

# Oracle Database 19c: AQ Sharded Queues for scalable, high performance Advanced Queuing

TECHNICAL BRIEF / DECEMBER 2019

## 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

Disclaimer ..... 2

Introduction ..... 4

Architecture..... 4

PERFORMANCE OF AQ-JMS QUEUES ..... 6

PERFORMANCE OF AQ-JMS TOPICS ..... 8

SYSTEM CONFIGURATION ..... 11

When NOT to use JMS sharded queues ..... 11

CONVERTING TO JMS sharded queues ..... 12

CONCLUSION ..... 12

FURTHER READING ..... 12

ADVANCED QUEUING USER'S GUIDE FOR ORACLE DATABASE 19c.....12

## INTRODUCTION

AQ sharded queues are an enhancement to Oracle Advanced Queuing (AQ) in Oracle Database 19c, first previewed in Oracle Database 12c. AQ sharded queues are high-performance, scalable, database-backed queues. AQ sharded queues are the preferred configuration for Oracle Advanced Queuing with:

- Queues that have enqueueers or dequeueers on multiple Oracle RAC instances
- High throughput queues
- AQ queues that consume too many system resources. AQ queues have been a feature of the Oracle database since Release 9.2.
- Queues with a large number of subscribers.

AQ allows businesses to take advantage of the Oracle Database 19c for enterprise messaging needs without the need for separate high-end, message-oriented middleware products. AQ is ideal for 3 Tier Architectures that store application state in a database. By being integrated with the database, AQ allows enqueue or dequeue to be atomically committed at the same time as other database operations without requiring the overhead of two phase-commit or XA interfaces. Persistent AQ messages can be backed up or recovered using traditional Oracle Database functionality.

Java Message Service Version 1.1 (JMS) is a popular standard API for accessing enterprise messaging systems from Java programs. Although AQ has supported a JMS interface since Oracle Database Release 9.2, AQ sharded queues provide significant improvements in performance, manageability and scalability in Oracle Database 19c, especially for high-concurrency architectures including Oracle Exadata, Autonomous Databases, and Oracle RAC.

This technical brief discusses the architecture and performance of AQ sharded queues. We provide guidelines of when and how to use them. For an overview of Advanced Queuing in general, please refer to the technical brief “Oracle Database 19c: Advanced Queuing.”

## ARCHITECTURE

A sharded queue is a single logical queue that is transparently divided into multiple, independent, physical queues through system-maintained partitioning. Below is a diagram of a single logical queue. Messages are typically enqueued into one side of the queue and dequeued from the other.



Figure 1. Diagram of a single logical queue

If the FIFO (first-in, first-out) ordering of messages in a queue is maintained by an IOT (index-organized table) or global index, the two sides of the queue can form hot-spots for high concurrency systems, resulting in excessive inter-instance block transfers in Oracle RAC. If the AQ IOT or global index is experiencing high block contention for the same block due to enqueues or dequeues, the sharded queue architecture is more appropriate. AQ sharded queues replace the global index of the queue with a partitioned table and an in-memory message cache to eliminate these performance issues. In Figure 2, we illustrate the AQ Sharded Queue architecture. While logically presenting a single queue to the application developer, the queue is transparently divided into shards (or partitions). One or more shards are allocated to each instance on a RAC cluster. Each enqueueing session is automatically assigned a shard so that the shard maintains the enqueue-time ordering of the messages within each session. Dequeues have an affinity for locally enqueued messages to minimize cross-instance activity. Underneath the sharded queue abstraction, a system-partitioned table is used. Oracle automatically manages the creation, allocation, population, and truncation of partitions (labeled “Part” in Figure 2) within the sharded queue table. AQ sharded queues use bulk operations such as partition truncates when appropriate to optimize performance.

