

# Multiple Buffer Pools

Agustin Amador

*This paper provides an overview of the buffer cache in Oracle7 and the different features that Oracle8 has added to improve buffer cache utilization. In Oracle7 the buffer cache is organized as one single buffer pool to keep/manage segment blocks (tables, indexes and clusters) in memory. Oracle8 introduces a new feature that allows users to use multiple buffer pools for different segments.*

*We describe the necessity of having multiple buffer pools and the different types of buffer pools provided in Oracle8. We also show how to configure and use multiple buffer pools as well as how to get the statistics to tune each of these pools. Lastly we give some internals about this new feature.*

**Buffer Cache Overview**

**Multiple LRU Latches**

**Multiple Buffer Pools**

**Using multiple buffer pools**

- Initialization parameters
- Example of Buffer Pools
- Guidelines for configuring initialization parameters
- Assigning a buffer pool to a segment
- Gathering statistics
- Multiple buffer pools internals

**Conclusions and Future Directions**

**References and Acknowledgments**

Copyright © Oracle Corporation 1998. All rights reserved. Printed in the USA.  
Author: Agustin Amador

Oracle is a registered trademark of Oracle Corporation. Oracle7 and Oracle8 are trademarks of Oracle Corporation. All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

**Disclaimer:** The algorithms mentioned in this paper are subject to change in newer Oracle releases.

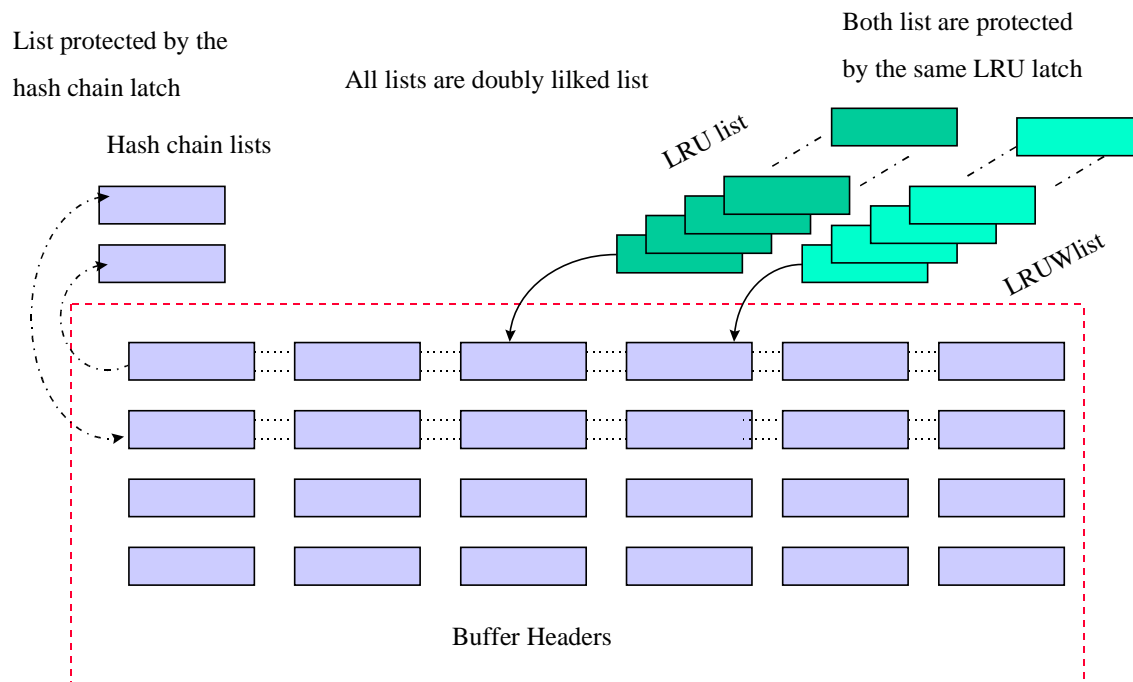
## Buffer Cache Overview

Oracle, like many other products (i.e., operating systems), uses a buffer cache to manage data blocks in memory. The buffer cache holds copies of data blocks read from the data files comprising the database. The buffer cache is located in the System Global Area (SGA) and is shared by all processes connected to an instance. Some advantages of using a buffer cache are eliminating physical I/O on frequently accessed blocks, providing fast access path to find block(s) in memory and help in maintaining concurrency control and multi-version consistency on blocks.

