MYSQL配置优化和参数解释

Extra: Using where;在结果集还要过滤一次
Using index直接使用索引就能得到数据

SELECT @@Innodb_buffer_pool_read_requests/1024/1024
配置内存大小的单位为字节/1024/1024
swappiness 如果是单台MYSQL可以设置为0，让MYSQL尽量不使用SWAP.修改方法：cat /proc/sys/vm/swappiness
Innodb缓冲池命中率不应该小于99%算法如下：
show status like 'innodb%read%'
缓冲泡命中率
=Innodb_buffer_pool_read_requests/(Innodb_buffer_pool_read_requests +Innodb_buffer_pool_read_ahead+Innodb_buffer_pool_reads)
innodb_buffer_pool_reads：表示从物理磁盘读取页的次数
innodb_buffer_pool_read_ahead：预读次数
innodb_buffer_pool_read_ahead_evicted：预读的页，但是没有被读取就从缓冲池中被替换的页的数量，一般用来判断预读的效率。
innodb_buffer_pool_read_requests：从缓冲池中读取页的次数。
innodb_data_read：总共读入的字节数。
innodb_data_reads：发起读取请求的次数，每次读取可能需要读取多个页.
优化的定律
1.对于一个占总响应时间不超过%5的查询进行优化，再优化也不会超过%5的收益。
2.如果花费100块钱去优化一个任务，但业务收入没有任何增加那就可以说反而导致业务被逆化了100元，如果优化的成本大于收益，就没必要进行。
Undo Log 是为了实现事务乩原子忏,在 MySQL 数掘库 InnoDB 存储弁擎中,迂用 Undo Log 来实现兜版本开收
控刽(简称:MVCC)。Undo Log 乩原理径简单,为了满足事务乩原子忏,在操作伕何数掘乀前,首兇将数掘备佯剐 UndoLog,然后迭行数掘乩修改。如果出现了错诣戒考用户执行了ROLLBACK询句,系统可仑刭用UndoLog中乩 备佯将数掘恢复剐事务开始乀前乩状忏。因此 Undo Log 乩 IO 忏能对二数掘捏入戒更新乔是径重要乩一个因素。
创健连接不要和并发搞混，如有成千上万的连接到mysql但其实有时候只有几个正在执行查询。
http://mirrors.sohu.com/mysql/
cpu饱和发生在MYSQL使用的数据能被装入内存，或能尽快根据需要从磁盘上读取的时候。密集的算法和执行无索引的连接操作都是引发高cpu的例子。
IO饱和一般发生在需要的数据比内存多得多的时候。如果应用程序分布在网络上，查询数据相当大或要求很低延迟的时候，瓶颈就会转移到网络上。
max_connections最大连接数(限制服务器的用户限制)
show variables like 'max_connections';
max_used_connections响应的连接数的历史高水位，可以告诉你某个时间点有个尖峰，如果它达到的max_connections说明客户端至少被拒绝了一次。
show variables like max_used_connections';
如果服务器的并发连接请求量比较大，建议调高此值，以增加并行连接数量，当然这建

立在机器能支撑的情况下，因为如果连接数越多，介于MySQL会为每个连接提供连接缓冲区，就会开销越多的内存，所以要适当调整该值
max_used_connections / max_connections * 100% （理想值≈ 85%）
如果max_used_connections跟max_connections相同那么就是max_connections设置过低或者超过服务器负载上限了，低于10%则设置过大数值过小会经常出现ERROR 1040: Too many connections错误，可以过’conn%’通配符查看当前状态的连接数量。

back_log(500)暂存的连接数量

当主要MySQL线程在一个很短时间内得到非常多的连接请求，这就起作用。如果MySQL的连接数据达到max_connections时，新来的请求将会被存在堆栈中，以等待某一连接释放资源，该堆栈的数量即back_log，如果等待连接的数量超过back_log，将不被授予连接资源。

TCP 连接优化(WEB服务器，不是MYSQL，可以参照)
/proc/sys/net/ipv4/tcp_keepalive_time：TCP发送keepalive探测消息的间隔时间（秒），用于确认TCP连接是否有效。7200优化为1800秒
/proc/sys/net/ipv4/tcp_max_syn_backlog:对于还未获得对方确认的连接请求，可保存在队列中的最大数目。如果服务器经常出现过载，可以尝试增加这个数字。
/proc/sys/net/ipv4/tcp_fin_timeout:对于本端断开的socket连接，TCP保持在FIN-WAIT-2状态的时间（秒）。对方可能会断开连接或一直不结束连接或不可预料的进程死亡。60,优化为30秒;
/proc/sys/net/ipv4/ip_local_port_range：表示TCP/UDP协议允许使用的本地端口号32768 61000调优化为1024 65535

更多参数请搜：Linux之TCPIP内核参数优化

back_log值指出在MySQL暂时停止回答新请求之前的短时间内有多少个请求可以被存在堆栈中。只有如果期望在一个短时间内有很多连接，你需要增加它，换句话说，这值对到来的TCP/IP连接的侦听队列的大小。 当观察你主机进程列表mysql> show full processlist，发现大量
264084 | unauthenticated user | xxx.xxx.xxx.xxx | NULL | Connect | NULL | login | NULL
的待连接进程时，就要加大back_log 的值了。默认数值是50，可调优为128，对于Linux系统设置范围为小于512的整数。
interactive_timeout一个交互连接在被服务器在关闭前等待行动的秒数。一个交互的客户被定义为对mysql_real_connect()使用CLIENT_INTERACTIVE 选项的客户。默认数值是28800，可调优为7200。
key_buffer_size(一般是40%-70%物理内存只对myisam有效)指定索引缓冲区的大小，它决定索引处理的速度，尤其是索引读的速度。通过检查状态值Key_read_requests(索引请求)和Key_reads(从磁盘读取索引的请求次数可以理解成没使用索引，这个数越少越好)，可以知道key_buffer_size设置是否合理。比例key_reads / key_read_requests应该尽可能的低，至少是1:100，1:1000(也就是小于0.01)更好（上述状态值可以使用SHOW STATUS LIKE ‘key_read%’获得）。
key_cache_miss_rate(命中率) =Key_reads / Key_read_requests * 100%，设置

在1/1000(0.001是千分之1)左右较好。在0.1%以下都不错了(0.1%就是千分之1也就是0.001每1000个请求有一个直接读硬盘)，
默认配置数值是8388600(8M)，主机有4GB内存，可以调优值为268435456(256MB)。
Key_blocks_unused表示未使用的缓存簇(blocks)数，Key_blocks_used表示曾经用到的最大的blocks数，比如这台服务器，所有的缓存都用到了，要么增加key_buffer_size，要么就是过渡索引了，把缓存占满了。比较理想的设置：
Key_blocks_used / (Key_blocks_unused + Key_blocks_used) * 100% ≈ 80% (0.08)。
query_cache_size 使用查询缓冲，MySQL将查询结果存放在缓冲区中，今后对于同样的SELECT语句（区分大小写），将直接从缓冲区中读取结果。通过检查状态值Qcache_*，可以知道query_cache_size设置是否合理（上述状态值可以使用SHOW STATUS LIKE Qcache%获得）。query_cache_size：查询缓存总空间，单位字节，必须是1024的整数倍。如果**Qcache_lowmem_prunes**的值非常大，则表明经常出现缓冲不够的情况，如果Qcache_hits的值也非常大，则表明查询缓冲使用非常频繁，此时需要增加缓冲大小；如果Qcache_hits的值不大，则表明你的查询重复率很低，这种情况下使用查询缓冲反而会影响效率，那么可以考虑不用查询缓冲。此外，在SELECT语句中加入SQL_NO_CACHE可以明确表示不使用查询缓冲。
查看是否开启查询缓存
使用缓存查询：**update,delete insert操作相比select要少很多才行，查询消耗大返回结果少的如count类似的**
show variables like 'query_cache_%'里面的
query_cache_type指定是否使用查询缓冲，可以设置为ON，OFF，DEMAND。DEMAND表示只有在查询语句中使用SQL_CACHE和SQL_NO_CACHE来控制是否需要缓存，该变量是SESSION级的变量。
query_cache_limit指定单个查询能够使用的缓冲区大小，缺省为1M。
query_cache_min_res_unit是在4.1版本以后引入的，它指定分配缓冲区空间的最小单位，缺省为4K。检查状态值Qcache_free_blocks(当前剩余的blocks)， 如果该值非常大，则表明缓冲区中碎片很多，这就表明查询结果都比较小，此时需要减小query_cache_min_res_unit。
query_cache_min_res_unit的估计值：(query_cache_size - Qcache_free_memory) / Qcache_queries_in_cache
可以通过Qcache_free_blocks来观察碎片，这个值反应了剩余的空闲块，如果这个值很多，但是Qcache_lowmem_prunes却不断增加，则说明碎片太多了。可以使用flush query cache整理碎片，重新排序，但不会清楚，清空命令是reset query cache。整理碎片期间，查询缓存无法被访问，可能导致服务器僵死一段时间，所以查询缓存不宜太大。
提高查询缓存的使用率对于写密集型的应用，建议关闭查询缓存。：

