

SHOW ENGINE INNODB STATUS

2025年6月20日 13:06

```
=====
2025-06-20 13:12:50 0x70e6381b700 INNODB MONITOR OUTPUT
=====
Per second averages calculated from the last 36 seconds          # 表示本次报告内"每秒平均"是统计自最后36秒的情况做的计算。0大概是频繁show导致的
-----
BACKGROUND THREAD
-----
srv_master_thread loops: 917 srv_active, 0 srv_shutdown, 2578785 srv_idle    # srv_master_thread loops : master线程的循环次数。每次循环时会选择一种状态来执行（active、shutdown、idle）
srv_master_thread log flush and writes: 2579702 --主线程日志刷新和写入次数 # 其中，active状态的数量增加与数据变化有关，与查询无关。可以通过srv_activesrv_idle的差值来获得系统的整体负载情况。
----- # active值越大表示系统越繁忙，当active远远高出idle时，说明这个数据库相对比较繁忙。
SEMAPHORES #报告了线程等待信号量的统计信息，并统计线程需要spin或等待mutex或rw-lock信号量的次数。
-----
OS WAIT ARRAY INFO: reservation count 9306      #OS的等待阵列信息。InnoDB分配槽的额度。
OS WAIT ARRAY INFO: signal count 8814           #OS的等待阵列信息。衡量的是线程通过阵列得到的信号的频度。（没懂）
RW-shared spins 0, rounds 5558, OS waits 2326    #spin共享锁(shared)的计数器。
RW-excl spins 0, rounds 25570, OS waits 422       #spin排它锁的计数器。
RW-sx spins 654, rounds 14220, OS waits 235        #spin共享写锁的计数器。（猜测sx应该是shared x，共享写锁常出现在事务中的update操作，环境中正是进行着大量update。）
Spin rounds per wait: 5558.00 RW-shared, 25570.00 RW-excl, 21.74 RW-sx #显示每次线程阻塞在 mutex（互斥锁）时，平均经历了多少次 spinlock 自旋尝试 才真正进入 OS 层面的等待
-----
LATEST DETECTED DEADLOCK
-----
2025-06-20 13:12:43 0x70e63755700 #检测最近一次的死锁，时间0x是写入时的OS句柄（内部追踪）
*** (1) TRANSACTION:          #事务(1)
TRANSACTION 1146215537, ACTIVE 7 sec starting index read      #事务ID，运行了7秒，当前执行阶段、正在根据索引读取行
mysql tables in use 1, locked 1      #此事务一打开1张表，并在该表上持有或等待1把行锁
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)      #说明事务在等待锁，内部用于管理等待锁结构数（这里是两个）、内存堆（heap）字节数1136
MySQL thread id 2672, OS thread handle 139699774904064, query id 76622 localhost root updating
DELETE FROM t WHERE i=1      #正在执行的SQL
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:      #在等待的锁
RECORD LOCKS space id 1851 page no 3 n bits 72 index GEN_CLUST_INDEX of table `csdev`.'t' trx id 1146215537 lock_mode X waiting #index gen_clust_index:指向主键聚集索引，lock_mode请求X锁，处于等待
Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info bits 0 #指在该页（page no 3）上的第2条记录，也就是主键i=1那一行
0: len 6; hex 00000021b98b; asc !;; #以下是InnoDB在“物理记录”（PHYSICAL RECORD）层面展示的每个字段的原始二进制值
1: len 6; hex 00004451dc68; asc DQ h;;
2: len 7; hex c700000e900110; asc ;;
3: len 4; hex 80000001; asc ;;
*** (2) TRANSACTION:          #事务(2)
TRANSACTION 1146215538, ACTIVE 18 sec starting index read, thread declared inside InnoDB 5000      #另一个事务ID，运行了18秒，表明这是INNODB内部创建的线程（不常见，业务层无需关注）
mysql tables in use 1, locked 1
4 lock struct(s), heap size 1136, 3 row lock(s)      #事务已持有3把锁
MySQL thread id 2671, OS thread handle 139699774904064, query id 76623 localhost root updating
DELETE FROM t WHERE i=1      #同样在执行删除 i=1
*** (2) HOLDS THE LOCK(S):      #持有的锁
RECORD LOCKS space id 1851 page no 3 n bits 72 index GEN_CLUST_INDEX of table `csdev`.'t' trx id 1146215538 lock mode S #对第1条记录(heap no 1, supremum)持有共享锁 (S)
Record lock, heap no 1 PHYSICAL RECORD: n_fields 1; compact format; info bits 0