There are three main lists used to organize the buffers in the Oracle buffer cache: the dirty list (also called the write list or LRUW), the least recently used list (also called the replacement list or LRU) and the hashed chain list. The LRUW list holds dirty buffers, that is, is a buffer that has been modified but has not been written to disk. The LRU list holds free and pinned buffers as well as dirty buffers that have not yet been moved to the LRUW list. A free buffer is a buffer that has not been modified and is available for reuse. Pinned buffers are buffers that are currently being accessed so are not candidates for replacement. The hashed chain list holds the same buffers as the LRU and LRUW lists, but buffers on this list are arranged depending on their data block addresses (DBAs); it is essentially used to cache blocks in the buffer pool.

When a user process needs to access a block, it first looks in the hashed chain list to see if the block is already in memory. If found, the block can be immediately used; if not, the block has to be read from a datafile on disk into a buffer in the cache. Before a block can be read into a free buffer needs to be identified and pinned.

Figure 1 depicts how a buffer cache is organized.



The number of hashed chain lists is given by the `_DB_BLOCK_HASH_BUCKETS` init.ora parameter. Each element of the LRU and LRUW lists points to any one buffer header within the chain list. A buffer can be in the LRU list or LRUW list but not in both.

When a process is looking for a free buffer it starts scanning from the LRU list tail (also called least recently used end). If a free buffer is found then the block is read into it and the buffer is placed at the LRU list head (also called most recently used end, or MRU). There is a particular case when new used buffers are placed at the LRU end of the list; this is when new buffer comes from blocks read through a full table scan and the table is not considered to be a small table.

The init.ora parameter `_SMALL_TABLE_THRESHOLD` is used by Oracle to determine whether or not a table is handled as a small table. By default this parameter is initialized to  $\max(4, \text{DB\_BLOCK\_BUFFERS}/50)$ ; in other words, a table is considered a small table if the number of blocks that it has is less than the 2% of the buffer cache size. For example, if the buffer cache has 1000 blocks (`DB_BLOCK_BUFFERS = 1000`) then a small table must have 20 or less blocks. Hence new used buffers that come from a table larger than 20 blocks are placed at the LRU end of the list to force this buffer to be moved out quickly from the cache, letting other more frequently used buffers to stay in memory.

The way information behaves in memory can be changed by using the `CACHE` clause during segment `CREATION` (`ALTER TABLE / CREATE TABLE CACHE` option) or with a `CACHE` hint in a query block. For example:

```
SQL> ALTER TABLE dept CACHE;
SQL> SELECT /*+ CACHE(a) */ ename, deptno FROM emp a;
```

Blocks of a table having the `CACHE` attribute on are stored on the MRU end of the LRU list during full table scans. Therefore, this option should be used only for small tables to prevent all other buffers from being moved out from the cache. The `CACHE` clause was introduced as a first attempt to better manage segments depending on their replacement policy and based on object(table) level control. However, it is still possible that a buffer for a `CACHED` segment be aged out from the buffer cache when a large (say twice the size of buffer cache) table is updated and blocks goes through the buffer cache. In other words, the `CACHE` option on the table does not guarantee that a table will be cached in the buffer pool.

A process needs to acquire a latch whenever it needs to access any of the lists described before. A latch is a low level serialization mechanism used to protect a region of code from concurrent execution. These are often used when the code needs to access shared data in the SGA but is performing an operation which needs to be done atomically. Both LRU and LRUW are protected by the same latch called the `LRU LATCH`.

## Multiple LRU Latches

Before Oracle 7.3 only one `LRU LATCH` could be defined to manage the whole buffer cache. However, on machines with a large number of CPUs and a high volume of concurrent users or operations using Parallel Query Option (PQO), severe LRU latch contention may be observed. In addition, the DBWR had to contend with other foreground or slave processes for the LRU latch for writing the dirtied blocks in the cache to disk. Starting Oracle 7.3, a new feature was added that allowed users to configure more than one `LRU LATCHES`.

