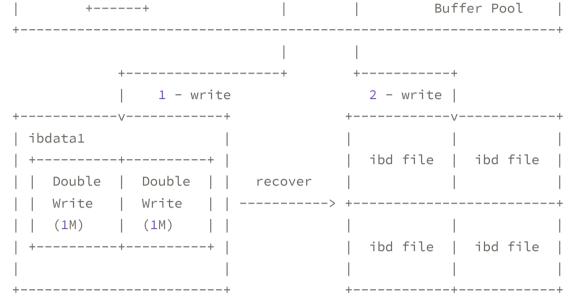
MySQL DBA学习笔记-----美河学习在线 www.eimhe.com 仅学习参考

## MySQL学习笔记(Day026:InnoDB\_7 – doublewrite/ChangeBuffer/AHI/FNP)

### —. Double Write

#### 1 Double Write介绍



### 2. Double Write的开销

1 row in set (0.00 sec)

```
假设每个页大小为16K,则2M的Double Write中存放了128个页,在使用了Double Write之后,IO从原来的128次IO变成了 128 + 1次IO,而 不是128 + 128次IO。
Double Write的2M数据是 顺序 刷入磁盘的,是一次10 ,该次IO的大小为2M。
开启Double Write的性能降低5% ~ 25%(IO Bound场景下降的最厉害)
slave服务器同样需要开启
```

# 3. Double Write可以关闭的前提

### 二. Insert/Change Buffer

# 1. Insert/Change Buffer介绍

### 2. Insert/Change Buffer举例

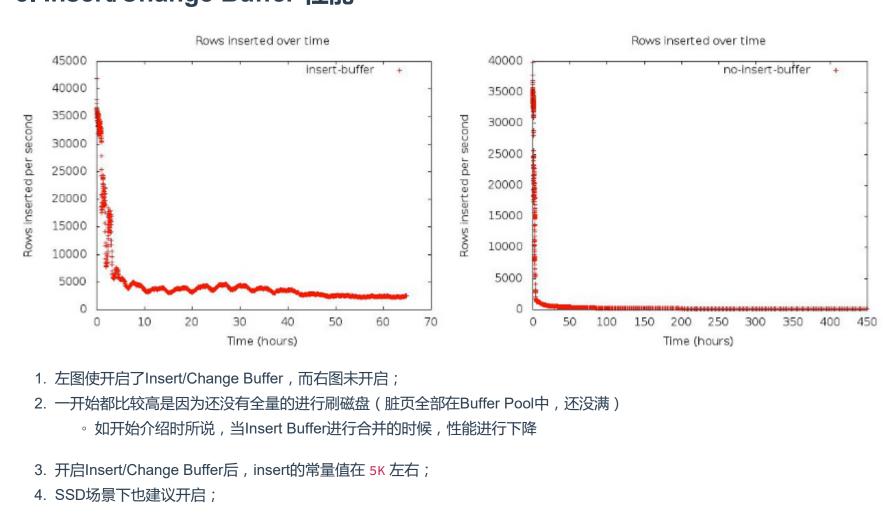
```
CREATE TABLE t(
a INT AUTO_INCREMENT, -- a 列是自增的
b VARCHAR(30), -- b 列是varchar
PRINARY KEV(a) -- a 是主體
key(b) -- b 是二级索引(如果是由mme之类的,可以看成是非唯一的)
);

• 对于主键(a 列),每次插入都要立即插入对应的 柴集索引 页中(在内存中就直接插入,不在内存就先读取到内存)
• 对于二级索引(secondary index)(b 列)

1. 在 没有 Insert/Change Buffer时,每次插入一条记录,就要读取一次页(读取内存,或者从磁盘读到内存),然后将记录插入到页中;
2. 在 有 Insert/Change Buffer时,当插入一条记录时,尤为所记录对应要插入的二级索引(secondary index)页 是否在Buffer Pool中:

• 如果该二级索引(secondary index)页 已给在Buffer Pool中,则 直接插入;
• 反之,先将其 Cache 起来,放到Insert/Change Buffer中,等到该二级索引(secondary index)页被读到 时,将Insert/Change Buffer中该页对应的记录合并(Merge)进去,从而减少I/O操作;
Insert/Change Buffer就是用来提升二级索引输入的性能。
使用空间换时间,批量插入的方式(二级索引耐以不急着插入,只要主键已经插入了即可)
```

# 3. Insert/Change Buffer 性能



MySQL DBA学习笔记-----美河学习在线 www.eimhe.com 仅学习参考

#### 4. Insert/Change Buffer 查看

```
mysql> show engine innodb status\G
 -- -----省略其他输出-----
 INSERT BUFFER AND ADAPTIVE HASH INDEX
 -----
Ibuf: size 1, free list len 0, seg size 2, 0 merges -- 这里为0 ,可能是Buffer Pool足够大,
                                         -- 数据页都缓存在内存中了,就用不到buffer了
 merged operations:
 insert 0, delete mark 0, delete 0
 discarded operations:
 insert 0, delete mark 0, delete 0
1. seg size: 页 的数量,例如当前页为8K,则 seg_size * 8K 就是Insert/Change Buffer使用的内存大小;
2. merges:合并了多少页;
3. merged insert:插入了多少条记录;
    。insert / merges 就是插入的效率 (插入一条记录,就要读取一次页);
4. discarded operations:应该是很小的值,或者为0;当记录写入到Insert/Change Buffer后,对应的表被删除了,则相应的Buffer中的记录就应该被丢弃;
 使用Insert/Change Buffer的前提时,需要使用随机IO,这时才放入Buffer中,如果页已经在Buffer Pool(内存)中,就不需要使用Insert/Change Buffer了
```

```
5. Change Buffer
  • 从MySQL 5.5 以后,改名为Change Buffer,表示不仅仅适用于insert。
     1. Insert
     2. Delete-Marking ( 标记删除 )
     3. Purge (清除)
     4. innodb_change_buffering = all
          all
          ■ none (禁用)
          inserts
          deletes
          changes = ( insert & delete-marking )
          purge
```

### 三. Adaptive Hash Index (自适应Hash索引)

```
• 搜索的时间复杂度
   。B+树 O(T), 只能定位到该记录所在的页;
    。哈希表 O(1), 可以直接定位到记录;
• 可以自己判断是否是活跃的页,如果是活跃的页,可以自动做Hash,管理员无需人工干预;
· MySQL5.6版本后,官方不推荐使用自适应Hash索引
   。CPU 使用率变高,但是性能没有提升;
• MySQL5.7中增加了 innodb_adaptive_hash_index_parts ,增加分片 ,以减少竞争 ;
• 只对等值的操作有意义;
```

#### 四. Flush Neighbor Page (FNP)

```
• 刷新 脏页所在区 (extent)的 所有脏页 , 合并IO , 随机转顺序的优化;
  。写入的数据太多
   。 如果业务确实是频繁更新,那刷新也会很频繁
• 对传统机械磁盘有意义;
  innodb_flush_neighbors={0|1|2} ( >=MySQL 5.6 )
  。0:表示关闭该功能
  。1:表示刷新一个区内的脏页
  。2:表示刷新几个 连续 的脏页
· SSD建议关闭次功能;
mysql> show variables like "%flush_neigh%";
+----+
+----+
+----+
```

1 row in set (0.00 sec)