000033d5ed3c2d in clone () from /lib64/libc.so.6
Thread 22 (Thread 0x42c3b940 (LWP 26607)):
#0  0x00000033d660ab99 in pthread_cond_wait@@GLIBC_2.3.2 ()
#1  0x000000000089f1b7 in os_event_wait_low ()
```

MySQL里有哪些线程，各司何职？

1. 当出现死锁等情况时，也可以用pstack各个线程在忙什么。
2. 启动后的18个线程，分别是做什么的？
3. 来一个新客户端连接什么情况？
4. 断开一个客户端连接什么情况？--为什么线程没少？

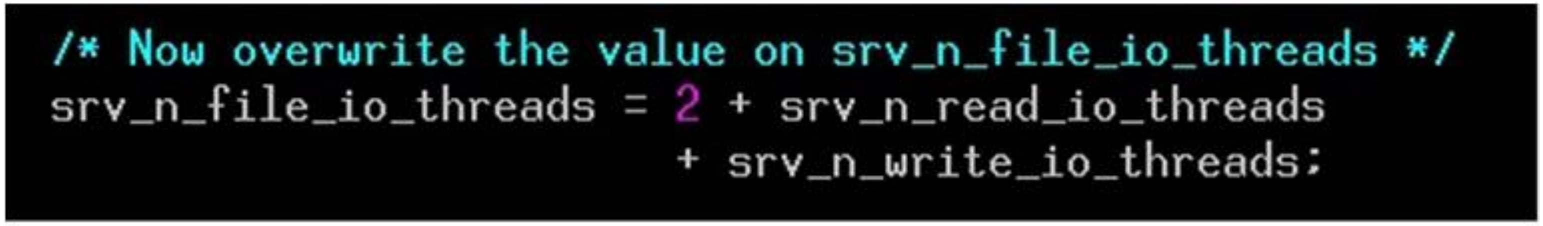
初始线程分析

io_handler_thread 10个

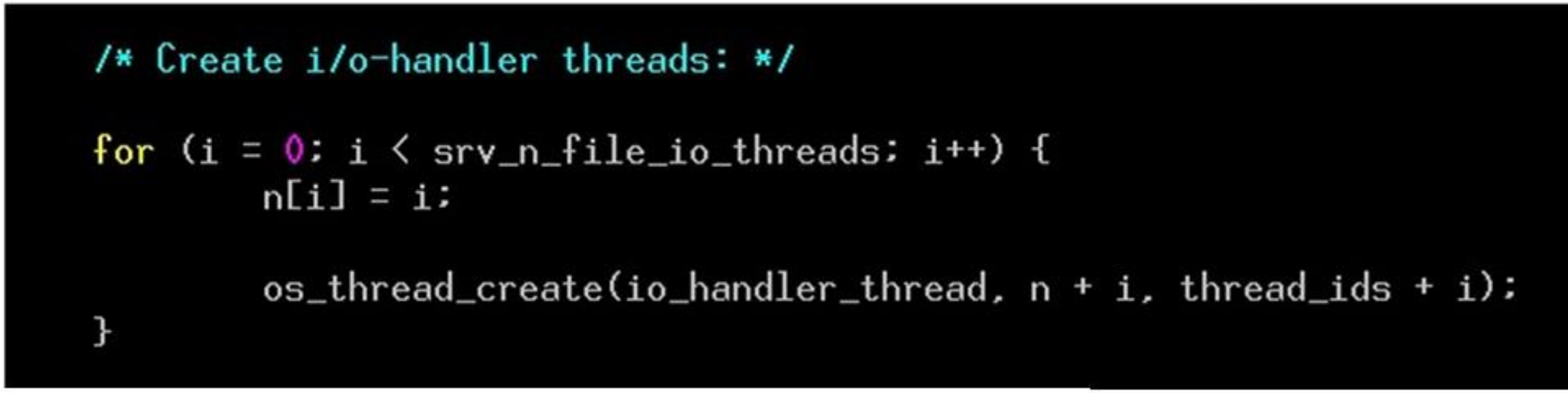
```
Thread 18 (Thread 0x42816940 (LWP 22173)):
#0  0x00000033d660ab99 in pthread_cond_wait@@GLIB
#1  0x00000000008c5797 in os_event_wait_low ()
#2  0x00000000008c430e in os_aio_simulated_handle ()
#3  0x000000000087f89e in fil_aio_wait ()
#4  0x0000000000806638 in io_handler_thread ()
#5  0x00000033d66064a7 in start_thread () from /lib64/lib
#6  0x00000033d5ed3c2d in clone () from /lib64/libc.so.6
```

初始线程分析

io_handler_thread 10个



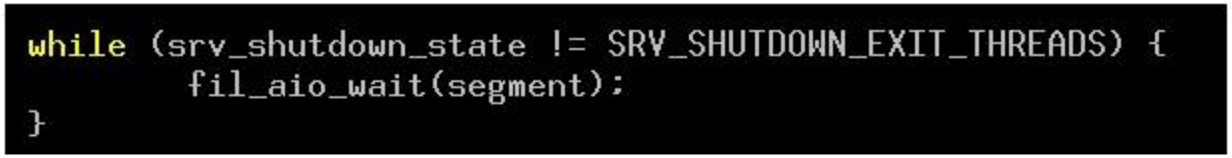
```
/* Now overwrite the value on srv_n_file_io_threads */
srv_n_file_io_threads = 2 + srv_n_read_io_threads
                        + srv_n_write_io_threads;
