Buffer Cache Performance Analysis And Multiple Buffer Pools in Oracle9i

Abstract
Is the tuning process complete once a high cache-hit ratio has been achieved? Often times not! This paper exposes analysis and techniques that prove buffer cache effectiveness can still be improved even after a cache-hit ratio of nearly ninety-percent has been achieved! Oracle9i offers high-resolution performance analysis through its internal V$ and X$ performance objects. Techniques and tips in this paper are based upon a large-scale OLTP application benchmark.

August 2001

Author: Kevin Closson

Introduction

Effective utilization of disk block buffering is a critical factor in maintaining acceptable OLTP transaction response times. Database Administrators are challenged with the balancing act of allocating sufficient memory for disk block buffering while ensuring adequate resources remain for the many other types of memory usage. Often times the tuning process is conducted only to the point where either there is no more memory available to allocate to the buffer cache, or the system has achieved an arbitrarily high cache-hit ratio.

Traditionally the exercise conducted by Database Administrators comprised of monitoring cache-hit ratio before and after increasing the size of the buffer cache. An overall cache-hit ratio of eighty to ninety percent is commonly acceptable.

This paper takes a different look at buffer cache tuning by first discarding the notion that the best performance has been realized once a high cache-hit ratio is achieved. Advanced database servers, such as Oracle9i, offer a richness of performance data that enable Database Administrators to make the best choices for how to utilize a given amount of buffer cache capacity.

Additionally, a useful feature known as multiple buffer pools was first introduced in the Oracle8 release. This paper exposes a case study of an OLTP benchmark based on Oracle9i in which the initial cache-hit ratio was nearly ninety percent. However, both transaction throughput and response times were improved through effective analysis of buffer pool utilization and the use of multiple buffer pools.

Multiple Buffer Pools in Oracle9i

Oracle introduced multiple buffer pools in the Oracle8 release[1]. As always, the total number of disk block buffers in the Oracle SGA (Shared Global Area) can be allocated based on the setting of the initialization parameter DB_BLOCK_BUFFERS. However, with the introduction of multiple database block size support in Oracle9i, there are new SGA buffer pool tunables. If multiple block sizes will be used, the DB_BLOCK_BUFFERS parameter is not supported. For this reason, this paper will center around the new Oracle9i cache sizing parameters.

If multiple pools are not defined, all of the buffers will be associated with the DEFAULT buffer pool. The DEFAULT pool is used to buffer all blocks from tablespaces that use the default block size[2]. However, the buffer cache must be partitioned into multiple pools if multiple block sizes will be used. The buffer

[1] The scope of this paper does not include a full tutorial on the multiple buffer pools feature. Oracle Documentation should be consulted for clear, complete coverage.
[2] The default block size in Oracle9i is established with the init.ora tunable *db_block_size*

cache can be partitioned into a maximum of five buffer pools by setting the following initialization parameters:

- DB_CACHE_SIZE
- DB_RECYCLE_CACHE_SIZE
- DB_KEEP_CACHE_SIZE
- DB_*N*K_CACHE_SIZE

These new Oracle9i init.ora parameters take a size assignment in Megabytes. For example, DB_CACHE_SIZE = 500M.

The intended usage of DB_RECYCLE_CACHE_SIZE and DB_KEEP_CACHE_SIZE is to establish partitioned sets of buffers from the DEFAULT buffer pool. These buffers will only be utilized for objects explicitly defined by the Database Administrator using the ALTER TABLE, ALTER INDEX and ALTER CLUSTER commands. There are no KEEP or RECYCLE pools allowed for any of the non-default block size pools.

The DB_*N*K_CACHE_SIZE parameter is used to configure up to four additional buffer pools beyond the default. A common usage for this functionality is to support transporting a tablespace from a different block size database. If a database expects to import a transportable tablespace from a database that is of a different block size, it is necessary to configure a buffer pool with the appropriate block size. For instance, if the default block size for the receiving database is 4K, but the transported tablespace is from an 8K database, the DB_8K_CACHE_SIZE needs to be set[1].

The RECYCLE buffer pool is best utilized to "protect" the default buffer pool from being consumed by randomly accessed blocks of data. With releases prior to Oracle8, the only method available to mitigate the affect of such access patterns was through the ALTER TABLE command with the NOCACHE option. Since it is an LRU based approach, the NOCACHE option can only offer limited benefit.

When the NOCACHE option is applied to an object, processes are limited to a low percentage of buffers from the least recently used end of the LRU lists when reading in blocks from that object. Any LRU list, of course, was fair game. Unlike most freshly read blocks however, these buffers are not moved to the most-recently-used end of its LRU list, thereby aging out of the cache quicker. This differs greatly from the effect of a RECYCLE buffer pool.