如果碎片不是问题，命中率却非常低，可能是内存不足，可以通过Qcache_free_memory 参数来查看没有使用的内存。如果2者都没有问题，命中率依然很低，那么说明缓存不适合你的当前系统。可以通过设置query_cache_size = 0或者**query_cache_type** 来关闭查询缓存。

query_cache_limit:如果查询结果大于这个值，则不会缓存。如果超出则qcache_not_cached就会增加。并把结果从缓存中删除。如果事先知道有这样结果发生可以在语句中加入sql_no_cache来避免缓存的额外开消。

query_cache_wlock_invalidate:在某个表被其它连接锁住，是否仍然从查询缓存中返回结果。默认为OFF，如果为ON则不会从缓存中读取这类数据。

mysql第三版书上：352页

判断查询缓存是否有效的直接命中率qcache_hits/(cache_hits+com_select)。发起一个SELECT要么增加cache_hits要么增加com_select。

根据cache_hits和cache_insert的比值来看命中和写入的比率，当这个比值大于3：1时通常查询是有效的，最好能达到10:1如果没达到最好是关闭。

mysql> SHOW VARIABLES LIKE '%query_cache%';可以看见以上的值。

查询缓存碎片率= Qcache_free_blocks / Qcache_total_blocks * 100% 如果查询缓存碎片率超过20%，可以用FLUSH QUERY CACHE整理缓存碎片，或者试试减小query_cache_min_res_unit，如果你的查询都是小数据量的话。查询缓存利用率= (query_cache_size - Qcache_free_memory) / query_cache_size * 100% 查询缓存利用率在25%以下的话说明query_cache_size设置的过大，可适当减小；查询缓存利用率在80%以上而且Qcache_lowmem_prunes > 50的话说明query_cache_size可能有点小，要不就是碎片太多。查询缓存命中率= (Qcache_hits - Qcache_inserts) / Qcache_hits * 100% 示例服务器查询缓存碎片率＝20.46%，查询缓存利用率＝62.26%，查询缓存命中率＝1.94%，命中率很差，可能写操作比较频繁吧，而且可能有些碎片。

如果cache_free_blocks大小恰好达到qcache_total_blocks/2那么查询就有严重碎片问题。如果你空闲块很多，但qcache_lowmem_prunes还不断增加，说明由于碎片导致了过早地在删除查询缓存结果。可以用FLUSH QUERY CACHE完成碎片整理。它会把查询缓存重新排序，并把所有空闲空间都集中到查询缓存的一块区域上。这个命令不会奖查询缓存清空，清空缓存由RESET QUERY CACHE完成。FLASH QUERY CACEH会访问所有查询缓存，在这期间任何其它连接都无法访问查询缓存，从而会导致服务器僵死一段时间。建议保持查询缓存空间足够小，以便维护时可以将服务器僵死时间控制在短时间内。

Mysql 目前只支持四种索引HASH B-TREE R-TREE FULL-TEXT。

1.B-TREE索引 在Mysql里用的较多，除了archive基本所有的存储引擎都支持它。b-tree在myisam里的形式和innodb稍有不同。

在 innodb里，有两种形态：一是primary key形态，其leaf node(叶节点)里存放的是数据，而且不仅存放了索引键的数据，还存放了其他字段的数据。二是secondary index(辅助索引)，其leaf node和普通的b-tree差不多，只是还存放了指向主键的信

息。

Threads_cached：提高为客户端创建连接过程性能，相当于连接泡。把空闲的连接放在里面不立刻销毁，如果有新的连接来会在这里面找空闲的连接，如没有才创建。有关Thread_Cache在MySQL有几个重要的参数,简单介绍如下:

thread_cache_size

Thread_Cache 中存放的最大连接线程数.在短连接效果明显。大了会浪费资源，一般和物理内存有关1G —> 8个2G —> 16

3G —> 32 >3G —> 64个如果短连接多的话可以适当加大.也可以查看当前线程缓冲的大小是否够(threads_createad/uptime),如果这个值跟0没差太多就意味着你的线程缓冲区太小了，新来的连接在线程缓冲区里找不到空闲的线程可用。

thread_stack：每个连接被创建的时候,mysql分配给它的内存。

```
mysql> show variables like ‘thread%’;
+——————-+——————————+
| Variable_name      | Value                        |
+——————-+——————————+
| thread_cache_size | 32                 |
| thread_handling     | one-thread-per-connection |
| thread_stack          | 196608           |
+——————-+——————————+
3 rows in set (0.01 sec)
mysql> show status like ‘%connections%’;
+——————-+—— - +
| Variable_name        | Value  |
+——————-+—— - +
| Connections           | 199156 |
| Max_used_connections | 31     |
+——————-+—— - +
2 rows in set (0.00 sec)
mysql> show status like ‘%thread%’;
+——————-+—— - +
| Variable_name     | Value  |
+——————-+—— - +
| Delayed_insert_threads | 0      |
| Slow_launch_threads  | 0      |
| Threads_cached        | 3      |
| Threads_connected   | 6      |
| Threads_created       | 8689 |
| Threads_running       | 5      |
+——————-+—— - +
6 rows in set (0.00 sec)
```

可以看到服务器的 thread_cache池中最多可以存放32个连接线程,为每个客户端球使用一个线程.为每个连接的线程分配192k的内存空间.服务器总共有199156次连接,最

大并发连接数为31,当前在thread_cashe池中的连接数为3个,连接数为6个,处于活跃状态的有5个,共创建 了8689次连接.显然这里以短连接为主.可以算出thread_cache命中率,公式为:

Thread_Cache_Hit=(Connections-Thread_created)/Connections*100%

当前服务器的Thread_cache命中率约为95.6%这个结果我还是比较满意的.但是可以看出 thread_cache_size有点多余改成16或8更合理一些.(解释：thread_cache_size设置不会立即生效，等到下一次线程关闭的时候，那时MYSQL检查缓存中是否有空间存线程，如果是，它会把线程缓存起来，供另一个连接使用。如果不是，它会直接结束掉线程。在这情况下，缓存中线程的数量，和线程缓存使用的内存数据不会马上就下降。只有当新连接为了使用线程把它从缓存中移走的时候才会下降。MYSQL只有在连接关闭的时候才会把线程加入缓存，也只有在创建新连接的时候才从缓存中移除线程。)

当一个新连接被创健出来并缓存中有一个线程时，MYSQL就会把这个线程从缓存中删除，并把它赋给连接。连接关闭时MYSQL会回收线程把它放回缓存中。如果缓存中没空间了，MYSQL就会销毁该线程。

thread_cache_size定义了MYSQL能在缓存中保存的线程数量，除非服务器有很多连接请求，否则就不要改变这个值，可以通过观察threads_created变量的值来确定线程缓存是否够大。如果每秒创建的线程数少于10个，缓存的大小就是够用的。如果therads_connected通常在100到200之间变化那就可以把线程缓存调成100

slow_launch_threads:创健时间超过了slow_launch_time秒的线程数。如果较大说明某些因素正在延迟联接分配的新线程，说明服务器有问题，通常表示系统过载，导致操作系统给创健的线程分配时间片。要诊断这个问题并修复它。

thread_pool_size定义为同时运行sql语句的mysql的线程数，是mysql thread最重要的性能参数.说是商业版才可能调整，但是我使用percona 的版本也可以动态调整.

表缓存

在mysql5.1中表缓存分了两部份：打开表table_open_cache和定义表table_definition_cache。表定义是全局的，可以在所有连接中共享。如果opened_tables值很大或正在上升，说明表缓存不够大，应该增加table_open_cache的值,把这个值变得很大唯一坏处就是在有很多myisam表的时候会导致较长的关闭时间，因为要冲刷健数据块，表还要被标记为不再打开。也会导致FLUSH TABLES WITH READ LOCK需要较长时间才能完成。

对于innodb来说并不是那么重要，因为innodb在很多方面都不会依赖于它因为innodb有自己的表缓存。但是innodb还是可以从解析后的.frm文件中获益。线程和表缓存都不会占用太多内存，但在高并发下开销就会上升很快

如果收到MYSQL报错不能打开更多文件的错误，可以在my.cnf文件中设定open_files_limit解决这个问题

1、open-files-limit

它限制了mysqld进程可持有的最大打开文件数，相当于是一个小区的总电闸，一旦超限，小区里所有住户都得停电。

5.6.7（含）以前，默认值0，最大和OS内核限制有关；

5.6.8（含）以后，默认值会自动计算，最大和OS内核限制有关。

在5.6.8及以后，其自动计算的几个限制规则见下，哪个计算结果最大就以哪个为上限:

```
1) 10 + max_connections + (table_open_cache * 2)
2) max_connections * 5
3) open_files_limit value specified at startup, 5000 if none
```

2、innodb-open-files

限制InnoDB引擎中表空间文件最大打开的数量，相当于自己家中电箱里的某个电路保险，该电路短路的话，会自动跳闸，而不会影响其他电路，去掉短路源后重新按上去就可以使用。

其值最低20，默认400，只计算了包含ibdata*、ib_logfile*、*.ibd 等三类文件，redo log不计算在内，5.6以后可独立undo log，我还未进行测试，应该也不会被计算在内，有兴趣的朋友可验证下。

