从案例看InnoDB表设计优化

叶金荣

2016.1.16

为什么对InnoDB表随机写入非常慢？

背景

- 向InnoDB表随机insert数据，最快每秒1万多，最慢每秒才500条
- 表t1

```
CREATE TABLE `t1` (
   `id` int(10) unsigned NOT NULL ,
   `c1` varchar(64) NOT NULL,
   `c2` varchar(255) DEFAULT NULL,
   `c3` int(10) unsigned DEFAULT  ‘0’ ,
   PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

如何写入的

- 采用下面的PHP代码随机写入数据

```
$i_min = 1;
$i_max = 999999999;
for ($i = $i_min; $i < $i_max; $i++)
{
  $x = round(mt_rand() * $i);
  $sql = "INSERT INTO t1 VALUES($x, 'name_$x', 'desc_$x', $i)";
}
```

insert过程记录

```
记数从100000 开始
ok 503 w, per: 12307 /s
ok 504 w, per: 13369 /s
...
ok 508 w, per: 14082 /s
ok 509 w, per: 11058 /s
ok 510 w, per: 12853 /s
…
ok 1201 w, per: 15592 /s
ok 1202 w, per: 11037 /s
ok 1203 w, per: 13454 /s
ok 1204 w, per: 12482 /s
ok 1205 w, per: 7162 /s
ok 1206 w, per: 6323 /s
…
ok 1210 w, per: 2559 /s
ok 1211 w, per: 2170 /s
```

```
…接上
ok 1282 w, per: 1079 /s
ok 1283 w, per: 1093 /s
ok 1284 w, per: 836 /s
ok 1288 w, per: 816 /s
ok 1291 w, per: 778 /s
ok 1292 w, per: 776 /s
...
ok 1360 w, per: 618 /s
ok 1361 w, per: 611 /s
ok 1362 w, per: 580 /s
ok 1363 w, per: 593 /s
...
ok 1372 w, per: 563 /s
ok 1373 w, per: 554 /s
ok 1374 w, per: 504 /s
ok 1375 w, per: 547 /s
ok 1376 w, per: 569 /s
...
```

案例分析

- 数据量

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
| 13719413 |
+----------+
1 row in set (12 min 14.59 sec)
```

- 表空间大小

```
-rw-rw---- 1 mysql mysql 8.5K 08-09 15:01 t1.frm
-rw-rw---- 1 mysql mysql 1.4G 08-09 15:50 t1.ibd
```

案例分析

- 查看基本情况：
  - 在slow log中可查看执行count时读取了多少distinct page，读了多少行；

```
mysql>set global log_slow_verbosity=full;
```

  - 备注：Percona分支版本才有这功能，或者打补丁

案例分析

- slow log

```
# Thread_id: 36  Schema: test  Last_errno: 0  Killed: 0
# Query_time: 719.255557  Lock_time: 0.000051  Rows_sent: 1  Rows_examined:
13719413  Rows_affected: 0  Rows_read: 18446744073693256119
# Bytes_sent: 70  Tmp_tables: 0  Tmp_disk_tables: 0  Tmp_table_sizes: 0
# InnoDB_trx_id: 4A0B331
# QC_Hit: No  Full_scan: Yes  Full_join: No  Tmp_table: No  Tmp_table_on_disk:
No
# Filesort: No  Filesort_on_disk: No  Merge_passes: 0
#   InnoDB_IO_r_ops: 74732  InnoDB_IO_r_bytes: 1224409088  InnoDB_IO_r_wait:
713.597767
#   InnoDB_rec_lock_wait: 0.000000  InnoDB_queue_wait: 0.000000
#   InnoDB_pages_distinct: 65254
SET timestamp=1312882181;
select count(*) from t1;
```

案例分析

- 物理读取字节数

```
mysql> select 65254*16/1024;
+---------------+
| 65254*16/1024 |
+---------------+
|     1019.5938 |
+---------------+
```

- 数据表理论大小

```
mysql> select (4+30+30+4)*13719413/1024/1024;
+-------------------------------+
| (4+4+30+30)*13719413/1024/1024 |
+-------------------------------+
|                    889.70192337 |
+-------------------------------+
```