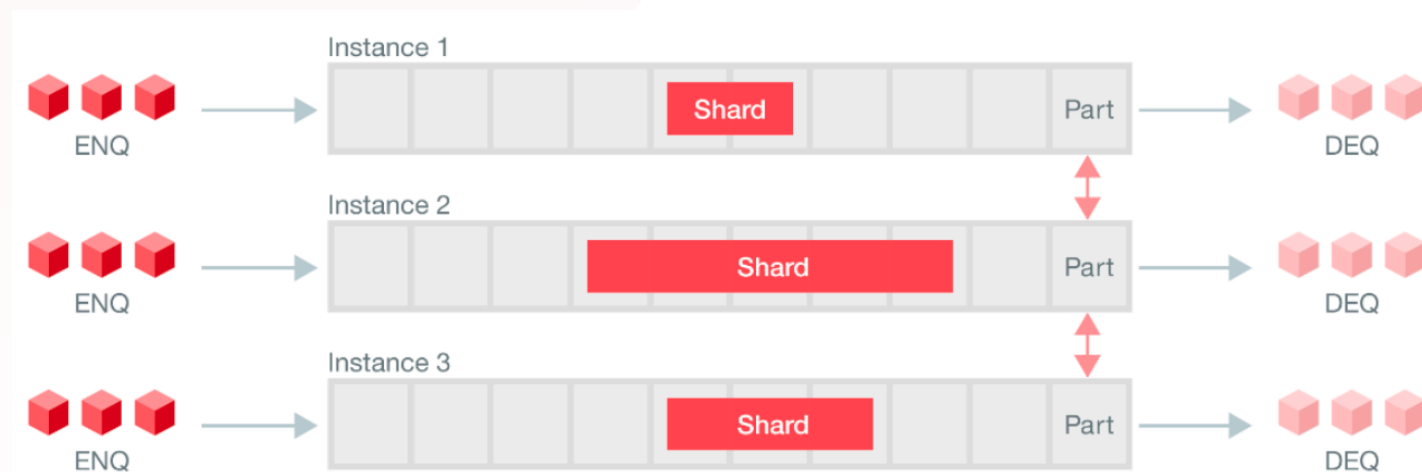


Figure 2. JMS Sharded Queue architecture

Instead of maintaining a global index or IOT, an AQ Sharded Queue uses a partitioned table and maintains an in-memory message cache. Local or partial indexes may be defined on a sharded queue table. In the event that the dequeuers are keeping up with the enqueueers, the in-memory message cache provides shard-specific ordering without requiring SQL to select from a shard’s partition. The message cache optimizes performance and reduces the disk and CPU overhead of AQ-JMS enqueues and dequeues. The message cache is also used to store nonpersistent messages. Factors such as queue backlogs, message size, and number of subscribers affect message throughput on all queuing systems. The architecture of Oracle’s sharded queues is well-designed for all of these dimensions. Since AQ sharded queues are database-backed, they can handle huge backlogs of large messages. Furthermore, to better utilize SGA (System Global Area), the message cache will not cache the full payload of larger messages, but it will still cache the appropriate metadata of larger messages. Queue metadata is maintained in-memory to avoid disk operations, to reduce the amount of SQL used to maintain the queues, and to handle JMS nondurable subscribers. Creation of a JMS durable subscriber is a relatively low overhead DML-like operation. To support a large number of subscribers efficiently, bitmaps are used to track subscribers to a message. AQ sharded queues support a streaming interface to reduce the memory requirement when dealing with large messages. Large message payloads are divided into small chunks rather than sending or receiving a large, contiguous array of bytes. You can use DBMS\_AQADM procedures such as SET\_MIN\_STREAMS\_POOL and SET\_MAX\_STREAMS\_POOL to fine-tune the allocation of SGA for the message cache. AQ sharded queues perform cross-instance communication but avoid simultaneous writes to the same block across RAC instances. Typically, dequeues occur on a shard that is local to a message’s enqueueing instance, but in certain situations, Oracle will efficiently forward messages across instances for dequeuing on another instance. For example, if an AQ Sharded Queue has a single enqueueing session on one Oracle RAC instance and a single dequeuing session on another instance, then the

queues will forward messages between the Oracle RAC instances. The forwarding of messages is done asynchronously to the enqueueing transaction to improve performance. Dequeueers may get an ORA-25228 if they are connected to an instance whose shards have no messages. The optimizations described above allow AQ sharded queues to improve the performance of queuing on a RAC database as well as a single-instance database significantly. Furthermore, since an AQ Sharded Queue is logically a single queue, there is no need to create your own multi-queue solution for instance affinity on RAC

## PERFORMANCE OF AQ-JMS QUEUES

A JMS Queue is a single-consumer queue typically used for point-to-point applications. The throughput and scalability of JMS queues using AQ-JMS sharded queues demonstrate better performance over AQ queues, especially on a RAC database. To illustrate this improvement, we perform a simple benchmark test where the number of enqueueer threads and dequeueer threads are incrementally increased for an AQ Sharded Queue on an Exadata X4-2 until performance tails off. Each message is persistent and has a 512-byte payload. Ten enqueue or dequeue operations were handled in each transaction.

As seen by the log-log graph below, sharded queues scales linearly with no regression as more instances are added until there are 32 concurrent enqueueer and dequeueer processes. In Oracle Database Release 12.1, the bottleneck for sharded queues was a combination of commit overhead and partition maintenance overhead. For an AQ sharded queue, the dequeueers were able to keep up with the equivalent number of enqueueers for the JMS Queue up to 76K dequeues per second. In contrast, AQ queues do not show a large increase of dequeue throughput as more threads or instances are added.

