MySQL 5.6新特性深入剖析

——InnoDB引擎

何登成
微博：@何_登成
网站：深入MySQL内核

Outline

- **MySQL 5.6简介**
- **MySQL 5.6新特性**
  - **InnoDB层新特性**
    - 性能优化
    - 功能增强
  - **Server层新特性**
    - 性能优化
    - 功能增强

MySQL 5.6简介

- 简介
  - MySQL 5.6版本，为MySQL最新的一个大版本，相对于MySQL 5.1/5.5，无论是MySQL Server层面，还是InnoDB Engine层面，都做了大量的改进(性能改进 vs 功能增强)。这些改进，无论是DBA，亦或是研发人员，都值得好好的学习、深入了解；

- 版本发布情况
  - MySQL 5.6.2(2011-04-11)： 第一个发布版本
  - MySQL 5.6.7(2012-09-29)： 第一个RC版本
  - MySQL 5.6.10(2013-02-05)： 第一个GA版本(**本PPT使用版本**)
  - MySQL 5.6.12(Not Released)： 最新研发版本
  - 详见：MySQL 5.6 Release Notes

MySQL 5.6简介—改进总览

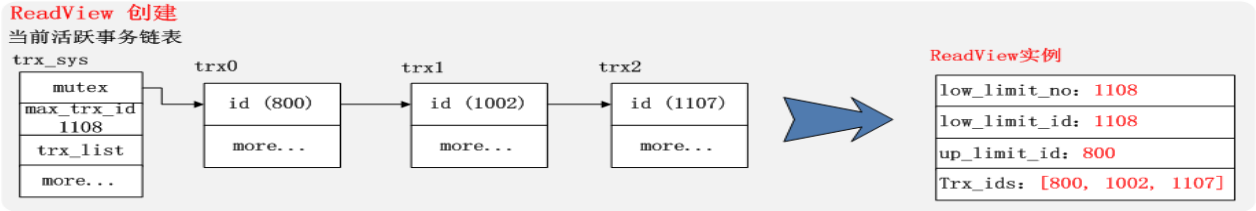
- **InnoDB Engine (本期内容)**
  - 性能
    - Read-Only Transaction, Buffer Pool Flushing, Page Cleaner, Purge, CRC32, SSD ...
  - 功能
    - Online DDL, Memcached Plugin, Transportable Tablespace, Buffer Pool Dump/Restore, FTS ...
- **MySQL Server (下期内容)**
  - **Optimizer**
    - Semi-Join, BKA, MRR, ICP, Join, In, Optimizer Tracing, Limit ...
  - **Replication**
    - GTID, Binlog Group Commit, Multi-Threaded Slaves ...
  - **Others**
    - Security ...

InnoDB—性能优化(总)

- **InnoDB性能优化**
  - Read-Only Transactions
  - Buffer Pool Flushing
  - Page Cleaner
  - Purge
  - CRC32
  - Compression
  - Data Dictionary LRU
  - Others
    - ssd, mutex, spinlock, memory allocation, read ahead, undo log tablespace...

InnoDB-Read Only Transaction

- **Read-Only Transactions**(双层优化)
  - 第一层
    - 瓶颈
      - 快照读操作，需要创建ReadView：获取trx_sys->mutex后遍历活跃事务链表，存在并发性能瓶颈；
      - InnoDB的MVCC策略(参考此处)；



    - 分析
      - 只读事务不会产生更新，也就不会产生历史版本；OLTP应用，读多写少；
      - 将活跃事务链表拆分为只读事务与更新事务两个链表，ReadView创建只需要遍历更新事务链表，能够极大的降低ReadView创建的开销；
    - 优化
      - 新增SQL语法：　start transaction read only;
      - 事务上新增标识： trx->read_only
      - 维护两个活跃事务链表：　ro_trx_list vs rw_trx_list
      - 注：　Autocommit = 1时，快照读一定是Read-Only事务

