InnoSQL 参考手册

5.5.20-v2

2012/8/21

目录

1 InnoSQL 概述

InnoSQL 是根据各生产环境业务需求以及由于硬件技术飞速发展而分支出的一个 MySQL 分支版本。此外，由于 MySQL 数据库在经历 Sun、Oracle 公司收购后，发展变得较为不明确，因此分支版本的 InnoSQL 可以自行为 MySQL 数据库添加各项功能，同时也能更了解 MySQL 数据库内部的工作原理。

InnoSQL 数据库会根据业务需求进行不断的开发和完善，同时也会整合开源社区的各类补丁，其目的是提高原有 MySQL 数据库的性能，提高数据库的高可用性，简化 DBA 的工作。同时，将一些富有创意的想法应用于数据库的生产环境中。

InnoSQL 完全兼容于 Oracle MySQL 版本，所有添加的补丁、插件、存储引擎都是动态的。如果不开启这些功能，那么它和原版本是完全一致的。迁移到 InnoSQL 数据库的成本为 0，因为其完全兼容于 MySQL。

InnoSQL 是一个开源的项目，官方网站为：http://www.innomysql.org/

2 InnoDB Flash Cache

2.1 概述

随着固态硬盘（solid state drive）的出现，硬盘技术得到了飞速的发展。与传统机械硬盘不同的是，固态硬盘没有磁头，是一个完全的电子设备，因此离散读取（random read）性能得到了极大的提高。然而，由于闪存的运作特性，数据不能像在普通机械硬盘里那样被直接覆盖。当数据第一次写入固态硬盘时，由于固态硬盘内所有页都为已擦除状态，所以数据能够以页为最小单位直接写入进去。但是要在上面再次写入的话，就需要首先擦除掉这个无效数据。而闪存的特性是，写入最小单位是页，而擦除最小单位是块，一般为 128~256 个页。因此，一般固态硬盘提供非对称的读写速度。

另一个值得关注的问题是闪存的写寿命。根据固态硬盘的不同，MLC 和 SLC 提供了不同的闪存写入寿命。不过一般固态硬盘提供商都提供了工具可以观察当前磁盘的写入情况，可以给终端使用用户一个较为明确的结果。

传统关系数据库系统对机械硬盘做了大量的优化，这些优化大多是针对于机械硬盘的特点，这些优化可能在固态硬盘中是不需要，甚至可能会带来性能的下降。故对于固态硬盘的优化，各数据库厂商还有很多问题需要解决。

目前对于固态硬盘的使用，一种是用做替代传统机械硬盘作为持久存储的方案，一种是将其作为 Cache。学界目前对这两种方案也有争议，但不管怎么说，固态硬盘的出现给数据库的工作方式带来了深远的影响。

InnoSQL 5.5.8 版本支持 InnoDB Secondary Buffer Pool。通过固态硬盘设备来做辅助缓冲池。该技术在大规模读的应用环境中有着非常不错的性能表现，但是在 TPC-C 这类测试中，可能

就显的比较平庸。因为其只是辅助缓冲池，而非真正意义的 Flash Cache。

InnoSQL 5.5.13 开始支持 InnoDB Flash Cache，虽然同样采用固态硬盘作为一个 Cache，但是两者的实现方式完全不同。其特点为：

- 不仅可以对读进行 Cache 存，也可以对写进行 Cache
- 可以进行 merge write，大大减少了实际页写次数
- 在固态硬盘上无离散写操作
- 预热时间大幅缩短

2.2 工作原理及架构设计

在 InnoDB 存储引擎中，为了避免 half-written 的问题，采用了 Doublewrite 的设计方式。其工作方式为：在刷新 128 个页前（最多 128 个页），先顺序地将 128 个页写入到 Doublewrite 中，在确保写入后则进行页的实际写入（更详细的说明请见《MySQL 技术内幕：InnoDB 存储引擎》）。

InnoSQL 的 Flash Cache 接管了原先 Doublewrite 的工作，并扩大了原先 Doublewrite 的尺寸。例如，用户可以拥有一个 300G 的 Doublewrite。因为 Doublewrite 的特性，对于 SSD 的写入都是顺序的。当页刷新到 Flash Cache 后建立对应的 Hash 表，那么之后读也可能会命中到 Flash Cache，因为刷新完成的脏页非常可能从缓冲池中移出。此外，当写入 Flash Cache 后，并不像 Doublewrite 那样去马上完成页的刷新操作，而是通过后台的 Flash Cache Thread 来完成这个操作。Flash Cache 中可能存在一个页的多个副本，当从 Flash Cache 中刷新回磁盘时，如果这个页已经有新的版本了，那老版本的页将不进行刷新，这个操作称为 merge write。

