InnoSQL 参考手册

David Jiang

2011/11/3

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：

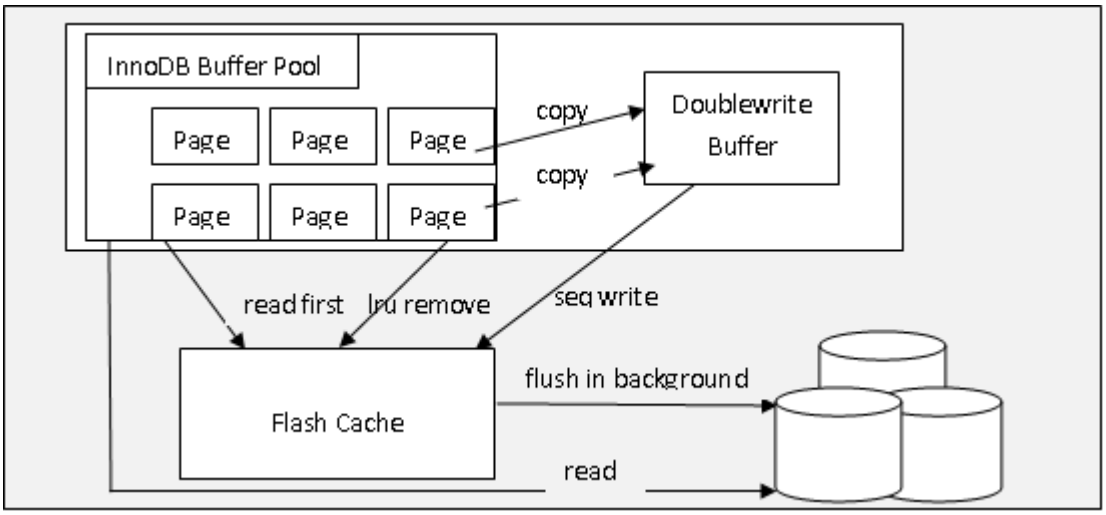

图 2-1 Flash Cache 架构

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)
```

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。

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。

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。

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。

innodb_flash_cache_warmup_table

该参数控制 flash cache 的冷预热。第一启用 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 的大小是，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.
```

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。

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。越大该则可以得到越高的 merge write 比率。但是如果太大时，写入 flash cache 的页，可能会需要等待。该参数默认值为 90。

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。

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

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 InnoDB IO Statistic

4.1 概述

InnoDB IO Statistic 是用来观察每句 SQL 语句中，InnoDB 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。

注意：该功能目前为 preview，下个版本可能对此作出一些调整

InnoSQL 将运行 SQL 语句得到的 InnoDB 逻辑读、物理读的信息放到默认的 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;
```

4.2 配置参数

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

io_slow_query

记录物理 IO 次数大于等于该值的 SQL 语句

slow_query_type

该参数可选的值为 0、1、2，表示记录 slow query 的类型。0 表示记录符合 long_query_time 条件的 SQL 语句，1 表示记录符合 io_slow_query 条件的 SQL 语句，2 表示记录所有符合条件的语句。

5 Page Cleaner Thread

5.1 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。

在 InnoDB 存储引擎中，页刷新存在四种情况。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。

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。

Page Cleaner Thread 的目的为了解决上述两个问题。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。

5.2 配置参数

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
```