An AQ queue performs best on a single RAC node with a throughput below 8K dequeues per second. Multiple RAC nodes experienced global index contention, as indicated by Automatic Database Diagnostic Monitor (ADDM) events such as “buffer busy” or “global cache wait”. Within a single node, CPU was the resource bottleneck for the AQ queue. When possible, AQ queues should leverage instance affinity on RAC and distribute the load across RAC nodes by using multiple AQ queues.

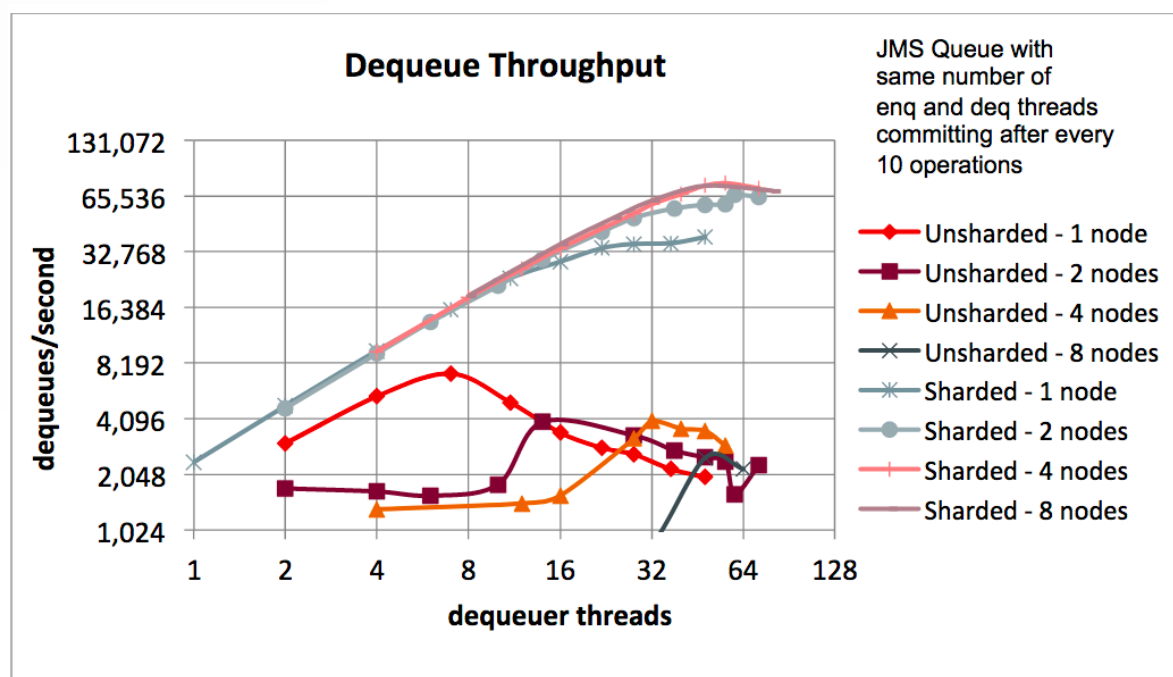


Figure 3. As the number of enqueueer and dequeueer threads is increased, dequeue throughput of an AQ Sharded Queue outperforms an AQ queue (marked as Unsharded above)

In the same test, enqueue throughput on the JMS Queue behaved similarly to the dequeue counterparts. An AQ Sharded Queue scales well across RAC nodes. In contrast, an unsharded queue performs best on a single node, but dequeues did not keep up with the enqueue rate when the number of enqueue and dequeue threads for the unsharded queue were kept the same.