InnoSQL Flash Cache 架构如图 2-1 所示：

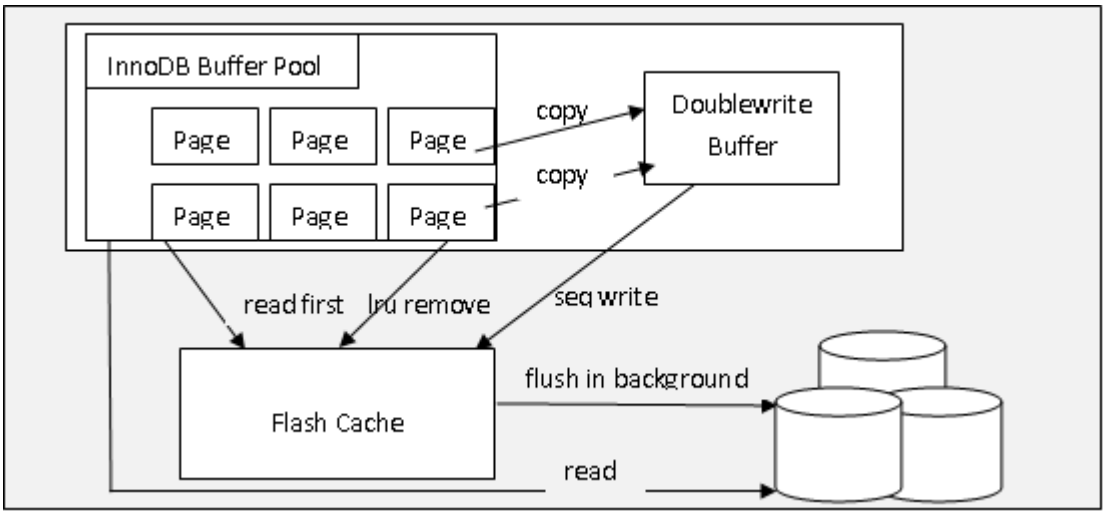

图 2-1 Flash Cache 架构

当从 flash cache 将页刷新回磁盘时，目前的算法为：

- 当 flash cache 中待刷新的页，小于 innodb_flash_cache_write_cache_pct* innodb_flash_cache_size/16384 时，不刷新页
- 当 flash cache 中待刷新的页，大于 innodb_flash_cache_write_cache_pct* innodb_flash_cache_size/16384 ，小于 innodb_flash_cache_do_full_io_pct** innodb_flash_cache_size/16384 时，刷新 10%的 innodb_io_capacity 页。
- 否则，刷新 innodb_io_capacity 个页

这个算法的目的是为了在 flash cache 尽可能的 cache 大量的写入页，以此来提高 merge write 的比例。

Flash Cache 架构设计符合了固态硬盘的特性，即充分利用了固态硬盘的高速离散读取性能以及避免了离散写入带来的性能下降和写入寿命问题。

2.3 配置参数

InnoSQL 提供了较多的配置参数来对 Flash Cache 进行各种不同的设置，可以通过 SHOW VARIABLES 命令来查看：

```
MySQL> show variables like 'innodb_flash_cache%';
+-----------------------------------------------------+----------------------+
| Variable_name                                       | Value                |
+-----------------------------------------------------+----------------------+
| innodb_flash_cache_do_full_io_pct                   | 90                   |
| innodb_flash_cache_file                             | d:\ib_fcfile         |
| innodb_flash_cache_is_raw                           | OFF                  |
| innodb_flash_cache_size                             | 1073741824           |
| innodb_flash_cache_use_shm_for_block | OFF                   |
| innodb_flash_cache_warmup_table                     | tpcc.*               |
| innodb_flash_cache_write_cache_pct                  | 10                   |
+-----------------------------------------------------+----------------------+
10 rows in set (0.06 sec)
```

2.3.1 innodb_flash_cache_file

flash cache 文件所在的位置。若没有手工创建，则 InnoSQL 在启动时，会根据 innodb_flash_cache_size 的大小自动生成该文件。因为 flash cache 较大，生成文件需要较长的时间，建议用户自动生成该文件。在 Linux 操作系统下，可以根据以下方式来创建 flash cache 文件：

```
root@db-51:/mnt/ddb/2# dd if=/dev/zero of=ib_fcfile bs=16384 count=63356
63356+0 records in
63356+0 records out
1038024704 bytes (1.0 GB) copied, 1.0725 s, 968 MB/s
root@db-51:/mnt/ddb/2# chown -R MySQL:MySQL ib_fcfile
```

上述通过 Linux 自带的 dd 命令产生了一个大小为 1G 的 flash cache 文件，最后不要忘记将 flash cache 文件的权限设置为数据库用户的权限。

2.3.2 innodb_flash_cache_size

启用 flash cache 的大小，注意该大小必须小于等于 flash cache 文件本身的大小。

2.3.3 innodb_flash_cache_is_raw

该参数可将基于固态硬盘的裸设备作为 flash cache。裸设备的设置与 Oracle 数据库下一致。该值默认为 OFF。

~~2.3.4 innodb_flash_cache_use_shm_for_block~~

该参数为 ON 时，可将 flash cache 中的内存信息存放在共享内存中，这样在 InnoSQL 数据库重启后，可以快速获得 flash cache 中的内容，加快数据的预热操作。该值默认为 OFF。InnoSQL 5.5.13.4 版本支持数据库正常关闭是将页信息 dump 到外存，不再支持该选项。

