Cache Fusion Demystified

By Arup Nanda

Can you do any better than a 100MB buffer cache for a 100MB database? What about in a RAC environment? As Oracle's Cache Fusion technology unifies the buffer cache across all nodes in a cluster, running a four-node cluster with a 25MB buffer cache per node should be plenty, right? Do you believe that?

Don't believe it! Mostly, it does make sense to have a buffer cache larger than your database. That's especially true in a RAC environment due to the overhead from Cache Fusion. Cache Fusion is at the core of RAC, and understanding the concepts is crucial to understanding the overhead it imposes. As stated earlier, Cache Fusion presents a unified buffer cache to an Oracle session. In other words, it works under the covers to make the *relevant* version of the database block buffers available on the required node for the required operation (i.e. read or write). To achieve that, it has to perform a lot of processing, which we will deep dive in this article.

Before we begin, here are some things you need to know to size and tune the buffer cache in a RAC environment.

- How buffers are allocated and de-allocated
- The difference between a *consistent read (CR)* block and a *current buffer*
- What buffer locks are and how they differ from row locks
- The different types of buffer locks, such as *Exclusive Current*, *Shared* and *Null*
- How Cache Fusion gets buffers from the different instances in the cluster

Oracle's documentation of these critical aspects of buffer cache operation is pitiful, leading to a great deal of confusion and error in the field. Your ability to tune a RAC database depends heavily upon a good understanding of these aspects, and of buffer cache and Cache Fusion operation in general. If you want to learn more about these topics, please continue reading.

Buffer State

Before we get too far into the details, I want to introduce a very important topic—the Buffer State. Buffer state refers to whether a buffer contains the most current copy of a block, or a copy that is consistent with some past point in time. Buffer state drives the manner in which Cache Fusion finds and delivers a version of a specific database block to a user.

Initially, you need to be aware of two buffer states.

- Consistent Read: The buffer contains a copy of the database block as it was at a specific point in time to provide consistent query results to a database user. The block may or may not have been modified; however, a buffer in this state is not necessary for instance recovery.
- Current: The buffer contains the most recent copy of the block after an update.

When a user requests a copy of a block, the server process searches the local buffer cache. If a copy of the block is *not* found in the local cache, the user has two options:

- Get a copy of the block from disk; or
- Get a copy of the block from one of the other instances, if found there.

If a copy of the block is found in the local cache, the user has three options:

- Send the data from the buffer containing the copy to the user;
- Examine other caches for a copy of the block in a more compatible state; or
- Get a copy of the block from the disk anyway.

To decide which option to take, Oracle relies on many factors, one of which is the buffer state.

Here I need to digress and make sure you understand the difference between a *block* and a *buffer*. A block is the least addressable unit of a database and it's on the disk. Blocks are read into buffers, which are said to contain "copies of blocks." SQL statements operate on those buffers, not on the blocks themselves.

The first session to request data from a block will cause the block's contents to be read from disk into a buffer. When another session requests the same block's data, but as of a different time (i.e., as of a different system change number, or SCN.), the server process must provide a read consistent image of the buffer. That read consistent image may or may not be the one already in the cache. The server process may have to generate that read consistent image by creating a new copy of the buffer. Consequently, there can be several consistent read copies of a single block but only one current copy. This is completely logical: The word "current" indicates the buffer is the most recent; there can't be more than one current copy. When the buffer is requested for the intention of being updated, it must be requested in current mode.

Remember, in a RAC database there is more than one buffer cache, i.e. there are as many buffer caches as there are "live" RAC nodes, one per node. Each cache may contain its own current copy of a given buffer. In that case, each current buffer is said to be in Shared Current mode. The Shared Current mode guarantees that the copies are identical to each other. If a buffer is modified, then that one is more recent that the others. That buffer is said to be in Exclusive Current mode. Only one buffer in the entire cluster can be Exclusive Current for a given block. Furthermore, when a buffer is upgraded to Exclusive Current mode, then all other shared current buffers for that block can no longer be called current; they become consistent read, or CR copies.

Block and Row Relationship

As blocks contain rows, you might expect some sort of hierarchical relationship between block- and row-locks. No such relationship exists. Instead, block- and row-level locking take place independently of each other.