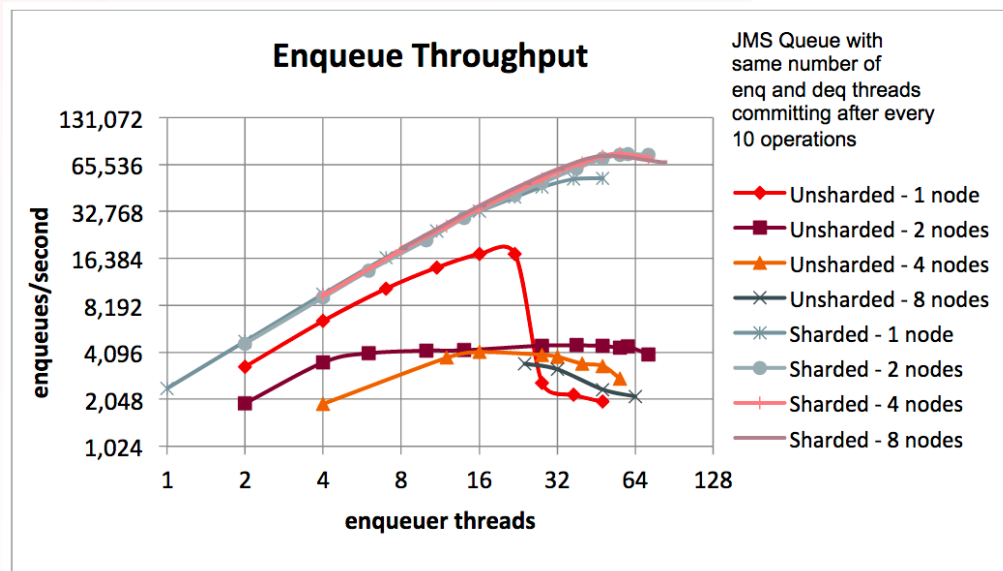


Figure 4. On the same test as in Figure 3, enqueue throughput of a sharded queue scales well across RAC nodes

Another interesting result of the test is that AQ sharded queues could dequeue over four times as many messages per CPU compared to the AQ queue at higher loads. In addition, AQ sharded queues spread the workload well between instances as shown in Figure 5. Although not illustrated, the latency of the dequeues in the AQ sharded queue was typically less than a millisecond.

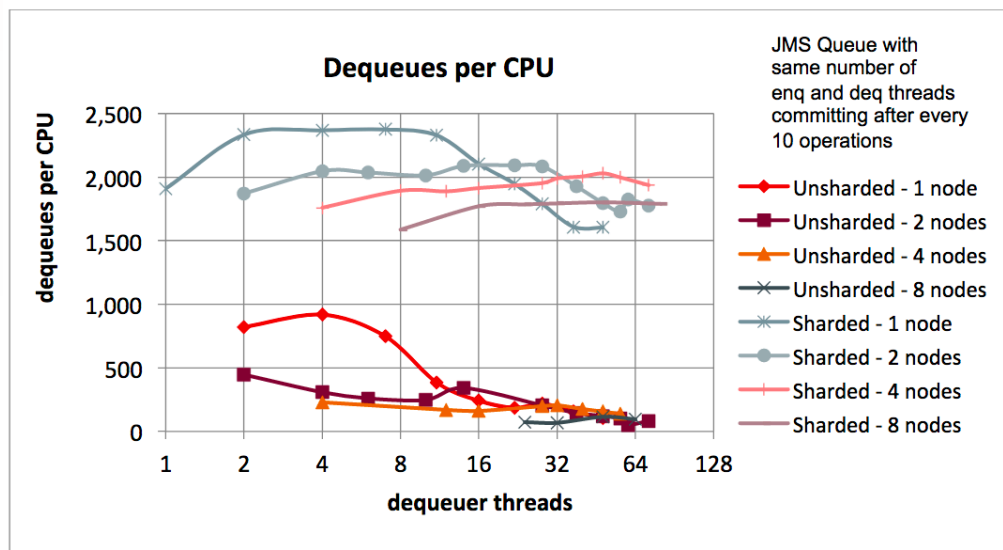


Figure 5. AQ sharded queues could support over four times the dequeue operations per CPU than an AQ queue (marked as Unsharded) at higher loads

In realistic application workloads, the commit overhead (as indicated by the “Log file sync” wait event in AWR reports) is typically not an issue because AQ enqueues and dequeues often are part of transactions that are sufficiently complex. The Exadata X4-2 Database Machine can handle plenty of additional non-AQ load without reducing messaging throughput for an AQ sharded queue. To emulate removal of the commit overhead as a bottleneck, we next modify the tests to enqueue and dequeue messages in much larger commit batches. As a result, AQ sharded queues perform even better with a limit near 100K dequeues per second. Dequeue processes again keep up with enqueue processes. Without the logarithmic scales, the benefits of adding more RAC instances are more clearly illustrated. In this test on Oracle Database Release 12.1, the bottleneck for a single AQ sharded queue is due to partition management of its underlying queue table.