Multiple latches are implemented using a new concept, *set*, which was also introduced in Oracle 7.3. A set is the pair of LRU and LRUW lists protected by the same latch. The number of sets (i.e., `LRU LATCHES`) can be defined via the init.ora parameter `DB_BLOCK_LRU_LATCHES`, this parameter is initialized by default to  $\#CPU's / 2$  and is limited to  $\#CPU's \times 2$  in Oracle 7.3 and to  $\#CPU's \times 6$  in Oracle 8.0.3. If the a value for `DB_BLOCK_LRU_LATCHES` is specified, Oracle does some internal sanity checks to prevent inappropriate values being used. A set must have at least 50 buffers to be protected by his own latch.

If more than one set is used, buffers are assigned to the lists in a round robin fashion, this cause all the buffers to be spread evenly across the lists. Also, when a user process want to get a free buffer it is assigned to a set in a random

fashion. If it could not get the latch then it start to look into the other sets in a round robin fashion until a latch is gotten or all the sets are visited. If all the sets have been visited and process could not get a latch then it is counted as a latch miss. Figure 2 shows a buffer cache configuration using multiple LRU LATCHES as well as multiple buffer pools.

The following query can be used to determine if you are having latch contention in any of the sets.

```
SELECT child#, sleeps/gets ratio
FROM V$LATCH_CHILDREN
WHERE name = 'cache buffers lru chain';
```

The ratio must be less than 1%; if it is above 1% then you should increase the number of sets.

All the threshold parameters for the DBWR/buffer cache remain the same except that they are adjusted by considering the number of sets. For example, the number of blocks a user process must scan before to post DBWR to make space on the list is determined by `_DB_BLOCK_MAX_SCAN_CNT / #sets`.

Lastly, Oracle8 allows a user to configure more than one buffer pool. We can configure up to three different types of buffer pools, and each buffer pool can have its own number of sets. The details of this feature are explained in the rest of this paper.

## Multiple Buffer Pools

Users have seen that Oracle segments (tables, indexes, etc.) must be treated differently depending on usage. For example, having blocks in memory from a table that is accessed very often differs from having blocks from a table accessed only at application initialization. We would like the blocks in the first case to be in memory as much as possible to eliminate the necessity of reading the blocks from disk; sometimes, however, these blocks have to be taken out from memory to create space for the blocks in the second case.

Oracle addressed part of this problem prior to Oracle7.3 by allowing users to specify a `CACHE` clause during segment creation. Blocks of segments using this option were loaded at the most recently-used end of the LRU list, so they were kept in memory as long as the space was needed to load other blocks. If a large table (for example, twice the size of the buffer pool) is accessed frequently in a random fashion, it is very likely that blocks from the `CACHED` table will be removed from memory. If we could segment the I/O on the `CACHED` table and the large table into different buffer pools then we could increase the probability of the `CACHED` table being in memory for a longer period of time.

With multiple buffer pools, we can assign segments to different pools depending how they are referenced and assign objects (tables) to those pools. Thus by segregating the I/O on objects to different buffer pools based on objects' usage pattern the number of actual physical I/Os are reduced and the utilization of buffer cache increased.

## Using multiple buffer pools

There are three different types of buffer pools that can be configured: `KEEP`, `RECYCLE` and `DEFAULT`. The same internal algorithms used to control a single buffer pool are used to control each type of pool; in other words the `keep`, `recycle` and `default` pools are treated in the same manner as if they were a single pool.

A `KEEP` buffer pool must be used when there are segments that have to be kept in memory as long as possible. Segments that are frequently accessed and that are 10% the size of the cache are good candidates to be assigned to this pool. However like the `CACHE` option, having an object in the `KEEP` pool does not guarantee it from being flushed to the disk. Hence for deriving maximum benefit from the `KEEP POOL`, the `KEEP` pool must be sized appropriately. The `KEEP` pool size should be greater than at least the sum of all the object sizes that are intended to be kept in the `KEEP` pool.

The RECYCLE pool is intended to have blocks of segments that are rarely accessed or large segments (twice the size of the buffer pool) that are accessed in a random fashion.

Segments that are not assigned to any of the previous pools are kept on the DEFAULT buffer pool, which always exists in every instance. It is equivalent to the single buffer cache in Oracle7.

There is no requirement for any buffer pool to be defined for another pool to be used.