3、table-definition-cache

该cache用于缓存 .frm 文件，该选项也预示着 .frm 文件同时可打开最大数量。

5.6.7 以前默认值400；

5.6.7 之后是自动计算的，且最低为400，自动计算公式：400 + (table-open-cache / 2)。

对InnoDB而言，该选项只是软性限制，如果超过限制了，则会根据LRU原则，把旧的条目删除，加入新的条目。

此外，innodb-open-files 也控制着最大可打开的表数量，和 table-definition-cache 都起到限制作用，以其中较大的为准。如果没配置限制，则通常选择 table-definition-cache 作为上限，因为它的默认值是 200，比较大。

4、table-open-cache

该cache用于缓存各种所有数据表文件描述符。

5.6.7 以前，默认值400，范围：1 - 524288；

5.6.8 - 5.6.11，默认值2000，范围：1 - 524288；

5.6.12以后，默认值2000（且能自动计算），范围：1 - 524288。

补充说明1：关于如何计算表文件描述符的建议：

table-open-cache 通常和 max-connections 有关系，建议设置为 max_connections * N，N的值为平均每个查询中可能总共会用到的表数量，同时也要兼顾可能会产生临时表。

补充说明2：MySQL会在下列几种情况把表从table cache中删掉：

> 1、table cache已满，并且正要打开一个新表时；
> 2、table cache中的条目数超过 table_open_cache 设定值，并且有某些表已经长时间未访问了；
> 3、执行刷新表操作时，例如执行 FLUSH TABLES，或者 mysqladmin flush-tables 或 mysqladmin refresh

补充说明3：MySQL采用下述方法来分配table cache：

1、当前没在用的表会被释放掉，从最近最少使用的表开始

2、当要打开一个新表，当前的cache也满了且无法释放任何一个表时，table cache会临时加大，临时加大的table cache中的表不用了之后，会被立刻释放掉。

InnoDB事务日志

INNODB是怎样刷新日志缓冲。当innoDB把日志缓冲刷新到磁盘日志时，先会使用一个Mutex锁住缓冲区，刷新到需要的位置，然后移动剩下的条目到缓冲区前面。当Mutex释放时，可能有超过一个事务已经准备好刷新其日志记录。有一个group commit功能可以在一个IO操作内提交多个事务。

Innodb使用日志来减少提交事务的开销，它不是每次提交事务就把缓冲池刷新到磁盘，而是记录了事务。事务对数据和索引做出的改变通常会被映射到表空间的随机位置，所以把这些改变写到硬盘就会引起随机IO。(随机IO需要时间在磁盘上寻找正确的位置，并还要等磁头移到相应的位置上)使用自身的日志把随机IO转换为顺序IO，日志文件是由

innodb_log_file_size和innodb_log_files_in_group控制，这两个文件默认大小5MB总计为10B，对于高性能的负载，这个大小不够，日志文件总大小上限是4GB，但是即使是写入负载极高的查询也高只需要几百MB(比如256MB)这两个参数如果太小innodb将会设置更多的检查点，并导致更多的日志写入。在极端情况下写入查询有可能会停下来，等日志的记录被应用到数据文件上。如果太大，innodb在恢复的时候就会做大量的工作，会极大的增加恢复时间(SSD就会效率更快可以调大)。如果有大型事务就可以增加缓存文件(默认为1MB)来减少IO动作。控制log缓冲大小的变量叫innodb_log_buffer_size，推荐值为1MB到8MB(如果分配一台大内存服务器时可以简单地分配32-128MB)，除非写入大量的巨型BLOB记录。innodb的日志想对正常数据要紧凑得多，它们不是基于页面的，所以不会在存储数据的时候浪费整个页面。

innodb_log_file_size:可以配置256M以上，如果对系统非常大写的情况下，也可以考虑用这个参数提高一下性能，把文件设大一点，减少checkpoint的发生。如果没大事务，控制在8M-16M即可。

检测Innodb的日志和日志缓冲IO性能：观察innodb_os_log_written的值了解innodb向日志文件写入的量在10秒到100秒时间间隔内的数据的最大值。如果最大数据是每秒写入100KB那么1MB的日志缓冲(innodb_log_buffer_size)可能就足够了。也可以使用这个指标来决定日志文件的合适大小，如最大值是每秒100KB，那256MB的日志文件就可以存储至少2560秒的日志记录。(100*2560=256000KB=256MB)一般一小时就够了

Innodb如何刷写日志缓冲

如果比起持久性更在意性能，就可以把**innodb_flush_log_at_trx_commit**的值来控制日志缓存被刷写到什么地方和刷写的频率。

0 把日志缓存写到日志文件中，并每秒刷一次，但在事务提交的时候不进行任何动用

1(事务提交时innodb_log_buffer中的内容会写到log file，并log file 也会刷新到磁盘)将日志缓冲写到日志文件中，并在事务提交的时候把缓存刷写到持久性存储中。这是默认(最安全)的设置。它保证不会遗失任何提交的事务，除非磁盘或操作系统“假冒”了刷写操作。

2 在每次提交的时候把日志缓冲写到日志文件中，但不进行清理。innodb安排每秒清理一次。它和0最大的区别是2在mysql进程崩溃的时候不会丢失任何事务。但是如果日志服务器崩溃或失去电力，还是有可能丢失事务。

大多数系统中是innodb的内存缓冲区移到操作系统的缓存中，它也在内存中。实际上不会把数据写入持久性存储中，因此设置0和2通常导致在崩溃或电力故障的时候最多失去一秒钟的数据，因为这时数据可能只在操作系统的缓存中。所以说”通常“原因是

innodb会每秒钟把日志文件写到磁盘上一次，但是在某些情况下也有可能失去多1秒的数据，例如刷写被停住了。把日志写到持久存储意味innodb把数据写入磁盘这会阻止IO调用直到数据被完全写入。由于数据写到磁盘比较慢所以当innodb_flush_log_at_trx_commit被设置到1时它会降低innodb每秒可以提交的事务。机械硬盘每秒只能执行几百次真正的磁盘事务，这是由驱动器旋转和寻址时间所决定的。固态硬盘性能会更好。
有时硬盘控制器或操作系统会假冒清写动作，它会把数据放入另外一个缓存中，比如磁盘自己的缓存。这样做速度较快但是很危险，因为数据在驱动器失去电力的时候还是会丢失，这甚至比把innodb_flush_log_at_trx_commit设置为1之外更糟糕，因为它不仅是导致丢失事务，还能导致数据损坏。(以上说的是普通硬盘)。高性能事务的最佳配置是把flush_log_at_trx_commit设置为1，并把日志文件放在有备用电池的写缓存的RAID上，这不仅安全速度也很快。
innodb_file_per_table导致每个文件都被单独地使用了fsync()函数，这个意味着想多个表写入不能被合并到单个io操作中，这可能要innodb执行较多的fsync()操作。
innodb_buffer_pool_size:innodb严重依赖于它，如果它设大预热和关闭都会花费很长时间，如果有很很多脏页在缓冲池里，在关闭时会花费较长时间，因为在关闭时要把脏页写到数据文件。如果事先知道什么时候关闭innodb可以在运行时修改innodb_max_dirty_pages_pct(它小并不保证innodb将在缓冲池中保持更少的脏页，它只是控制innodb是否可以偷懒的值，innodb默认会用一个后台线程来刷新脏页，并合并写入，更高效顺序写出到磁盘，这个行为称为偷懒)变量把值改小，等待刷新线程清理缓冲池，然后在脏页数量较少时关闭，可以监控innodb_buffer_pool_pages_dirty状态或用innotop来监控show innodb status来观察脏页的刷新量。当有很大的缓冲池，在重启服务器时需要几小时或几天的预热的时间，尤其是硬盘很慢的时候，可以搜(快速预热buffer_pool)或进行全表，全索引扫描。或把SQL放在一个文件里。用init_file来完成。
innodb_max_dirty_pages_pct(75):当脏页的百分比超过了这个值，innodb将快速地刷写脏页，尝试让脏页的数量更低。(如过大内存很大服务器压力也大，如太小，硬盘的压力也会增加)当事务日志没有足够的空间时，innodb也将进入“激烈刷写(Furious Flushing)”模式，这就是大日志可以提升性能的一个原因。
show innodb status里的LOG里log sequence number(日志号)和log flushed up to(刷新日志)如果相差大，和BUFFER POLL AND MEMORY的修改的页modified db*16k/1024=就是没有写到数据文件。(如果是写入量大，而写入的数据不是太活跃，可以考虑把这个值设的低一点，如果写入或是更新的数据也就是热数据就可以把这个值设为95%)
旧数据版本和表空间
如果有很多未被清理的事务并表空间因它而增长，就可以强制MYSQL变慢，来让清理线程能跟上数据的变化。否则innodb就会不停地写入数据并填充硬盘，直到耗尽磁盘空间或让表空间达到规定上限。为了减缓写入可以把**innodb_max_purge_lag:**变量设为0之外的值。这个值表示等待清理的事务的最大数量，一旦超过这个值innodb就会延迟更新数据的事务，要知道自己的负载才能决定这个值的最佳大小。如：事务平均影响1KB数据，并能容忍100MB未清理的数据，那么这个值就是1 000 00.要记录未被清理的行会影响所有的查询，因为它会让表和索引很快变大。如果清理线程跟不上节奏，性能就会下降，设置**innodb_max_purge_lag**也会降低性能，但是它的危害比前者小。此