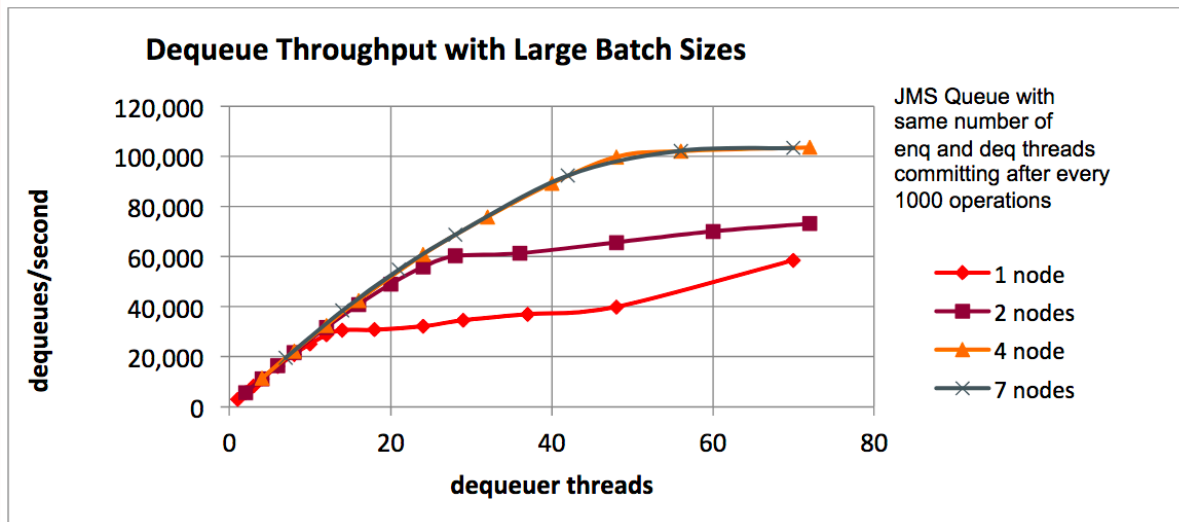


Figure 6. An AQ sharded queue supports over 100K dequeues per second with large commit batch sizes

## PERFORMANCE OF AQ-JMS TOPICS

A JMS Topic is a multi-consumer queue typically used for publish-subscribe applications. In this variation of the simple benchmark, we demonstrate the throughput of an AQ-JMS Topic by configuring 10 subscribers for each 512-byte message. The database connections of the enqueueer and dequeueer threads and their associated sessions are spread evenly over multiple instances. Similarly, dequeueer threads are spread evenly across subscribers. For example, if we have 4 instances and 4 enqueueers and 10 subscribers, one enqueueer is connected to each RAC node and each node has 10 dequeueer threads, with each of the threads for a different subscriber. With 10 subscribers for each enqueued message, AQ sharded queues achieved approximately 100K dequeues per second. Again, the bottleneck was a combination of commit overhead and partition management overhead. AQ queue throughput did not significantly improve with more threads and did not scale well on RAC due to global index contention, as indicated by the statistics such as buffer busy waits. AQ sharded queues used less than half the CPU consumed by AQ queues.



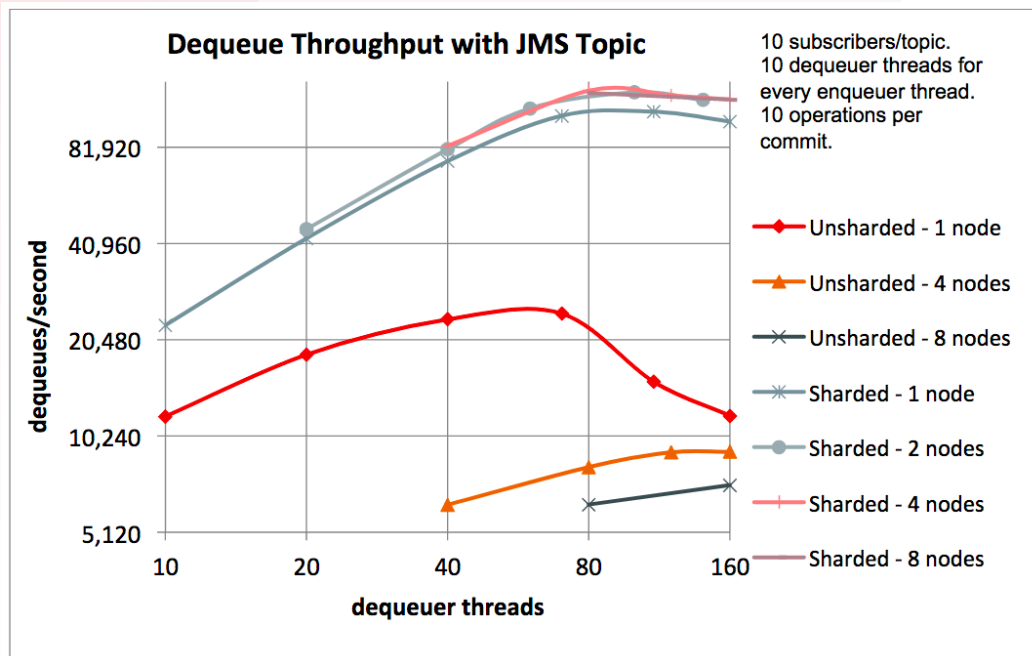


Figure 7. With 10 subscribers for each message, AQ sharded queues achieved approximately 100K dequeues per second.