## **Initialization parameters**

There are two new initialization parameters used to configure the pools, BUFFER\_POOL\_KEEP and BUFFER\_POOL\_RECYCLE. We also must consider the parameters that already exist, DB\_BLOCK\_BUFFERS and DB\_BLOCK\_LRU\_LATCHES in configuring the buffer pools.

The syntax for the new parameters is:

```
BUFFER_POOL_KEEP = (buffers:<value>,lru_latches:<value>) or  
BUFFER_POOL_KEEP =<value>  
BUFFER_POOL_RECYCLE =(buffers:<value>,lru_latches:<value>) or  
BUFFER_POOL_RECYCLE =<value>
```

You can specify the number of blocks as well as the number of LRU latches assigned to each pool (first option) or you also can specify just the number of the blocks (second option); in the latter case, only one LRU latch is assigned to the pool.

We cannot define the blocks and the number of LRU latches for the DEFAULT pool explicitly. Rather, those values are calculated from the total number of blocks (i.e. DB\_BLOCK\_BUFFERS) and total number of LRU latches (i.e. DB\_BLOCK\_LRU\_LATCHES) minus the blocks and lru latches assigned to the KEEP and RECYCLE pools

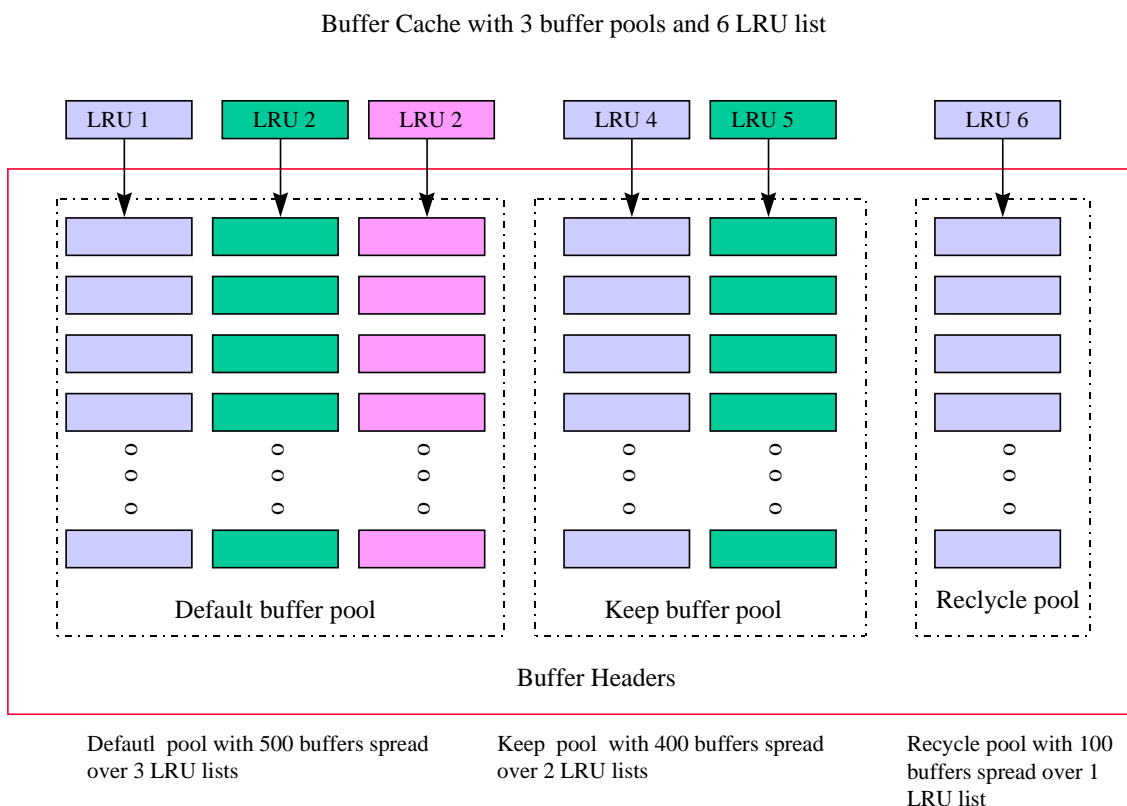
## Example of Buffer Pools

Let say we have the following init.ora file configuration:

```
DB_BLOCK_BUFFERS=1000
DB_BLOCK_LRU_LATCHES=6
BUFFER_POOL_KEEP =(buffers:400, lru_latches:2)
BUFFER_POOL_RECYCLE=100
```

We will have a SGA with three buffer pools, KEEP pool using 400 blocks and 2 LRU latches, RECYCLE pool using 100 blocks and 1 LRU latch (default number of latches) and DEFAULT pool using 500 blocks (1000 - 400 - 100) and 3 LRU latches (6 - 2 - 1)

Figure 2 depicts this configuration. (We omit LRUW and hashed chain lists to simplify the diagram.)