InnoDB-Read Only Transaction

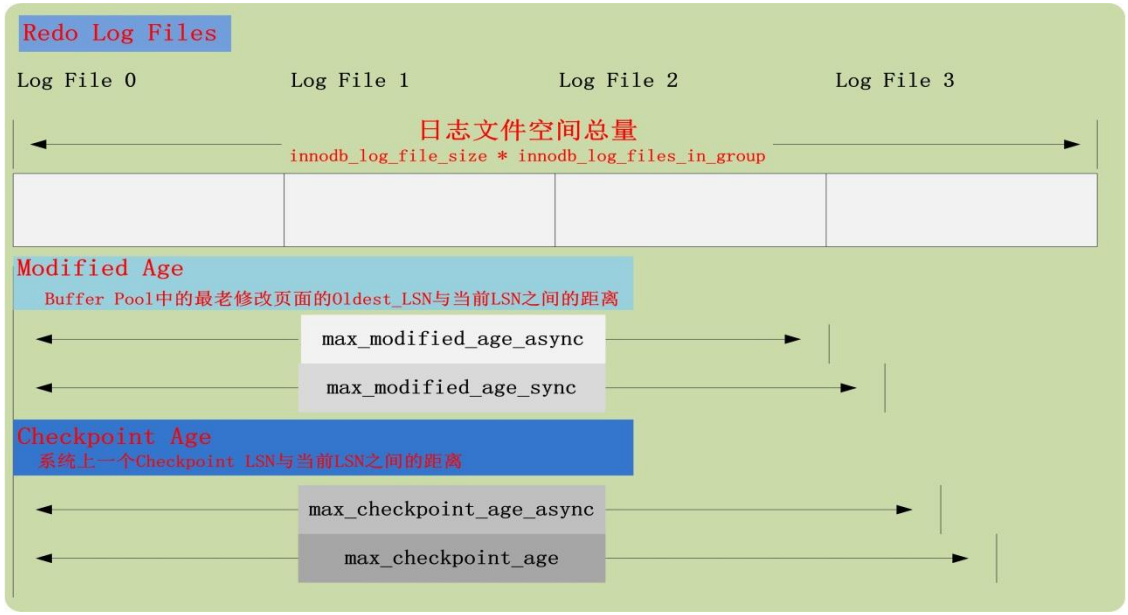
- **Read-Only Transactions**(双层优化)
  - 第二层
    - 瓶颈
      - 将事务划分为只读/更新事务之后，InnoDB系统的并发效率并未有明显提升；
    - 分析
      - 在row0sel.cc::row_search_for_mysql()函数中，每读取一条记录，都会增加一个count计数；
      - 多线程并发修改此count(srv_n_rows_read)，就会导致cache coherence问题；
    - 优化
      - 重新设计计数器：N个计数对象，按照**CACHE_LINE_SIZE**对齐；
      - 每个事务，根据事务ID映射到不同的计数对象上，进行统计，减少碰撞；
      - Type m_counter[(**N** + 1) * (**CACHE_LINE_SIZE** / sizeof(Type))];

InnoDB-Buffer Pool Flushing

- **Buffer Pool Flushing(Flush List Flush)**
  - InnoDB的Fuzzy Checkpoint策略，按照内部脏页链表(Flush List)，逐步将最老的一部分脏页写出磁盘，推进系统的检查点(Checkpoing LSN)；(参考此文)
  - 存在的问题
    - InnoDB原生Flushing算法不够稳定(Percona提出了3种改进策略)
  - 新的算法
    - 系统平均刷脏页速度： avg_page_rate = ½过去**innodb_flushing_avg_loops**秒 + ¼ + ... 的速度
    - 系统平均日志速度： lsn_avg_rate
    - 根据系统脏页比率与日志年龄，计算本次应该Flush的脏页数量：npages
    - 根据平均刷脏页速度进行调整： npages = (npages + avg_page_rate) / 2
    - 根据lsn_avg_rate，计算本次日志应该Flush到的位置：lsn_limit
    - 最后，根据**npages**与**lsn_limit**，进行本次Flush；
  - 新引入的参数
    - innodb_adaptive_flushing_lwm
    - innodb_max_dirty_pages_pct_lwm
    - innodb_max_io_capacity
    - innodb_flushing_avg_loops

InnoDB-Page Cleaner

- **两种Flush策略**
  - LRU List Flush
    - 写出LRU链表尾部的脏页，释放足够的页面，以满足前端用户的需求；
    - 原由用户线程触发，用户线程处理；
  - Flush List Flush：
    - 将系统中最老的部分脏页写出，推进系统的检查点(Checkpoint LSN)；
    - 根据Checkpoint Age的不同，由不同的线程处理(Master Thread vs User Thread)；



InnoDB-Page Cleaner

- **Page Cleaner流程**
  - 将LRU List Flush与Flush List Flush全部移到Page Cleaner后台线程中处理，减少Master Thread与User Thread的压力；
  - Page Cleaner线程，每秒启动一次；
  - LRU List Flush
    - 从LRU链表尾部开始遍历：将未使用的Clean Page从LRU链表摘除；将未使用的Dirty Page写出，然后从LRU链表摘除；
    - 外部参数：innodb_lru_scan_depth (控制LRU链表尾部遍历的长度)；
    - 内部参数：PAGE_CLEANER_LRU_BATCH_CHUNK_SIZE (默认100：控制一个处理批次的大小，防止长时间持有buffer pool mutex，导致系统出现并发瓶颈)；
  - Flush List Flush
    - 使用前页中介绍的New Adaptive Flush算法；