2.3.5 innodb_flash_cache_warmup_table

该参数控制 flash cache 的预热。第一次启用 flash cache 时，用户可以指定将表读取到 flash cache 中，因为是顺序读，因此可极大的提高 flash cache 的预热速度。该参数可以设置为：

```
innodb_flash_cache_warmup_table=tpcc.*
innodb_flash_cache_warmup_table=tpcc.warehouse:tpcc.orders
innodb_flash_cache_warmup_table=tpcc.*:user.*
```

tpcc.*表示预热 tpcc 架构下的所有表。冒号用来分区预热的各个表。

当预热的文件大于 flash cache 大小时，flash cache 会预热 innodb_flash_cache_size 大小的文件，然后停止。在启动时，可以从.err 文件中观察到：

```
111103 11:35:19    InnoDB: start to warm up tablespace tpcc.history to flash cache.
111103 11:35:19    InnoDB: warm up table tpcc.history to space: 21 offset 512.(100%)
111103 11:35:19    InnoDB: flash cache is full, warm up stop.
111103 11:35:20    InnoDB: flash cache warm up finish.
```

2.3.6 innodb_flash_cache_write_cache_pct

该参数控制刷新的算法。当 flash cache 小于该比例时，flash cache 不回刷到磁盘。这个阶段可称为 cache 阶段。该参数默认值为 10。

2.3.7 innodb_flash_cache_do_full_io_pct

该参数控制刷新的算法，当 flash cache 大于该比例时，则刷新 innodb_io_capacity 个页回到磁盘上，小于时则刷新 10%*innodb_io_capacity 个页回到磁盘上行。该值越大则可以得到越高的 merge write 比率。但是如果太大时，写入 flash cache 的页，则可能会遇到需要等待。该参数默认值为 90。

2.3.8 innodb_flash_cache_enable_write

该参数控制 Flash Cache 的 Write Cache 功能。默认为 1，表示将 Flash Cache 作为 Write Cache。0 表示不将 Flash Cache 作为 Write Cache，这时页的写入都将采用原来的方式，即 doublewrite -> disk。在 xtrabackup 备份时，该参数需设置为 0，并在备份结束后迅速的设置为 1。其余情况都不应设置为 0。

2.3.9 innodb_flash_cache_backup_dir

flash cache 的备份路径。若不进行设置，则默认在 datadir 目录下。注意该参数只指定备份路径，备份文件名都为 ib_fc_backup。

2.3.10 innodb_flash_cache_backuping

将 flash cache 中状态为 READY_FOR_FLUSH 的页备份到文件 ib_fc_backup。启用该参数时，必须将 innodb_flash_cache_enable_write 设置为 0。

2.4 观察状态

可以通过 SHOW GLOBAL STATUS 来查看 flash cache 的状态：

```
MySQL> show global status like 'innodb_flash%';
+-----------------------------------------+----------+
| Variable_name                           | Value |
+-----------------------------------------+----------+
| Innodb_flash_cache_aio_read       | 0        |
| Innodb_flash_cache_wait_aio       | 0        |
| Innodb_flash_cache_used           | 8192 |
| Innodb_flash_cache_read           | 0       |
| Innodb_flash_cache_write          | 0        |
| Innodb_flash_cache_flush          | 0        |
| Innodb_flash_cache_merge_flush | 0        |
| Innodb_flash_cache_move          | 0        |
+-----------------------------------------+----------+
9 rows in set (0.00 sec)
```

Innodb_flash_cache_read

通过 flash cahce 读取得到页的次数，该数据可反映 flash cache 的命中率

Innodb_flash_cache_aio_read

通过 flash cache 进行 AIO 读取得到页的次数

Innodb_flash_cache_write

写入 flash cache 中页的数量

Innodb_flash_cache_flush

刷新 flash cache 中的页回磁盘的数量

Innodb_flash_cache_merge_flush

merge write 的次数，通过该变量可得到 merge write 的效率

Innodb_flash_cache_move

只读页从 InnoDB Buffer Pool 刷新到 flash cache 的次数

Innodb_flash_cache_used

flash cache 中已经使用的页

另一种更直观的方法是通过 SHOW ENGINE INNODB STATUS 来查看当前 flash cache 的状态，如：

```
MySQL> show engine innodb status\G
……
----------------------
FLASH CACHE INFO
----------------------
flash cache size: 8192
flash cache thread status: flash cache thread is idle
flash cache location is: 0(0), flush to 0(0), distance 0 (0.00%), used 8192(100.00%), wait aio 0.
flash cache reads 117:, aio read 0, writes 0, flush 0(0), migrate 0, move 0
FIL_PAGE_INDEX reads: 116(99.15%): writes: 0, flush: 0, merge raio -1.#J%
FIL_PAGE_INODE reads: 1(0.85%): writes: 0, flush: 0, merge raio -1.#J%
FIL_PAGE_UNDO_LOG reads: 0(0.00%): writes: 0, flush: 0, merge raio -1.#J%
FIL_PAGE_TYPE_SYS reads: 0(0.00%): writes: 0, flush: 0, merge raio -1.#J%
FIL_PAGE_TYPE_TRX_SYS reads: 0(0.00%): writes: 0, flush: 0, merge raio -1.#J%
FIL_PAGE_OTHER reads: 0(0.00%): writes: 0, flush: 0
flash cache read hit ratio 5.94% in 55 second(total 4.85%), merge write ratio -1.#J%
flash cache 2.13 reads/s, 0.00 writes/s. 0.00 flush/s, 0.00 merge writes/s, 0.00 migrate/s, 0.00
move/s
……
```