0: len 8; hex 73757072656d756d; asc supremum;;
Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info bits 0      #对第2条记录(heap no 2, 即 i=1)已经持有一把共享锁 (S)
0: len 6; hex 00000021b98b; asc !;;
1: len 6; hex 00004451dc68; asc DQ h;;
2: len 7; hex c700000e900110; asc ;;
3: len 4; hex 80000001; asc ;;
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:      #还在等待获取的锁
RECORD LOCKS space id 1851 page no 3 n bits 72 index GEN_CLUST_INDEX of table `csdev`.'t' trx id 1146215538 lock_mode X waiting #正尝试将对 i=1 的锁从共享锁升级为 排他锁 (X)，所以也要等待
Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
0: len 6; hex 00000021b98b; asc !;;
1: len 6; hex 00004451dc68; asc DQ h;;
2: len 7; hex c700000e900110; asc ;;
3: len 4; hex 80000001; asc ;;
*** WE ROLL BACK TRANSACTION (1)      #死锁判定与回滚
-----
TRANSACTIONS
-----
Trx id counter 1146215539      #每创建一个新事务就会累加。
Purge done for trx's n:o < 1146215537 undo n:o < 0 state: running but idle      #小于1146215537这个ID的事务的历史数据都被清理掉了，undo n:o < 0这个是innodb清理进程正在使用的撤销日志编号，为0时说明清理进程处于空闲状态。
History list length 5      #历史记录中包含多少个事务的列表长度。就是InnoDB数据文件undo里的未清除事务的数量。当一个事务执行了更新并提交后，这个数字就会增加。当清除进程移除一个旧版本数据时，它就会递减（purge thread也会更新purge done for ...也就是上一行的数据。）。
LIST OF TRANSACTIONS FOR EACH SESSION:
--TRANSACTION 421175890223616, not started #每个session的事务情况。事务ID超级大的就是只读事务。一旦有写的动作，才会切换为读写事务，分配正常的事务ID。not started表示这个事务已经提交并且没有再发起影响事务的语句，可能刚好空闲。看事务ID，这是一个查询事务。一旦事务落盘，就not started。
0 lock struct(s), heap size 1136, 0 row lock(s)
--TRANSACTION 421175890221792, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
--TRANSACTION 421175890220880, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
--TRANSACTION 1146215538, ACTIVE 25 sec      #已运行 25 秒未提交或回滚
4 lock struct(s), heap size 1136, 4 row lock(s), undo log entries 1      #InnoDB为此事务分配了 4 个锁结构（lock struct），对应了它曾申请或升级过的 4 次锁，这 4 个锁结构在内存堆（heap）中占据了共 1,136 字节，用于存放锁的元信息（类型、对象指针、事务信息等）。该事务在执行过程中至少有 1 条 undo 日志记录（如对某行做了更新或删除）
MySQL thread id 2671, OS thread handle 139699774904064, query id 76623 localhost root      #MySQL线程ID及其他信息
-----
FILE I/O
-----
I/O thread 0 state: waiting for completed aio requests (insert buffer thread)      #插入缓冲区线程。负责change buffer的合并
I/O thread 1 state: waiting for completed aio requests (log thread)      #日志线程。负责异步的日志刷新。
I/O thread 2 state: waiting for completed aio requests (read thread)      #读线程，负责读前置的操作（read-ahead），预测InnoDB将要使用的数据并读入内存。读线程的数量受参数 innodb_read_io_threads 控制。
I/O thread 3 state: waiting for completed aio requests (read thread)
I/O thread 4 state: waiting for completed aio requests (read thread)
I/O thread 5 state: waiting for completed aio requests (read thread)
I/O thread 6 state: waiting for completed aio requests (write thread)      #写线程，刷新脏页。写线程的数量受参数 innodb_write_io_threads 控制。
I/O thread 7 state: waiting for completed aio requests (write thread)
I/O thread 8 state: waiting for completed aio requests (write thread)
I/O thread 9 state: waiting for completed aio requests (write thread)
```