InnoDB-Purge Thread

- **Purge Thread**
  - Purge操作：InnoDB读取提交事务的Undo记录，然后将事务更新所产生的历史版本(标记为删除，并且对所有活跃事务不可见的版本)从数据文件(聚簇索引、辅助索引)中删除的操作；
  - MySQL 5.1，Purge操作在InnoDB Master Thread中完成；MySQL 5.5，一个Purge后台线程；
  - 存在的问题
    - Purge操作不及时，导致Undo空间膨胀；数据文件中存在大量历史无用记录；
  - Multi-Purge Threads
    - MySQL 5.6，多个Purge线程，并发回收历史版本；
    - 新增参数：
    - innodb_purge_threads　　默认1，取值[1, 32]
    - innodb_purge_batch_size　　默认300
    - **注意1**： innodb_purge_threads只是规定了Purge线程的上限，InnoDB会根据事务负载自动调节(详见srv0srv.cc::srv_do_purge()函数)；
    - **注意2**：innodb_purge_batch_size可以理解为每次Purge，回收的提交事务数量；

InnoDB-CRC32

- **Page Checksum**
  - InnoDB在其页面的头部与尾部，维护了两个Checksum(详见InnoDB Page Structure)，通过页面的内容计算而来，用于校验页面内容是否被破坏；
  - 脏页从内存写出时，需要重新计算Checksum (buf0flu.cc::buf_flush_init_for_writing())；
  - 页面从外存读取进内存时，需要计算页面Checksum，判断其与存储的Checksum是否相同，进而验证页面是否损坏 (buf0buf.cc::buf_page_is_corrupted())；
- **原有Checksum算法**
  - 软件计算(buf0checksum.cc::buf_calc_page_new_checksum())，逐个读取4字节int，然后做^运算；
  - 一个InnoDB Page，默认为16K，软件计算Checksum，性能低下；
- **CRC32 Checksum**
  - 通过CPU指令(SSE4.2)计算CRC32 Checksum，提升Checksum的计算性能 (ut0crc32.cc::ut_crc32_sse42())
  - 新增参数：innodb_checksum_algoritm

InnoDB-Compression

- **Compression**
  - InnoDB以Page为单位进行压缩，采用zlib压缩工具；
- **优化措施**
  - 新增参数
    - innodb_compression_level
      - 控制zlib压缩级别[1..9]；
    - innodb_compression_failure_threshold_pct
      - InnoDB的压缩页面经过更新之后，再次压缩可能会导致压缩失败，需要分裂；
      - 此参数控制压缩失败的比率，超过此比率，压缩前的页面需要进行Padding；
      - 所谓**Padding**，就是在16K的页内填充一些无效内容，降低页面利用率，保证压缩成功率；
    - innodb_compression_pad_pct_max
      - 非压缩页Padding的大小；
      - 默认：50%

InnoDB-Data Dictionary LRU

- **Data Dictionary**
  - 每一个用户表，在InnoDB的系统表(SYS_TABLES, SYS_COLUMNS, SYS_INDEXES, ...)中都存储着一些元数据信息；
  - 当前端SQL操作用户表时，此表的元数据信息会被读取出来，存放于InnoDB Data Dictionary Cache之中；
- **原有问题**
  - Data Dictionary Cache，会占用大量的内存
    - 表打开时，会将元数据加载入Dictionary Cache，但是表关闭时，并不会从Dictionary Cache中删除元数据。大量表的情况下，Dictionary Cache会占用大量内存(详见此文)；
- **改进措施**
  - 将Dictionary Cache中的所有表元数据，维护为一个LRU链表；
  - 借用MySQL Server层的参数：**table_definition_cache** (默认400，软限制)
  - 后台Master Thread，每**SRV_MASTER_DICT_LRU_INTERVAL**(47)秒，遍历一次Dictionary Cache，清除LRU链表尾部不使用(ref count = 0)的表(保证数量小于**table_definition_cache**即可)；

InnoDB-Others

- **SSD**
  - Compression优化；4K、8K Page支持；Undo log tablespace；...
- **Mutex**
  - CAS原子指令
- **Spinlock**
  - 根据innodb_sync_spin_loops参数，InnoDB Spinlock在无法获取锁时，会反复重试innodb_sync_spin_loops次；
  - 改进
    - 重试前，根据innodb_spin_wait_delay参数，Relax CPU的使用，通过PAUSE指令实现；
    - 关于PAUSE指令，可参考此文；
- **memory allocation**
  - 除了Buffer Pool之外，InnoDB需要使用一些其他的内存，原由内部实现内存的分配与释放；
  - 改进
    - 直接使用操作系统提供的tcmalloc，ptmalloc等更为高效的内存分配方法；