这里可以观察到 flash cache 的命中率并细化到各种类型页的命中率，merge write 比率等。

总之，SHOW ENGINE INNODB STATUS 这可以得到一切比较细化的数据。建议 DBA 通过该命令来观察 flash cache 的状态。

2.5 Xtrabackup 支持

InnoSQL 5.5.20-v2 版本开始支持对于带有 Flash Cache 功能的数据库热备。其热备的实现方式如下所示：

- 设置 innodb_flash_cache_enable_write=0
- 设置 innodb_flash_cache_backuping=1，备份 flash cache 中的页
- Xtrabackup 备份 InnoDB 数据文件以及 redo log
- 设置 innodb_flash_cache_enable_write=1

注意当 innodb_flash_cache_enable_write 设置为 0 时，页不再向 Flash Cache 中写入，即将 Flash Cache 作为 Write Cache 失效，写入采用原 Doublewrite -> Disk 方式。随着大量的新页写入，也会导致 flash cache 的命中率下降。因此在备份完成后要重新将该值设置为 1。

对于恢复，其过程如下：

- 通过工具 fcbck2databck 将 flash cache 的页拷贝回磁盘
- xtrabackup apply log
- xtrabackup copy back

fcbck2databck 工具使用方法如下：

```
root@db-78:/ssd2/fcbk# ./fcbck2databck --data_bck_dir=/ssd2/backup
--fc_bck_path=/mnt/ddb/2/mysql_data_sbtest/ib_fc_backup
```

参数--data_bck_dir 表示数据库备份文件所在的目录，参数--fc_bck_path 表示 flash cache 的备份目录

3 InnoDB 共享缓冲池内存

3.1 概述

通常，基于磁盘的数据库系统都有缓冲池（buffer pool），用来缓存页的读取或写入，以此来提高数据库的整体性能。相应地，缓冲池越大，数据库的性能越好。

但是随着内存容量的不断增大，缓冲池的预热时间变得越来越长。这里，预热是指将磁盘上的页放入缓冲池。如果可能，用户可能更希望的是在上次数据库关闭的时候，缓冲池中的页放入缓冲池，这样能保证数据库应用的连续性。

假设磁盘的离散读取为 20M/s，这样预热一个 32G 容量的缓冲池需要差不多 30 分钟。在这

30 分种内数据库的负载会相当之高。

InnoSQL 的共享缓冲池内存特性支持 InnoDB 存储引擎。启用共享内存后，InnoDB 会通过共享内存技术来申请缓冲池页的内存。这样做的好处是，可以大大缩短数据库预热的时间。第二次启动时，通过扫描共享内存中保存的页到缓冲池即可。这种预热是及其快速的，因为是在内存中完成的，用户甚至察觉不到这个过程。同时，当 InnoSQL 数据库 Crash 后，同样可以使用共享内存中的页，InnoSQL 会通过比较写入到磁盘的 LSN 和每个页的 LSN，把未刷新的脏页从缓冲池中移除，这样也能极大的缩短 InnoDB 恢复的时间。

3.2 实现

InnoSQL 对原版 MySQL 在申请 buf_pool 内存这做了些修改。如果使用共享内存，则通过 shmget 来申请，然后通过 shmat 把共享内存映射到进程内部的地址。

这里有几个方面需要注意：

1. 每次申请共享内存的大小是固定的，如果我们需要修改 buf_pool 的大小，则在 MySQL 重启的时候，会自动判断原先的共享内存大小与当前的大小是否一致，如果不一致，则共享内存会被重建，原来的数据将被丢失，这样就起不到预热的效果。
2. 如果在配置了共享内存后，又关闭了该配置，为了保持数据的一致性，在关闭共享内存配置后启动时，原来的共享内存会被删除。
3. 由于 MySQL 内部 buf_pool 默认为 16K 对齐，而 linux 下共享内存默认为 4K 页对齐，所以为了保证得到的共享内存为 16K 对齐的，我们在进行 shmat 时需要制定一个进程内部 16K 对齐的地址给共享内存进行 shmat。
4. 由于 MySQL 服务出现异常而 crash，在重启启动 MySQL 时，系统内部共享内存初始化只初始化到最大 checkpoint 值部分，其它的数据页通过 recovery 恢复。
5. 由于操作系统在共享内存上的限制，在系统进行重新启动后，共享内存会被销毁，同样起不到预热的作用。

3.3 配置参数

3.3.1 innodb_use_shm_preload

在 MySQL 的配置文件 my.cnf 中增加一项用于控制是否使用共享内存作为 MySQL 的 buf_pool 的配置，如下：

```
[innodb]
Innodb_use_shm_preload=1
```

如果缺省上面的配置或者配置值为 0，则不使用共享内存作为 buf_pool，只有当上面的配置项为 1 的时候，才开启共享内存的配置。

由于操作系统在共享内存的申请上有某些限制，如在 Ubuntu 上默认的设置只支持 8G 的共享内存。所以在 buf_pool 比较大的情况下，需要修改系统的某些配置才能进行，修改系统的配置需要 root 权限。

涉及到的相关配置如下：

/proc/sys/kernel/shmmax：里面的值表示共享内存段的大小，单位为 Byte

/proc/sys/kernel/shmall：里面的值表示共享内存的总量，单位为 4KByte(页的大小)