Pending normal aio reads: [0, 0, 0, 0], aio writes: [0, 0, 0, 0], #读写线程挂起操作的数量。aio就是异步io。
ibuf aio reads:, log i/o's:, sync i/o's: #insert buffer thread挂起的fsync操作数量。
Pending flushes (fsync) log: 0; buffer pool: 0 #log thread 挂起的fsync操作数量。
28699 OS file reads, 294401 OS file writes, 40916 OS fsyncs #读、写以及fsync操作的次数。
0.00 reads/s, 0 avg bytes/read, 0.25 writes/s, 0.25 fsyncs #在头部区域"Per second averages calculated from the last 36 seconds" 的时间段内，平均每秒的执行次数。

INSERT BUFFER AND ADAPTIVE HASH INDEX #此部分显示InnoDB插入缓冲区（也称为更改缓冲区）和自适应哈希索引的状态。

lbuf: size 1, free list len 2157, seg size 2159, 0 merges #size 1已经合并记录页的数量。Free list len 插入缓冲中空闲列表的长度。Seg size, 当前insert buffer的长度，单位为页(16k) merges, 代表合并插入的次数。
merged operations:
insert 0, delete mark 0, delete 0 #三个项目分别表示merge操作合并了多少个insert buffer、delete buffer（标记删除）、purge buffer（真正删除）。
discarded operations:
insert 0, delete mark 0, delete 0 #三个项目分别表示当change buffer发生merge时，表可能已经被删除了，就不再需要合并到辅助索引中了。
Hash table size 276671, node heap has 1 buffer(s) #自适应哈希索引状态。AHI大小，及AHI的使用情况（存有多少个buffer）。

Hash table size 276671, node heap has 1 buffer(s)
Hash table size 276671, node heap has 1 buffer(s)
Hash table size 276671, node heap has 1 buffer(s)
Hash table size 276671, node heap has 2 buffer(s)
Hash table size 276671, node heap has 1 buffer(s)
Hash table size 276671, node heap has 1 buffer(s)
Hash table size 276671, node heap has 1 buffer(s)

0.00 hash searches/s, 0.00 non-hash searches/s #根据头部的时间段，哈希索引查找次数和非哈希索引的查找次数。哈希索引仅适用于等值查询。可以根据searches的比例来结合业务情况，了解是否需要继续启用AHI特性（默认启用）。注意：AHI在高并发业务会导致哈希倾斜，竞争加剧会导致实例hang住或者短暂hang住

LOG #显示有关InnoDB日志的信息。内容包括当前的日志序列号，将日志刷新到磁盘的距离以及InnoDB上次执行检查点的位置。

Log sequence number 253630028025 #当前日志LSN
Log flushed up to 253630028025 #日志已刷新落盘到的位置的LSN
Pages flushed up to 253630028025 #pages flushed up to指的是下一次即将做checkpoint lsn的位置；在没有新数据的写入的情况下，pages flushed up to取的是Log sequence number (log_sys->lsn)；在没数据写入的情况下，为什么last checkpoint point不等于pages flushed up to？是因为做checkpoint是同时redo日志会写MLOG_CHECKPOINT，而MLOG_CHECKPOINT占用九个字节，所以会出现pages flushed up to-last checkpoint point=9；
Last checkpoint at 253630028016 #最新ckpt的LSN
0 pending log flushes, 0 pending chkp writes #还未完成的日志操作及统计信息
10257 log i/o's done, 0.14 log i/o's/second #已经发生的日志操作统计信息