InnoDB-Others

- **File Extension**
  - 5.5中，数据文件扩展时，需要锁住全局的file system mutex，数据文件扩展串行化，并且会堵塞前端用户操作；
  - 改进
    - 每个扩展中的文件，新增一个标识，无需长时间持有file system mutex，问题解决；
- **read ahead**
  - **linear read ahead**
    - 每次从外存成功读取一个page之后，都判断是否需要进行linear read ahead(buf0rea.cc::buf_read_ahead_linear())；
    - linear判断方法：遍历page所属extent，判断后一个page的访问时间是否大于前一个page；
    - 控制参数：innodb_read_ahead_threshold (默认56)
  - **random read ahead**
    - 若page所属extent中，超过**BUF_READ_AHEAD_RANDOM_THRESHOLD(13)**数量的页面均在LRU链表的热端，并且参数innodb_random_read_ahead(默认关闭)开启，则预读extent中的所有其他Pages；

InnoDB—功能增强(总)

- **InnoDB功能增强**
  - Online DDL
  - Memcached Plugin
  - Transportable Tablespace
  - Buffer Pool Dump/Restore
  - Persistent Statistics
  - FullText Search(略)

InnoDB-Online DDL

- **DDL发展历程**
  - Copy Table
    - MySQL最早的DDL操作方式，DDL通过Copy Table方式实现：
      - 新建Temp Table；
      - 锁原表，原表可读不可写；
      - 将原表数据Copy到Temp表；
      - 删除原表，重命名Temp表，解锁；
    - 缺点：并发低；两倍存储空间；
  - Inplace
    - 直接在原表上进行DDL(Add/Drop Index ...)，锁表进行；
    - 缺点：并发低；
  - Online
    - DDL操作过程中不长时间锁表，并发操作可读可写，提供高并发；
    - 两种方式
      - Inplace Online DDL：　　Add/Drop Index, ...
      - Copy Table Online DDL：　　Add/Drop Column, ...

InnoDB-Online DDL

InnoDB Inplace Online DDL处理流程

STEP 1	检查InnoDB引擎是否支持当前DDL的Inplace Online操作？Add Index是支持的； (handler0alter.cc::check_if_supported_inplace_alter())
STEP 2	创建新索引(标识 ONLINE_INDEX_CREATION)；为新索引创建Row Log(Block组织方式)； (handler0alter.cc::prepare_inplace_alter_table())
STEP 3	读取聚簇索引最新项，构造新索引项，排序并插入新索引；中间过程中，表可读可写， 用户DML操作产生的I/U/D记录，存储在Row Log之中； (handler0alter.cc::inplace_alter_table())
STEP 4	重放Row Log中的操作至新索引上，重放Row Log中间的Block过程不加锁，用户DML操作 产生新的Row Log，Append到Row Log最后的一个Block中； 当前Block为Row Log最后一个Block，则锁住索引树，禁止读写，重放最后一个Block； (handler0alter.cc::inplace_alter_table())
STEP 5	完成最后的收尾工作，Inplace Online DDL操作完成； (handler0alter.cc::commit_inplace_alter_table())

