Oracle Real Application Clusters (RAC) Cache Fusion Performance Optimizations on Exadata

May, 2021

Public

Disclaimer

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle software license and service agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle. Due to the nature of the product architecture, it may not be possible to safely include all features described in this document without risking significant destabilization of the code.

Table of contents

Executive Summary

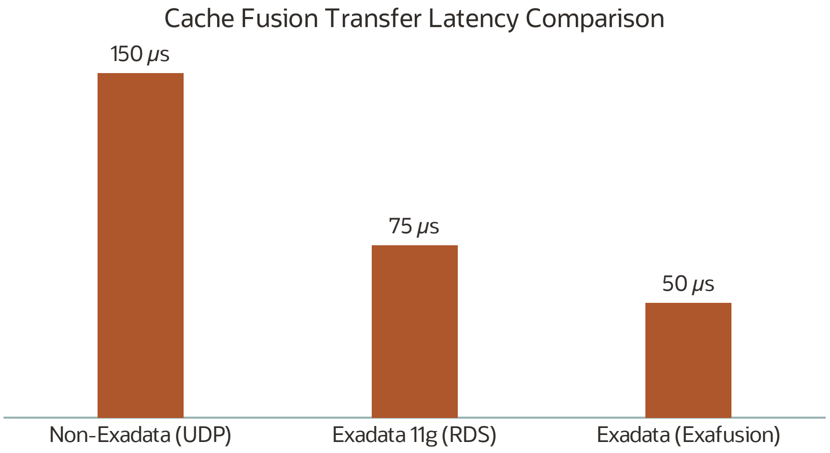
Oracle Real Application Clusters commonly referred to as Oracle RAC is an option to the Oracle Database that provides linear horizontal scalability and high availability. Oracle RAC Cache Fusion is a component of Oracle RAC responsible for synchronizing the caches among Oracle RAC instances making it possible for applications to seamlessly utilize the computing resources of all the Oracle RAC instances without making any changes. Cache Fusion utilizes a dedicated private network for cache synchronization. Application scalability therefore relies on the latency and bandwidth provided by the underlying private network.

Exadata, with its adoption of **advanced networking components like RDMA over Converged Ethernet (RoCE) or InfiniBand, enables Oracle to further improve performance and scalability.** In addition to benefiting from the improved wire speed of the underlying network, we re-engineered significant portions of Oracle RAC Cache Fusion layer to leverage the advanced protocols and RDMA capabilities available on Exadata. For example, on Exadata, Oracle RAC instances directly transfers buffers to the wire and bypasses the Operating System (OS) kernel. This results in block transfers that have ultra-low latency and that incur dramatically lower CPU cost. Oracle RAC on Exadata also uses new protocols that eliminate waits in the performance critical parts of transaction commits. This paper will explain these Exadata-specific optimizations that have been implemented since Oracle 12c.

List of Performance Optimizations

Exafusion

Traditionally, Oracle RAC messaging was implemented using the commonly used networking model using network sockets. In this model, all communications (sends and receives) would go through the OS kernel, thus requiring context switches and memory copies between user space and OS kernel for every RAC message being exchanged. Exafusion is the next generation networking protocol available on Exadata since 12c (on both RoCE and InfiniBand), which allows for **direct-to-wire messaging** from user space, **completely bypassing the OS kernel.** By eliminating the context switches and OS kernel overhead, Exafusion enables Oracle to process round trip messages in less than 50 µs (micro-seconds), **which is 3x faster than a traditional socket-based implementation, and a further 33% improvement compared to the first generation of Exadata** which used the RDS protocol for messaging. Additionally, the CPU cost associated with sending and receiving messages is lower with Exafusion, allowing for higher block transfer throughput and increased headroom in LMS processes before they could become saturated. **Faster messaging not only benefits runtime application performance, it also makes every Oracle RAC operation faster** - this includes dynamic lock remastering (DRM), Oracle RAC reconfiguration (associated with instance or PDB membership changes), and instance recovery.



The adoption of Exafusion is the foundation of subsequent performance optimizations for RAC on Exadata, including zero copy transfers and adoption of RDMA.

Exafusion and the subsequent optimizations described in this document do not require extra OS resources to operate. When Exafusion is enabled, one may notice that the IPC0 background process uses high RSS memory usage in “ps”, however this is due to the fact that Oracle instance registers (pins) all IPC buffers with the Host Channel Adaptor (HCA) on behalf of all processes running in the instance, and **does not indicate excessive memory usage or memory leaks.** Further details can be found in MOS note 2407743.1.