In Figure 8, the concurrent enqueue throughput for topics behaved similarly to their dequeue counterparts in Figure 7. AQ sharded queue performance did not degrade as more RAC nodes were added

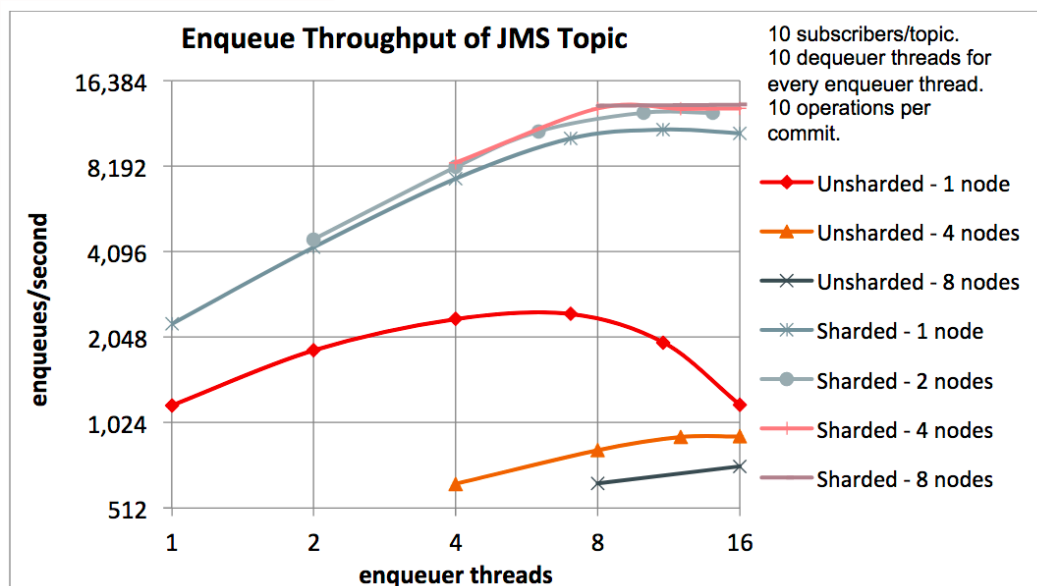


Figure 8. AQ sharded queue enqueue performance did not degrade as more nodes were added

With larger commit batch sizes to eliminate the commit overhead bottleneck not experienced in typical application environments, AQ sharded queues performed better with approximately 150k combined enqueue and dequeue operations per second or over a half billion operations an hour on a single JMS Topic.

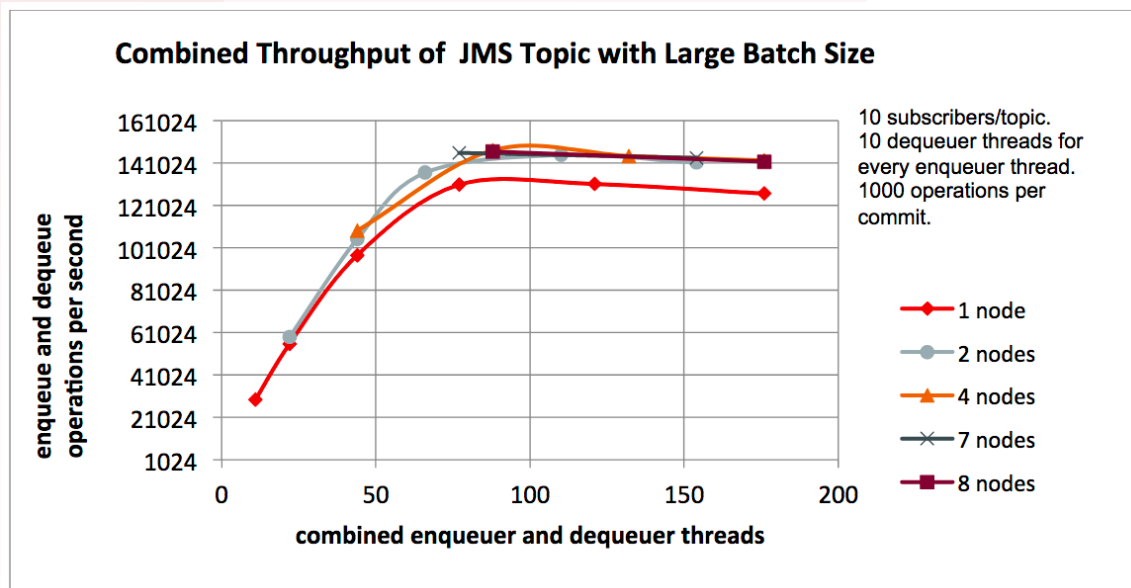


Figure 9. AQ sharded queues supported approximately 150k combined enqueue and dequeue operations per second or over a half billion operations an hour on a single JMS Topic

As with JMS Queues, the bottleneck for JMS Topics was largely due to partition management of a single AQ sharded queue, so system resources weren't being stressed. As nodes were added, more resources were available for other processing and other queues. For example, the CPU utilization on each of the 8 nodes on the X4-2 was less than 10% for the AQ sharded queue on a single JMS Topic.