InnoDB-Online DDL

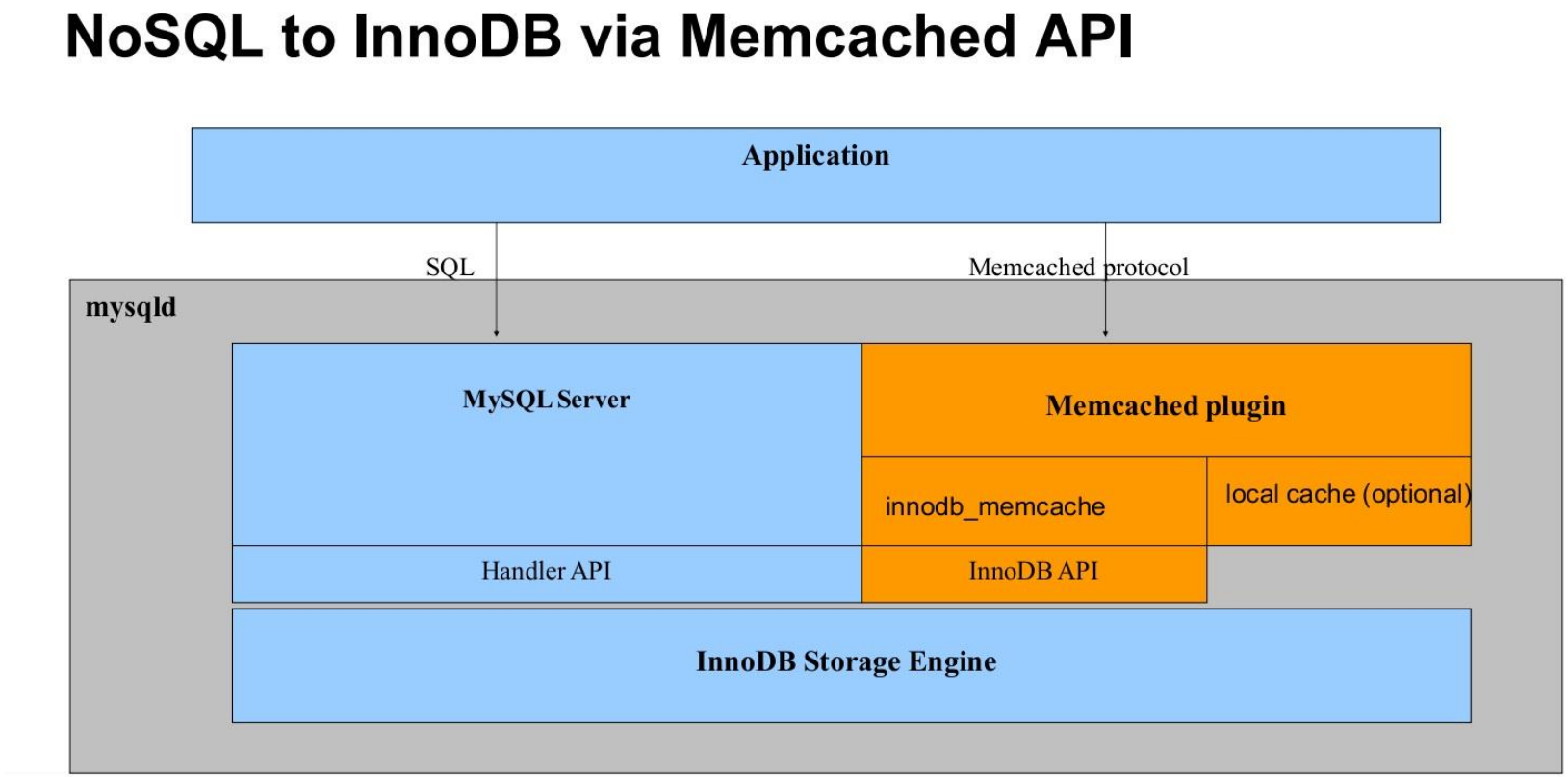
- **注意事项**
  - **是否支持Online Add Unique Index？**
    - 支持；
    - 在创建过程，以及Row Log回放过程中，都会进行Unique约束检查；
  - **Online操作，新的索引缺乏版本信息，如何处理？**
    - **问题**：读取最新记录构建新索引，因此新索引上缺乏版本信息；
    - **解决**：索引字典上，新增一个trx_id属性，标识索引创建过程中系统的最大事务ID，所有小于此trx_id值的事务，均不可使用新索引；
  - **Online操作性能，是否可以优化？**
    - 可通过增加**innodb_sort_buffer_size**参数，优化Online (Add Index/Column)操作性能；
    - 创建索引，排序过程，使用内存大小为**innodb_sort_buffer_size**的3倍；
    - Row Log Block大小，等于**innodb_sort_buffer_size** ；

InnoDB-Online DDL

- **Copy Table Online DDL流程如何？**
  - Add/Drop Column，无法进行Inplace Online DDL，需要创建临时表 (**handler0alter.cc::prepare_inplace_alter_table()**)；
  - 原表聚簇索引，创建Row Log (**handler0alter.cc::prepare_inplace_alter_table_dict()**)；
  - 读取原表记录，构建新表聚簇索引/辅助索引记录，排序并顺序插入；
  - 重放原表聚簇索引上的Row Log到新表之上；(同样，重放Row Log中间Block时不锁原表，重放最后一个Row Log Block时，锁住原表，禁止更新操作)
  - 删除原表，将临时表重命名为原表，更新部分持久化统计信息，在线加列操作完成 (**handler0alter.cc::commit_inplace_alter_table()**)；

InnoDB-Memcached Plugin

- Memcached Plugin
  - MySQL 5.6，对外提供了通过Memcached接口直接访问InnoDB引擎中的记录的方式；



InnoDB-Memcached Plugin

InnoDB Memcached Plugin简析

ONE	InnoDB层面提供一批直接操作InnoDB内部方法的接口，包括：Scan / Insert / Delete/ Update/ Transaction ... (api0api.cc)
TWO	Memcached，作为MySQL的一个Plugin，在初始化时，获取InnoDB提供的所有接口；
THREE	InnoDB内部，维护一个系统表：Container Table；Container Table中存储的是InnoDB Table中的各个列，与Memcached接口中的【Key,Value】间的对应关系； 例如：CONTAINER_KEY(c1)、CONTAINER_VALUE(c2,c3,c4)
FOUR	用户通过Memcached接口发起一个请求，Memcached Plugin根据接口的映射关系，将操作转换为InnoDB内部接口方法，完成操作之后返回用户；
FIVE	默认(META_CACHE_OPT_INNODB)直接操作InnoDB的Buffer Pool，没有自己的Cache； 也可以通过参数调整Cache策略，使得Memcached Plugin也有自己的Cache；
SIX	通过Memcached Plugin，支持事务操作，支持多版本读取，支持Binlog；

