ORACLE

Oracle Real Application Clusters (RAC) Optimizations on Exadata

April 2025, Version 23ai

ORACLE

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, timing, and pricing 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.

ORACLE

Table of contents

List of figures

Executive Summary

Oracle Real Application Clusters, commonly referred to as Oracle RAC, is an extremely popular Oracle Database capability that provides linear horizontal scalability and high availability. Oracle RAC Cache Fusion is a component of Oracle RAC, responsible for synchronizing the caches among multiple 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 such as RDMA over Converged Ethernet (RoCE), enables Oracle to further improve performance and scalability**. In addition to benefiting from the improved wire speed of the underlying network, Oracle RAC Cache Fusion has been further optimized to leverage the advanced protocols and RDMA capabilities available on Exadata. This paper explains these optimizations that are available to Oracle databases deployed on Exadata.

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 Oracle Database 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 30 µs (micro-seconds), **which is 5x faster than a traditional socket-based implementation, as it would happen if RAC were to be implemented on generic servers**. Additionally, the CPU cost associated with sending and receiving messages is lower with Exafusion, allowing for higher block transfer throughput and increased headroom in the Cache Fusion server background processes (LMS processes) before they could become saturated. **Faster messaging not only benefits runtime application performance, but it also makes every Oracle RAC operation faster** - this includes dynamic lock redirection (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 block sends and adoption of RDMA.

Exafusion and the subsequent optimizations described in this document do not require extra OS resources to operate.

Zero Copy Block Sends

RoCE and InfiniBand network adapters support Zero Copy messaging. User space buffers are registered with the Host Channel Adapter (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. Elimination of the CPU cycles required for copying buffers further optimizes RAC Cache Fusion transfer latencies on Exadata.

Undo Block RDMA Reads

Undo blocks need to be fetched from other Oracle RAC instances when there are transaction rollbacks etc. In RAC on Exadata, undo block transfers have been optimized to use an RDMA-based transfer protocol, replacing the traditional messaging-based protocol. **By leveraging RDMA, foreground processes are able to directly read undo blocks from another instance's SGA**. The undo block reads **no longer invoke processes on the other instance**, removing the server-side CPU and context switch overheads which occur in RAC deployments on non-Exadata. Additionally, the transfer latencies are no longer affected by OS process congestion or overall system CPU load on the other instance, **which helps sustain deterministic read latencies even in the case of a load spike or stability issue** on another instance in the cluster. RDMA reads would typically complete in **less than 10 µs**, which is a **3x improvement** over the best latencies obtained with the messaging-based protocol.

In-Memory Commit Cache

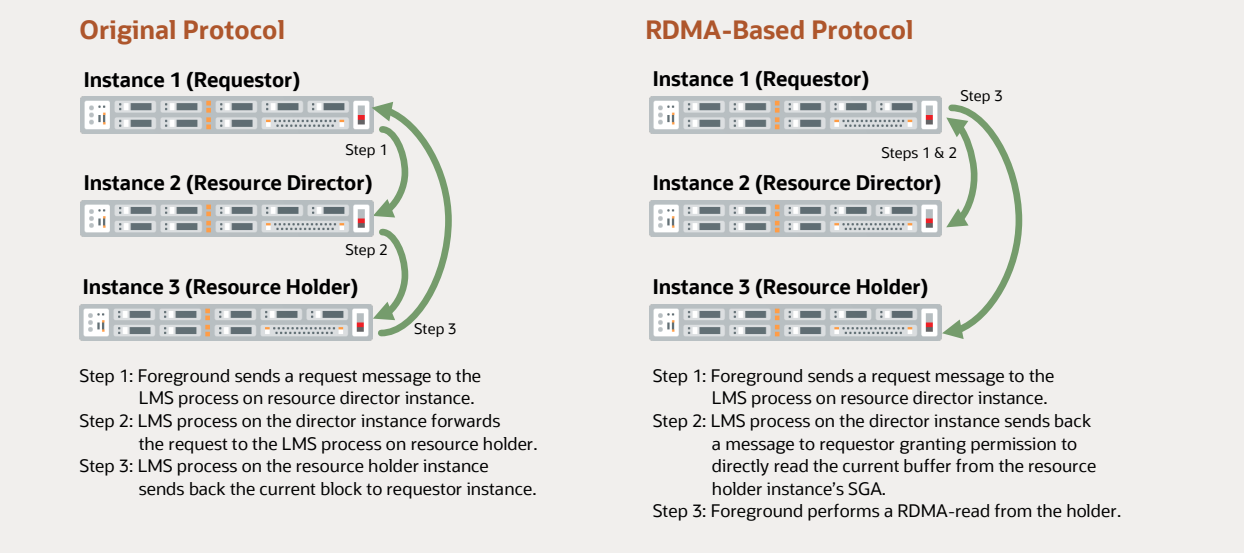
Applications that have long running batch jobs and concurrent queries may exhibit high volumes of "undo header" CR block transfers. To address this issue, Exadata supports **an in-memory commit cache**. With this capability, each instance maintains a cache of local transactions and their respective states (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 another instance. In addition, the state of multiple transaction ID's (XID's) can be looked up in a single message, which helps further reduce the number of roundtrip messages in Oracle RAC, as well as the CPU overhead in LMS processes which are responsible for responding to the commit cache lookup requests. With the in-memory commit cache, **we can 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, a lot of the "*gc cr block 2-way*" waits corresponding to "undo header" transfers are likely to be replaced with a smaller number of "*gc transaction table 2-way*" waits (renamed from "*gc transaction table*" waits in releases prior to Oracle 23ai). A single "*gc transaction table 2-way*" wait represents a lookup of multiple XID's in one roundtrip.

