Oracle 的排序与布隆过滤器（Bloom Filter）

中国 Oracle 用户组

作者：黄玮 （Fuyuncat）

http://www.acoug.org

版本	发布时间
1.0	2010/4/16

1. Oracle 的排序

本节主要介绍 Oracle 的排序算法与跟踪。

1.1 排序(sort)算法

在 Oracle 的排序过程中，首先将获取到的数据放入一块私有内存区（Sort Area）中进行排序。如果需要排序的数据太大，无法一次在 Sort Area 中完成全部数据的排序，就会将 Sort Area 中排好序的数据直接写入（Direct Write，数据不被 cache）临时空间作为一个数据集存在。当所有数据都在内存中排过序并写入了磁盘后，就会将磁盘上的数据集进行合并排序 （Merge Sort）。合并排序是一个递归过程，直到所有数据集都被合入一个数据集，排序才算完成。

1.2 初始化运行（Initial Runs）

数据最初在 Sort Area 中排序的过程被称为初始化运行。Sort Area 的 80~90%的空间需要被用于读缓冲，其他空间则被用于写缓冲。如果我们知道有多少数据（可以通过需要排序的记录数、记录平均长度已经数据块大小估算出来）需要进行排序，那么就可以用下面的公式来估算初始化运行的次数（也就是会产生多少个初始数据集）：

```
Initial Runs = CEIL(SORT_DATA_BLOKCS   /
                    ROUND(0.8*SORT_AREA_SIZE/BLOCK_SIZE))
```

1.3 合并（Merges）

在进行合并时，可以同时合并 2 个或 2 个以上的数据集。一次合并多少个数据集就称为合并宽度（Merge Width）。合并也是在 Sort Area 中完成的，进行合并之前，需要将数据从磁盘直接读入（Direct Read）内存中。和全表扫描的多数据块读（MBRC）类似，对排序数据块的读取也可以是多个的，由 sort_multiblock_read_count（SMRC，这个参数在 9i 后是隐含参数）控制一次读取的数据块数。不过，请注意，这个参数控制的 是一次能读取的最大数据块数，而实际上一次能读取的数据块数是由 sort_area_size、数据块大小等数据计算得来的。要进行合并操作，最少需要夺 取 2 个数据集；如果使用了异步 IO（disk_asynch_io=TRUE），需要有 2 块读缓冲。因此实际的 SMRC 可以按照以下公式计算：

SMRC = MIN(ROUND(SORT_AREA_SIZE/(2*2*BLOCK_SIZE))-1, _sort_multiblock_read_count)

有了 SMRC，合并宽度也可以计算出来了，

Merge Width = ROUND(SORT_AREA_SIZE/(2*SMRC*BLOCK_SIZE))-1

合并是个递归过程，直到所有数据集被合为一个数据集。每一轮合并过程中，如果还有少于合并宽度的数据集没有被合并，则会在下一轮中再进行合并。我们创建了以下函数来计算合并次数。

注意：除最后一次合并外，所有合并都被称为中间运行（Intermediate run）。

```
HELLODBA.COM>create or replace function sort_merges(init_runs number, width number)
  2  return number
  3  as
  4    i number:=init_runs;
  5    n number:=0;
  6  begin
  7      while i>width loop
  8          n := n+floor(i/width);
  9          i := floor(i/width)+ mod(i,width);
 10      end loop;
 11      if i > 0 then
 12          n := n+1;
 13      end if;
 14      return n;
 15  end;
 16  /

Function created.
```