BUFFER POOL AND MEMORY #提供有关已读和已写页面的统计信息。您可以从这些数字中计算出查询目前正在执行多少个数据文件I/O操作

Total large memory allocated 1099431936 #分配给InnoDB Buffer Pool的总内存，单位bytes。
Dictionary memory allocated 3055390 #分配给InnoDB数据字典的内存，单位bytes。
Buffer pool size 65528 #分配给IBP的内存，单位pages。
Free buffers 8192 #Buffer Pool Free List 总大小，单位pages
Database pages 57327 #Buffer Pool LRU List 总大小，单位pages
Old database pages 20998 #IBP old LRU 总大小，单位pages (冷端)
Modified db pages 0 #当前IBP中脏页的数量，单位pages
Pending reads 0 #等待读入IBP的页数量，单位pages
Pending writes: LRU 0, flush list 0, single page 0 #LRU，从LRU列表的底部开始写入的旧脏页数。（没懂）；flush list，ckpt期间要刷新的缓冲池页面数；single page，IBP中暂挂的独立页面写入数。
Pages made young 519, not young 10710 #LRU中被made young的页面数(LRU列表中页移动到前端的次数),not young LRU中保持在old子列表中的页面数(没被移动到前端，因innodb_old_blocks_time设置)。0.00 youngs/s, 0.00 non-youngs/s #youngs/s度量标准仅用于old pages，基于对page的访问次数，而不是页的数量。对页进行多次访问都会被计算。如果见到非常低的值，可能需要减小延迟或增加old page LRU list的比例。增大后，页面需要更长的时间才会移动到尾部，这就增加了再次访问page，从而使他们made young的可能性增大。Not young，如果在执行大表扫描时未看到较高的non-young和non-youngs/s，请增加innodb_old_blocks_time。
Pages read 28040, created 254765, written 259415 #从磁盘到缓冲池读取的页面总数。在缓冲池中创建的页面总数（从ibp分配，但并未从文件中读入数据的页面，可能属于一个被删除的表）。从缓冲池到磁盘写入的页面总数。
0.00 reads/s, 0.00 creates/s, 0.00 writes/s #平均每秒读的页数，创建的页数，写的页数。
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000 #buffer pool hit，缓冲池的命中率，用来衡量innodb在缓冲池中查找到所需页的比例，（不确定是不是：它度量自上次innodb快照输出后到本次输出这段时间内的命中率，因此，如果服务器自那以后一直很安静，你将会看到No buffer pool page gets since the last printout。它对于度量缓存池的大小并没有用处。）；young-making rate，指对所有buffer pool的访问，不仅仅是old page LRU list。Young making和not young通常不会合计到整个IBP的命中率（因为old命中了就移动到new，而new的命中只有在与head有一定距离时才会移动到头部；not，因innodb_old_blocks_time设置而不足以移动到new，或命中new但是并没使其向头部移动的全ibp的命中率，不仅仅是old LRU list）。
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s #每秒平均预读操作次数；每秒因未被访问而被逐出ibp的页数；每秒平均随机预读次数。
LRU len: 57327, unzip_LRU len: 0 #LRU len, IBP中LRU列表包含页面的总数(Database pages)；压缩页的unzip_LRU列表包含页面的总数(0则表示没有)。
I/O sum[16]:cur[0], unzip sum[0]:cur[0] #最近50秒内访问的缓冲池LRU列表页面的总数；已访问的缓冲池LRU列表页面的总数；已访问的缓冲池unzip_LRU列表页面的总数。

INDIVIDUAL BUFFER POOL INFO

--BUFFER POOL 0
Buffer pool size 8191
Free buffers 1024
Database pages 7164
Old database pages 2624
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 53, not young 655
0.00 youngs/s, 0.00 non-youngs/s
Pages read 3382, created 31529, written 32127
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 7164, unzip_LRU len: 0
I/O sum[2]:cur[0], unzip sum[0]:cur[0]
--BUFFER POOL 1
Buffer pool size 8191
Free buffers 1024
Database pages 7167
Old database pages 2625
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 67, not young 953
0.00 youngs/s, 0.00 non-youngs/s
Pages read 3478, created 31451, written 31810
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 7167, unzip_LRU len: 0
I/O sum[2]:cur[0], unzip sum[0]:cur[0]
--BUFFER POOL 2
Buffer pool size 8191
Free buffers 1024
Database pages 7165
Old database pages 2624
Modified db pages 0

```

Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 46, not young 731
0.00 youngs/s, 0.00 non-youngs/s
Pages read 3448, created 32126, written 32544
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 7165, unzip_LRU len: 0
I/O sum[2]:cur[0], unzip sum[0]:cur[0]
---BUFFER POOL 3
Buffer pool size 8191
Free buffers 1024
Database pages 7166
Old database pages 2625
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 77, not young 538
0.00 youngs/s, 0.00 non-youngs/s
Pages read 3552, created 31520, written 32579
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 7166, unzip_LRU len: 0
I/O sum[2]:cur[0], unzip sum[0]:cur[0]
---BUFFER POOL 4
Buffer pool size 8191
Free buffers 1024
Database pages 7166
Old database pages 2625
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 62, not young 460
0.00 youngs/s, 0.00 non-youngs/s
Pages read 3588, created 32426, written 32941
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 7166, unzip_LRU len: 0
I/O sum[2]:cur[0], unzip sum[0]:cur[0]
---BUFFER POOL 5
Buffer pool size 8191
Free buffers 1024
Database pages 7166
Old database pages 2625
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 49, not young 378
0.00 youngs/s, 0.00 non-youngs/s
Pages read 3636, created 32099, written 32536
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 7166, unzip_LRU len: 0
I/O sum[2]:cur[0], unzip sum[0]:cur[0]
---BUFFER POOL 6
Buffer pool size 8191
Free buffers 1024
Database pages 7167
Old database pages 2625
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 69, not young 1350
0.00 youngs/s, 0.00 non-youngs/s
Pages read 3515, created 31814, written 32344
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 7167, unzip_LRU len: 0
I/O sum[2]:cur[0], unzip sum[0]:cur[0]
---BUFFER POOL 7
Buffer pool size 8191
Free buffers 1024
Database pages 7166
Old database pages 2625
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 96, not young 5645
0.00 youngs/s, 0.00 non-youngs/s
Pages read 3441, created 31800, written 32534
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 7166, unzip_LRU len: 0
I/O sum[2]:cur[0], unzip sum[0]:cur[0]
-----
```