如果开启的 buf_pool 的大小为 32G，则可以设置 shmmax 中的值大小为 34359738368，shmall 中的值大小为 8388608。

3.4 共享内存的管理

可以通过命令 IPCS 来查看当前启用的共享内存：

```
innosql@db-62:~$ ipcs -a

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x0008c231 4653056    innosql    600        549715968  0

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```

可以手工通过 IPCRM 命令来删除共享内存：

```
innosql@db-62:~$ ipcrm -m 4653056
```

4 InnoSQL 预热

4.1 概述

预热是指将磁盘上的页放入缓冲池。如果可能，用户可能更希望的是在上次数据库关闭的时候，缓冲池中的页放入缓冲池，这样能保证数据库应用的连续性。

通过共享缓冲池的方式已经可以很好的对数据库进行预热，但是当数据库发生宕机或者系统重启之后，贡献缓冲就启不到预热的效果。这时可以通过读取指定的文件 ib_buffer_pool 来进行预热。

导出的预热文件 ib_buffer_pool 只包含 space，offset 以及在 LRU 列表中的位置，而不是导出

这个缓冲池内存的信息，因此速度是非常快的。

将 Buffer Pool 中的数据库导出到文件 ib_buffer_pool:

```
mysql> set global innodb_buffer_pool_dump=1;
Query OK, 0 rows affected (0.00 sec)

mysql> set global innodb_buffer_pool_dump_now=1;
Query OK, 0 rows affected (0.00 sec)
```

可以指定间隔时间将 Buffer Pool 中的页导出到 ib_buffer_pool，如每隔 1 小时导出：

```
mysql> set global innodb_buffer_pool_dump=1;
Query OK, 0 rows affected (0.00 sec)

mysql> set global innodb_buffer_pool_dump_interval=3600;
Query OK, 0 rows affected (0.00 sec)
```

4.2 配置参数

4.2.1 innodb_buffer_pool_dump

是否启用 BP 的 dump 功能

4.2.2 innodb_buffer_pool_dump_interval

BP dump 的时间间隔

4.2.3 innodb_buffer_pool_dump_now

是否现在就导出

4.2.4 innodb_buffer_pool_load_at_startup

在 InnoDB 启动时读取 ib_buffer_pool 文件进行预热

4.2.5 innodb_buffer_pool_load_now

在数据库运行时读取 ib_buffer_pool 文件进行预热

5 InnoDB IO Statistic

5.1 概述

InnoDB IO Statistic 是用来统计 SQL 语句的 IO 请求统计。数据库底层的开发人员可以更了解 InnoDB 存储引擎的工作机制。对于 DBA 和 SQL 编程人员来说，也可以从这个特性中观察 SQL 语句是否运行合理，是否需要额外的索引来提高 SQL 语句的执行速度。目前仅支持 InnoDB 存储引擎，对于 NTSE 引擎的支持将会在稍后的版本中得到实现。

InnoSQL 将运行 SQL 语句得到的逻辑读、物理读的信息保存到默认的 slow log 查询中去。当启用记录 IO 信息时，可在 slow log 中看到类似如下内容：

```
# Time: 111103 13:29:06
# User@Host: root[root] @ localhost [::1]
# Query_time: 119.293823 Lock_time: 119.274822 Rows_sent: 1 Rows_examined: 1 Logical_reads: 198 Physical_reads: 3
use tpcc;
SET timestamp=1320298146;
select * from warehouse where w_id=1;
# Time: 111103 13:31:28
# User@Host: root[root] @ localhost [::1]
# Query_time: 0.335019 Lock_time: 0.333019 Rows_sent: 1 Rows_examined: 1 Logical_reads: 164 Physical_reads: 50
SET timestamp=1320298288;

select * from history;
```

5.2 配置参数

5.2.1 long_query_time

记录时间大于该参数的 SQL 语句，并同时记录下 InnoDB 的逻辑、物理读取次数

5.2.2 long_query_io

将逻辑读次数大于该值的 SQL 语句记录到 Slow Log，默认值为 100

5.2.3 slow_query_type

该参数可选的值为 0、1、2、3。表示记录 slow query 的类型。0 表示不开启该功能。1 表示记录符合 long_query_time 条件的 SQL 语句。这和原 MySQL 数据库的工作方式一样，不记录

IO 的次数。2 表示记录符合 io_slow_query 条件的 SQL 语句。3 表示记录所有符合条件的语句。

6 SQL Profiler with IO statistics

6.1 概述

通过 InnoDB IO Statistics 已经可以得到 SQL 语句的 IO 物理和逻辑读取的信息。但是得到的结果是放入到 slow log 中，DBA 和开发若想直接在命令行界面看到结果就显得不够直观和方便。MySQL Profiler 是 MySQL 数据库自带的一个 Profiler 工具，可以观察和分析 SQL 语句的执行过程以及执行的开销。现在将 IO 的信息集成到 Profiler 中，可以使 DBA 和开发人员更为直观的观察 SQL 语句运行的 IO 开销。而 slow log 记录的 IO 信息用于在运维过程中发现存在问题的 SQL 语句。

6.2 使用方法