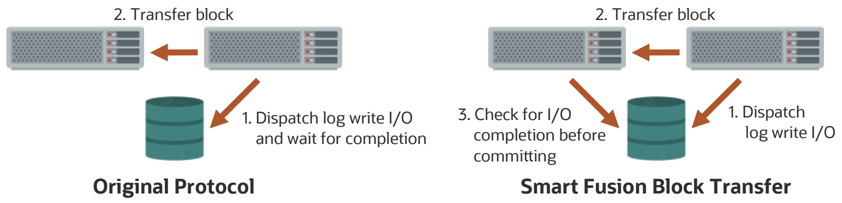
Zero Copy Block Sends

RoCE and InfiniBand network adapters support Zero Copy messaging. User space buffers are registered with the HCA and the HCA directly places the contents of user space buffers on the wire, unlike traditional messaging protocols where the OS kernel first makes a copy of the user space buffer and then places them on the wire. Since Oracle RAC 12c, we use this feature on Exadata for inter-instance communications. Elimination of the CPU cycles required for copying buffers **improved the transfer latencies by up to 5% compared to Exafusion without Zero Copy sends.**

Smart Fusion Block Transfer

Traditionally, Oracle RAC instance would have to wait for redo log flush to complete before sending a dirty block to another instance. This is a common access pattern in OLTP systems with frequent DML's. The redo flush is done to ensure database consistency in the event of an instance failure. This means that inter-instance transfer latency for frequently modified blocks which have redo pending was always dependent on redo flush I/O latency, and was subject to outliers caused by intermittent spikes in I/O performance.

Oracle RAC 12c utilizes Smart Fusion Block Transfer optimization, which allows an Oracle RAC instance to send the block once the **redo I/O is in-flight** to the Exadata storage server. Oracle RAC LMS process is permitted to initiate a block transfer before receiving I/O completion acknowledgment, allowing sessions on the requestor instance to start accessing that block while the redo I/O may still be pending. The requestor instance checks for I/O completion before it commits further changes to the same block. The committing process is required to wait for the "*remote log force - commit*" wait event if the I/O is yet to complete. This is a rare occurrence, which is only seen when there are extreme I/O outliers. Such I/O outliers are mostly eliminated on Exadata with the Smart Flash Logging feature. Smart Fusion Block Transfer optimization allows for improved concurrency across Oracle RAC instances to improve overall application performance. This optimization results in reducing the **"*gc current block busy*" wait times by 3x times** for workloads that updates hot blocks concurrently**.**


2. Transfer block
1. Dispatch log write I/O and wait for completion
Original Protocol
2. Transfer block
3. Check for I/O completion before committing
1. Dispatch log write I/O
Smart Fusion Block Transfer

Undo Block RDMA Reads

Undo blocks need to be fetched from other Oracle RAC instances when there are transaction rollbacks etc. In Oracle RAC 18c, undo block transfers have been optimized to use a RDMA-based transfer protocol, replacing the traditional messaging-based protocol. **By leveraging RDMA, foreground processes are able to directly read the undo blocks from the remote instance's SGA.** The undo block reads **no longer invoke processes** on the remote instance, removing the server-side CPU and context switch overheads which were always part of traditional Oracle RAC communications. Additionally, the transfer latencies are no longer affected by OS process or overall system CPU load on the remote instance, **which helps sustain deterministic read latencies even in the case of a load spike** on the remote instance. RDMA read of a remote block would typically complete in **less than 10 µs**, which is a **5x improvement** over the best latencies we would get with the traditional message-based protocol using Exafusion.

In-Memory Commit Cache

Applications that have long running batch jobs and concurrent queries may exhibit high volumes of "undo header" CR block transfers. In Oracle 18c, **an in-memory commit cache has been added on Exadata.** Each instance would maintain a cache of local transactions and their respective states (committed or not) in the SGA, and the cache can be looked up remotely. This is faster than transferring the undo header blocks, each sized 8kb, to the remote instance. The state of multiple transaction ID's (XID's) can be looked up in a single message, which helps reduce the number of roundtrip messages in Oracle RAC, and also the CPU overhead in LMS processes which is responsible for responding to remote lookup requests. With the in-memory commit cache, **we are able to batch up to 30 XID lookups in a single roundtrip message** which would have been 30x 8k block transfers prior to this optimization.