参数是全局动态参数用来控制history list的长度，如history list长度大于它时会“延缓”DML的操作，默认为0表示不做限制，大于0就会延缓DML的操作，其延缓算法为：delay=((length(history_list)-innodb_max_purge_lag)*10)-5，delay单位为毫秒，对像是行，而不是一个DML操作。例如一个update操作需要更新5行数据时，每行数据的操作都会被delay，所以总延时时间为5*delay。而delay的统计会在每一次purge操作完成后，重新进行计算。在innodb1.2中引入了新的动态参数innodb_max_purge_lag_delay用来控制delay的最大毫秒数。也就是当上述计算得到的delay值大于该参数时，将delay设置为indoor_max_purge_lag_delay，避免由于purge操作缓慢导致其它SQL线程出现无限制的等待。

双写缓存

innodb在对页面进行部分写入的时候使用了双缓冲来防止数据损坏。可以通过把innodb_doublewrite设为0禁用。

sync_binlog:控制mysql如何把二进制刷写到磁盘，默认为0，这意味不会执行任何刷写，并何时把日志写到持久设备上取决于操作系统。如果大于0就规定了把二进制日志刷写到磁盘期间可以运行多少次写入。一般就是0或1很少有设更大的。如果为0就有可能导致二进制日志和事务的数据不同步。

expire_logs_day选项自动地删除老的二进制日志，单位是天，在flush logs的时候去check是否过期。如选择了它删除就不要使用rm来删除，否则服务器会不知道到底哪一个起作用，就会不能正常工作，并PURGE MASTERL LOGS也会停。解决办法就是手动使用磁盘上的文件列表和hostname-bin.index文件进行重新同步。

MyIsam并发优化

concurrent_insert变量配置myisam的并发插入行为。

0 不允许并发插入，每一次都会把表锁住。

1 默认值。只要表中没有空缺，就允许并发插入。

2 在mysql5.0更高的版本可用。它强制并发插入到表尾，即使表有空缺也不例外。如果没有线程从表中读取数据，mysql就会把新数据插入到空缺中。使用了该值表的碎片会增多，所以就需要更经常地对表进行优化。

Innodb并发优化

innodb两种并发:1逻辑并发(应用可以看到资源的竞争，如表或行锁争用)，2内部并发(缓冲池页的资源争用，改变服务器设置或用不同硬件，这类问题只能缓解不能解决)

只有控制线程。最基本的方式就是使用**innodb_thread_concurrency**(8)变量，它限制了一次有多少线程能进入内核。0表示不限制进入内核的数量。如果有Innodb并发问题该变量是最重要的配置项。没有办法为所有的架构和工作负载确定最佳的并发数理，在理论上可以用以下公式：并发＝CPU的数量X磁盘的数量X2如果内核中已经有了允许数量的线程，那么线程就不能再进入内核了。innodb采用了一种两阶段的过程来保证线程可以尽可能高效地进入内核。线程首先睡眠**innodb_thread_sleep_delay**(150000)所规定的微秒数，然后再次进行尝试。如果还是不能进入，它就会进入一个等待线程的队列中并把控制权交给操作系统。

第一阶段默认的眨眼时间是10000微秒，当有很多线程都处于“正在等待进入队列”这一状态时，改变这个值有助于高并发性系统。默认值在有大量小查询的时候会太大了，因为它给查询增加了10毫秒延时。一但线程进入了内核，它就会得到一个确定的数字作为“凭据”，它再次进入内核的时候，就不会再进行任何的并发检查。该数字限定了它再次回到等待队列之前能做多少工作。**innodb_concurrency_tickets**(5000)选项控制了

凭据的数量。除非有大量的运行很长时间的查询，否则我们极少改动这个选项。凭据只是为每个查询授权，而不是为每个事务授权。一旦查询结束，凭据就会被丢掉。

innodb_commit_concurrency:控制有多少个线程可以在同一时间提交。

优化filesort

去掉不必要的字段，用覆盖索引，当查询和order by时用到不是同一索引就会引起，加大sort_buffer_size参数，增加它并不是为了让MYSQL选择改进使用哪种排序，而是减少在排序过程中对须要排序的数据进行分段，因为分段会造成MYSQL使用临时表来进行交换排序。

双路排序：是首先根据相应的条件取出相应的排序字段和可以直接定位行数据的行指针信息，然后在sort buffer 中进行排序。

单路排序：是一次性取出满足条件行的所有字段，然后在sort buffer中进行排序。

max_length_for_sort_data的大小和Query语句中order by 字段大小总和来判定需要使用单/双路排序，如果超过此值则使用单路排序，如果sort_merge_pass值上升就要强制使用单路排序。

aborted_clients:由于客户端没有正确关闭连接导致客户端终止而中断的连接数。如果随时增加，那么就要确定是不是正确关闭连接，如果不是就要检查网终性能，并检查max_allowed_packet，如果超过了max_allowed_packet的查询会被强制中断。

max_allowed_packet:这个设置防止服务器发送太大的包，也会控制多大的包可以被接收。如果太小，复制有时候会出现问题，通常表现为备库不能接收主库发过来的复制数据，也许要增加到16MB或更大，如果超过这个值，它们可能会被截断或者设置为NULL。

aborted_connections:式图连接到mysql服务器而失败的连接数。这个值应该接近0，如果不是就有可能是网络问题。有几个被中断的连接是正常的。如：某些人试着从错误的主机连接，使用了错误的用户名和密码，或定义了无效的数据库。

binlog_cache_disk_use和binglog_cache_use:如果它两之间比率很大，就应该增加binglog_cache_size的值。大部份事务最好落在二进制日志缓存里面，偶尔一个发生在磁盘也没关系。一但未中率下降到某一点，就不会再从加大缓存中受益。假如未中率每秒一个，并加了缓存大小，让未中率减少到每分钟一次就够好了。如果再下降也得不到多少好处，还不如把内存节约起来做别的事情。

binlog_cache_disk_use:使用临时二进制日志缓存，但超过binglog_cache_size的值并使用临时文件来保存事务中的语句的事务数量。

binglog_cache_use:使用临时二进制日志缓存的事务数量。

bytes_received(从所有客户端接收到的字节数)和bytes_sent(发送给客户端的字节数):这两值可以帮你找到服务器的数据是发送太多还是读取太多引起的，也许会指出代码中的其它问题，比如查询提取了超出自己需要的数据。

created_tmp_disk_tables:服务器执行语句时在磁盘上自动创健临时表的数量和繁忙程度。如果值高就有可能在查询选择BLOB或TEXT列的时候，或tmp_table_size和max_heap_table_size可能不够大(这两个参数最好一样大,是控制Memory引擎的内存临时表能使用内存的大小，如果超过将会被转为磁盘MyIsam表,如果太大会内存溢出)。

created_tmp_tables:服务器执行语句时自动创健的内存中的临时表数量。如果created_tmp_disk_tables较大，可能要增加tmp_table_size值使临时表基于内存而不基于硬盘。

handler_read_rnd_next:在数据文件读下一行的请求数。如果你正进行大量的表扫

描，些值会很高，说明索引不正确或写入的查询没有利用索引。
handler_read_rnd_next/handler_read_rnd显示了全表扫描的大致平均值，如果较大应该优化架构和索引包括查询。
handler_read_rnd:根据固定位置读一行请求数，如果正执行大量查询并需要对结果进行排序些值会较高。你可能使用了大量需要MYSQL扫描整个表的查询或你的连接没有正确使用健。
handler_read_prev:按照健顺序读前一行的请求数。该方法主要用于优化ORDER BY ….DESC。
handler_read_next:按照健顺序读下一行的请求数。如果使用范围约束或如果执行索引扫描来查询索引列，就会增加。
handler_read_key:根据健读一行的请求数。如果高说明查询和表的索引正确。
key_blocks_used:如果key_blocks_used*key_cache_block_size的值远小于已经充分热身的服务器上的key_buffer_size值，那就意味着key_buffer_size的值太大了，内存被浪费掉求了。
open_files:它不应该和open_files_limit的值接近。如果接近了就应该增加open_files_limit。
open_tables:当前打开表的数量。如果很大或一直增长，可能是因为表缓存table_open_cache不够大,如果太大会导致关机和关闭会很长时间，因为它会把打开的表关闭并标为不再打开。也可能FLUSH TABLES WITH READ LOCK操作更长时间。
opened_tables(0):已经打开表的数量。应该把值和table_cache进行对照，如果每秒有太多opened_tables,说明table_cache还不够大。表缓存没有被全利用上时，显示的临时表也能导致opened_tables增加。
select_full_join:没有使用索引的联接数量。如果不为0，检查表的索引。全联接，真正的性能杀手。就算是一分钟一次也太多。
select_full_range_join(0):如果变量值高说明用了许多使用范围查询联接表。范围查询比较慢，也是一个较好的优化点。
select_range_check(0):记录了在联接时，对每一行数据重新检查索引的查询计划数量，它的性能开销很大。如果比较高或正在增加，那就意味着一些查询没有找到好索引。
sort_merge_passes(0):如果值大应该增加sort_buffer_size，也许只是为某些查询。检查查义并查明哪一个导致了文件排序，最好的办法只有优化。
table_locks_waitead:不能立即获得表锁的次数，如果值高并在增加就得INNODB和分表。
slave_parallel_workers:默认为0，表示不开启并行复制，有效值0-1024
在数据库存中并发分两种
逻辑并发：解决对应用程序可见的资源竞争，如表锁和行锁，通常需要的策略是改变应用程序，使用不同的存储引擎，改变服务器配置，或使用不同的锁定提示或事务隔离层次。
内部并发：对信号量，innoDB缓冲池中的页面访问的争夺，可以能过改变服务器设置，改变操作系统或使用不同的硬件来解决，但是问题可能一直会存在。有时候使用不同的存储引擎或存储引擎补丁有助于减轻这类问题。
skip_name_resolve:在生产环镜打开是个好主意，因为失效或缓慢的域名解析对于大多数程序都是问题，MYSQL尤其严重。当MYSQL收到请求时，它会进行正向和反向的DNS查找。有很多因素会导致出错。当错误发生时，就会拒绝连接，减慢了连接到服务器的过程，并通常会导致灾难，后果甚至会像受到拒绝服务攻击一样。如果打开skip_name_resolve时MYSQL就不会进行任何DNS查找。