```
mysql> set profiling=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from tpcc.warehouse;
……
mysql> select * from tpcc.warehouse limit 1;
……
mysql> show profiles\G;
*************************** 1. row ***************************
        Query_ID: 1
        Duration: 0.00118225
  Logical_reads: 72
Physical_reads: 0
           Query: select * from tpcc.warehouse
*************************** 2. row ***************************
        Query_ID: 2
        Duration: 0.00022750
  Logical_reads: 3
Physical_reads: 0
           Query: select * from tpcc.warehouse limit 1
```

若想让 show profiles 只输出最后的 SQL 语句，可以按如下设置：

```
mysql> set profiling_history_size=1;
Query OK, 0 rows affected (0.00 sec)
```

7 死锁检测优化

7.1 概述

InnoSQL 对 InnoDB 的死锁检测进行了优化。原 InnoDB 的处理方式是采用递归的深度优先算法，当并发量大的时候，递归的方式会产生瓶颈。InnoSQL 采用 Stack 的方式模拟递归的调用方式，以此提高该函数模块的整体性能。

7.2 配置参数

7.2.1 innodb_optimize_deadlock

默认值为 0，表示采用原版 MySQL 的处理方式。1 表示采用优化的方式。

8 UNDO information_schema

8.1 概述

MySQL 可以通过 SHOW ENGINE INNODB STATUS 来观察事物信息，但是对于 Undo 的信息显示较少。InnoSQL 扩展了对于 undo 信息的显示，DBA 和开发人员可以更为直观的了解到每个事务产生的 Undo 信息。

InnoSQL 在 information_shema 架构下多产生两张表 INNODB_ROLLBACK_SEGMENT 和 INNODB_TRX_UNDO。前者用来显示回滚段的信息，后者用来显示每个事务产生的 UNDO 信息。当事务结束时，INNODB_TRX_UNDO 表中的信息将会消失，因此 INNODB_UNDO 表保存的只是当前活跃事务的 UNDO 信息。

8.2 INNODB_ROLLBACK_SEGMENT

字段	说明
segment_id	段 ID
space	段的 space ID
page_no	段的 Page no
last_page_no	最近使用的 und log 页的 page not
last_offset	最近使用的 undo log 页中的偏移量

last_trx_no	最近使用该回滚段的事务 no
update_undo_list	update undo 页的数量
update_undo_cached	被 cache 的 update undo 页的数量
insert_undo_list	insert undo 页的数量
insert_undo_cache	被 cache 的 insert undo 页的数量

8.3 INNODB_TRX_UNDO

字段	说明
trx_id	事务 ID
rseg_id	回滚段的 ID
undo_rec_no	undo 记录编号，从 1 开始
undo_rec_type	undo 记录类型： ❒ TRX_UNDO_INSERT_REC ❒ TRX_UNDO_UPD_EXIST_REC ❒ TRX_UNDO_UPD_DEL_REC ❒ TRX_UNDO_DEL_MARK_REC
size	undo 记录的大小（字节）
space	undo 记录所在 space
page_no	undo 记录所在的 page no
offset	undo 记录所在页的偏移量

8.4 使用方法

8.4.1 查看回滚段信息

查看回滚段的信息，MySQL 5.5 开始一共支持 127 个回滚段，即回滚段 ID 0～126，127 回滚段不使用。如：

```
mysql> SELECT * FROM INNODB_TRX_ROLLBACK_SEGMENT \G;
*************************** 1. row ***************************
```

```
            segment_id: 0
                 space: 0
               page_no: 6
        last_page_no: NULL
         last_offset: 9183
         last_trx_no: 31654D
   update_undo_list: 0
update_undo_cached: 1
   insert_undo_list: 0
insert_undo_cached: 1
……
128 row in set (0.00 sec)
```

8.4.2 查看事务 UNDO 信息

查看每个事务的每个具体 UNDO 记录信息：

```
mysql> SELECT * FROM INNODB_TRX_UNDO LIMIT 1\G;
*************************** 1. row ***************************
         trx_id: 3C1AB5
       rseg_id: 25
   undo_rec_no: 0
undo_rec_type: TRX_UNDO_UPD_EXIST_REC
           size: 33
          space: 0
       page_no: 81943
         offset: 5958
……
40 row in set (0.00 sec)
```

计算每个事务产生的 UNDO 量：