```



```
    /* Create i/o-handler threads: */

    for (i = 0; i < srv_n_file_io_threads; i++) {
            n[i] = i;

            os_thread_create(io_handler_thread, n + i, thread_ids + i);
    }
```



```
while (srv_shutdown_state != SRV_SHUTDOWN_EXIT_THREADS) {
        fil_aio_wait(segment);
}
```

Insert_buffer \ log \ read\write   thread

初始线程分析

Srv.*thread 6个

1. srv_lock_timeout_thread
2. srv_error_monitor_thread
3. srv_monitor_thread
4. srv_LRU_dump_restore_thread (Percona)
5. srv_master_thread
6. srv_purge_thread (5.5)

初始线程分析

剩下两个

所以初始时**MySQL**层就两个线程

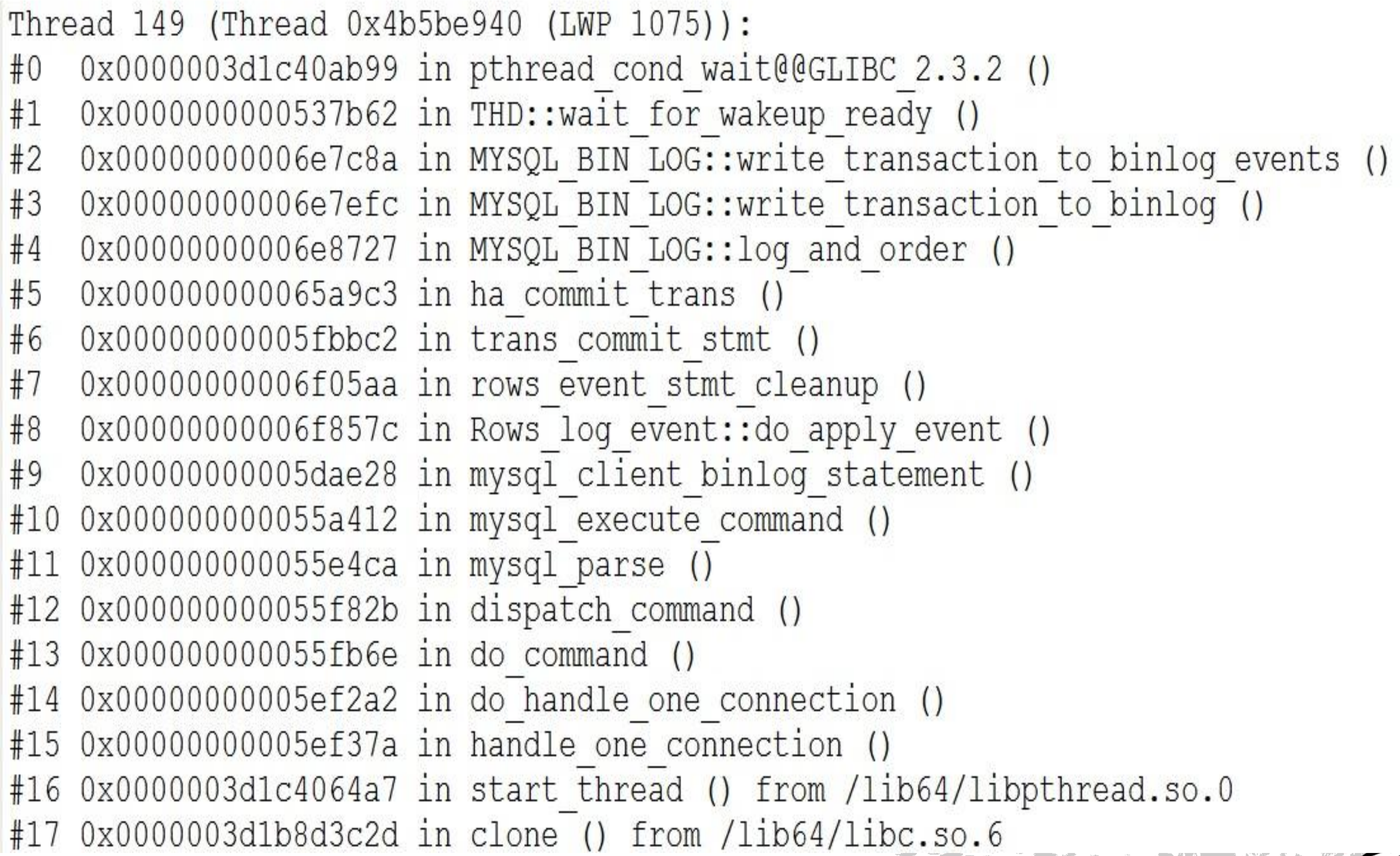
1. main 线程
2. signal_handler

从库谁管？

新连接

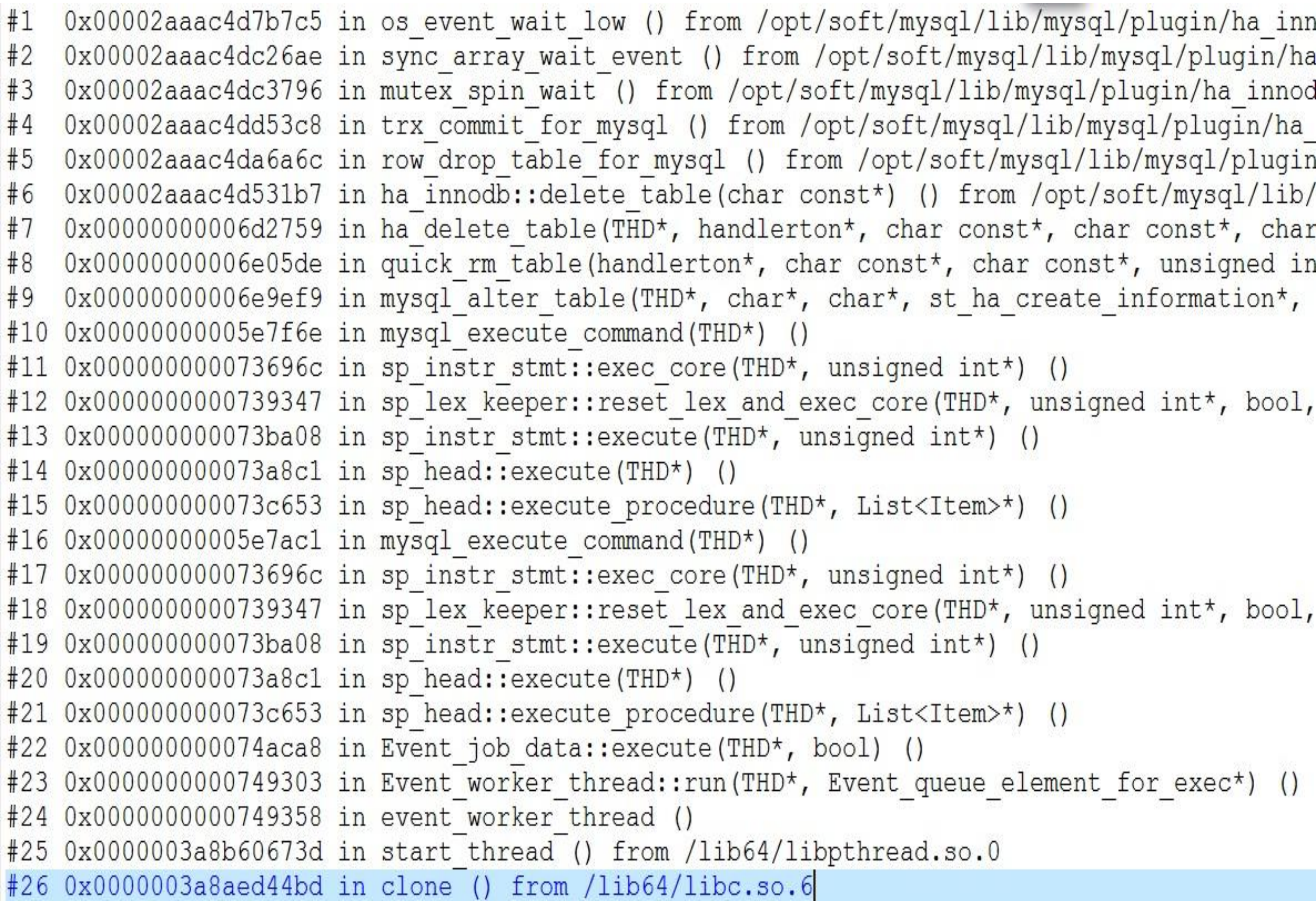
```
Thread 2 (Thread 0x41c7e940 (LWP 26056)):
#0  0x00000033d660ab99 in pthread_cond_wait@
#1  0x00000000004fbd4a in one_thread_per_connection_end ()
#2  0x00000000005f85ef in do_handle_one_connection ()
#3  0x00000000005f871a in handle_one_connection ()
#4  0x00000033d66064a7 in start_thread () from
/lib64/libpthread.so.0
#5  0x00000033d5ed3c2d in clone () from
```

两个例子



```