[1] The ALTER SYSTEM command can also be used to dynamically add the 8K buffer pool.

Figure 1.0 depicts the initialization parameter settings required to allocate a buffer cache of one-hundred Megabytes in total size with ten percent allocated to a recycle buffer pool.

```
db_cache_size = 100M
db_recycle_cache_size = 10M
```

Figure 1.0: Initialization parameters allocating a ten- percent *recycle* pool

Often times an application will have a few very critical objects, such as indexes, that are small enough to fit in the buffer cache but are quickly pushed out by other objects. This is the perfect case for using the initialization parameter called DB_KEEP_CACHE_SIZE. The KEEP buffer pool is aptly named. It is intended to be used for objects that take absolute priority in the cache. For instance, critical indexes or small look-up tables. An approach to sizing the KEEP buffer pool will be described in detail later in this paper. However, it should be considered a dedicated portion of the buffer cache for very special objects.

Objects that are not explicitly assigned to either the KEEP or RECYCLE buffer pool will be cycled through the DEFAULT buffer pool. The default buffer pool is comprised of the remaining buffers once the RECYCLE and KEEP buffer pools have been allocated

Figure 1.1 is a depiction of an Oracle SGA buffer cache with a total size of one hundred Megabytes. In the illustration, ten Megabytes have been allocated to the RECYCLE buffer pool. In addition, sixty Megabytes have been configured for the KEEP buffer pool. The remaining thirty Megabytes are for the DEFAULT buffer pool.

```
db_cache_size = 100M
db_recycle_cache_size = 10M
db_keep_cache_size = 60M
```

Figure 1.1: Initialization parameters employing all default-block-size buffer pools

Monitoring Buffer Pool Effectiveness

Oracle9i contains a richness of runtime performance data specifically related to the buffer cache. The internal performance objects and views (X$ and V$) provide high-resolution data that is critical to assessing the root cause of performance degradation. The Oracle documentation contains a good deal of information on these performance objects and is augmented by third-party industry publications.

For the sake of this paper, the performance data of most interest will be gleaned from the following performance analysis objects:

- xbh, xkcbwds
- v$buffer_pool
- v$filestat, obj$
- v$sysstat
- dba_data_files

The information derived from these objects can be summarized into the following three categories:

- **Buffer cache-hit ratio.**

  The v$sysstat performance view can be used to calculate the cache-hit ratio. The columns contain cumulative data since the instance was booted.

  Database Administrators typically perform delta operations over periods of time with this data. In the case of this paper, however, the tests were conducted with a fresh boot of the database instance before each run. The formula requires the sum of both types of Oracle logical reads known as *db block gets* and *consistent gets* as well as the number of physical reads.

  Figure 2.0 contains both the formula and the SQL statement used to calculate the cache-hit ratio. For reference, this script will be referred to as *hit_ratio.sql* for the remainder of this paper.

```
REM  hit ratio =  1 – (preads  /  lreads )
column lreads format 9999999999 heading 'LOGICAL  READS'
column preads format 9999999999 heading 'PHYSICAL READS'
select sum(value) lreads  from           v$sysstat
where          name in ('db block gets', 'consistent gets');
select   value  preads
from           v$sysstat
where          name in ('physical reads');
```

Figure 2.0: *hit_ratio.sql* - SQL statement provides data for cache-hit ratio calculation

- **Object presence in the buffer cache on an individual buffer pool basis.**

  This script provides a fine grain view of each pool within the buffer cache. Figure 2.1 contains the SQL statement for this data collection. For reference, this script will be referred to as *cache_content.sql* for the remainder of this paper.

```
select buff_pool.name  pool, o.name object, sum(ct) blocks
from
( select set_ds, obj, count(*) ct from x$bh group by set_ds, obj) bh, obj$ o,x$kcbwds
kcbw,v$buffer_pool buff_pool
where o.dataobj# = bh.obj
and o.owner# > 0 and bh.set_ds = kcbw.addr
and kcbw.set_id between buff_pool.lo_setid and
buff_pool.hi_setid  and     buff_pool.buffers != 0
group by buff_pool.name, o.name, o.subname
order by buff_pool.name, o.name, o.subname ;
```

Figure 2.1: *cache_content.sql* - SQL statement providing buffer pool content

- **Datafile physical reads by object.**

  This script provides data that facilitates determining which objects in the database are cycling through the cache. Note, this script presumes a database schema that has an established like-named tablespace for each table. Figure 2.2 contains the SQL statement for this data collection. For reference, this script will be referred to as *object_reads.sql* for the remainder of this paper.

```
select tablespace_name,sum(PHYRDS) reads
from dba_data_files,v$filestat
where dba_data_files.file_id = v$filestat.file#
group by tablespace_name
order by reads desc;
```

Figure 2.2: *object_reads.sql* - SQL statement attributing datafile reads to each tablespace

Benchmark Study

In order to clearly depict the benefits of multiple buffer pools in Oracle9i, the following benchmark analysis is provided.