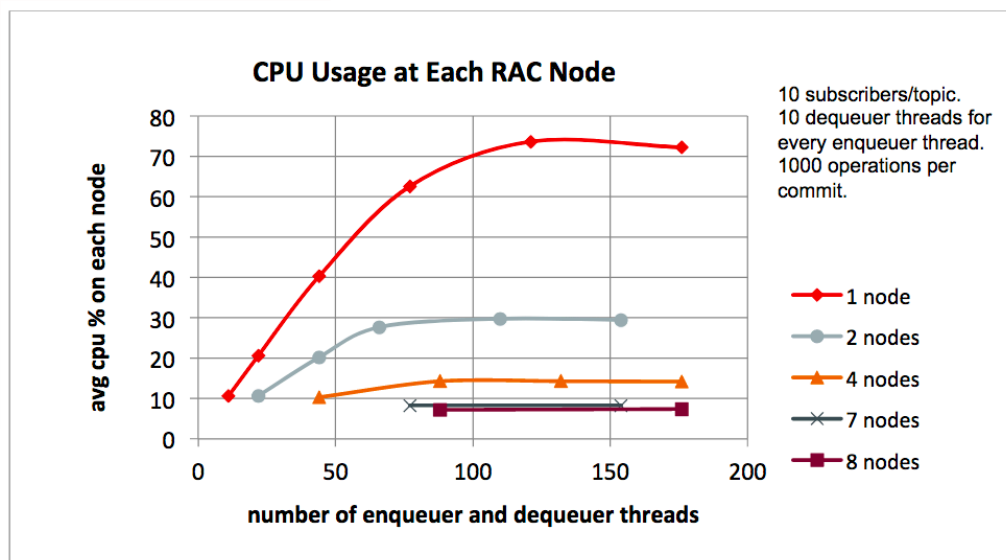


Figure 10. The CPU utilization on each of the 8 nodes on the X4-2 was less than 10% for the AQ sharded queue.

In this same JMS Topic test, the average response time for a dequeue operation stayed relatively flat for AQ sharded queues, whereas the response time for AQ queues increased with more RAC instances or with more dequeuers as seen in the graph below. With AQ sharded queues, dequeuers kept up with enqueueers.

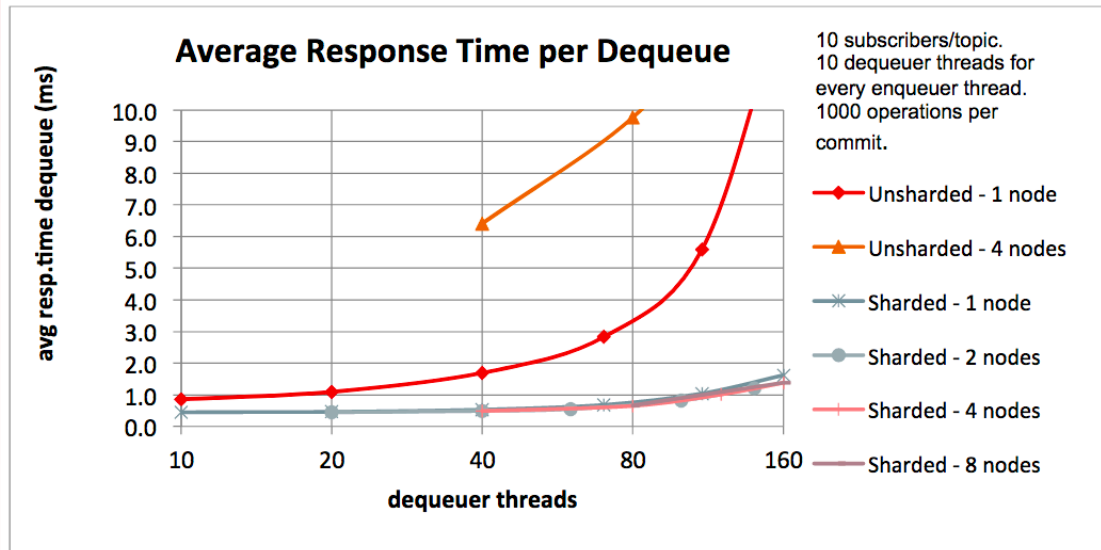


Figure 11. The response time for AQ queues(marked Unsharded above) increased with more instances or with more dequeuers