With the commit cache optimization, we can expect a lot of the "*gc cr block 2-way*" waits corresponding to "undo header" transfers to be replaced with a smaller number of "*gc transaction table 2-way*" waits. A single "*gc transaction table 2-way*" wait represents a remote lookup of multiple XID's in one roundtrip.

NOTE: The "*gc transaction table 2-way*" wait is used in releases starting with Oracle 21c.
Earlier releases (Oracle 18c and 19c) would use the "*gc transaction table*" wait event instead.

Fast Index Split

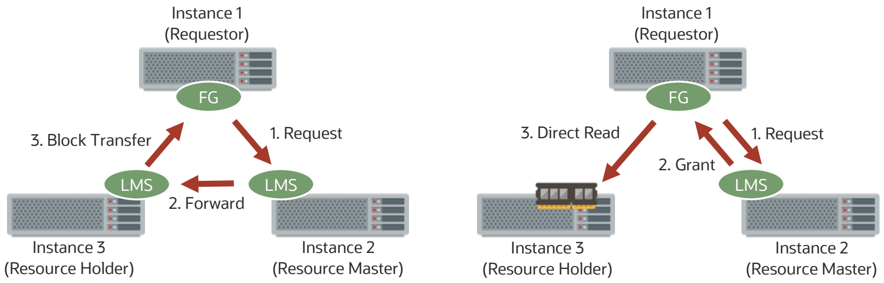
When there is a B-tree index leaf block split (frequently seen in OLTP workloads with right-growing indices), applications accessing the splitting leaf & branch blocks on all Oracle RAC instances would need to wait for the split operation to complete. This may cause intermittent hiccups (periods of almost zero activity) in application performance. Traditionally, these waits were implemented under a TX enqueue ("*enq: TX-index contention*" waits). These split waits have been optimized on Exadata in Oracle 19c, to use a less expensive Cache Fusion based mechanism in lieu of global enqueues. **The fast index split waits will be under the new "*gc index operation*" wait event ("*index split completion*" in 21c onwards), which replaces the traditional TX enqueue waits.**

Persistent Memory Commit Accelerator

Exadata X8M introduces the Persistent Memory Commit Accelerator, which implements **redo log I/O with RDMA writes to persistent memory on the storage servers.** This optimization significantly improves redo flush I/O performance, which would further improve inter-instance concurrency on systems experiencing high volumes of dirty buffer sharing (see Smart Fusion Block Transfer).

Shared Data Block and Undo Header RDMA Reads

In Oracle 21c, **RDMA support for Cache Fusion has been extended to support reads for data blocks, space blocks and undo header blocks.** Similar to the Undo Block RDMA read optimization in 18c, this will contribute to faster reads of data cached in remote instances, and further reduction in LMS CPU since LMS will not be invoked when data is read via RDMA. Traditionally, a foreground process would send a request to read a block to the master instance, then the master instance would forward the request to the holder instance, and the request is fulfilled by a 3-way Cache Fusion transfer ("*gc current block 3-way*"). This is a common access pattern in read intensive OLTP workloads running on large clusters of 3+ nodes. In large clusters, the size of each instance is typically small, which means that it is less likely that data is cached on the local instance, but chances are higher that it is cached on another instance. With data & space block RDMA, the master instance will respond to the requestor with a lock grant (permission to read the data), along with information about the holder instance for the block requested. The requesting client can then RDMA-read the block directly from the holder instance. This will remove the master-holder messaging, which will help improve read latency and reduce LMS CPU on the holder instance (who traditionally had to send back the block to the requestor).



Original Protocol **RDMA-based Protocol**

In this case, the foreground will see the following sequence of wait events instead of the traditional "*gc current block 3-way*" wait:

- "*gc current grant 2-way*" wait, followed by,
- A short "*gc current block direct read*" wait event

The "*gc current block direct read*" waits are **typically less than 10us**, and the combined wait time for the grant & read is usually shorter than the traditional 3-way transfer latency.

If the requestor is also the master instance, the "*gc current grant 2-way*" in the example above can be eliminated, because the instance can grant itself permission to read data without involving any messaging. In this case, the request can be quickly fulfilled by a single "*gc current block direct read*". This would replace some "*gc current block 2-way*" waits that were traditionally seen in Oracle RAC, including 2 node clusters.