The workload chosen was based upon an Order Entry and Product Tracking OLTP application running on Oracle9i.

The database consisted of approximately forty Gigabytes of table storage and index overhead. On average, each transaction consists of twelve DML statements. The transaction mix can be broken down into four primary areas as shown in Figure 3.0.

Transaction Type	Percent of Total Transactions
Insert Intensive:	
Taking New Orders	18%
Adding New Customers, New Products and Warehouse Stock	16%
Select Intensive:	
Reports (Shipment Status, Stock on Hand, etc)	54%
Update Intensive:	
Order Amendment, Customer Updates, Pricing	10%
Deletes:	
Closing Customer Accounts, Item Discontinuation	2%

Figure 3.0: Benchmark Transaction Breakout

The block size chosen for the database was four Kilobytes. In all tests, the total capacity of the buffer cache was roughly two Gigabytes.

The primary metric of success in this suite of tests is the ability to improve transaction response times by only changing how the buffer pools are configured while keeping the total capacity of the cache constant.

The test suite was first executed with no specialized buffer pool tuning.

Default Buffer Pool Test

In order to analyze default buffer cache effectiveness of Oracle9i with the test workload described above, the initialization parameters DB_KEEP_CACHE_SIZE and DB_RECYCLE_CACHE_SIZE were simply commented out.

At the end of the thirty-minute run, the following data was collected.

First, the *hit_ratio.sql* script was executed to determine the buffer cache-hit ratio. Figure 3.1 contains the result that shows a cache-hit ratio of 87.9%[1]. As suggested in the introduction section of this paper, most tuning efforts are concluded once this level of cache-hit ratio has been achieved. However, a closer look at cache efficiency is needed.

```
LOGICAL  READS
--------------
      72701090

PHYSICAL READS
--------------
       8787180
```

Figure 3.1: Cache-hit ratio for default buffer pool test is 87.9%

Using the *object_reads.sql* script, the next performance aspect to consider is the number of datafile reads per database object. Figure 3.2 reveals that 53.5% of all physical reads originate from a combination of the WAREHOUSE and CUSTOMER tables[2].

[1] 1 – (8787180 / 72701090)
[2] WARE (2,716,321) + CUST (1,992,713) = 4,709,034 / TOT_PHYS_READS (8,787,180)

```
TABLESPACE_NAME                      READS
------------------------------ ----------
WAREHOUSE                          2716321
CUSTOMER                           1992713
ITM_IDX                            1466615
ITEM                               1324614
ORD_IDX                             451631
ORDERS                              303800
NAME_IDX                            153945
PRODUCT                             112353
WHR_IDX                             106085
ROLL_2                               59901
ROLL_1                               58980
CUS_IDX                              26312
```

Figure 3.2: Number of physical reads per database object

This attribute is not necessarily troubling provided the application concentrates on small portions of these tables. Database objects that are causing reduced cache efficiency tend to exhibit two main characteristics. First, they will account for large percentages of all physical reads. Second, once the blocks are in the cache they will not be shared by other processes and will therefore be quickly replaced based upon the LRU policy. The *cache_content.sql* script reports the number of blocks in the cache from each database object. Output from this script appears in Figure 3.3.

Any table or index sustaining a large number of reads should also have a commensurate presence in the cache. If not, it is safe to surmise that the blocks are being read at random and therefore are not being shared. Figure 3.3 reveals that the WAREHOUSE and CUSTOMER tables combined only represent 28% of all buffers in the cache[1] while Figure 3.2 shows reads against these two tables account for 54% of all physical reads.

```
POOL_NAME OBJECT                       BLOCKS
--------- ------------------------ ----------
DEFUALT   WAREHOUSE                     86273
          PRODUCT                       63843
          ITM_IDX                       57925
          CUSTOMER                      54943
          WHR_IDX                       51725
          ORD_IDX                       46701
          ITEM                          44929
          CUS_IDX                       18223
          ORDERS                        10844
```

Figure 3.3: Buffer cache content by object

[1] Note, Figure 3.3 does not itemize the entire 500,000 buffers for the sake of brevity. Figure 3.3 accounts for the main application tables and indexes. The 28% WAREHOUSE + CUSTOMER cache footprint is 28% of the entire 500,000 buffers.