```
mysql> SELECT trx_id,SUM(size) FROM INNODB_TRX_UNDO GROUP BY trx_id;
```

```
+------------+------------------+
| trx_id     | SUM(size)        |
+------------+------------------+
| 3D0329 |              841 |
| 3D0330 |              790 |
| 3D0331 |              522 |
| 3D033C |              559 |
| 3D0344 |              383 |
| 3D0353 |              207 |
| 3D0357 |              173 |
| 3D035E |              154 |
| 3D0365 |               84 |
+-------------+-----------------+
9 rows in set (0.00 sec)
```

9 InnoSQL IO 优化

9.1 概述

InnoSQL 对与数据库的 IO 做了如下的优化工作

- 增大 redo log file 可设定的值
- 控制是否启用刷新邻接页
- Page cleaner thread
- Group commit 补丁整合

默认 InnoDB 存储引擎对于 redo log file 的限制为 4G，InnoSQL 将其增大为 16G。增大 redo log file 的好处是可以避免 async 刷新，使得刷新变得更为平缓。缺点是恢复需要的时间会有所延长。

InnoDB 存储引擎在刷新时会检测刷新页所在的区（extent）中所有页的情况，如果是脏页的则会刷新。这样做的好处是可以将多次 IO 合并成一次 IO 进行刷新，以此提高 IO 的性能。然而，这么处理的缺点是可能将不怎么脏的页刷新到磁盘，同时对于 SSD 这里有着较高 IOPS 性能磁盘，这么处理也显得不那么必要。InnoSQL 通过参数的方式来控制是否启用刷新邻接页的功能。

page cleaner thread 功能在 InnoSQL 5.5.20-1 版本中移除掉。因为增大 redo log file 的尺寸就能使得刷新变得较为平稳。而这块的代码又显得非常复杂。

InnoSQL 还 merge 了 MariaDB 对于 MySQL group commit 修复的补丁。在某些特定场景下，性能有大幅的提高。

9.2 配置参数

9.2.1 innodb_flush_neighbors

默认为 1，表示启用邻接页的刷新。0 表示不启用。对于 SSD 这类有着较高 IOPS 的磁盘，建议将该值设置为 0。

9.3 状态信息

InnoSQL 还增强了 SHOW ENGINE INNODB STATUS 对于 IO 的显示：

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 16483614720; in additional pool allocated 0
Dictionary memory allocated 74706
Buffer pool size      983040
Free buffers            202072
Database pages          765996
Old database pages 282780
Modified db pages    0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 12658, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 1534480, created 0, written 956181
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: 765996, unzip_LRU len: 52752
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
Async Flush: 0, Sync Flush: 0, LRU List Flush: 861792, Flush List Flush: 1050570
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
1 read views open inside InnoDB
Main thread process no. 21752, id 140206699194112, state: waiting for server activity
Purge thread process no. 21753, id 140206690801408
Number of rows inserted 0, updated 0, deleted 0, read 0
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
```

在 SHOW ENGINE INNODB STATUS 的 BUFFER POOL AND MEMORY 的部分，最后显示了不同类

型刷新的次数。而在 ROW OPERATIONS 的部分，显示了 main thread 和 purge thread 线程的进行 ID，这样可以通过 iotop 这类工具来观察这两个线程的 IO 使用情况。

10 Page Cleaner Thread

10.1 概述

在 InnoSQL 中对 InnoDB 存储引擎新增加了一个后台线程，称为 Page Cleaner Thread。自适应的刷新（adaptive flush），以及脏页的刷新(modified buffer pool page flush)，异步的刷新（async flush），关闭数据库时的脏页刷新（flushing when shutdown）都在该线程中完成。

注意：InnoSQL 5.5.20-1 版本开始不再支持 page cleaner thread

在 InnoDB 存储引擎中，页刷新存在五种情况。在继续说明前，首先介绍一些概念，这将有助于更好的理解 InnoDB 中的刷新算法。

checkpoint_age：current_lsn – checkpint_lsn（检查点最后发生时的 LSN）

async_water_mark: ~78%*Log_Group_Size

sync_water_mark: ~90%*Log_Group_Size

在 InnoDB 存储引擎中，Redo Log 是循环写的结构，默认不支持对于 Redo Log 的归档。因此为了避免覆盖写 Redo Log 时，脏页还未刷新到磁盘，设置了 async_water_mark 和 sync_water_mark。故 InnoDB 刷新页的算法为：

checkpoint_age < async_water_mark 这种情况以为这有足够的 Redo Log 空间，当前没有必要非常迫切的去刷新脏页到磁盘上。在这种情况下，根据是否启用 adaptive_flusing 策略来刷新一部分页。所有的刷新操作都在 Master Thread 中完成，不会阻塞任何其他线程。

async_water_mark < checkpoint_age < sync_water_mark 当查询用户线程（User Thread）检查到当前 checkpoint_age 在（async_water_mark，sync_water_mark）区间范围内时，这意味了当前 Redo Log 空间已经出现紧张情况，需要马上刷新一部分的页到磁盘上。这种情况下，只有第一个发现问题的查询用户线程会被阻塞，并立刻刷新脏页，刷新在用户线程中完成，而非后台线程。但是其余的查询用户线程依然可以继续工作，不会被阻塞。

checkpoint_age > sync_water 这种情况代表 Redo Log 空间严重不足，可视为最坏的一种情况。发现该现象的查询用户线程立即刷新脏页，其余查询用户线程等待刷新操作完成。

n_dirty_pages > innodb_max_dirty_page_pct 当脏页的数量大于 innodb_max_dirty_page_page_pcty* buffer_pool_size 时，刷新 innodb_io_capacity 个页。这个操作时在 Master Thread 线程中完成，不会阻塞其他查询用户线程。

当 InnoDB Buffer Pool 的缓冲池已经满了，用户线程需要申请新的内存时，需要释放 LRU 列表中页。当该页是脏页，即同时存在于 FLUSH LIST 中时，那么用户线程会刷新 LRU 列表中的脏页，保证缓冲池有不少于 64 个的空闲页。这个刷新不同于前面所说的刷新，称为 FLUSH LRU_FLUSH_LIST。

在原 MySQL 数据库中，上述的刷新算法存在以下两个问题：

- 当工作负荷比较大时，Master Thread 可能过于繁忙，导致没有足够的时间来刷新脏页。
- 异步的刷新脏页操作在查询用户线程（User Thread）中完成，这将导致该用户线程的操作被阻塞。
- 当 FLUSH LRU_FLUSH_LIST 时，同样会阻塞用户线程。