我们来看一个例子：sort area 大小为 64k，估算到有 721 个数据块需要进行排序，数据块大小为 8K。通过以下查询语句，我们就可以得到这次排序的相关统计数据。

```
HELLODBA.COM>select init_runs,SMRC,merge_width,sort_merges(init_runs,merge_width) merges from
  2  (select init_runs,
  3                SMRC,
  4                round(sort_area_size / (2 * SMRC * block_size), 0) - 1 as merge_width
  5      from (select sort_blocks, sort_area_size, block_size,
               ceil(sort_blocks / round(0.8 * sort_area_size / block_size, 0)) init_runs,
```

```
  6                                round(sort_area_size / (2 * 2 * block_size), 0) - 1 SMRC
  7                        from (select 721 as sort_blocks, 65536 as sort_area_size, 8192 as b
lock_size from dual)));
 INIT_RUNS       SMRC  MERGE_WIDTH      MERGES
----------  ----------  -----------  ----------
      121           1            3          60
1 row selected.
```

1.4 新的排序算法

从 10gR2 之后，oracle 引入了新的排序算法以减少排序时内存和 CPU 的消耗。新算法由参数"_new_sort"控制，默认为 TRUE。新算法在以下几个方面进行了改进。

*内存排序

对于那些仅需要在内存中就可以完成的小数据集的排序，新算法需要更少的内存

*合并排序

初始化运行完成后，会保存初始数据集的键值到内存中，在进行数据集进行合并时，会根据键值来选择数据集。从 trace 文件中可以看到一条新的统计数据：

Comparisons while searching for key in-memory 120

新算法过程中，合并过程中用到的临时数据块更容易被重用，节省磁盘空间

1.5 Trace - 跟踪

我们可以通过 10031, 10032, 10033 事件对排序过程进行跟踪，跟踪内容可以帮助我们更好的理解排序过程及其性能。下面的跟踪文件就是我们之前例子的产生的。

```
Recording run at 401ca9 for 6 blocks
Recording run at 401cb0 for 6 blocks
Recording run at 401cb6 for 6 blocks
Recording run at 401d3c for 6 blocks
Recording run at 401d42 for 6 blocks
Recording run at 401d48 for 6 blocks
...
---- Sort Parameters ------------------------------
sort_area_size                          65536
sort_area_retained_size                 65536
sort_multiblock_read_count                  1
max intermediate merge width                3
```

```
Merging run at 401d1a for 1 blocks
Merging run at 401c3a for 6 blocks
Total number of blocks to read: 7 blocks
Recording run at 401d1b for 7 blocks
Merging run at 401c40 for 6 blocks
Merging run at 401c46 for 6 blocks
Merging run at 401c5c for 6 blocks
Total number of blocks to read: 18 blocks
Recording run at 401d22 for 17 blocks
...
---- Sort Statistics ------------------------------
Initial runs                                          121
Intermediate runs                                      59
Number of merges                                       60
Input records                                       47582
Output records                                      47582
Disk blocks 1st pass                                  721
Total disk blocks used                                896
Total number of comparisons performed            651813
    Comparisons performed by in-memory sort      320660
    Comparisons performed during merge           331033
    Comparisons while searching for key in-memory  120
Temp segments allocated                                1
Extents allocated                                     56
Uses version 2 sort
Uses asynchronous IO
        ---- Run Directory Statistics ----
Run directory block reads (buffer cache)             240
Block pins (for run directory)                         1
Block repins (for run directory)                     239
        ---- Direct Write Statistics -----
Write slot size                                     8192
Write slots used during in-memory sort                 2
Number of direct writes                             2884
```

```
Num blocks written (with direct write)          2884
Block pins (for sort records)                       2884
Waits for async writes                              1731
        ---- Direct Read Statistics ------
Size of read slots for merge phase               8192
Number of read slots for merge phase                6
Size of read slots for output                    8192
Number of read slots for output                     8
Number of direct sync reads                      1498
Number of blocks read synchronously              1498
Number of direct async reads                     1386
Number of blocks read asynchronously             1386
Waits for async reads                             347
---- End of Sort Statistics -----------------------
```