Suppose there is a two node RAC database in which there is a table with only four rows. Further suppose that all four rows fit into just one block. Figure 1 shows this database while a transaction is in process.

continued on page 8

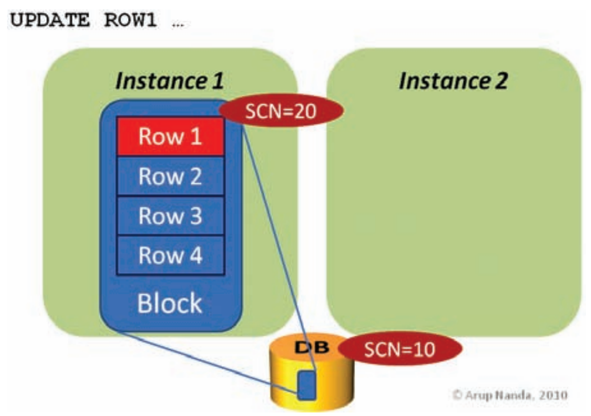


Figure 1. Four rows in one block during a transaction initiated from Instance 1.

The SCN number is 10 at the beginning of the transaction. A session connected to instance 1 updates only a single row of the table—Row 1. The server process will retrieve the block from the database in Current Mode (because the intention is to modify the block) and modify Row 1, as shown in Figure 1. The SCN number of the buffer now is 20. Instance 2 does not have the buffer because no one there has asked for any row in the block.

Now a session connected to instance 2 updates a different row—Row 2. It can do that because session 1 has placed a lock on Row 1, not on Row 2. The server process on instance 2 finds that a copy of the block is available in the buffer cache of instance 1. That server process requests a copy of the buffer in Current Mode (*current* mode, since the *intention* is to modify). The requested copy is transferred to instance 2 via cache fusion.

To modify the buffer, instance 2 must have the buffer to itself with no other instance potentially modifying it. Instance 2 accomplishes that objective by placing a lock on the buffer. In this case, instance 2 requests an Exclusive Current lock. Until it gets that lock, instance 2 is not allowed to modify the buffer and the session waits. After obtaining the lock, the mode of the buffer in the cache of the other instance becomes CR, as it is no longer current. Figure 2 shows the resulting state of affairs.

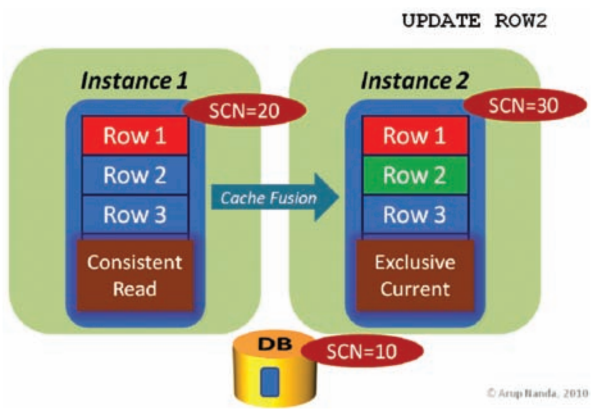


Figure 2. A copy of the block in Instance 2's buffer cache, ready for update.

Row 2 is modified. Instance 2's buffer cache now contains the most recent copy of the block. The block on disk represents SCN 10. The copy in instance 1's buffer cache represents SCN 20. The copy in instance 2's buffer cache is most current at SCN 30.

A checkpoint occurs, and the dirty buffer is written to disk as shown in Figure 3. The SCN number of the block on disk now matches the SCN of the copy in instance 2's buffer cache. The buffer in the cache is no longer exclusive; a process reading from either the cache or the disk will get the same result. exclusive current mode is no longer needed for that buffer. Thus, the buffer in instance 2's cache is demoted to consistent read mode, i.e. the Exclusive Current lock is removed.

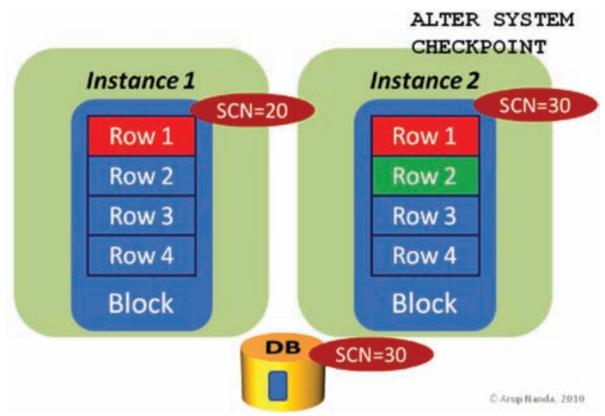


Figure 3. Buffer and SCN states after the block has been rewritten on disk.