Additionally, if a remote master instance is also the holder instance, LMS would respond with a grant message, then the requestor will RDMA-read the data from the holder (who is also the master). This is similar to the 3-way scenario described above, except that the master and holder instances are the same. In this case, the traditional "*gc current block 2-way*" waits are replaced by a "*gc current grant 2-way*" and "*gc current block direct read*". While the read latencies won't improve much in this case, **the cost for LMS to grant a lock is cheaper compared to sending back a data block, so the RDMA optimization will help reduce LMS CPU usage.**

Broadcast-on-Commit over RDMA

Before committing a transaction, the Broadcast-on-Commit protocol ensures that the system change number (SCN) on all the instances in a cluster is at least as high as the commit SCN. This is required to ensure the Consistent Read (CR) property of Oracle transactions. Traditionally, the Broadcast-on-Commit protocol used messages to broadcast the SCN to all the instances in a cluster. The LGWR process sends the SCN in a message to the LMS process on all instances. LMS process, upon receiving an SCN message, updates its instance's SCN and sends back an SCN ACK message to the LMS process on the initiating instance. Once the redo I/O completes, LGWR checks whether the redo SCN has been acknowledged by all instances. If so, LGWR notifies the foreground processes waiting for the transaction that the commit operation has completed. If the redo SCN was not acknowledged by the time the redo I/O completes, then the commit won't complete until all SCN ACKs have been received. Clients will see high "*log file sync*" wait times in this case.

In Oracle 21c, Broadcast-on-Commit has been optimized to use RDMA for the following reasons:

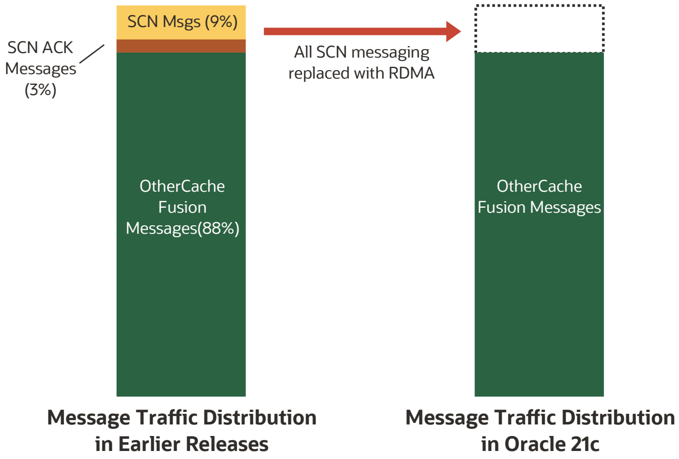
1. RDMA latency is lower than messaging:

As I/O latency improves on Exadata, broadcasting SCN using messaging could potentially become a bottleneck.

2. Reducing load on LMS processes:

Running OLTP applications, we see that SCN messages account for a measurable portion of messaging traffic, especially on clusters with large number of instances. Although these messages are rarely in the critical path latency-wise (because the actual IO would typically take longer), reducing these messages will have a benefit of reducing LMS load, giving us more headroom so that the system can better tolerate load spikes.

For example, running a large CRM (OLTP) workload on a 3 instance cluster, we saw that 12% of overall RAC messages were for SCN broadcasts. With RDMA, these messages will no longer invoke the LMS process.



In the Broadcast-on-Commit over RDMA mode, the LGWR process directly updates the SCN on each remote instance in the cluster using remote atomic operations. This makes the commit protocol faster as it is not affected by the remote LMS process's context switch latency or the CPU load on the remote instances.

Conclusion

These are some examples of how Oracle RAC leverages the advancements in hardware on Exadata to further optimize Oracle RAC Cache Fusion performance resulting in dramatic application scalability improvements without requiring any application changes. Oracle continues to invest in further innovations, by engineering the software to take advantage of the latest hardware technologies available in the market.

References

- Oracle Real Application Clusters (RAC) White Paper
- Oracle RAC Internals – The Cache Fusion Edition
- Oracle RAC 12c Practical Performance Management and Tuning
- Oracle RAC features on Exadata
- Oracle RAC 12c Release 2 – New Availability Features

Connect with us

Call **+1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle

Authors: Atsushi Morimura, Namrata Jampani, Anil Nair
Contributing Authors: Neil Macnaughton, Avneesh Pant, Michael Zoll