2. 布隆过滤器（Bloom Filter）

布隆过滤器（Bloom Filter）是用于判断一个元素是否属于一个数据集的数据结构。其基本思想就是用一个或多个 hash 函数对数据集中的每个成员做映射，映射结果不是存在完整的 hash 表中，而是存在一个位向量（bit vector）中。位向量所有位初始都为 0，根据 hash 结果将位向量中相应位置 1。对数据集中的所有成员的 hash 计算完成后，就得到了该数据集的位向量。当需要判断一个元素是否属于该数据集时，也用相同的 hash 函数对其映射得到它的位向量，然后将其位向量上所有为 1 的位与数据集位向量上相应位比较， 如果发现数据集的位向量上某个位为 0 的话，可以判断这个元素不属于该数据集，这样的一个结果是肯定的。而如果所有相应位都为 1 的话，那么该元素可能属于这个数据集。

如果由布隆过滤判断某个元素属于一个集合，但事实上却不是，那就是误判。误判率可以这样计算：

假如集合成员数为 n，用 k 个函数映射后，m 位向量上某个位为 0 的概率为

(1-1/m)^(k*n)

某个位为 1 的概率就是

1-(1-1/m)^(k*n)

而如果要判断一个数是否属于该集合，其所有映射位和集合向量的所有匹配的概率就是

(1-(1-1/m)^(k*n))^k

而假设期望的误判率为 p，那相应 m 值的计算就是

1/(1-(1-p^1/k)^1/(k*n)))

从上面的计算式可以看出，当 m 越大，误算的概率就越低。我在这里用一段 perl 代码模拟了 bloom filter。通过调整 m 值，可以看到 m 越大，误算率越低：

```
HELLODBA.COM>host perl bloomfilter.pl 10 100
array size:20; hash num:10; vector members:100
29 missed in 100(0.29)

HELLODBA.COM>host perl bloomfilter.pl 10 150
array size:20; hash num:10; vector members:150
5 missed in 100(0.05)

HELLODBA.COM>host perl bloomfilter.pl 10 200
array size:20; hash num:10; vector members:200
2 missed in 100(0.02)
```

代码可以在这里下载：

http://www.Hellodba.com/Download/bloomfilter.zip

根据上面计算式，假如用 10 个 hash 函数判断一个元素是否在一个成员数为 20 的数据集当中，我们希望其误判率在 0.01（1%） 以内，那么 m 值最少为 202。

```
HELLODBA.COM>select ceil(1/(1-power(1-power(p,1/k),1/(k*n)))) m from (select 0.01 p, 10 k, 20 n from dual);
         M
----------
       202
```

Oracle 在做并行 join、分区裁剪（bloom filter pruning，11g 引入）时会用到 bloom filter。 以下是 10.2.0.3 上的相关参数：

```
HELLODBA.COM>select name, value, description from all_parameters where name like '%bloom%';
NAME                           VALUE  DESCRIPTION
------------------------------ ------ ------------------------------------------
_bloom_filter_enabled          TRUE   enables or disables bloom filter
_bloom_filter_debug            0      debug level for bloom filtering
```

以下是 11g 上的相关参数：

```
HELLODBA.COM>conn demo/demo@ora11
```

```
Connected.
NAME                          VALUE  DESCRIPTION
----------------------------- ------ -------------------------------------------
_bloom_filter_enabled         TRUE   enables or disables bloom filter
_bloom_filter_debug           0      debug level for bloom filtering
_bloom_vector_elements        0      number of elements in a bloom filter vector
_bloom_predicate_enabled      FALSE  enables or disables bloom filter predicate pushdown
_bloom_pruning_enabled        TRUE   Enable partition pruning using bloom filtering
```

其中_bloom_vector_elements 就是控制上面计算式中的 m。

黄玮的个人简介

黄玮，网名 Fuyuncat

99 年开始从事 DBA 工作，有多年的水利、电信及航运行业大型数据库开发、设计和维护经验。

曾供职于南方某电信设备制造公司。作为项目组长及 DB 组长，承担项目管理和数据库系统的设计、开发和维护。

目前供职于东方海外货柜有限公司珠海 ISDC，负责全球电子物流系统——CargoSmart 的数据库开发、维护工作。

个人网站：http://www.HelloDBA.com/
Email: fuyuncat@gmail.com
msn: fuyuncat@hotmail.com