When the buffer is demoted to CR, there is no need to have a buffer lock. In the cache, however, every buffer has a lock. When there is no lock on a buffer, the lock is said to be a Null Lock.

Now a new different session—we'll call it session 3—connected to instance 1 updates yet another row—Row 3. Instance 1 requests a copy of the buffer from instance 2. The copy in Instance 1's cache is made Exclusive Current, because the intention is to modify it. The buffer in instance 2, which was earlier in Exclusive Current, now becomes CR, as shown in Figure 4.

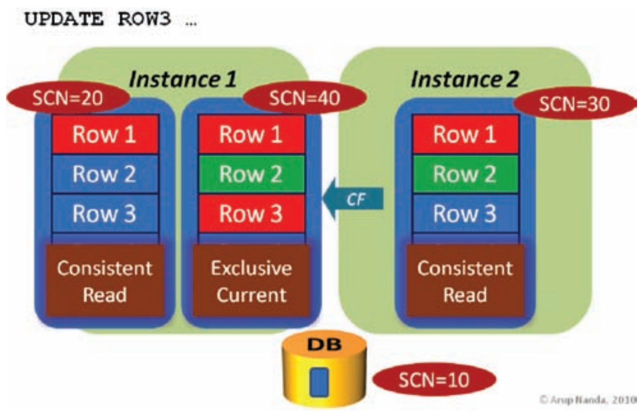


Figure 4. A second copy of the block in instance 1's buffer cache.

You can see from our discussion so far that buffer locks are the mechanism through which instances are provided with exclusive access to buffers for purpose of modifying those buffers. Buffer locks are also known as Parallel Cache Management (PCM) locks. They are different from row locks and completely independent from them as well.

Refer to Figure 4. The rows are locked as follows. The sessions involved happen to be spread across two instances, but the locks are owned by the sessions, not by the instances per se.

- Row 1 is locked by session 1.
- Row 3 is locked by session 3.
- Row 2 is locked by session 2.

The situation with the buffers is independent from that of the locks.

- There are two copies of the block, or two buffers on instance 1.
- One buffer on instance 1 is locked in Exclusive Current mode; the other with a Null lock (for the Consistent Read copy). The lock for guaranteeing an Exclusive Current copy of the buffer is known as XCUR.
- On instance 2, there is only one copy of the buffer with a NULL lock.

Remember that these buffer locks are different from the row locks. Had the sessions committed after every modification, there would not now be any row level locks, but the buffer locks would still be there. Similarly, had a checkpoint occurred after each modification, the buffer locks would have changed from XCUR to NULL, but the row locks would still be in place as they go away only after the corresponding transaction commits. The bottom line is that row locks protect the integrity of rows, and buffer locks the integrity of buffers. The two lock systems operate independently from each other.

Speaking of CR copies, there is a limit of six CR copies per block. When the seventh CR copy is necessary, Oracle ages out the oldest CR copy to make room for the new one. Otherwise, CR copies might proliferate, flooding the buffer cache and denying other blocks free buffers.

Past Image Mode

One more possible buffer mode is the Past Image mode. The term Past Image is not documented in Oracle Manuals. It's just widely understood and acceptable.

To understand Past Image mode, consider the possibility of instance recovery following the crash of instance 2 during the state shown in Figure 2. What is the impact on the block on the disk from instance 2's crash? The answer is that it has no impact. The buffer was in CR mode, i.e. it was not modified. Actually, it was modified, but the most recent copy is not in the buffer cache of that instance; it's in instance 1. So the instance recovery process will not have to consider the buffer at all. On the other hand, if instance 1 crashes, the instance recovery process will not need to even consider anything in any buffer cache; rather, it will merely get the relevant redo entries from the online redo log and apply them to the blocks in the database. This is due to the fact that the CR copy of the buffer is identical to the block on the disk and does not need to be flushed to the disk.