ROW OPERATIONS #显示了主线程在做什么，包括每种行操作的数量和性能比率。

```

0 queries inside InnoDB, 0 queries in queue
0 read views open inside InnoDB
Process ID=11210, Main thread ID=139699617376000, state: sleeping
Number of rows inserted 9636804, updated 15, deleted 6, read 2704      #累计值。
0.00 inserts/s, 0.00 updates/s, 0.03 deletes/s, 0.06 reads/s          #前面的统计时间，平均下来每秒值。
-----
```

END OF INNODB MONITOR OUTPUT

补充：

SEMAPHORES:

更详细的 Mutex 信息:

SHOW ENGINE INNODB MUTEX;

问题表现	可能原因	建议
Threads waiting for semaphores 很高	并发争用严重、磁盘慢、mutex 热点	调低 innodb_thread_concurrency、分析热点 SQL、优化 I/O
Spin rounds per wait 很高	高并发自旋效率低	关注是否有锁粒度过大、热点更新等问题
OS waits 很多	mutex 总是抢不到，频繁 sleep	优化表结构或 SQL，减小临界区、加锁粒度

InnoDB有一个多阶段的等待策略。

首先会对锁进行自旋（spin），如果经历了一个自旋周期后还没有持有锁，则进入到操作系统等待状态（os wait），等待被唤醒。

如果在一秒中看到几十万个spin wait，则需要关注show engine innodb mutex;

WE ROLL BACK TRANSACTION:

- InnoDB 检测到 **两个事务都在等待对方持有的锁**:
 - 事务1等待事务2已经持有的排他锁
 - 事务2等待将自己持有的共享锁升级为排他锁，而此时该行又被事务1请求了排他锁
- 于是触发死锁检测，系统根据成本（一般是“回滚成本最小”的原则）选择回滚事务1。回滚后，事务1的所有修改都会被撤销并释放锁，事务2则可以继续完成它的锁升级和后续操作。

事务1: DELETE ... WHERE i=1 FOR UPDATE → 锁等待 X(id=1)

事务2: DELETE ... WHERE i=1 → 先获 S(id=1)，后请求 X(id=1) → 锁升级等待

↑ ↓
←—————互相等待————→ 死锁

关键：互相等待对方释放锁，形成环路，InnoDB 自动检测并回滚其中一方。

TRANSACTIONS:

减少此类开销的最好办法就是确保事务已成就立即提交，不要让事务长时间地处于打开状态，因为一个打开的事务即使不做任何操作，也会影响到innodb清理旧版本的行数据。

FILE I/O:

三行挂起读写线程、缓冲池线程、日志线程的统计信息的值(Pending normal aio reads、ibuf aio reads、Pending flushes (fsync) log)是检测I/O受限的应用的一个好方法，如果这些I/O大部分有挂起操作，那么负载可能I/O受限。

在linux系统下使用参数: innodb_read_io_threads和innodb_write_io_threads两个变量来配置读写线程的数量，默认为各4个线程。

insert buffer thread: 负责插入缓冲合并，如：记录被从插入缓冲合并到表空间中

log thread: 负责异步刷事务日志

read thread: 执行预读操作以尝试预先读取innodb预感需要的数据

write thread: 刷新脏页缓冲