replicate_do_db:规则可以指定复制某个库的数据。如：
replicate_do_db=hello
本来可以复制hello这个库数据，use test;然后在test里CREATE TABLE hello.tes这样回hello里面insert tes这个表就会报错了。
internal_tmp_disk_storage_engine：可定义磁盘临时表的引擎类型为InnoDB，而在这前只能使用MyIsam。
default_tmp_storage_engine：是在5.6.3后增加的系统选项，是控制CREATE TMPORARY TABLE创建的临时表引擎类型，以前默认的是MEMORY，不要把它和internal_tmp_disk_storage_engine搞混了。set default_tmp_storage_engine="innodb";
big_tables:将所有临时表存储在磁盘，而非内存中。
sql_log_bin=0:的作用和目的是禁止将自己的语句记录到二进制日志文件的binlog中。千万不要加globle这样会让所有的都不记录。
skip_networding：选项告诉mysql不要监听任何TCP socket，但是它任然会允许Unix socket连接。启动不带网络支持的mysql也很简单，只要在my.cnf里加入skip_networking。但是它会阻止你使用如stunnel这样的工具来做安全的远程连接和复制。bind_address=127.0.0.1这样只能本机连接。
max_connect_errors:(100)负责过多尝试失败的客户端连接，来防止暴力破解。这和性能无关如果值太小就会出现以下错误
ERROR 1129 (00000): Host 'gateway' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'
如果要重置要么重启服务要么执行FLUSH HOSTS
last_query_cost:这个变量显示了查询优化器的查询计划在最近一次执行查询时的开销。
innodb_sync_spin_loops:
wait_timeout(28800):是指mysql在关闭一个非交互的连接之前所要等待的秒数。
interactive_time:是指mysql在关闭一个交互的连接之前所要等待的秒数。
innodb_data_file_path=/disk1ibdata1:1G;/disk2/ibdata2:1G
———以下是控制复制行为参数
read_only:禁止没有特权的用户在备库做变更。只接受主传输过来的变更，不接爱应该来的。
skip_slave_start:禁用，这个是在MYSQL崩溃或其它问题后自动启动。
slave_net_timeout:这个选项控制备库发现跟主库的连接已经失败并需要重连接之前等待的时间，一般为1分钟或更短。
sync_master_info(10000),sync_relay_log,sync_relay_log_info:在mysql5.5以后可用，解决了复制中备库长期存在的问题，不把它们的状态文件同步到磁盘，所以服务器崩溃后可能要人来猜测复制的位置实附上在主库是哪个位置，并且可能在中继日志(relay log)里有损坏。这些选项使得更容易从崩溃中恢复。默认是不打开的，因为它们会导致备库额外的fsync()操作，可能会降低性能，如果硬件牛B还是建议打开，如果复制中出现fsync造成的问题就应该关闭它们。
sync_relay_log:默认为10000，即每10000次sync_relay_log事件会刷新到磁盘。为0则表示不刷新，交由OS的cache控制。
sync_relay_log_info：若relay_log_info_repository为FILE，当设置为0，交由OS刷新磁盘，默认为10000次刷新到磁盘；若relay_log_info_repository为TABLE，且为INNODB存储，则无论为任何值，则都每次evnet都会更新表。

要INNODB不丢数据----------
innodb_flush_log_at_trx_commit = 1 以及 sync_binlog = 1，在master还有一个Xa什么的忘求咯
如果同时还想保证slave端能和master保持一致性的话，把master_info_repository 和 relay_log_info_repository 都设置为TABLE默认为(FILE)，并且relay_log_recovery=1即可
master_info_repository = TABLE
relay_log_info_repository = TABLE
这两个参数会将master.info和relay.info保存在表中(5.7多源复制也要改为TABLE)，默认是Myisam引擎，官方建议用
alter table slave_master_info engine=innodb;
alter table slave_relay_log_info engine=innodb;
alter table slave_worker_info engine=innodb;
改为Innodb引擎，防止表损坏后自行修复。
relay_log_recovery(off):在5.5.X增加了它，作用是当slave从库宕机后，假如relay-log损坏，导致一部份中继日志没处理，则自动放弃没有执行的relay-log，并重新从master上获取日志这样就保证了relay-log的完整性。
innodb_autoinc_lock_mode(1):这个选项控制innodb如何生成自增主健值，在高并发插入时自增主健可能是个瓶颈，如果有很多事务等待自增锁(可以在show engine innodb status看到)应该审视这个变量设置。0全部使用表锁，1可预判行数时使用新方式，不可用时表锁。2全部使用新方式(不安全)使用1就够了。
innodb_buffer_pool_instances(0):8个innodb buffer是在5.5和更高版中出现，可以把缓冲池切分为多段，这可能是在高负载多核机器上提升MYSQL可扩展性最重要的一个方式了。多个缓冲池分散了工作压力，所以一些全局mutex竞争就没那么大了。
innodb_io_capacity(200):这个参数控制了innodb checkpoint时的IO能力，一般可以按一块SAS 15000转的磁盘200个计算，6块盘的SAS做的Raid10这个值可以配到600即可，如果是普通的SATA一块盘只能按100算。可以告诉innodb服务器有多大的IO能力，有时候需要把这个设置得相当高，在像PCI-E SSD这样很快的存储设备上需要设置为上万才能稳定地刷新脏页。2000-20000范围，主要是看处理的IPOS(每秒输入/输出)。
innodb_read_io_threads和innodb_write_io_thread:这两项控制后台有多少线程可以被IO操作使用，默认4个，对于大部份服务器都够了，尤其是5.5里可以用操作系统原生的异步IO以后。如果有很多硬盘并工作负载并发很大，可以发现这些线程很难跟上，这种情况下可以增加线程数，可以简单地把这个选项值设置为可以提供IO能力的磁盘数量(即使后面是一个RAID控制器)
innodb_strict_mode(OFF):在有些条件下把警告改成抛错，如CREATE TABLE时候有风险的语句等。
innodb_old_blocks_time:innodb有两段缓冲泡LRU链表，目的是为防止换出长期使用很多次的页面。像mysqldump产生的这种一次性的大查询，通常会读取页面到缓冲池的LRU列表，从中读取需要的行，然后移动到下一页。(在411页第三版的)默认为1000毫秒(1秒)
innodb_support_xa：设置为1，标志支持分布式事物，主要保证binary log和其它引擎的主事务数据保持一致性。如果设置为0就是异步操作，这样就会一定程度上减少磁盘刷新次数和磁盘的竞争。(5.5是互斥，5.6是先刷盘，里面是队列)

Innodb_page_size:页大小，默认为16384。16K
----------其它对IO有影响的参数(以5.6为准）开始----------
innodb_adaptive_flushing 默认即可
innodb_change_buffer_max_size 如果是日值类服务，可以考虑把这个增值调到 50
innodb_change_buffering 默认即可
innodb_flush_neighors 默认是开的， 这个一定要开着，充分利用顺序IO去写数据。
innodb_lru_scan_depth: 默认即可 这个参数比较专业。控制LRU列表中的可用页数量，默认为1024
innodb_max_purge_lag 默认没启用，如果写入和读取都量大，可以保证读取优先，可以考虑使用这个功能。
innodb_random_read_ahead 默认没开启，属于一个比较活跃的参数，如果要用一定要多测试一下对用passport类应用可以考虑使用
innodb_read_ahead_threshold 默认开启：56 预读机制可以根据业务处理，如果是passprot可以考虑关闭。如果使用innodb_random_read_ahead,建议关闭这个功能
innodb_read_io_threads 默认为：4 可以考虑8
innodb_write_io_threads 默认为：4 可以考虑8
sync_binlog 默认即可： 0
innodb_rollback_segments 默认即可: 128
----------其它对IO有影响的参数(以5.6为准）结束----------
slave_parallel_workers:
————————innodb 内幕 innodb后台线程----------
innodb是多线程模型因此后台有不同的后台线程:
1.Master Thread核心线程负责刷新缓冲池的导步刷新，包括脏页，合并插入缓冲(insert buffer)和undo页回收等。
2.Io Thread 有4个IO Thread分别是write read insert buffer和log Io thread。从1.0x开始read thread和write thread分别增加到了4个，分别用innodb_read_io_threads和innodb_write_io_threads进行设置
innodb_purge_threads(1)
3.Purge Thread 事务被提交后，undolog可能不再需要，所以用purge thread来回收已经使用并分配的undo页。在innodb1.1开始purge操作被独立到单独的线程中来减轻master thread的工作，从而提高CPU的使用，可以用**innodb_purge_threads**=1来控制。进一步加快undo页的回收，同时由于purge thread需要离散地读取undo页，这样也能更进一步利用磁盘的随机读性能，可以设置4个purge thread
4.Page Cleaner Thread是在1.2.x引入的，作用是把之前版本中脏页的刷新操作都放入到单独线程中完成。减轻了原来Master Thread的工作及对用户查询线程的阻塞，进一步提高innodb的性能。
innodb_old_blocks_pct(37)：差不多5/8 LRU midpoint算法。midpoint位置由innodb_old_blocks_pct控制，在innodb存储引擎中midpoint之后的列表称为Old列表，之前的被称为new列表，可以简单的理解为new列表中的页都是最为活跃的热点数据。为了防止全表扫描多页来放到原生的LRU列表前，刷掉了真正的热数据，新增加了一个参数:
innodb_old_blocks_time(1000)：用于表示页读取到的mid位置后需要等待多久才会被加入到LRU列表的热端。
innodb_log_buffer_size:控制redo log(控制事务的原子性持久性,undo是保证一致性)缓冲池中还有重做日志缓冲，innodb首先将重做日志放入到这个缓冲，然后按一定的频率刷新到重做日志文件，这个缓冲一般不要太大，因为一般每秒会刷新到文件一次，