Consider the transition from Figure 2 to Figure 4. The buffer is flushed to disk in Figure 3, and then transferred to instance 1 in Figure 4. But what if the buffer is transferred from instance 2 to instance 1 before it is flushed to disk? Then what happens if instance 1 crashes? In that case, the CR buffer in instance 2 will not be the same as on the disk. Its SCN is 30 while that of the block on the disk is 10. If instance 1 crashes, the block on the disk must undergo instance recovery. However, a more recent copy of the block is available on instance 2, current up to SCN 20. Applying redo entries from SCN 10 to 30 is definitely more work than applying redo from only 20 to 30. Clearly, it makes a lot of sense to flush the buffer from instance 2 to disk and recover from that more recent image. Unfortunately, what I've just described will not happen because the state of the buffer is CR, and by its very definition the CR mode presumes the buffer to be in sync with the disk (i.e., that the buffer has not been modified).

To avoid the quandary I've just described, and to open up the possibility of saving on work during recovery, Oracle does not mark the buffer in instance 2 as CR, but rather as Past Image (PI). That mode means that the block has been modified in the current instance (and, therefore, is not CR), and modified later in another instance (therefore, not Exclusive Current). If recovery is required, that PI buffer can be used to reduce the number of redo entries that must be applied. The buffer remains in the PI state until it is flushed to the disk by a checkpoint, or until it you manually flush the buffer. Flushing the buffer makes its contents identical to the block on disk, converting the buffer into a regular, CR copy.

Cache Fusion Processes

The part of the RAC infrastructure that transfers buffers from one instance to the other is called the Global Cache Service (GCS). GCS makes sure buffers are served up as appropriate; but it does not know which instance has what type of lock on a given buffer. That information is provided by another component, Global Enqueue Service (GES). GES used to be called Dynamic Lock Manager (DLM) in previous versions of the database engine.

GES holds information about locks on the buffers. These locks are exposed through the view V$LOCK_ELEMENT, which is built upon the X$LE table. Another helpful view is V$BH, which shows the buffer headers in the buffer cache, along with the lock element on that buffer. If a buffer is locked, you can see the name of the lock element in the LOCK_ELEMENT column of V$BH. You can examine the state of all the buffers of an object and of the locks on those buffers, by issuing the following query:

```
select file#, block#,
          decode(class#,1,'data block',2,'sort block',3,'save undo block', 4,
'segment header',5,'save undo header',6,'free list',7,'extent map',
8,'1st level bmb',9,'2nd level bmb',10,'3rd level bmb', 11,'bitmap block',
12,'bitmap index block',13,'file header block',14,'unused',
15,'system undo header',16,'system undo block', 17,'undo header',
18,'undo block') class_type, status, lock_element_addr
from v$bh
where objd = <DataObjectId of the Object>
order by 1,2,3;
```

The output should tell you various buffers of the blocks of the segment in the instance and the state of the buffer. If a buffer is locked by either a shared or exclusive lock, the lock address is also shown, which when joined with V$LOCK_ELEMENT, shows the details on the lock.

Lock Queuing

To summarize, when an instance wants to change the state of a buffer from CR to Exclusive Current, or vice versa, it must get a lock on that buffer. Such a lock is called a Buffer Lock and it is different from a row lock. When a process wants to acquire a buffer in Exclusive Current state, it must get the XCUR lock on the buffer. If the buffer was already in CR mode, this additional restrictive locking is known as a lock upgrade. Similarly, when an instance wants to upgrade one of its own buffers to a higher state, which means the corresponding buffer on the other instance must be converted to a CR copy, the lock on the remote instance is converted down from XCUR in a process known as a lock downgrade. The lock is never said to go away in a downgrade; it simply becomes a NULL lock. As long as a buffer is there in one of the buffer caches, there is a lock on the buffer. In case of the buffer being in CR mode, the lock is NULL.

In the normal course of database operation, there will be a lot of requests to upgrade and downgrade locks on a specific buffer. To make sure the locking

continued on page 10

Cache Fusion Demystified *continued from page 9*

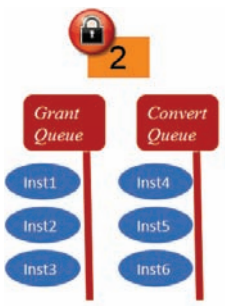
process is smooth, there has to be a queue. Each buffer in a RAC instance has two queues:

- Grant Queue – Contains incoming lock requests; and
- Convert Queue Holds the outgoing notifications, through which requesting processes are notified of having been awarded their locks.

A process wanting a lock to be downgraded or upgraded on a buffer must place a request in the Grant Queue, shown in Figure 5. When the request is fulfilled, notification is placed into the convert queue to let the process know that its request has been satisfied.