InnoDB-Memcached Plugin

- **注意事项**
  - **数据缓存策略**
    - 在innodb_memcache.cache_policies表中进行设置；
    - **innodb_only** (Only InnoDB BP)， **cache_only** (Only Memcached)， **caching** (Both)；
    - 可单独设置每个操作：set，get，delete，…
  - **事务操作策略**
    - **daemon_memcached_r_batch_size** vs **daemon_memcached_w_batch_size**; 默认(1)
    - 同一连接：多少次read创建一个事务 vs 多少次write提交一次事务；
    - **innodb_api_bk_commit_interval**; 默认(5 S)
    - Memcached Plugin后台线程，5S定期清理Idle连接；
  - Others
    - 参考此文；
  - **总结**
    - 较为难用；与SQL (DDL/DML)相互并发，存在一定的约束，较难理解；

InnoDB-Transportable Tablespaces

- **Transportable Tablespaces**
  - Transportable Tablespaces的功能，就是将一个表数据文件从当前数据库拷贝出去，然后导入到另外一个数据库之中；
- 原有约束
  - MySQL 5.6之前，无法通过拷贝数据文件的方式实现数据的转移；
    - 数据文件ibd中不包含最新纪录；
    - 数据文件的日志信息与其他数据库不符；
    - Purge与Change Buffer存在影响；
    - …
- 改进
  - 通过提供新的命令，使得Transportable Tablespaces成为可能；
    - 注意：innodb_file_per_table参数必须开启；

InnoDB-Transportable Tablespaces

InnoDB-Transportable Tablespaces

- **Discard & Import Tablespace主要流程**
  - **Discard Tablespace**
    - Discard过程，上层并发由MDL锁保护；底层需要等待后台操作结束；
    - 删除Change Buffer中所有相关的缓存项；
    - 设置表元数据信息，标识Tablespace删除状态：
    - 重新生成表的ID，保证所有基于表ID的操作后续均会失败(Purge)；
    - 删除数据文件，Discard成功；
  - **Import Tablespace**
    - 读取cfg文件：表定义；索引定义；索引Root Page；列定义；...
    - 读取Import文件的每一个Page，检查完整性；
    - 根据读取的CFG文件，重新设置当前表的元数据信息；

InnoDB-BP Dump/Restore

- **Buffer Pool Dump/Restore**
  - **Dump**：　将InnoDB Buffer Pool中的页面标识Dump到外存；
  - **Restore**：　读取Dump文件，根据其中保存的页面标识，读取对应的页面填充Buffer Pool；
- **功能优势**
  - **原有问题**
    - 重启MySQL服务器，InnoDB的缓存预热，一直是系统较大的瓶颈。在预热过程中，I/O压力过大(随机I/O)，影响用户的使用；
  - **解决方案**
    - 通过BP Dump/Restore，Dump过程，将内存页面标识写出；Restore过程，读取Dump文件，将页面标识排序，顺序读取外存页面进入Buffer Pool；将预热的随机I/O转换为顺序I/O；

InnoDB-BP Dump/Restore

- **Dump/Restore处理流程**
  - **Dump流程**
    - 遍历所有的Buffer Pool Instances；
    - 获取当前Buffer Pool Instance的Mutex(排他)，遍历LRU链表，将LRU链表中的每一个页面标识记录下来；
      - 页面标识：【**space_id，page_no**】
    - 释放Buffer Pool Mutex；
    - 将所有的页面标识写出到Dump文件，Dump完成；
    - **注意**
      - 由于Dump需要长时间持有Buffer Pool Mutex，因此会影响前端应用，低峰期进行；
  - **Restore流程**
    - 读取Dump文件，获取所有页面标识；
    - 对页面标识进行排序(**保证顺序I/O**)，然后顺序读取页面标识对应的数据页面；

InnoDB-Persistent Statistics

- **Persistent Statistics**
  - 统计信息持久化；

- **原有问题**
  - **InnoDB中的统计信息是不持久化的，在以下情况下会更新**
    - 表打开时；
    - 表中的大量数据被修改时；(> 2 000 000 000) or > (stat_n_rows/16)
    - Analyze Table；
    - Show table/index Status；
  - **统计信息精准度不够**
    - 随机采集8个页面，估算全表的统计信息；

- **Persistent Statistics优势**
  - 更为精准的统计信息；
  - 更为固定的统计信息；

InnoDB-Persistent Statistics

- **持久化哪些统计信息？(参考这个)**
  - **Table**
    - n_rows　　表记录数量
    - clustered_index_size　　聚簇索引大小
    - sum_of_other_index_sizes　　其他索引总大小
  - **Index**
    - number of index pages　　索引大小
    - number of index leaf pages　　索引叶页面数量
    - n_diff[]　　索引前缀组合不同取值数量
- **持久化统计信息存储在哪？**
  - mysql.innodb_table_stats;
  - mysql.innodb_index_stats;
- **持久化统计信息如何修改？**
  - Analyze Table;
  - 直接修改统计信息表；(可通过此操纵执行计划：见下页实例)

