Mysql优化（出自官方文档） - 第十篇（优化InnoDB表篇）

秋风五丈原 2022-11-12 原文

Mysql优化（出自官方文档） - 第十篇（优化InnoDB表篇）

目录

2 Optimizing InnoDB Transaction Management

3 Optimizing InnoDB Transaction Management

4 Optimizing InnoDB Redo Logging

5 Bulk Data Loading for InnoDB Tables

6 Optimizing InnoDB Queries

7 Optimizing InnoDB DDL Operations

8 Optimizing InnoDB Disk I/O

9 Optimizing InnoDB Configuration Variables

10 Optimizing InnoDB for Systems with Many Tables

1 Optimizing Storage Layout for InnoDB Tables

如果一个表的大小已经变得非常大了（M级别），那么使用OPTIMIZE TABLE对表进行重组织和合并浪费的空间，该命令会拷贝表中的部分数据和重建索引。

在InnoDB中，如果PRIMARY KEY很长，会浪费非常多的空间，因为primary key会在每一个二级索引中拷贝一次，这个时候，请考虑创建一个AUTO_INCREMENT列作为primary key或者使用VARCHAR的一部分作为primary key。

使用VARCHAR来代替CHAR类型列，因为VARCHAR占用更小的空间，CHAR(N)类型占用空间最少为N个字符，无论长度小于N或者为NULL。

如果一个表中存储着大量的数字或者重复文本，考虑使用COMPRESSED格式。

2 Optimizing InnoDB Transaction Management

如果一个事务只包含SELECT语句，那么，打开AUTOCOMMIT能够协助InnoDB识别**只读事务**从而优化它们。

如果有一个很大的事务，包含巨量的数据更改删除操作，此时如果发生回滚，将会导致Mysql性能非常的糟糕，所以应该尽量避免回滚的发生。如果无法避免，可采取下面的措施来加快回滚操作速度：

增加buffer pool的大小，这样子数据操作就可以缓存在内存中，而不需要频繁写入磁盘，从而减少磁盘I/O

设置innodb_change_buffering=all ，这样子除了insert操作外，update和delete操作也可以被缓存

定时COMMIT，或者拆分大的事务。

经过上面的操作，可以让rollback操作变为CPU-bound型操作，能够大幅度加快处理速度。

如果一个长事务在修改表，那么该表上面的其他事务将无法使用covering index 技术，只能使用二级索引来获取对应的列。

如果一个二级索引的PAGE_MAX_TRX_ID较新，或者一个二级索引对应的记录被标记为DELETED，InnoDB可能使用聚集索引来查找对应的记录。

3 Optimizing InnoDB Transaction Management

InnoDB会避免为只读事务创建transaction ID(TRX_ID field)，因为一个transaction ID只有当进行写操作，或者类似于SELECT ... FOR UPDATE这样的语句时才会有用。对于只读事务，消除transaction ID能够减少内部数据结构的大小。

InnoDB是通过下面的方式来识别只读事务的：

一个事务以 START TRANSACTION READ ONLY开头，在这样的事务下，任何更改数据库的操作都会抛出错误。

autocommit被打开，此时，一个单独的语句都会被视为一个事务，所以对于那些不带FRO UPDATE或者LOCK IN SHARED MODE的SELECT语句，将会被视为只读事务。

事务没有READ ONLY，但是同样的没有UPDATE或者lock rows的操作，此时，InnoDB会视该事务为只读，后面一旦有了UPDATE或者lock rows操作，那么，该事务将不再处于只读状态。

4 Optimizing InnoDB Redo Logging

配置redo log的文件大小和buffer pool一样大，虽然较大的redo log可能会导致较长时间的recovery，但是目前的recovery速度非常快，所以可以配置较大的redo log。通过 innodb_log_file_size and innodb_log_files_in_group 两个参数可以进行对应的配置。

考虑增大 log buffer， 较大的log buffer可以避免事务在commit进行日志的磁盘I/O。通过 innodb_log_buffer_size可进行配置。

配置合适的 innodb_log_write_ahead_size的值可以避免文件的"read-on-write"，配置的过小，可能会导致操作系统或者文件系统频繁的进行"read-on-write"，配置过大可能会影响fsync的效率，因为会导致过多的磁盘block读写。尽量配置该值和操作系统，文件系统的cache block size相匹配。