Shared Data Block and Undo Header RDMA Reads

In Oracle Database 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 Reads optimization, this contributes to faster reads of data cached in other 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 director instance, then the director 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 director 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 foreground could then RDMA-read the block directly from the holder instance. This will remove director-to-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).

Figure 1. RDMA-based Cache Fusion protocol



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

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

The "*gc current block direct read*" waits are **typically less than 10 µs**, 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 director instance, the "*gc current grant 2-way*" in the example earlier could be eliminated, because the instance can grant itself permission to read data without any messaging. In this case, the request could 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 non-local director 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 director). This is like the 3-way scenario described earlier, except that the director 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**.

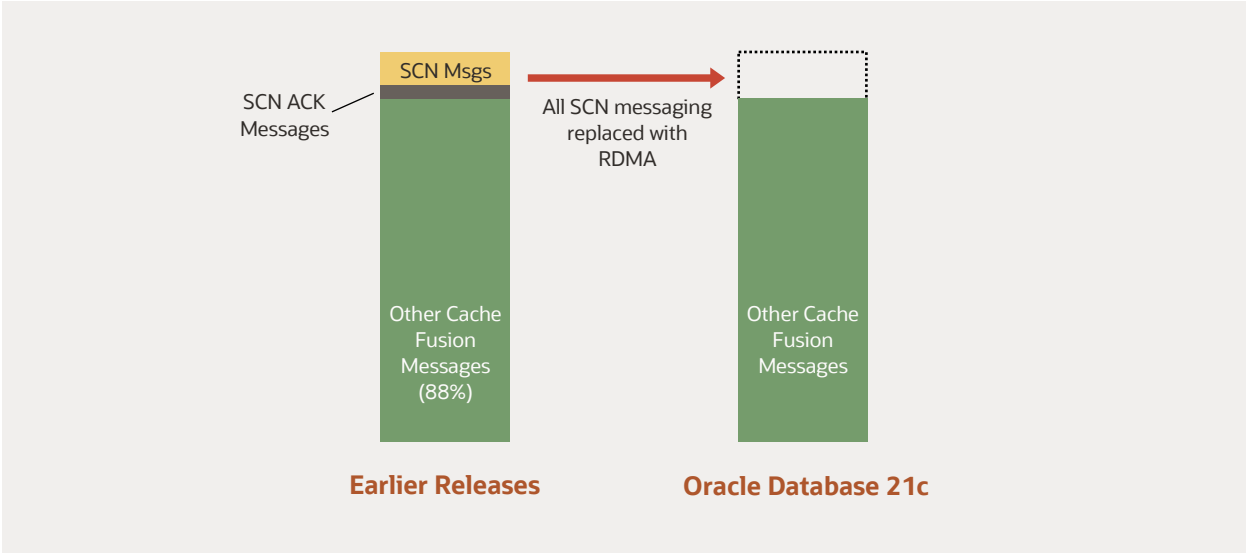
Broadcast-on-Commit Over RDMA

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. Foregrounds will see high "*log file sync*" wait times in this case.

In Oracle Database 21c, Broadcast-on-Commit has been optimized to use RDMA. Leveraging RDMA ensures lower latency than messaging, and also reduced load on LMS processes, especially for OLTP applications, where it was observed 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 has a benefit of reducing LMS load, yielding more headroom so that the system can better tolerate load spikes.

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

Figure 2. SCN message traffic reduction in Oracle Database 21c



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

Optimized Object Checkpoints

Workloads involving frequent Exadata Smart Scans or PQ scans may encounter performance bottlenecks associated with object checkpoints. Common symptoms involve lots of time spent waiting for "*enq: KO - fast object checkpoint*" and "*reliable message*" waits. This is especially true for applications involving very short scans running at high concurrency, because the checkpoints are requested at a very high frequency.

In Oracle Database 23ai, a quick RDMA-based check has been added to probe the cluster if an object checkpoint is required. The object checkpoint can be completely skipped if no dirty buffers are found in the global cache for the object. The check is performed under the "*gc obj ckpt direct read*" wait event and would **typically complete in less than 10 µs without invoking any background processes in the cluster instances**. This optimization significantly improves Exadata Smart Scans for the Vector Database in Oracle Database 23ai, as AI Vector Search scans tend to be extremely short. An internal study has proven that this optimization helps eliminate 99% of the object checkpoints for an AI Vector Search workload.

Conclusion

This document highlights several examples of how Oracle RAC leverages Exadata to further optimize Oracle RAC Cache Fusion performance resulting in dramatic application scalability improvements without requiring any application changes. With Exadata, Oracle continues to invest in further innovations by engineering the database software to take advantage of the latest hardware capabilities, unlocking significant productivity gains for Oracle's customers.

References

- Oracle Real Application Clusters (RAC) White Paper
- Oracle RAC Internals – The Cache Fusion Edition
- Oracle RAC features on Exadata

ORACLE

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