Given the tendency of the test application to randomly access the WAREHOUSE and CUSTOMER tables, there are an inordinate number of reads against all the other objects. With OLTP workloads, index blocks are generally revisited by other processes, however given the pervasiveness of the WAREHOUSE and CUSTOMER tables they are getting pushed out of the cache. Indeed, Figure 3.3 suggests that index blocks account for only roughly 35% of the entire buffer cache[1].

Performance Summary with Default Buffer Cache

Given the inefficient cache profile for index buffering with the test application, the benchmark performance achieved was limited to 858 transaction per second with an average response time of .11 seconds.

Configuring Multiple Buffer Pools

Given the random access nature of the WAREHOUSE and CUSTOMER tables in the test application, the cache effectiveness for INDEX buffering is compromised. With Oracle9i technology, there are two typical methods for addressing this issue. One technique is to utilize a small RECYCLE buffer pool and force all buffering of the WAREHOUSE and CUSTOMER tables through this pool. The other approach is to utilize a KEEP buffer pool that will host mostly indexes. The later is the method chosen for this study.

With Oracle9i, configuring multiple buffer pools consists of two main challenges. The first step is to determine which objects to assign to each buffer pool. The second step is to determine the size of each buffer pool. Monitoring cache content with the *cache_content.sql* script provides data showing how many blocks from each object are in the cache. However, the more elusive information is how many blocks from each object *need* to be in the cache.

When utilizing a KEEP buffer pool it is generally acceptable to assign the most active indexes to it. However, it may not be appropriate to limit the KEEP buffer pool to indexes alone. In the case of the test workload, the PRODUCT table is also a candidate for the KEEP buffer pool. Although it is not intuitive, the data collected from the default SGA tests indicates the PRODUCT table has an extremely high access rate in the cache. Figure 3.2 reveals that the PRODUCT table only represents 1.3% of the total physical reads yet Figure 3.3 suggests it accounts for 13% of all buffers in the cache. This irregular ratio of physical reads to cache presence is the essential characteristic of a high cache-hit rate object. That is, while not suffering a significant portion of physical reads it still accounts for significant cache

[1] All indexes in the benchmark application have a suffix of IDX and are stored in tablespaces named after the table they contain. Hence, reads accounted against the ITM_IDX tablespace are the results of query plans that access the ITM_IDX index

presence. For this sake, the PRODUCT table will also be assigned to the KEEP buffer pool in the multiple buffer pool test.

For this case study, the absolute number of buffers allocated to the KEEP buffer pool is based upon the cache data from the default SGA test in Figure 3.3. The CUSTOMER and WAREHOUSE tables account for 28% of all buffers in the cache. Since these are the tables we want to exclude from the KEEP buffer pool, the 28% would typically be configured as the remainder after the KEEP pool is allocated. However, the WAREHOUSE table is 35% (86273 buffers) more pervasive than the second runner-up which is the PRODUCT table (63843 buffers). To compensate for this effect, the size of the KEEP pool was adjusted by roughly the same 35%. Therefore, the KEEP buffer pool size chosen was 80% of the entire buffer cache as depicted in Figure 4.0.

```
db_cache_size = 2000M
db_keep_cache_size = 1600M
```

Figure 4.0: Configuration parameters for multiple buffer pools test

Multiple Buffer Pools Test

The Oracle instance was first booted with the buffer pool configuration listed in Figure 4.0. The commands in Figure 5.0 were then executed to assign the desired objects to the KEEP buffer pool.

```
ALTER TABLE product   BUFFER POOL KEEP;
ALTER INDEX itm_idx   BUFFER POOL KEEP;
ALTER INDEX ord_idx   BUFFER POOL KEEP;
ALTER INDEX whr_idx   BUFFER POOL KEEP;
ALTER INDEX cus_idx   BUFFER POOL KEEP;
ALTER INDEX prd_idx   BUFFER POOL KEEP;
```

Figure 5.0: Commands to associate objects with a KEEP buffer pool

Running the test workload under this buffer cache configuration yielded substantially improved cache efficiency while offering an overall cache-hit ratio of 89% - a 1% variance from that achieved in the default buffer cache test.