InnoDB-Persistent Statistics

```
mysql> select * from mysql.innodb_table_stats;
+---------------+------------+---------------------+--------+----------------------+---------------------------+
| database_name | table_name | last_update         | n_rows | clustered_index_size | sum_of_other_index_sizes |
+---------------+------------+---------------------+--------+----------------------+---------------------------+
| test          | t1         | 2013-02-22 20:07:15 |  33258 |                   97 |                        62 |
| test          | t2         | 2013-02-22 20:07:21 |      1 |                    1 |                         0 |
+---------------+------------+---------------------+--------+----------------------+---------------------------+
2 rows in set (0.00 sec)

mysql> select * from mysql.innodb_index_stats;
+---------------+------------+------------+---------------------+--------------+------------+-------------+-----------------------------------+
| database_name | table_name | index_name | last_update         | stat_name    | stat_value | sample_size | stat_description                  |
+---------------+------------+------------+---------------------+--------------+------------+-------------+-----------------------------------+
| test          | t1         | PRIMARY    | 2013-02-22 20:07:15 | n_diff_pfx01 |      33258 |          20 | a                                 |
| test          | t1         | PRIMARY    | 2013-02-22 20:07:15 | n_leaf_pages |         69 |        NULL | Number of leaf pages in the index |
| test          | t1         | PRIMARY    | 2013-02-22 20:07:15 | size         |         97 |        NULL | Number of pages in the index      |
| test          | t1         | idx_t1_b   | 2013-02-22 20:07:15 | n_diff_pfx01 |      32768 |          30 | b                                 |
| test          | t1         | idx_t1_b   | 2013-02-22 20:07:15 | n_diff_pfx02 |      32768 |          30 | b,a                               |
| test          | t1         | idx_t1_b   | 2013-02-22 20:07:15 | n_leaf_pages |         30 |        NULL | Number of leaf pages in the index |
| test          | t1         | idx_t1_b   | 2013-02-22 20:07:15 | size         |         31 |        NULL | Number of pages in the index      |
| test          | t1         | idx_t1_c   | 2013-02-22 20:07:15 | n_diff_pfx01 |          1 |          30 | c                                 |
| test          | t1         | idx_t1_c   | 2013-02-22 20:07:15 | n_diff_pfx02 |      32768 |          30 | c,a                               |
| test          | t1         | idx_t1_c   | 2013-02-22 20:07:15 | n_leaf_pages |         30 |        NULL | Number of leaf pages in the index |
| test          | t1         | idx_t1_c   | 2013-02-22 20:07:15 | size         |         31 |        NULL | Number of pages in the index      |
| test          | t2         | PRIMARY    | 2013-02-22 20:07:21 | n_diff_pfx01 |          1 |           1 | a                                 |
| test          | t2         | PRIMARY    | 2013-02-22 20:07:21 | n_leaf_pages |          1 |        NULL | Number of leaf pages in the index |
| test          | t2         | PRIMARY    | 2013-02-22 20:07:21 | size         |          1 |        NULL | Number of pages in the index      |
+---------------+------------+------------+---------------------+--------------+------------+-------------+-----------------------------------+
14 rows in set (0.00 sec)

mysql> explain select * from t1,t2 where t1.b=t2.a and t1.c=t2.a;
+----+-------------+-------+-------+-------------------+----------+---------+-----------+------+-------------+
| id | select_type | table | type  | possible_keys     | key      | key_len | ref       | rows | Extra       |
+----+-------------+-------+-------+-------------------+----------+---------+-----------+------+-------------+
|  1 | SIMPLE      | t2    | index | PRIMARY           | PRIMARY  | 4       | NULL      |    1 | Using index |
|  1 | SIMPLE      | t1    | ref   | idx_t1_c,idx_t1_b | idx_t1_b | 5       | test.t2.a |    1 | Using where |
+----+-------------+-------+-------+-------------------+----------+---------+-----------+------+-------------+
2 rows in set (0.01 sec)

mysql> update mysql.innodb_index_stats set stat_value = 600 where index_name = 'idx_t1_b' and stat_name = 'n_diff_pfx01';
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update mysql.innodb_index_stats set stat_value = 60000 where index_name = 'idx_t1_c' and stat_name = 'n_diff_pfx01';
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> flush table t1;
Query OK, 0 rows affected (0.00 sec)

mysql> explain select * from t1,t2 where t1.b=t2.a and t1.c=t2.a;
+----+-------------+-------+-------+-------------------+----------+---------+-----------+------+-------------+
| id | select_type | table | type  | possible_keys     | key      | key_len | ref       | rows | Extra       |
+----+-------------+-------+-------+-------------------+----------+---------+-----------+------+-------------+
|  1 | SIMPLE      | t2    | index | PRIMARY           | PRIMARY  | 4       | NULL      |    1 | Using index |
|  1 | SIMPLE      | t1    | ref   | idx_t1_c,idx_t1_b | idx_t1_c | 5       | test.t2.a |    1 | Using where |
+----+-------------+-------+-------+-------------------+----------+---------+-----------+------+-------------+
```