默认为8MB。这个参数是innodb事务日志所使用的缓冲区，在写事务日志的时候为了提高性能，也是先写入到innodb_log_buffer中，当满足innodb_flush_log_trx_commit所设的条件或日志缓冲区写满后，才会把日志写到文件或磁盘中，在下列三种情况会刷新到文件中一次：
1.Master Thread 每一秒将重做日志缓冲刷新到重做日志文件。
2.每个事务提交时。
3.当重做日志缓冲池剩余空间小于2/1时。

checkpoint技术

为了解决CPU速度和磁盘速度的鸿沟，因此页的操作都是在缓冲中完成的，如一条DML语句UPDATE或DELETE改变了页中记录，那么这时页是脏的。为了避免发生丢失数据当前事务都用了Write Ahead Log(预写日志)，当事务提交时先写重做日志，再修改页。这也是事务ACAD中的D持久性要求。checkpoint技术目的是：
A 缩短恢复时间
B 缓冲不够用时，将脏页刷新到磁盘
C 重做日志不可用时刷新脏页(根据LUR算法会溢出最少使用的页，若是脏页就会强制checkpoing将脏页刷回磁盘)重做日志是循环使用的。
checkpoint所做的事情就是将缓冲池中的脏页刷到磁盘，不同的是每次刷多少页，从哪里取脏页，和啥时候触发，在innodb存储引擎内部有两种checkpoing分别为sharp和Fuzzy Checkpoint
Sharp Checkpoint发生在数据库关闭时将所有的脏页都刷磁盘，这是默认的即参数：innodb_fast_shutdown=1

Fuzzy Checkpont又包括了：

Master Thread Checkpoint
FLUSH_LRU_LIST Checkpoint
Async/Sync Flush Checkpoint
Dirty Page too much Checkpoint

对于Master Thread差不多每秒或十秒的速度进行脏页刷新表中一定比例的页，是异步的不会阻塞
FLUSH_LRU_LIST 是因为存储引擎要保证LRU列表中要有差不多100个空闲页可供使用，在innodb1.1.x前，要检查LUR列表中是否有可用的空间操作发生在用户查询线程中，显然这个会阻塞用户查询操作，如没有100个可用空闲页，innodb会把LRU尾部页移除，如果这些页有脏页，就要进行checkpoint，这些页是来自LRU列表的，所以称为FLUSH_LRU_LIST Checkpoint在5.6后也就是1.2.x这个检查被单独的page clear线程中进行，并可以用**innodb_lru_scan_depth**控制LRU列表中的可用页数量，默认为1024
Async/Sync Flush 指的是重做日志文件不可用的情况，这时要强制把一些页刷回磁盘，此时脏页是在脏页列表中选取的，如将已经写入重做日志的LSN记为**redo_lsn**，将已经刷新回磁盘最新页的LSN记为**checkpoint_lsn**
最后一种情况是Dirty Page too much，即脏页数量太多，导致innodb引擎强制进行checkpoint。目的来说还是为了保证缓冲池中有足够可用的页。可由参数innodb_max_dirty_pages_pct来控制。默认为75当缓冲池中脏页占据75%时强制进行checkpoint，刷新一部分的脏页到磁盘。

Master Tthread线程的工作方式

在innodb1.0.x之前其内部由多个loop组成，主循环loop，后台循环backdrop loop,刷新

循环flush loop，暂停循环suspend loop。主线程会根据数据库存运行状态在这几个loop中进行切换。
loop为主循环有每秒一次和每10秒一次的操作是用sleep来实现的，只是大概的时间，负载高的时候会超过10秒。**每一秒的操作包括：**
1.把日志缓冲刷新到磁盘，即使这个事务还没有提交。（总是）
2.合并插入缓冲（可能）
3.最多刷新100个缓冲池中的脏页到磁盘（可能）
4.如果没有当前用户活动，则切换到background loop（可能）
即使事务没有提交innodb也会每秒把重做日志缓冲中的内容刷到生做日志文件。这就是为什么**再大的事务commit的时间也很知的原因**
合并插入缓冲(insert buffer)它不是每秒都发生的，innodb会判断当前一秒内发生的IO次数是否小于5次，如果小于它为认为当前IO压力小，可以执行合并缓冲操作。
刷新100个脏页也不是每秒都会发生的，innodb会判断当前缓冲池中脏页的比例（buf_get_modified_ratio_pc）,是否超过了innodb_max_dirty_pages_pct这个值，如果超了innodb会认为要做同步了，将100个脏页FLUSH
接下来是**10秒的操作**，包括以下内容：
1 刷新100个脏页到磁盘（可能的情况）
2 合并最多5个插入缓冲（总是）
3 将日志冲刷到磁盘（总是）
4 删除没用的undo页（总是）
5 刷新100个或10个脏页到磁盘（总是）
在以上过程，innodb会先判断过去10秒内磁盘IO操作是否小于200次，如果是，它为认为有这个能力操作，因此把100个脏页刷新到磁盘。接着会合并插入缓冲，接着innodb会进行一上执行full puerge（清除所有）操作，即删除无用的undo页。在full perge过程中innodb会判断当前事务系统中已被删除的行是否可以删除，比如有时候可能还有查询操作要读取之前版本的undo信息，如果没有则删除，innodb在执行full purge时每次最多尝试回收20个undo页。然后Innodb会判断缓冲池中脏页比例(buf_get_modified_ratio_pct)如果有超过面分之70的脏页，则刷新100个到磁盘，如小于70%则刷新10%到磁盘。
如没有用户活动(数据库空闲时)或关闭(shutdown)就会切换到这个循环。background loop会执行以下操作：
1 删除无用的undo页（总是）
2 合并20个插入缓冲（总是）
3 跳回到主循环（总是）
4 不断刷新100个页直到符合条件（可能，跳转到flush loop中完成）
如果flush loop中也没有事做了，innodb会切换到suspend loop将Master Thread挂起，等事件发生。如果启用innodb引擎但是没有innodb表，那么Master Thread总是处于挂起状态。
以上是1.0.x之前,在1.0.x之后由于现在有了磁盘的高速发展出现了SSD这个高IO性能的硬件，以前的innodb最大只会刷新100个脏页，合并20个插入缓冲。如果是密集写入中每秒产生大于100的脏页，如果是产生大于20个的插入缓冲，Master Thread就会忙不过来，或很慢。同时当宕机恢复也会很久，因为很多数据还没被FLUSH到磁盘，尤其是insert buffer。所以在1.0.x以后提供了一个参数：**innodb_io_capacity**来表示磁盘IO