优化用户线程等待flushed redo的spin delay，在高并发场景下将会特别有用，通过下面几个变量可以优化spin delay，这里不再进行赘述。

innodb_log_wait_for_flush_spin_hwm，innodb_log_spin_cpu_abs_lwm和innodb_log_spin_cpu_pct_hwm

5 Bulk Data Loading for InnoDB Tables

这些技巧只是对INSERT技巧的一个简单补充：

当导入数据到InnoDB中时，关闭autocommit，因为autocommit会导致每一个insert都进行flush log操作。

如果在二级索引上面有UNIQUE约束，那么可以临时关闭uniqueness checks，这样子可以省略Mysql检查unique的开销，但是**必须要自己保证数据中没有重复key**。

```
SET unique_checks=0;
... SQL import statements ...
SET unique_checks=1;
```

如果表上面有外键约束，那么可以临时关闭外键检查来加快速度，对于大表来讲，会减少大量的磁盘I/O，但是同时需要保证外键的正确性。

```
SET foreign_key_checks=0;
... SQL import statements ...
SET foreign_key_checks=1;
```

使用INSERT插入多行的语法，可以减少server和client之间的网络开销。

如果表中有AUTO-INCREMENT列，设置 innodb_autoinc_lock_mode为2（interleaved）（默认是1（consecutive））

对于批量插入操作，按照PRIMARY KEY顺序进行插入的速度要更快

当loading数据到一个带有FULLTEXT索引的表中，采用下面的步骤可以优化插入下利率：

在创表的时候，创建一个BIGINT UNSIGNED NOT NULL，名为FTS_DOC_ID的列，并且在上面创建一个FTS_DOC_ID_INDEX的唯一索引，举例如下：

```
CREATE TABLE t1 (
FTS_DOC_ID BIGINT unsigned NOT NULL AUTO_INCREMENT,
title varchar(255) NOT NULL DEFAULT '',
text mediumtext NOT NULL,
PRIMARY KEY (`FTS_DOC_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
CREATE UNIQUE INDEX FTS_DOC_ID_INDEX on t1(FTS_DOC_ID);
```

loading数据到表中

当数据load完成之后，创建索引。

> NOTE:
>
> 当创建FTS_DOC_ID列的时候，确保当FULLTEXT索引列被更新的时候，FTS_DOC_ID也会被更新。如果不创建该列，那么InnoDB会隐式的管理 DOC ID，InnoDB会在调用 CREATE FULLTEXT INDEX的时候添加一个FTS_DOC_ID隐藏列，因为执行该操作，需要重建表，此时会较大的影响性能。

6 Optimizing InnoDB Queries

因为InnoDB无论如何都会创建primary key，所以显式选择一些会被高频访问的列作为primary key

不要创建包含过多或者过长列作为primary key，primary key会在每个二级索引中复制一遍。

如果某列不可能为NULL，那么声明为 NOT NULL

7 Optimizing InnoDB DDL Operations

当loading数据的时候，如果没有二级索引，速度将会大大提升，因此，尽量在loading完数据后再创建二级索引。

如果需要删除表中的所有数据，使用TRUNCATE TABLE 来取代DELETE FROM *tbl_name*。同理， DROP TABLE and CREATE TABLE的效率也会比普通的DELETE FROM要高很多。

InnoDB表是由primary key进行组织的，所以修改primary key的定义会导致整张表都被重新组织，开销较大，因此，尽量不要去修改primary key的定义，在创建表的时候就指定好。

8 Optimizing InnoDB Disk I/O

如果发现CPU的利用率不足70%，那么很有可能是因为磁盘到了瓶颈，可以采取下面的优化方式：

增加buffer pool的大小，一般配置为系统内存的50%-75%

修改flush函数，InnoDB默认为fsync，但是在有些系统上这种类似的函数会非常慢，因此通过 innodb_flush_method 进行相应的调整。

为write buffer设置一个threshold size，通过设置 innodb_fsync_threshold 参数可以配置write buffer在多大的时候才进行flush操作，可以减少一定的磁盘I/O