The Grant and Convert queues are memory structures. To replicate them on all instances would be time-consuming and would itself affect the locking process. Instead, the queues for a given buffer are kept in only one instance. This instance is known as the Master Instance of the buffer.

A buffer has only one Master Instance and each instance may hold queues of many buffers. When an instance requests a lock on a specific buffer, that instance's own Lock Management Service (LMS) process contacts the LMS process of the master instance for the buffer in question. When the request is fulfilled, the requesting instance gets notified via the LMS process.



The master instance of a buffer is not hard-coded. It may change. If an instance requests a buffer lock several times, but is not the master, then the cluster may make that instance the master of that buffer. This process of automatically changing the master instance is known as Dynamic Resource Mastering. (Buffers can be manually remastered as well).

When an instance wants to get a lock on a given buffer, it has to check with the master instance for that buffer. You may be wondering by now how the requesting instance knows where the master instance is located. The answer is that the requesting instance looks up the master instance in what is called a Global Resource Directory (GRD).

Think of the GRD as a phone book. When you want to use someone's office for a private meeting, you phone the person to find out if the office is available. To call, you need to know the phone number—which is in the phone book. You have a copy of the phone book, because it is replicated and sent to everyone. The GRD is replicated among all instances just as the phone book is given out to all phone company subscribers. Any instance can consult its copy of the GRD to determine the master instance for a given block. Then, a lock request can be sent to that master instance.

You can find out the master instance for a block yourself by executing a query against the X$ tables. To find the master instance of all the blocks holding part of a specific object, first find the Data Object ID of the object in question. Then issue the following query:

```
select  b.dbablk, r.kjblmaster master_node
from x$le l, x$kjbl r, x$bh b
where b.obj = <DataObjectId>
and b.le_addr = l.le_addr
and l.le_kjbl = r.kjbllockp
```

When a block is remastered, the master instance information is updated and replicated across all the instances. To prevent problems, the entire GRD is frozen until the updated information appears in all instances. This causes the sessions requesting blocks to wait until remastering is complete—another cause of performance issues.

Putting it all Together

An instance locks a buffer in one of two modes: Current or Consistent Read, depending upon whether the instance will modify the buffer. Every time a node wants a buffer from a remote node, the buffer is copied and then sent—a process known as CR processing.

There can be only one current state of a buffer in an instance in Shared Mode. In that case, the buffers on all instances are identical. If a node modifies a buffer, the node must acquire the buffer in Exclusive Current mode. RAC enforces the different modes by locking the buffers. For example, Exclusive Current mode is enforced by an XCUR lock on the buffer. When an instance requests a block for modification, the instance must look up that block's master instance in the Global Resource Directory. Then, the requesting instance sends a message to the master instance to get the lock. The request goes to the Grant Queue and is fulfilled in the order it was received. Until the request is received, the session requesting the buffer lock must wait.

You should also realize that for every block on disk, there can be more than one copy in the buffer caches of the various RAC instances. There can be one Shared Current copy, one Past Image copy and many CR copies. Thus, creating only 100MB of buffer cache per instance for a 100MB database will not be enough. Based on how often the data on the different instances is requested and how often checkpoints occur, the buffer cache requirement could be as much as six times the database size to be effective.

Putting everything together, you should be able to deduce that the best way to improve performance in a RAC database is to make sure that requests for a specific buffer go to only one instance—a technique known as Application Partitioning. However, such partitioning may make one node overly loaded while keeping other one relatively idle; so you should evaluate it carefully in the context of your specific application. In a later article, I will explain more on Application Partitioning and how to enable it in RAC.

About the Author

Arup Nanda (arup@proligence.com) has been an Oracle DBA for more than 15 years, with 10 years in RAC. He leads the global database architecture at a multinational company in the New York metro area. He has coauthored four books, written nearly 300 articles, presented some 100 sessions and blogs regularly at arup.blogspot.com. He is an Oracle ACE Director, a member of the Oak Table Network and was the DBA of the Year in 2003 by Oracle. He would like to take this opportunity to express his sincere gratitude to Jonathan Gennick for his detailed review and suggestions to improve this article.

Security Tip #40 | Understand the architecture – Understand exploits

Find exploit information and play with hacks on test databases.

- www.appsecinc.com
- www.red-databasesecurity.com
- www.oxid.it/cain.html
- www.petefinnigan.com