MySQL 5.6—改进总结

- **InnoDB Engine (本期内容)**
  - 性能
    - Read-Only Transaction, Kernel Mutex Split, Buffer Pool Flushing, Page Cleaner, Purge, CRC32 ...
  - 功能
    - Online DDL, Memcached Plugin, Transportable Tablespace, Buffer Pool Dump/Restore, FTS ...
- **MySQL Server (下期内容)**
  - **Optimizer**
    - Semi-Join, BKA, MRR, ICP, Join, In, Optimizer Tracing, Limit ...
  - **Replication**
    - GTID, Binlog Group Commit, Multi-Threaded Slaves ...
  - **Others**
    - Security ...

参考资料

- MySQL. MySQL 5.6.10 http://www.mysql.com/downloads/mysql/
- MySQL Doc. MySQL 5.6 Release Notes http://dev.mysql.com/doc/relnotes/mysql/5.6/en/
- MySQL Doc. InnoDB Performance and Scalability Enhancements *http://dev.mysql.com/doc/refman/5.6/en/innodb-performance.html*
- Oracle. MySQL 5.6: What's New in Performance, Scalability, Availability *https://blogs.oracle.com/MySQL/entry/mysql_5_6_is_a*
- Oracle. Better scaling of read-only workloads *https://blogs.oracle.com/mysqlinnodb/entry/better_scaling_of_read_only*
- 何登成. MVCC(Oracle, InnoDB, PostgreSQL) *http://vdisk.weibo.com/s/aUltP*
- Mikael Ronstrom. MySQL team increases scalability by >50% for Sysbench OLTP RO in MySQL 5.6 labs release april 2012 *http://mikaelronstrom.blogspot.com/2012/04/mysql-team-increases-scalability-by-50.html*
- 何登成. Buffer Pool Implementaion InnoDB vs Oracle http://vdisk.weibo.com/s/grZnZ
- Percona. Improved InnoDB I/O Scalability *http://www.percona.com/docs/wiki/percona-xtradb:patch:innodb_io*
- Oracle. New flushing algorithm in InnoDB *https://blogs.oracle.com/mysqlinnodb/entry/new_flushing_algorithm_in_innodb*
- Oracle. Introducing page_cleaner thread in InnoDB *https://blogs.oracle.com/mysqlinnodb/entry/introducing_page_cleaner_thread_in*
- Oracle. MySQL 5.6: Multi threaded purge *https://blogs.oracle.com/mysqlinnodb/entry/mysql_5_6_multi_threaded*
- 何登成. InnoDB Page Structure(InnoDB页面结构详解) *http://hedengcheng.com/?p=118*
- Oracle. InnoDB Compression Improvements in MySQL 5.6 *https://blogs.oracle.com/mysqlinnodb/entry/innodb_compression_improvements_in_mysql*
- Oracle. MySQL 5.6: Data dictionary LRU *https://blogs.oracle.com/mysqlinnodb/entry/mysql_5_6_data_dictionary*
- MySQLPerformance. How much memory Innodb Dictionary can take? *http://www.mysqlperformanceblog.com/2010/05/06/how-much-memory-innodb-dictionary-can-take/*
- Oracle. InnoDB 5.6.4 supports databases with 4k and 8k page sizes *https://blogs.oracle.com/mysqlinnodb/entry/innodb_5_6_4_supports*
- Oracle. Online ALTER TABLE in MySQL 5.6 *https://blogs.oracle.com/mysqlinnodb/entry/online_alter_table_in_mysql*
- Oracle. April 2012 Labs Release – Online DDL Improvements *https://blogs.oracle.com/mysqlinnodb/entry/april_2012_labs_release_online*
- MySQL Doc. InnoDB Integration with memcached *http://dev.mysql.com/doc/refman/5.6/en/innodb-memcached.html*
- 何登成. InnoDB Memcached Plugin源码实现调研 *http://hedengcheng.com/?p=619*
- Oracle. InnoDB transportable tablespaces *https://blogs.oracle.com/mysqlinnodb/entry/innodb_transportable_tablespaces*
- Todd. Understanding InnoDB transportable tablespaces in MySQL 5.6 *http://mysqlblog.fivefarmers.com/2012/09/07/understanding-innodb-transportable-tablespaces-in-mysql-5-6/*
- Oracle. InnoDB Persistent Statistics at last *https://blogs.oracle.com/mysqlinnodb/entry/innodb_persistent_statistics_at_last*
- Oracle. InnoDB persistent stats got a friendly UI *https://blogs.oracle.com/mysqlinnodb/entry/innodb_persistent_stats_got_a*
- 何登成. MySQL查询优化浅析 *http://vdisk.weibo.com/s/rpWqf*

Questions?

Thanks