的吞量，默认为200，对于刷写到磁盘页的数量，会按照innodb_io_capacity的百分比来进行控制如下：
1 在合并插入缓冲时，合并插入缓冲的数量为innodb_io_capacity值的%5
2 在从缓冲区刷新脏页时，刷新脏页的数量为innodb_io_capacity
如使用了SSD类的磁盘，或做了RAID，当存储设备有更高的IO速度时，可以把它的值调的再高点，直到符合磁盘IO的吞吐量。
innodb_adaptive_flushing(ON):自适应地刷新，在1.0.x带来的新参数。它影响每秒刷新脏页的数量，原来的刷新机制是：脏页在缓冲池中所占的比例小于innodb_max_diry_pages_pct时，不刷新脏页，大于它时才刷新100个。随着这个新参数引入，innodb会通过一个名为buf_flush_get_desired_flush_rate的函数来判断需要刷新脏页最合适的数量，它是通过判断产生redo log的速度来决定最合适的刷新脏页数据，所以当脏页小于innodb_max_dirty_pages_pct时，也会刷新一定量的脏页。还有就是之前full purge时最多回收20个Undo页，从1.0.x开始引入了
innodb_purge_batch_size(300)，它可以控制每次full puerge回收的undo页数量。(通常来说此值越大每次回收的undo page也就越多，可供重用的undo page就越多，减少了磁盘存储空间与分配的开销，但是如果太大则每次要purge处理更多的undo page，从而导致CPU和磁盘IO过于集中对于undo log的处理，使性能下降。普通用户不用调整)
innodb1.2.x的Master Thread
由于Master Thread对性能起到关健作用，在1.2.x中进行了优化，对于刷新脏页的操作，从Master Thread线程分离到一个单独的Pate Cleaner Thread，从而减轻了Master Thread的工作，同时进一步提高了系统并发性。
innodb关健特性这些特性为innodb带来更好的性能以及更高的可靠性。
插入缓冲(insert buffer)，
两次写(double write)，
自适应哈希索引(adaptaive hash index)，
异步IO(async IO)，
刷新邻接页(flush neighbor page)
插入缓冲
insert buffer(是B+树，非叶节点存放的是查询的健值search key)的升级Change Buffer在1.0.x开始引入
Innodb可以对DML操作INSERT DELETE UPDATE都进行缓冲，分别是insert buffer delete buffer purge buffer，和之前的insert buffer一样，change buffer适用对像也是非唯一的辅助索引，对一条对象UPDATE可能分为两步：
1 把记录标记为已删除
2 真正删除记录
因此Delete buffer对应的是第一个过程，purge buffer对应的是UPDATE的第二个过程。**innodb_change_buffering**就是控制各种buffer的选项，参数可选值为(inserts deletes purges changes all none)changes表示启用inserts和deletees，all表示启用所有。默认为all。**innodb_change_buffer_max_size(25):**可以通过它来控制Change Buffer最大使用内存数量。默认25表示最多用1/4的缓冲池内存空间，它的最大有效值为50，可以在show innodb里看到每种buffer操作的次数。总之insert buffer是给innodb带来性能上的提升。
两次写Double write

double write 带给innodb的是数据页的可靠性。宕机时innodb可能正在写入某个页到表中，而这个页只写了一部分，如16KB只写了4KB就挂了，这被称为部分写失效(partial page wirte)。注意在重做日志记录的是对页的物理操作，如果这个页本身已损坏，再重做是无意义的。doublewrite由两部份组成，一部分是内存中的doublewrite buffer为2MB，另一个就是物理磁盘上共享表空间中连续的128个页，即2个分区(extent)同样为2MB。在对缓冲池的脏页进行刷新时，并不直接写磁盘，而是通过memcpy函数将脏页先复制到内存中的doublewrite buffer然后通过doublewrite buffer丙分两次，每次1MB顺序地写入共享表空间的物理磁盘上，然后马上调用fsync来同步磁盘，避免缓冲写带来的问题，在这个过程中因为doublewrite页是连续的，因此这个过程是顺序写的，开销并不是很大。在doublewirte完成页的写入后，再把doublewrite buffer中的页写入各个表空间文件，此时写入则是离散的，可以用show global status like 'innodb_dblwr%'查看。
show status **innodb_dblwr_pages_written**代表doblewrite一共写入的页数,会变的，innodb_dblwr_written代表实际写入量要符合64:1如果小于64:1可以说明系统压力并不是很高。用**innodb_dblwr_pages_written/innodb_dblwr_written**就等于innodb_dblwr_pages_written:innodb_dblwr_written。
show global status like '**innodb_buffer_pool_pages_flushed**%'会看见当前从缓冲池中刷新到磁盘页的数量，它的值和innodb_dblwr_pages_written是一样的，只是在版本不同加上的。所以看innodb_dblwr_pages_written就行了。
skip_innodb_doublewrite：可以禁用使用doublewrite功能，但可能发生写失效问题。有些文件系统本身就有这项功能，如ZFS就可以不用启动双写了。

自适应哈希索引

HASH是一种非常快的查找方法，一般查找时间复杂为O(1)即只需要查找就能定位数据，而B+树查找次数，取决于B+树的高度，一般为3-4层，所以要3-4次查询。innodb会监控对表上各索引查询，如果观察到建立哈希索引可以带来提升速度，就会建立，所以称为自适合哈希索引(Adaptive Hash Index,AHI)可以在INSERT BUFFER AND ADAPTIVE HASH INDEX里看到

异步IO(AIO)

为了提高磁盘操作性能，当前数据库存都是用的AIO，innodb也是。AIO对应的是Sync IO即每进行一次IO操作，要等此次操作结束后才能接下来继续工作，如发起一条SQL要扫描多个索引页，也就是要进行多次的IO操作，在每扫描一个页并等其完成后再进行下一次的扫描，这是没有必要的，用户可以在发出一个IO请求后再发另一个，当全部IO请求完毕后，等所有IO操作完成，这就是AIO。AIO的一个优势就是可以IO Merge，把多个IO合为1个，这样可以提交IOPS性能。例如用户要访问页的(space-表空间ID唯一的,page_no页偏移量)为(8,6) (8,7) (8,8)每个页大小为16KB，那么同步IO要3次IO操作，而AIO会判断这三个页是连续的，可以根据space,page_no得知，因此AIO底层会发送一个IO请求从(8,6)开始一次读取48KB的页。可以通过iostat里的rrqm/s和wrqm/s得到合并值，在1.1.x之后AIO提供了内核级别支持称为Native AIO,编译时要libaio库支持，以下有一个参数**innodb_use_native_aio**：用来控制是否启用Native AIO，LINUX下默认为开启ON。

刷新邻接页 Flush Neighbor Page

工作原理是，当刷新一个脏页时，innodb会检测该页所在区(extent)的所有页，如果是脏页，会一起刷新，这样的好处就是通过AIO可以将多个IO写入操作合并为一个IO操作，所以工作机制在传统机械磁盘下有显著优势，也要考虑以下：
1 是不是可能将不怎么脏的页进行了写入，而该页之后又会很快变成脏页

2 固态硬盘有较高的IOPS，是否还要这个特性

innodb_flush_neighbors:在innodb从1.2.x开始提供了这个参数来控制是否启用该特性。如果是机械硬盘建义启用它，如果是固态有超高IOPS性能磁盘，可以关闭。

innodb_fast_shutdown：仟可以为0 1 2默认为1 在72页

0 在关闭mysql时innodb要完成所有full purge和merge insert buffer 并刷新所有脏页到disk，有时要几个小时来完成，innodb升级可以为0；

1 是默认值，表示不要上面的full purge和merge操作，但在缓冲池中的一些数据脏页还是会刷新到disk

2 表示不完成full purge和merge insert buffer操作，也不刷新缓冲池的脏数据页到disk，而是将日志写入日志文件，这样不会有任何事务丢失，但是在启动时会进行恢复操作 recovery。

innodb_force_recovery：直接影响整个innodb恢复的状况，默认为0，代表当发生要恢复时，进行所有的恢复操作，还有好多参数在72页。

当设置参数值大于0后，可以对表进行select,create,drop操作,但insert,update或者delete这类操作是不允许的。

1(SRV_FORCE_IGNORE_CORRUPT):忽略检查到的corrupt页。

2(SRV_FORCE_NO_BACKGROUND):阻止主线程的运行，如主线程需要执行full purge操作，会导致crash。

3(SRV_FORCE_NO_TRX_UNDO):不执行事务回滚操作。

4(SRV_FORCE_NO_IBUF_MERGE):不执行插入缓冲的合并操作。

5(SRV_FORCE_NO_UNDO_LOG_SCAN):不查看重做日志，InnoDB存储引擎会将未提交的事务视为已提交。

6(SRV_FORCE_NO_LOG_REDO):不执行前滚的操作。

日志文件

以下参数影响着二进制日志记录的信息和行为：

1 max_binlog_size
2 binlog_cache_size
3 sync_binlog
4 binlog-do-db
5 binlog-ignore-db
6 log-slave-update
7 binlog_format

max_binlog_size：binlog文件的大小，如果超过则新健文件后缀名+1默认为1073741824(1G),事务的表所有未提交的二进制日志会被记录到一个缓存中等该事务提交时直接将缓冲中的二进制日志写入二进制文件，而该缓冲的大小由binlog_cache_size决定，默认为32K。binlog_cache_size是基于会话的，也就是当一个线程开始一个事务时，MYSQL会自动分配一个大小为binlog_cache_size的缓存，因此这个值的设置要相当小心，不能设置太大。当一个事务的记录大于binlog_cache_size时MYSQL会把缓冲中的日志写入一个临时文件中，所以值也不能太小。show status like 'binlog_cache'

查看这几个binlog_cache_use,binlog_cache_disk_use的状态，可以判断当前binlog_cache_size的设置是否合适。binlog_cache_use记录了使用缓冲写二进制日志的次数，binlog_cache_disk_use记录了使用临时文件写二进制日志的次数。

二进制日志默认并不是每次写的时候同步到磁盘，sync_binlog = 1是控制同步写，就是从binlog_cach同步到磁盘的时间，设置为1时在一个事务发出commit之前，由于sync_binlog为1，因此会把二进制日志立即写入磁盘。如果这时已经写入了二进制日志，但提交还没有发生，这事务就会被回滚掉。但是二进制日志已经记录了该事务信