Thread 149 (Thread 0x4b5be940 (LWP 1075)):
#0  0x0000003d1c40ab99 in pthread_cond_wait@@GLIBC_2.3.2 ()
#1  0x0000000000537b62 in THD::wait_for_wakeup_ready ()
#2  0x00000000006e7c8a in MYSQL_BIN_LOG::write_transaction_to_binlog_events ()
#3  0x00000000006e7efc in MYSQL_BIN_LOG::write_transaction_to_binlog ()
#4  0x00000000006e8727 in MYSQL_BIN_LOG::log_and_order ()
#5  0x000000000065a9c3 in ha_commit_trans ()
#6  0x00000000005fbbc2 in trans_commit_stmt ()
#7  0x00000000006f05aa in rows_event_stmt_cleanup ()
#8  0x00000000006f857c in Rows_log_event::do_apply_event ()
#9  0x00000000005dae28 in mysql_client_binlog_statement ()
#10 0x000000000055a412 in mysql_execute_command ()
#11 0x000000000055e4ca in mysql_parse ()
#12 0x000000000055f82b in dispatch_command ()
#13 0x000000000055fb6e in do_command ()
#14 0x00000000005ef2a2 in do_handle_one_connection ()
#15 0x00000000005ef37a in handle_one_connection ()
#16 0x0000003d1c4064a7 in start_thread () from /lib64/libpthread.so.0
#17 0x0000003d1b8d3c2d in clone () from /lib64/libc.so.6
```

两个例子



```