Blocks are spread uniformly over the LRU queues assigned to each pool. In our example, DEFAULT queues (1, 2 and 3) have 167, 167 and 166 blocks respectively and KEEP queues (4 and 5) have 200 blocks each; RECYCLE queue has 100 blocks.

Each queue must have at least 50 blocks; if we configure a pool with less blocks then an error is signaled (e.g., BUFFER\_POOL\_KEEP =(buffers:100, lru\_latches:3) ). Therefore, we have to be aware of all initialization parameters involved when multiple buffer pools are configured.

## Guidelines for configuring initialization parameters

### DB\_BLOCK\_LRU\_LATCHES

This parameter must be set at least to  $1 + \text{\#latches for keep pool} + \text{\#latches for recycle pool}$ . The maximum number of LRU latches that can be specified is limited to  $2 \times \text{\#CPUs} \times \text{\#possible buffer pools}$ , in Oracle8 release 8.0.3 is limited to  $6 \times \text{\#CPUs}$  (we can have 3 different pools).

### DB\_BLOCK\_BUFFERS

This parameter must be set at least to  $50 \times (\text{DB\_BLOCK\_LRU\_LATCHES} - \text{\#latches for keep pool} - \text{\#latches for recycle pool}) + \text{\#buffers for keep pool} + \text{\#buffers for recycle pool}$

If either blocks or LRU latches assigned are outside of the limits, the error "ORA-00378: buffer pools cannot be created as specified" is signaled, and we have to look into the alert file to see what is causing the error. If in our example we change DB\_BLOCK\_BUFFERS value from 1000 to 600 the following lines are shown in the alert.log file:

```
Total number of buffers (600) is less than buffers required for buffer pools
Number of buffers allocated to BUFFER_POOL_KEEP = 400
Number of buffers allocated to BUFFER_POOL_RECYCLE = 100
Default buffer pool needs at least 50 buffers per lru latch
Number of LRU latches allocated to default buffer pool = 3
```

Also, if we specified BUFFER\_POOL\_KEEP =(buffers:100,lru\_latches:3), we will get "Incorrect parameter specification for BUFFER\_POOL\_KEEP" message signaled in the alert.log file.

## Assigning a buffer pool to a segment

A new storage clause parameter is included to assign a buffer pool to a segment. The BUFFER\_POOL parameter is used to define a default buffer pool for a segment. All blocks in the segment are stored in the specified pool. If a buffer pool is defined for a partitioned table or index, the partitions inherit the buffer pool from the table or index definition unless overridden by a partition level-definition.

BUFFER\_POOL is not valid for creating or altering tablespaces or rollback segments, nor is it permitted for a clustered table. The buffer pool for a clustered table is specified at the cluster level.

Values that BUFFER\_POOL can have are KEEP, RECYCLE or DEFAULT. Parameter and values are case insensitive. In the next example we define keep\_table to be stored in the KEEP pool and blocks belong to the recycle\_table to be stored in the RECYCLE pool.

```
CREATE TABLE keep_table ( n number(2) , c varchar2(10))
STORAGE (buffer_pool KEEP);
ALTER TABLE recycle_table storage (buffer_pool RECYCLE);
```

## Dictionary views

A new column called BUFFER POOL was added to some dictionary views to indicate the default buffer pool for a given segment. Example of these views is:

USER\_TABLES, DBA\_SEGMENTS, USER\_TAB\_PARTITIONS and ALL\_INDEXES.

In addition two new V\$ views have been added:

V\$BUFFER\_POOL and V\$BUFFER\_POOL\_STATISTICS.