而实际上，数据表物理大小是：
1.4GB

案例分析

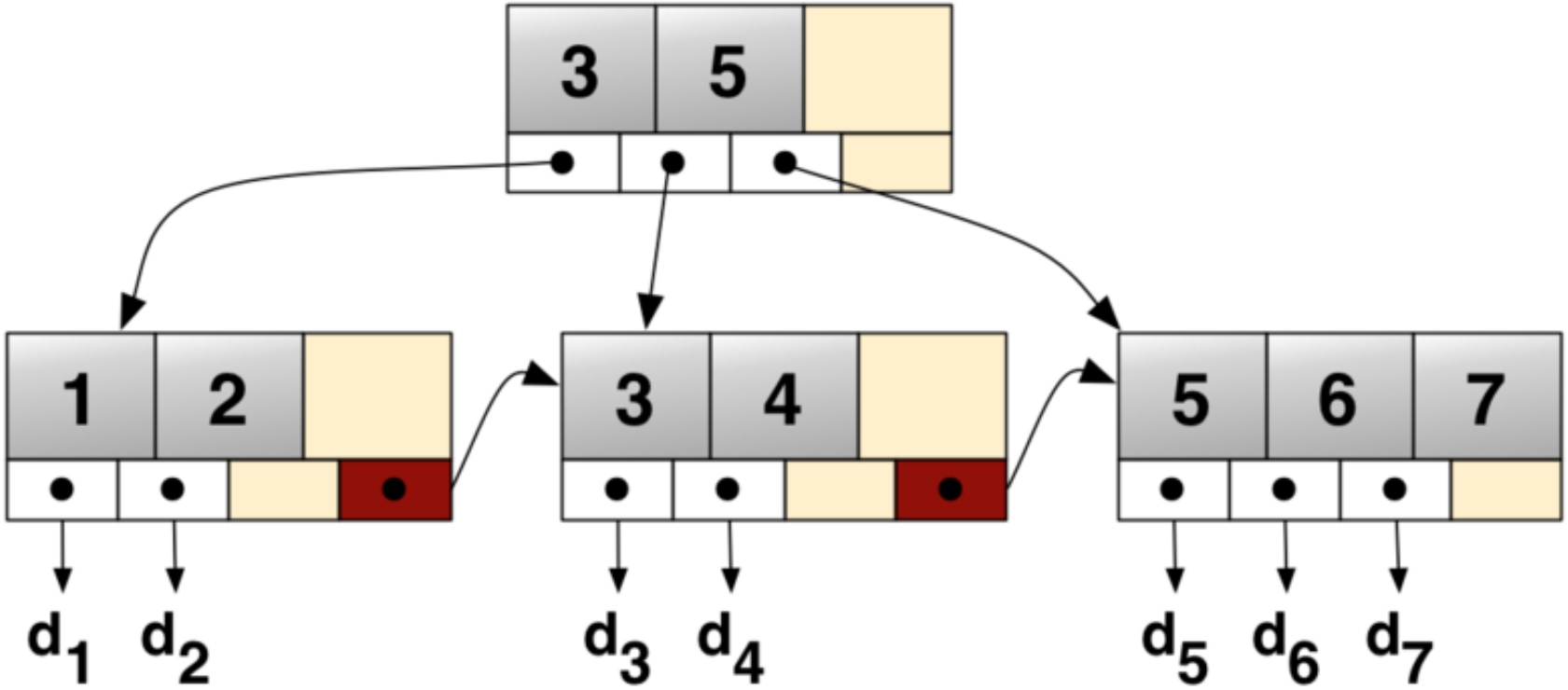
- 思考
  - 为什么数据表理论大小和实际尺寸差距那么大？
    - 是因为索引而导致物理尺寸增加吗？
    - 实际上，对于InnoDB表而言，没有普通索引时就没有额外的索引空间，主键本身就是整个表数据
  - 为什么count那么慢
    - Rows_read:18446744073693256119，怎么会这么大？
    - count执行慢可能是因为文件不够连续，随机I/O多
    - Rows_read值这么高，是Percona Server在5.1.56前的bug

案例分析

- 数据表物理尺寸和实际有差距的原因
  - 物理存储时产生碎片，原因：
    - InnoDB表是基于b+ tree的聚集索引组织表；
    - InnoDB表物理存储实际上是根据主键（PK）的顺序存储；
    - 数据写入顺序如果和主键顺序一致则最优；
    - 本案例中，数据是随机写入的，所以。。。