#1  0x00002aaac4d7b7c5 in os_event_wait_low () from /opt/soft/mysql/lib/mysql/plugin/ha_inn
#2  0x00002aaac4dc26ae in sync_array_wait_event () from /opt/soft/mysql/lib/mysql/plugin/ha
#3  0x00002aaac4dc3796 in mutex_spin_wait () from /opt/soft/mysql/lib/mysql/plugin/ha_innod
#4  0x00002aaac4dd53c8 in trx_commit_for_mysql () from /opt/soft/mysql/lib/mysql/plugin/ha_
#5  0x00002aaac4da6a6c in row_drop_table_for_mysql () from /opt/soft/mysql/lib/mysql/plugin
#6  0x00002aaac4d531b7 in ha_innodb::delete_table(char const*) () from /opt/soft/mysql/lib/
#7  0x00000000006d2759 in ha_delete_table(THD*, handlerton*, char const*, char const*, char
#8  0x00000000006e05de in quick_rm_table(handlerton*, char const*, char const*, unsigned in
#9  0x00000000006e9ef9 in mysql_alter_table(THD*, char*, char*, st_ha_create_information*,
#10 0x00000000005e7f6e in mysql_execute_command(THD*) ()
#11 0x000000000073696c in sp_instr_stmt::exec_core(THD*, unsigned int*) ()
#12 0x0000000000739347 in sp_lex_keeper::reset_lex_and_exec_core(THD*, unsigned int*, bool,
#13 0x000000000073ba08 in sp_instr_stmt::execute(THD*, unsigned int*) ()