V\$BUFFER\_POOL

This view displays information about all buffer pools available for the instance. The "sets" pertain to the number of LRU latch sets.

Column	Datatype	Description
INST_ID	NUMBER	Instance ID
ID	NUMBER	Buffer pool ID number
NAME	VARCHAR2	Buffer pool name
LO_SETID	NUMBER	Low set ID number
HI_SETID	NUMBER	High set ID number
SET_COUNT	NUMBER	Number of sets in this buffer pool. This is HI_SETID - LO_SETID + 1
SIZE	NUMBER	Number of buffers allocated to the buffer pool
LO_BNUM	NUMBER	Low buffer number for this pool
HI_BNUM	NUMBER	High buffer number for this pool

## V\$BUFFER\_POOL\_STATISTICS

This view display statistics for each buffer pool. It is not created by default: script catperf.sql must be run from server manager as internal to create it.

Column	Datatype	Description
ID	NUMBER	Buffer pool number
NAME	VARCHAR2	Buffer pool name
SET_MSIZE	NUMBER	Buffer pool maximum set size
CNUM_REPL	NUMBER	Number of buffers on replacement list
CNUM_WRITE	NUMBER	Number of buffers on write list
CNUM_SET	NUMBER	Number of buffers in set
BUF_GOT	NUMBER	Number of buffers gotten by the set
SUM_WRITE	NUMBER	Number of buffers written by the set
SUM_SCAN	NUMBER	Number of buffers scanned in the set
FREE_BUFFER_WAIT	NUMBER	Free buffer wait statistic
WRITE_COMPLETE_WAIT	NUMBER	Write complete wait statistic
BUFFER_BUSY_WAIT	NUMBER	Buffer busy wait statistic
FREE_BUFFER_INSPECTED	NUMBER	Free buffer inspected statistic
DIRTY_BUFFERS_INSPECTED	NUMBER	Dirty buffers inspected statistic
DB_BLOCK_CHANGE	NUMBER	Database blocks changed statistic
DB_BLOCK_GETS	NUMBER	Database blocks gotten statistic
CONSISTENT_GETS	NUMBER	Consistent gets statistic
PHYSICAL_READS	NUMBER	Physical reads statistic
PHYSICAL_WRITES	NUMBER	Physical writes statistic

### Gathering statistics

Currently the scripts utlstat.sql and utlstat.sql have not been updated to work with multiple buffer pools. However, you can easily write your own scripts based on the v\$buffer\_pool\_statistics view. You can take two different snapshots of this view and the use the delta values of the statistics columns to calculate the desired statistic.

On instances with only one buffer pool we wanted to have a hit ratio greater that 90% if possible. Using multiple pools we can still calculate the hit ratio for each pool, but we should not expect a 90% hit ratio on all the pools that is the case for the RECYCLE buffer pool. We could still size the KEEP and DEFAUL pools based on the hit ratio and we could determine the proper size for the RECYCLE pool based on the "Free buffer wait" statistic.

The KEEP pool hit ratio can be calculated using the formula:

$$\text{hit ratio} = 1 - \frac{\text{PHYSICAL\_READS}}{(\text{DB\_BLOCK\_GETS} + \text{CONSISTENT\_GETS})}$$

where values are obtained from v\$buffer\_pool\_statistics. Remember to select only the rows for the KEEP pool:

```
SELECT name, db_block_gets , consistent_gets, physical_reads
FROM V$BUFFER_POOL_STATISTICS
WHERE name = 'KEEP';
```

If the RECYCLE pool is too small, we can see higher values for the wait event "Free buffer wait"

```
SELECT name, free_buffer_wait
FROM v$buffer_pool_statistics
WHERE name ='recycle';
```

Keep in mind that the final goal is to reduce the number of physical I/Os.

## Multiple buffer pools internals

Two new internal tables were added to support this feature: X\$KCBWBPD and X\$KCBWDS.

X\$KCBWBPD (buffer pool descriptor) is the base table for the V\$BUFFER\_POOL view, which depicts how buffer pools are configured. It specifies how many sets (i.e. LRU latches) are assigned to each pool (SET\_COUNT column); the low and high set id (LO\_SETID and HI\_SETID columns); and the number of buffers (blocks) that each buffer pool has (BUFFERS column). All buffers in the buffer cache are numbered from 0 to db\_BLOCK\_buffers - 1 (BUF# column on X\$BH). LO\_BNUM and HI\_BNUM columns are the low and high buffer numbers associated with the pool.