案例分析

- InnoDB B+ tree index

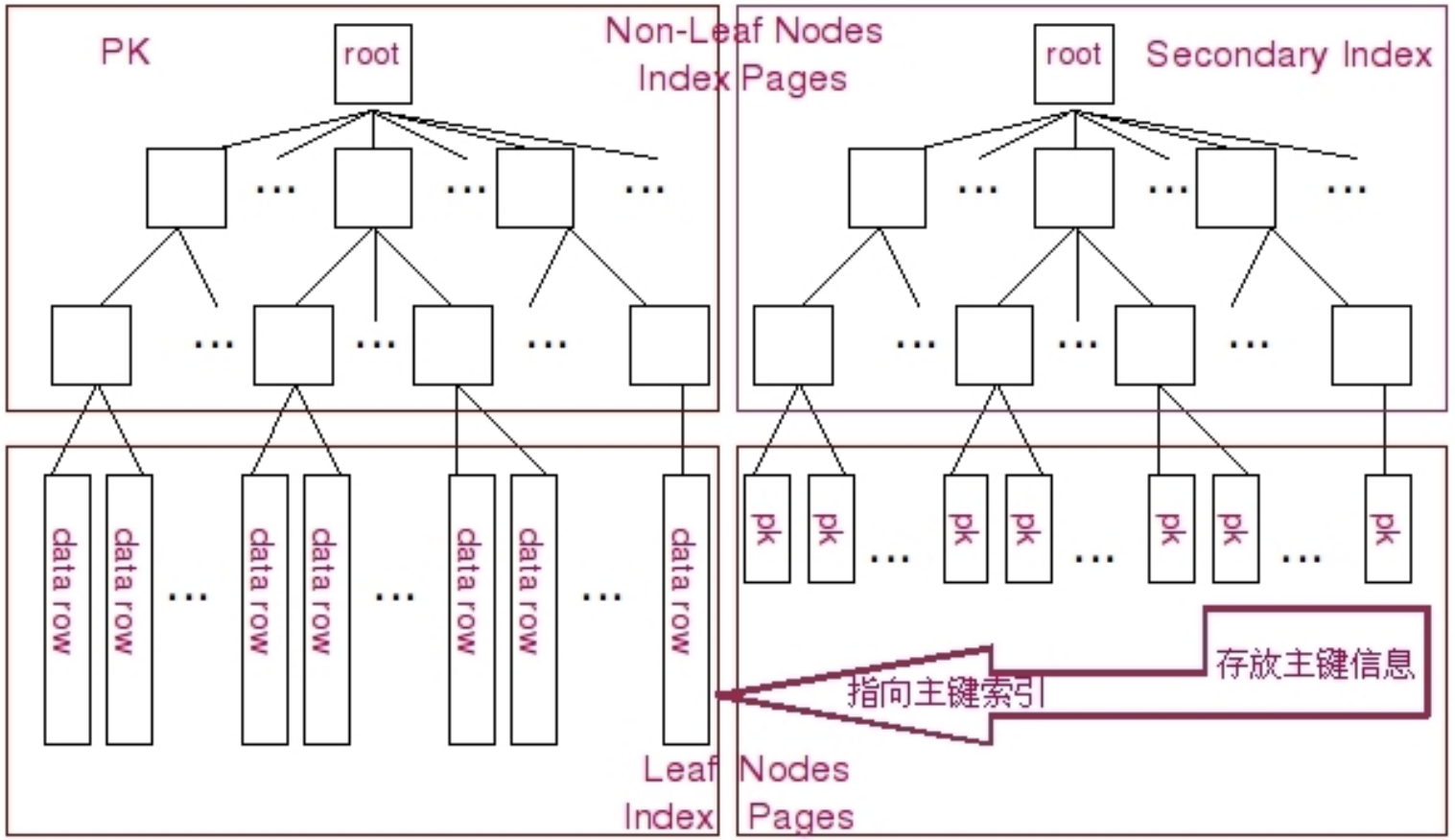


案例分析

- count(*)统计太慢的原因
  - InnoDB是事务表， 没有像MyISAM表那样有个计数器，执行count(*)时需要实时统计，通常优先选择二级索引，不得已才选择主键；
  - 所以如果是不带WHERE的count(*)实际上相当于作了一次全表（有二级索引时则优先选择二级索引）扫描；
  - 该表本身碎片情况很严重，即便是全表扫描时的I/O代价也很高；
  - slow log中的Read_Rows值很大，实际上是bug引起的，虽然确实需要读取到很多记录数。