#14 0x000000000073a8c1 in sp_head::execute(THD*) ()
#15 0x000000000073c653 in sp_head::execute_procedure(THD*, List<Item>*) ()
#16 0x00000000005e7ac1 in mysql_execute_command(THD*) ()
#17 0x000000000073696c in sp_instr_stmt::exec_core(THD*, unsigned int*) ()
#18 0x0000000000739347 in sp_lex_keeper::reset_lex_and_exec_core(THD*, unsigned int*, bool,
#19 0x000000000073ba08 in sp_instr_stmt::execute(THD*, unsigned int*) ()
#20 0x000000000073a8c1 in sp_head::execute(THD*) ()
#21 0x000000000073c653 in sp_head::execute_procedure(THD*, List<Item>*) ()
#22 0x000000000074aca8 in Event_job_data::execute(THD*, bool) ()
#23 0x0000000000749303 in Event_worker_thread::run(THD*, Event_queue_element_for_exec*) ()
#24 0x0000000000749358 in event_worker_thread ()
#25 0x0000003a8b60673d in start_thread () from /lib64/libpthread.so.0
#26 0x0000003a8aed44bd in clone () from /lib64/libc.so.6
```

MySQL里有哪些线程，各司何职？

1. 当出现死锁等情况时，也可以用pstack各个线程在忙什么。
2. 启动后的18个线程，分别是做什么的？
3. 来一个新客户端连接什么情况？
4. 断开一个客户端连接什么情况？--为什么线程没少？

与线程数目有关的参数

- innodb_purge_threads