Since Oracle overrides the init.ora parameters setting for inappropriate settings for each set, we can determine how the sets are configured by the following query.

```
SQL> SELECT name, set_count, lo_bnum, hi_bnum, buffers FROM v$buffer_pool;
```

```
SQL> select name, set_count, lo_bnum, hi_bnum, buffers from v$buffer_pool;
```

NAME	SET_COUNT	LO_BNUM	HI_BNUM	BUFFERS
KEEP	2	0	399	400
RECYCLE	1	400	499	100
DEFAULT	3	500	999	500

Default pool has 500 buffers spread over 3 LRU latches; buffers are numbered from 500 to 999.

We also can write a query that shows which segments are stored in which buffer pool and how many buffers the segment is using.

```
SQL> select 'KEEP' POOL, o.name, count(buf#) BLOCKS
from obj$ o, x$bh x
where o.dataobj# = x.obj
and x.state !=0
and o.owner# !=0
and buf# >= (select lo_bnum from v$buffer_pool where name='KEEP'
and buffers > 0)
and buf# <= (select hi_bnum from v$buffer_pool where name='KEEP'
and buffers > 0)
group by 'KEEP',o.name
union all
select 'DEFAULT' POOL, o.name, count(buf#) BLOCKS
from obj$ o, x$bh x
where o.dataobj# = x.obj
and x.state !=0
and o.owner# !=0
and buf# >= (select lo_bnum from v$buffer_pool where name='DEFAULT'
and buffers > 0)
and buf# <= (select hi_bnum from v$buffer_pool where name='DEFAULT'
and buffers > 0)
group by 'DEFAULT',o.name
union all
select 'RECYCLE' POOL, o.name, count(buf#) BLOCKS
from obj$ o, x$bh x
where o.dataobj# = x.obj
```

```

and buf# >= (select lo_bnum from v$buffer_pool where name='RECYCLE'
            and buffers > 0)
and x.state !=0
and o.owner# !=0
and buf# <= (select hi_bnum from v$buffer_pool where name='RECYCLE'
            and buffers > 0)
group by 'RECYLCE',o.name
/

```

POOL	NAME	BLOCKS
KEEP	EMP_KEEP	9
RECYCLE	RECYCLE_TABLE	2
DEFAULT	T2	2

X\$KCBWDS (set descriptor) was first introduced in Oracle7.3 is used to store statistics for each set. The V\$BUFFER\_POOL\_STATISTICS view is based on the X\$KCBWDDS table. From this table we can see how buffers are spread over sets.

```
SQL> SELECT set_id, set_msize FROM x$kcwds;
```

SET_ID	SET_MSIZ
1	167
2	167
3	166
4	200
5	200
6	100

From the previous output we can see how buffers are evenly spread over system sets. For example, sets 4 and 5 (which correspond to KEEP buffer pools) have the same number of buffers.

## Conclusions and Future Directions

Multiple buffer pool feature is an attempt to provide better control over buffer cache utilization for sophisticated users. Instead of a single buffer pool a user can now configure multiple buffer pools and assign objects to the pools based on the pattern of usage. The ability to segregate IO by object to different buffer pools based on usage pattern should result in improved system performance. There are a number of other enhancements that can be made to the multiple buffer pool feature in the future releases of Oracle 8/9. The ability to pin an object in memory to incur no I/Os, to specify different replacement strategies (round robin, random, etc.) for each pool, provide query/session level hint/control over objects and pools are some of them. All these enhancements when available in Oracle will make the utilization of buffer cache even more better and further improve system performance and throughput.

## References and Acknowledgments

Oracle confidential developments documents, by *Alex Ho*  
Oracle confidential development documents, by *Ahmed K. Ezzat*  
Oracle8 Server Concepts Release 8.0 June, 1997. Par No. A54644-01  
Oracle8 Server Tuning Release 8.0 June, 1997. Par No. A54638-01  
Oracle8 Server Reference Release 8.0 June, 1997. Par No. A54645-01  
To reviewers Prabhaker Gongloor, Nitin Vengurlekar and Vijay Lunawat for their helpful comments.