## SYSTEM CONFIGURATION

The above performance tests were run on an Exadata X4-2 Database. The Exadata X4-2 has 8 compute nodes and 14 cells. Each node has two 2.70Ghz Xeon E5-2697 v2 processors. Each processor on the node has 12 cores with 24 threads. The nodes each have 30MB caches and 256GB memory. The Exadata cells have two 2.60Ghz Xeon E5-2630 v2 processors. Each processor on the cell has 6 cores with 12 threads. The cells each have 15MB cache and 94GB memory.

The tests ran Oracle Database Release 12.1. The configuration included up to eight RAC instances. JMS clients typically ran on the non-database compute nodes on the Exadata X4-2. STREAMS\_POOL\_SIZE was configured to 40GB so that the message cache size was more than sufficient to cache the message backlog.

## WHEN NOT TO USE JMS SHARDED QUEUES

AQ sharded queues provide many advantages over AQ queues for high end users of JMS. However, AQ queues are a mature, popular feature of the Oracle Database. If AQ queues satisfy your requirements, there is no compelling need to migrate to AQ sharded queues. Furthermore, AQ sharded queues in Oracle Database Release 19.1 have some restrictions that do not apply to AQ queues. Oracle-specific JMS extensions such as array enqueue and dequeue are not supported. AQ sharded queues must use the thin JDBC driver. Only AQ queues support features that are not part of the JMS standard including transactional grouping, exception queues, the messaging gateway, non-JMS ordering, and recipient lists. In 19.1, JMS applications that heavily depend upon dequeue by message ID should evaluate AQ queues.

## CONVERTING TO JMS SHARDED QUEUES

Existing applications built upon the JMS standard should not require changes to use AQ sharded queues beyond using the 19.1 JMS and JDBC drivers. You simply need to recreate the queue when it is empty. The first step to convert an AQ queue to a AQ sharded queue is to stop the AQ queue for enqueue operations using the `DBMS_AQADM.STOP_QUEUE` procedure. Then, drain the queue by dequeuing all the messages in the AQ queue. Next, you stop the queue for dequeues, verify that the queue is empty, and drop the empty AQ queue. Then, set the `STREAMS_POOL_SIZE` parameters. Finally, use `DBMS_AQADM.CREATE_SHARDED_QUEUE()` to create an AQ sharded queue of the same name, start the queue to enable enqueues and dequeues, and grant any required privileges on the queue.

AQ sharded queues introduce new or modified monitoring views or tables. Monitoring scripts based upon views or tables for AQ queues will require changes.

## CONCLUSION

An AQ-JMS sharded queue provides impressive performance that can exceed a half-billion enqueues and dequeues per hour for a single JMS Topic while consuming less than 10% of the CPU on an Oracle Exadata X4-2 Database Machine. AQ sharded queues are database-backed, and enqueues and dequeues therefore can be committed in the same transaction as other database operations without requiring two phase commit. AQ sharded queues perform well on a single-instance database as well as on RAC. AQ sharded queues have backup and recovery functionality like other database tables and support high availability via Oracle Data Guard. AQ sharded queues have set a new bar for database-backed messaging systems.

## FURTHER READING

### ADVANCED QUEUING USER'S GUIDE FOR ORACLE DATABASE 19c

<https://docs.oracle.com/en/database/oracle/oracle-database/19/adque/index.html>

## ORACLE CORPORATION

### Worldwide Headquarters

500 Oracle Parkway, Redwood Shores, CA 94065 USA

### Worldwide Inquiries

TELE + 1.650.506.7000 + 1.800.ORACLE1

FAX + 1.650.506.7200

oracle.com

## CONNECT WITH US

Call +1.800.ORACLE1 or visit [oracle.com](https://www.oracle.com). Outside North America, find your local office at [oracle.com/contact](https://www.oracle.com/contact).



[blogs.oracle.com/oracle](https://blogs.oracle.com/oracle)



[facebook.com/oracle](https://facebook.com/oracle)



[twitter.com/oracle](https://twitter.com/oracle)

## Integrated Cloud Applications & Platform Services

Copyright © 2020, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. 0920

White Paper Oracle Database 19c: AQ Sharded Queues for scalable, high performance Advanced Queuing  
December 2019

Author: Sanjay Goll

Contributing Authors: Mahesh Girkar, Amit Ketkar, Amitabha Samanta



Oracle is committed to developing practices and products that help protect the environment

ORACLE®