Page Cleaner Thread 的目的为了解决上述三个问题。所有之前在 Master Thread 中完成的刷新操作都移入到该后台线程中。与此同时，查询的用户线程不再需要负责异步的刷新以及 LRU_FLUSH_LIST 的刷新操作，查询的用户线程只负责完成同步刷新（Sync Flush）操作，即 checkpoint_age > sync_water 这种情况。

10.2 配置参数

10.2.1 innodb_enable_page_cleaner_thread

表示是否启用 Page Cleaner Thread，默认为 OFF，表示原 MySQL 5.5 的刷新策略。如果要启用 Page Cleaner Thread，可在 MySQL 数据库的配置文件中进行如下的设置：

```
[mysqld]

innodb_enable_page_cleaner_thread=ON
```

11 MySQL Profiler

11.1 概述

从 InnoSQL 5.5.20-v2 版本开始引入 MySQL Profiler，用来控制每个用户可以使用的资源。目前可控制的内容包括：

- 每个用户可以使用 CPU 资源的总量
- 每个用户每个事务可以使用 CPU 的资源
- 每个用户可以使用逻辑 IO 资源的总量
- 每个用户每个事务可以使用逻辑 IO 资源的总量

在 mysql 架构下新增了表 user_profiler，用来保存用户可使用的资源，其结构如下所示：

ysql> desc mysql.user_profiler;

```
+----------------------+-----------------+------+-----+---------+-------+

| Field                | Type            | Null | Key | Default | Extra |
```

```
+----------------------+-----------------+------+-----+---------+-------+
| Host                 | char(60)        | NO   | PRI |         |       |
| User                 | char(16)        | NO   | PRI |         |       |
| Max_cpu_times        | int(11) unsigned | NO   |     | 0       |       |
| Curr_cpu_times       | int(11) unsigned | NO   |     | 0       |       |
| Max_io_reads         | int(11) unsigned | NO   |     | 0       |       |
| Curr_io_reads        | int(11) unsigned | NO   |     | 0       |       |
| Max_mem_size         | int(11) unsigned | NO   |     | 0       |       |
| Max_cpu_times_per_trx | int(11) unsigned | NO   |     | 0       |       |
| Max_io_reads_per_trx | int(11) unsigned | NO   |     | 0       |       |
| Operate_priv         | enum('N','Y')    | NO   |     | N       |       |
+----------------------+-----------------+------+-----+---------+-------+
10 rows in set (0.00 sec)
```

其中 cpu_time 的单位为 1/100 秒。Operate_priv 表示用户是否有权限操作 user_profiler 表。

当前需要用户手工的往其中插入数据，在之后的版本中将提供类型 CREATE PROFILER 的命令来支持往 user_profiler 表中插入数据。插入数据后必须执行 FLUSH PRIVILEGES。

11.2 配置参数

11.2.1 current_profiler_record

用来控制将当前状态写入到 user_profiler 的频率，有效值为：

- 0 表示当用户线程退出后对当前用户的资源限制重置为 0
- 1 表示当用户线程退出后保存当前用户的资源使用情况，下次该用户连接时，继续统计资源的使用情况
- 2 表示每个 SQL 语句执行完后，对用户使用的资源情况更新到表 user_profiler

11.3 状态观察

当用户需要查看当前已使用 cpu 和 io 的总资源，可以通过如下命令：

```
mysql> show status like 'Threads%used'\G;
*************************** 1. row ***************************
```

```
Variable_name: Threads_cpu_times_used

        Value: 0/0

*************************** 2. row ***************************

Variable_name: Threads_io_used

        Value: 0/0

2 rows in set (0.00 sec)
```

12 Binlog Enhance

12.1 记录 user 和 ip

默认 MySQL 数据库的 binlog 不记录 user 和 ip 信息，当产生问题时不能定位是哪个用户执行了该语句。InnoSQL 增加了参数 binlog-user-ip，将 user 和 ip 信息记录到 binlog 中。注意如果要搭建 replication 环境，需要确保两边该参数的值必须设置为相同。同样的，通过 mysqlbinlog 查看时，同样需要增加参数—binlog –user-ip，最后的显示效果如：

```
mysqlbinlog --binlog-user-ip /data/mysqld.000001

……

#root@localhost

# at 191

#120821 17:18:21 server id 1  end_log_pos 295  Query  thread_id=1  exec_time=0
      error_code=0

use test/*!*/;

SET TIMESTAMP=1345540701/*!*/;

delete from t where a = 1

/*!*/;

#root@localhost

# at 295

#120821 17:18:23 server id 1  end_log_pos 380  Query  thread_id=1  exec_time=0
      error_code=0

SET TIMESTAMP=1345540703/*!*/;

COMMIT

/*!*/;
```

```
#root@localhost

DELIMITER ;

# End of log file

ROLLBACK /* added by mysqlbinlog */;

……
```

13 InnoSQL 推荐配置

```
[mysqld]
innodb_flush_neighbors=0（for SSD）
innodb_purge_threads=1
slow_query_type=3
innodb_use_shm_preload=1
innodb_buffer_pool_dump_interval=3600
```