在linux AIO上，使用noop 或者 deadline I/O调度方式，通过innodb_use_native_aio可进行对应的配置。

在Solaris 10上面的x86_64架构使用direct I/O

在Solaris 2.6或者之后的版本，使用row storage来存储数据和日志文件

考虑使用非旋转存储介质，比如ssd这种，非旋转存储设备的随机读写性能要远高于传统存储设备，对于Mysql，通常来讲，面向随机I/O的文件包括： file-per-table and general tablespace data files, undo tablespace files, and temporary tablespace files，而面向顺序写I/O的文件包括：InnoDB system tablespace files (due to doublewrite buffering and change buffering) and log files such as binary log files and redo log files。

通过配置下面的参数可以提高非旋转存储介质的效率：

innodb_checksum_algorithm， innodb_flush_neighbors， innodb_io_capacity， innodb_io_capacity_max， innodb_log_compressed_pages， innodb_log_file_size， innodb_page_size and binlog_row_image

增加I/O capacity从而避免backlogs，如果性能出现较大范围的波动（因为InnoDB进行checkpoint的缘故），考虑增加 innodb_io_capacity 参数，该值越大，flush的频率也会越高，就可以避免较多的backlog操作。

如果flushing没有落后的话，那么尝试适当降低I/O capacity， 但是不要过低，过低的I/O capacity会导致性能抖动。

将系统tablespace files存储在 Fusion-io 设备上，因为这种设备支持原子写操作，因此，当使用这种设备的时候，doublewrite buffering (innodb_doublewrite) 会自动禁用以提高性能。

禁用compressed pages的日志，通过innodb_log_compressed_pages系统变量进行控制，默认情况是启用的，这是为了防止在恢复的时候，使用了不同的zlib算法，会导致mysql挂掉，如果你确定不会使用不同的zlib算法，那么可以禁用该项来提高效率。

9 Optimizing InnoDB Configuration Variables

本小节主要提供一些优化InnoDB效率的手段，由于条数较多，且有很多在上文已经提到过，所以这里仅挑选一些重要的条目列出来:

对于InnoDB的线程数进行一个限制，过高的线程并发可能会导致上下文切换开销较大。

对于read-ahead操作，控制prefetching的数量，因为过多的read-ahead会导致性能抖动，详情可以看InnoDB的Buffer Pool Prefetching技术。

控制InnoDB后台的I/O量，过多的后台I/O会导致性能抖动，主要通过master thread来配置。

调整后台写的算法，不同的算法适应不同的场景，详情可通过Buffer Pool Flushing来配置。

尽可能的利用多核处理器和其缓存内存的优势，这样子可以减小上下文的时延，详情可通过配置Spin Lock Polling来实现。

避免单词操作如table scan，对buffer pool中高频访问数据的干扰，可通过配置Buffer Pool的一些淘汰算法来实现。

在以前版本的Mysql（指Mysql.5.5之前）中，为了避免系统因为恢复启动时间过长，通常会将log的大小设置的较小。在现在的版本中，通过redo log的恢复流程被大大优化，现在可以考虑适当增大redo log的大小了，因为这样子可以减少I/O

对于拥有较大内存的机器，调整buffer pool的实例数和大小；增加事务并发的最大数量，可以大幅度的提升业务量大的数据库的拓展性，在Undo logs的介绍中有介绍原因。

将purge线程设置为后台线程。

为了避免频繁的线程上下文切换，在现代机器上，可将 innodb_thread_concurrency配置到32，innodb_concurrency_tickets 配置5000，通过这两个参数，每个线程就可以在被换出前做尽可能多的事情。

10 Optimizing InnoDB for Systems with Many Tables

如果配置了 non-persistent optimizer statistics（非默认设置），InnoDB会在启动后且当该表第一次被访问的时候计算index [cardinality](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_cardinality)的值，这个过程非常耗时，但是只有第一次open table的时候才会做，后面的访问将不会做这样的事情，为了提前给table“热身”，可以使用SELECT 1 FROM *tbl_name* LIMIT 1这样的语句来触发。

可通过innodb_stats_persistent 参数来配置优化数据是否需要持久化。

Mysql优化（出自官方文档） - 第十篇（优化InnoDB表篇）的更多相关文章

1. Mysql优化（出自官方文档） - 第十二篇（优化锁操作篇）
   Mysql优化(出自官方文档) - 第十二篇(优化锁操作篇) 目录 Mysql优化(出自官方文档) - 第十二篇(优化锁操作篇) 1 Internal Locking Methods Row-Leve ...
2. Mysql优化（出自官方文档） - 第三篇