- innodb_read_io_threads / innodb_write_io_threads

与线程控制有关的参数

- thread_cache_size
- thread_concurrency （已经废弃）
- innodb_thread_concurrency
- thread_handling (one-thread-per-connection)

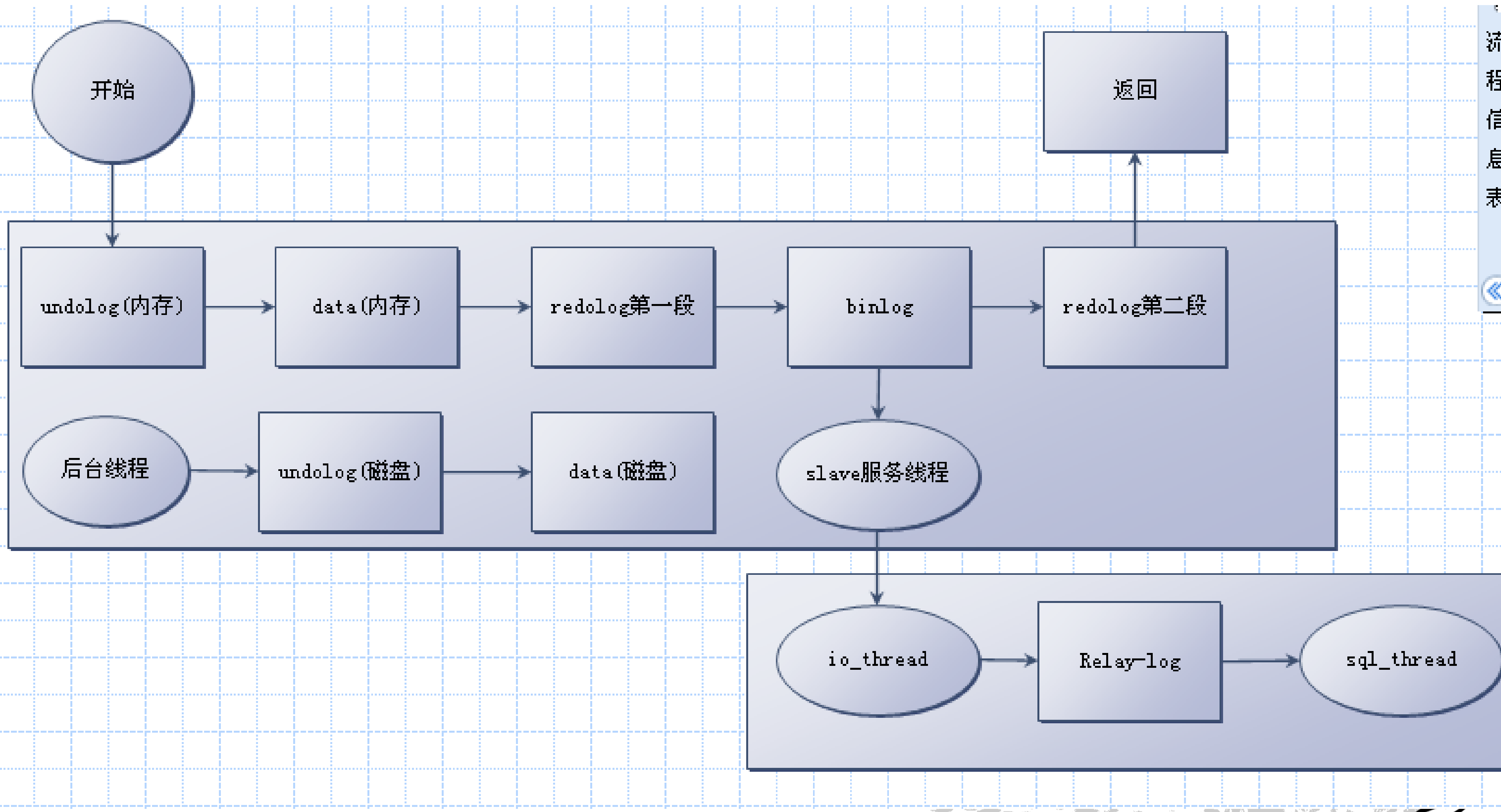
查询涉及到哪些线程

访问数据的方式与此有关

- Select 1 行
- Select 全表

异步read的触发时机 / innodb_read_ahead_threshold

更新涉及到哪些线程



开始
返回
undolog(内存)
data(内存)
redolog第一段
binlog
redolog第二段
后台线程
undolog(磁盘)
data(磁盘)
slave服务线程
io_thread
Relay-log
sql_thread

更新涉及到哪些线程

除用户线程外

- Insert buffer thread
- Log thread
- Io write thread
- Master thread
- Purge thread

并发线程和性能的关系

要达到最大性能，单线程好还是多线程?

- 多核机器当然要多线程
- 但是不是越多越好
- 线上一个例子及解决方法（治标 or 治本）

查询的各个cpu消耗阶段 及 一些优化手段

- **SQL 解析**
  - 注意mysql_parse 不只是解析
  - Batch 操作
- **数据查询**
  - querycache
  - 索引、排序
  - Spin lock
- **数据拷贝**
  - 引擎层与Server层拷贝，大小头问题

大小头问题

- 现象
- 导致的cpu问题
- 改进空间

Spin lock

- 什么是spinlock
- 工作过程
- 什么情况下会成为cpu杀手
- 解决方向

排序操作与CPU，优化空间

- 什么是排序
- 有哪几种排序
- 执行过程
- Cpu都耗在哪里？
- Cpu消耗的模式下可以怎么改进

Batch操作对性能的影响

- 像MySQLdump那样的insert
- Load data的实际流程
- Transfer增加batchinsert的思路和效果
- Analyse table 的流程和优化—fast index creation?

谢谢

追風堂