案例分析

- InnoDB B+ tree index



思考

- 从测试结果来看，在写入1200万行数据前，写入性能还能接受，那么
  - 碎片对写入性能有影响么，如何避免碎片；
  - InnoDB表上执行count(*)时都发生了什么；
  - 是否考虑进行表拆分；
  - 不过，拆分后，如果对该表的请求存在跨多表需求时，是否反而更麻烦。

解决方案

- 如何消除碎片
  - 重整表空间即可消除碎片

```
mysql> alter table t1 engine = innodb;
```

  - 整理后表空间大小

```
-rw-rw---- 1 mysql mysql 848M 08-09 18:23 t1.ibd
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
| 13719413 |
+----------+
1 row in set (4.92 sec)
```

- 从12分钟缩短到5秒内，嗯，还有提升空间吗？

解决方案

- count(*)还能加速吗

```
mysql> alter table t1 add index idx_c3 (c3);

mysql> select count(*) from t1_rnd;
+----------+
| count(*) |
+----------+
| 13719413 |
+----------+
1 row in set (1.84 sec)
```

- 从12分钟直接提升到5秒内，现在又提升到了1.84秒，看来还没到尽头

解决方案

- 先看看InnoDB表上执行count(*)时做了什么

```
# Thread_id: 44  Schema: test Last_errno: 0  Killed: 0
# Query_time: 1.841461  Lock_time: 0.000051  Rows_sent: 1  Rows_examined:
13719413  Rows_affected: 0  Rows_read: 13719413
# Bytes_sent: 70  Tmp_tables: 0  Tmp_disk_tables: 0  Tmp_table_sizes: 0
# InnoDB_trx_id: 4A0B89B
# QC_Hit: No  Full_scan: Yes  Full_join: No  Tmp_table: No  Tmp_table_on_disk:
No
# Filesort: No  Filesort_on_disk: No  Merge_passes: 0
# InnoDB_IO_r_ops: 0  InnoDB_IO_r_bytes: 0  InnoDB_IO_r_wait: 0.000000
# InnoDB_rec_lock_wait: 0.000000  InnoDB_queue_wait: 0.000000
# InnoDB_pages_distinct: 12252
SET timestamp=1312970168;
select count(*) from t1_rnd;
```

- 思考：为什么加了二级索引后count提速了？

经验总结

- InnoDB表主键是有序的整型
- 写数据时，主键值和主键定义顺序规则一致；
- 执行count(*)时，如果执行计划选择了PK，需要引起注意，是否改成二级索引；
- 对于Innodb表的使用要保持一个好的习惯，尽量基于唯一索引去访问数据（排除range）
- 如果对于Innodb有大量的update操作，同时又有大量的range操作，请考虑定期进行整理碎片（注意锁表影响）

经验总结

- 结论验证：写入程序稍作调整

```
$i_min = 1;
$i_max = 999999999;
for ($i = $i_min; $i < $i_max; $i++)
{
  $x = $i;
  $sql = "INSERT INTO t1 VALUES($x, 'name_$x', 'desc_$x', $i)";
}
```

经验总结

记数从100000 开始
ok 498 w, per: 21769 /s
ok 499 w, per: 21384 /s
ok 500 w, per: 21770 /s
ok 501 w, per: 21376 /s
ok 502 w, per: 21749 /s
...
ok 1380 w, per: 21624 /s
ok 1381 w, per: 21329 /s
ok 1382 w, per: 21632 /s
ok 1383 w, per: 21592 /s
ok 1384 w, per: 21377 /s
ok 1385 w, per: 18434 /s
ok 1386 w, per: 19711 /s
...

…接上
...
ok 2991 w, per: 21515 /s
ok 2992 w, per: 21772 /s
ok 2993 w, per: 18120 /s
ok 2994 w, per: 19615 /s
ok 2995 w, per: 21163 /s
ok 2996 w, per: 21782 /s
ok 2997 w, per: 21377 /s
ok 2998 w, per: 21705 /s
ok 2999 w, per: 13881 /s
ok 3000 w, per: 20969 /s
...

写入到3KW行后，性能依旧不差

经验总结

- 数据量

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
| 32719524 |
+----------+
1 row in set (1.93 sec)
```

- 表空间大小

```
-rw-rw---- 1 mysql mysql 8.5K 08-09 15:50 t1.frm
-rw-rw---- 1 mysql mysql 1.9G 08-09 16:19 t1.ibd
```

举一反三

- 下面这些业务表该如何设计呢
  - 用户表（User）
  - LOG表