息，不能被回滚。这个问题可以用**innodb_support_xa**=1来解决，虽然它和XA事务有关，但它同时确保了二进制日志和innodb引擎数据文件的同步。

binlog-do-db和binlog-ignore-d：b表示需要写入或忽略写入哪些库的日志。默认为空，表示需要同步所有库的日志到二进制日志里。

log-slave-update：如果是master=>slave=>slave就要开启它，记录自己的二进制日志文件。

binlog_format：设置binlog的格式：STATEMENT ROW MIXED有三个

存储引擎文件

以上都是mysql数据库存本身的文件，和存储引擎无关。

表空间文件(tablespace)这是innodb的存放设计。ibdata1等，可以用**innodb_data_file_path**进行设置。

重做日志文件redu log

ib_logfile0类似，当实例介质失败时它会用上，如掉电innodb会用它恢复为了达到高可用可以设置为mirrored log groups，把不同文件的文件组放在不同磁盘上。redu log是循环写的，第一个文件写满再写第2个这样。以下是影响redu log的属性：

1.innodb_log_file_size：指定了日志文件大小。innodb1.2.x前最大4GB之后为512GB

2.indoob_log_files_in_group：指定重做日志文件组中文件的数量，默认为2。即ib_logfile0 ib_logfile1

3.innodb_mirrored_log_group：指定日志镜像文件组的数量，默认为1，只有一个日志组，没镜像，若磁盘做了高可用方案如磁盘陈列就可以不开启重做日志的镜像功能。

4.innodb_log_group_home_dir：指定了日志文件组所在路径，默认为./，表示在mysql数据库存的数据目录下。

SHOW VARIABLES LIKE 'innodb%log%'可以得到以上几个参数的值。

重做日志文件大小对于innodb引擎的性能有着非常大的影响，如果太大，在恢复时可能需要很长时间，如果太小会导致一个事务的日志要多次切换重做日志文件，还会导致频繁地发生async checkpoint，导致性能抖动，可能在错误日志中看到如下警告信息：

0909 24…. innodb:error:the age of the last checkpoint is …….

这是因为重做日志有一个capacity变量，这个值代表了最后的检查点不能超过这个阀值，如果超了则必须把缓冲池(innodb buffer pool)中脏页列表(flush list)中的部分脏数据页写回磁盘，这时会导致用户线程阻塞。

它和之前二进制的区别是：之前介绍的会记录所有与MYSQL数据库有关的日志记录，包括innodb myisam等其它存储引擎的日志，都是记录的逻辑日志，而重做日志是记录关于每个页(page)更改的物理情况。此外写的时间也不同，二进制日志文件仅在事务提交前进行提交，即只写磁盘一次，不论事务多大。而在事务进行的过程中，却不断有重做日志条目(redo entry)被写到重做日志文件中。redo log写入磁盘是按512个字节，也就是一个扇区大小进行写入，因此可以保证写入必定是成功的，所以重做日志的写入过程中不需要doublewrite。

在前面分析了从日志缓冲写入磁盘上的重做日志是按一定件件进行的。主线程，master thread每秒会将重做日志缓冲写入磁盘的重做日志文件中，不论事务是否提交，另一个触发写磁盘的过程是由innodb_flush_log_at_trx_commit控制，表示在提交commit时，处理重做日志的方式。有效值有0 1 2。

0 代表当提交事务时，不会刷新，而是等主线程每秒的刷新，1和2不同的地方在于：1表示执行commit时将重做日志缓冲同步写到磁盘，即伴有fsync的调用。2表示将重做日志异步写到磁盘，即到文件系统的缓存中，所以不能完全保证在执行commit时肯定

会写入重做日志文件，只是有这个动作发生。当mysql数据库存发生宕机而操作系统级服务器没有发生宕机时，由于此时未写入磁盘的事务日志保存在文件系统缓存中，当恢复时同样能保证数据不丢失。

innodb表的概念

在innodb里如果没有主健会自动选择一个：先看是否有非空的唯一索引如果有就用它(如有三个，会按健索引的第一个，注意只限于单列索引)，如果没有，innodb会用rowid自动创健一个6字节大小的指针。

4.2Innodb逻辑存储结构

表空间，段，区，页，行组成

行溢出，如果有三个列varchar的总长度超过65532也会报错，会存储在页之外。没超出时放在页类型为B-tree node中，但是发生行溢出时数据是放在页类型为Uncompress BLOB(未压缩)页中

分区表

可以SHOW VARIABLES LIKE '%partition%';来查看当前是否支持分区表，使用分区并不能提高数据库性能，只是对某些SQL带来提高，在OLTP中分区要小心，不要一味使用分区。当前MYSQL支持以下分区类型：

RANGE分区：行数据基于属于一个给定连接区间的列值被放入分区。5.5开始支持。(range如果是NULL则放在最左边也就是第一区)

LIST分区：和RANGE分区类型，只是LIST分区面向的是离散的值。也是5.5后。如果LIST分区请允许NULL不会报错如：PARTITION p0 VALUES IN (1,2,NULL),如果不许则insert into tb_name select 1,NULL

HASH分区：根据用户自定义的表达式返回值来进行分区，值不能为负数。

KEY分区：根据MYSQL数据库提供的哈希函数来进行分区。

不论哪种分区，如果表中存在主健或唯一索引时，分区列必须是唯一索引的一个组成部分(不需要整个唯一索引的所有列)。唯一索引可以是允许NULL值。如没有健唯一索引和主健，可以指定任何一列为分区列。

对原表进行分区：

```
alter table b partition by range (id)(
PARTITION p0 VALUES LESS THAN (10),
PARTITION p1 VALUES LESS THAN (20),
PARTITION p3 VALUES LESS THAN(20)
)
```

```
explain partitions select * from b where id >3 and id <=12
```

SELECT * FROM information_schema.PARTITIONS WHERE table_schema='库名' AND table_name='表名'可以查看

innodb_adaptive_hash_index：控制innodb自适应HASH的。禁用或启动此特性，默认为开启在SHOW INNODBSTATUS里的HASH里non-hash searches/s

innodb_lock_wait_timeout：用来控制事务等待的时间。默认为50秒，可以动态修改SET @@innodb_lock_wait_timeout=60;

(重要)innodb_rollback_on_timeout：用来设定是否在等待超时时对进行中的事务进行回滚，默认为OFF，代表不回滚。如果为ON是回滚整个事务，OFF为回滚到上一条语句的操作

innodb在默认情况下不会回滚超时引发的错误导演。其实innodb在大部份情况下都不会对异常进行回滚。列锁例外。

ERROR 1213就是死锁报错

innodb_print_all_deadlocks：默认为OFF，改为ON会记录所有的死锁信息记录到error.log

undo log解释（第320页内幕）

innodb_undo_directory：用于设置rollback segment文件所在的路径。默认是放在表空间的。默认值为“.”。

innodb_undo_logs：用来设置rollback segment的个数，默认为128。如果观察到同回滚日志有关的互斥争用，可以调整这个参数，官方建议先设小，如发现竞争大再调大。

innodb_undo_tablespaces：**独立undo文件**用来设置构成rollback segment文件的数量，这样rollback segment可以较为平均地分布在多个文件中。设置后会在路径innodb_undo_directory看到undo为前缀的文件。这个参数无法后期设定。

事务在undo log segment分配页并写入undo log的过程同样要写入重做日志。当事务提交后innodb会做以下两件事情：

1.奖undo log放入列表中以供之后的purge操作

2.判断undo log所在的页是否可以重用，若可以分配给下个事务使用事务提交后并不能马上删除undo log以及undo log所在的页。这是因为可能还有其他事务需要通过undo log来得到行记录之前的版本。事务提交时将undo log放入一个链表中，是否可以删除undo log和他所在的页由purge线程来判断。如果每个事务分配一个单独的undo页会非常浪费空间，因为事务提交可能并不能马上释放。如某应用的删除和更新操作的TPS(transaction per second)为1000，为每个事务分一个undo页那么一分钟就要1000*60个页，约要1GB惹每秒puerge页的数量20这样对磁盘空间有着相当高的要求，所以innodb要对undo页进行重用。当事务提交时，先将undo log放入链表中，然后判断undo页的使用空间是否小于3/4，若是则表示该undo页可以被重用，之后的新undo log记录在当前undo log的后面。由于存放undo log的列表是以记录进行组织的，而undo页可能存放着不同事务的undo log因此purge操作需要涉及磁盘的离散读取操作，是一个比较慢的过程。可以在show innodb status来看链表中undo log的数量：**History list length 12**当purge操作会减少该值。

如果对事务操作的统计

如果autocommit=0时记录以下，

com_commit：

com_rollback：

autocommit = 1记录以下，

handler_commit：

handler_rollback：

计算TPS（事务处理能力）(com_commit+com_rollback)/time

innodb_additional_mem_pool_size： 20971520/1024/1024=20 参数这个参数用来设置 InnoDB 存储的数据目录信息和其它内部数据结构的内存池大小。应用程序里的表越多，你需要在这里分配越多的内存。对于一个相对稳定的应用，这个参数的大小也是相对稳定的，也没有必要预留非常大的值。如果 InnoDB 用光了这个池内的内存，InnoDB 开始从操作系统分配内存，并且往 MySQL 错误日志写警告信息。默认值是 1MB ，当发现错误日志中已经有相关的警告信息时，就应该适当的增加该参数的大小。推荐此参数至少讵置为 2MB,实际上,是雹要根掘顷目乩 InnoDB 表乩数目相应地增加