Figure 5.1 shows the cache-hit data using *hit_ratio.sql* script.

```
LOGICAL  READS
--------------
      78517177

PHYSICAL READS
--------------
       8564204
```

Figure 5.1: Cache-hit ratio for multiple buffer pools is 89%

Most notable was the improvement in index block caching. While the default buffer cache test results show that index blocks represent 35% of the whole, with a KEEP buffer pool that figure is improved by 91%. Figure 5.2 shows that index presence in the cache is increased to a respectable 67%. Moreover, the caching of ITM_IDX index alone improved by 260% from the 11% achieved with a default cache to the 39.6% seen in Figure 5.2.

```
POOL_NAME OBJECT                         BLOCKS
--------- ------------------------ ----------
DEFAULT   WAREHOUSE                     24225
          ITEM                          23409
          CUSTOMER                      13586
          ORDERS                         6140
KEEP      ITM_IDX                      198207
          PRODUCT                       63980
          ORD_IDX                       62661
          WHR_IDX                       52371
          CUS_IDX                       18255
          PRD_IDX                        4526
```

Figure 5.2: Buffer cache content by object with multiple buffer pools

Another dramatic effect was the reduction in physical reads for index objects. Data from Figures 3.1 and 3.2 show that with a default buffer pool, the 2,204,588 physical disk reads for index objects accounted for roughly 25% of the total. With a KEEP buffer pool the total physical reads for index objects was 1,056,471, as shown in Figure 5.3, representing only 12% of the total reads - a 52% improvement in buffering index blocks.

```
TABLESPACE_NAME                     READS
------------------------------ ----------
WAREHOUSE                         3180670
CUSTOMER                          2145999
ITEM                              1404929
ITM_IDX                            667863
ORDERS                             318160
ROLL_1                             182327
ROLL_2                             180950
NAME_IDX                           166831
ORD_IDX                            131900
PRODUCT                             80686
WHR_IDX                             68408
CUS_IDX                             21469
```

Figure 5.3: Physical reads per database object with multiple buffer pools

Additional performance gains were realized by assigning the PRODUCT table to the KEEP buffer pool. Figure 3.2 shows that in the default buffer cache case

112,353 disk read operations occurred for the PRODUCT table. That figure was reduced by roughly 28% down to 80,686 reads.

Using multiple buffer pools usually results in wide variation in physical reads for at least certain tables or indexes. In the case of this performance study, the tradeoff was increased physical reads on the WAREHOUSE and CUSTOMER tables for the sake of better cache efficiency. While the sum of physical disk reads for WAREHOUSE and CUSTOMER increased 13% from the default buffer cache case as seen in Figure 5.4, the total physical reads actually decreased by approximately 2.5% as seen in Figures 3.1 and 5.1.

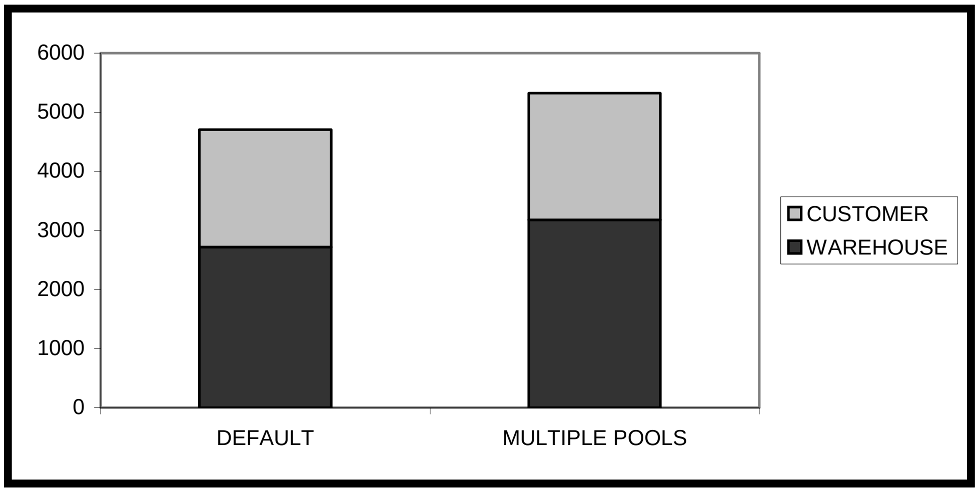

Figure 5.4: Comparison of CUSTOMER and WAREHOUSE reads

Performance Summary with Multiple Buffer Pools

The improved cache efficiency, due to utilizing multiple buffer pools, resulted in a performance increase of 11% for a transaction per second rate of 952. The average response time improved by 13% - down to .096 seconds.

Summary

With multiple buffer pool support in Oracle9i, traditional performance tuning methodology is not always sufficient. As demonstrated with the performance study in this whitepaper, additional performance is possible even after reports of extremely high cache-hit ratios - provided there is remaining system bandwidth.