ORACLE DATABASE 12c

Oracle Database 12*c*: JMS Sharded Queues

For high performance, scalable Advanced Queuing

ORACLE WHITE PAPER | MARCH 2015

ORACLE

Table of Contents

Introduction

JMS Sharded Queues are an enhancement to Oracle Advanced Queuing (AQ) in Oracle Database 12cR1. JMS Sharded Queues are high-performance, scalable, database-backed queues. JMS Sharded Queues are the preferred JMS configuration for Oracle Advanced Queuing with:

- JMS queues that have enqueuers or dequeuers on multiple Oracle RAC instances
- High throughput JMS queues
- Unsharded JMS queues that consume too many system resources. Unsharded JMS queues have been a feature of the Oracle database since Release 9.2
- JMS Topics with a large number of subscribers

AQ allows businesses to take advantage of the Oracle Database 12c for enterprise messaging needs without the need for 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 enqueues or dequeues 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 failed over to a standby database, backed up or recovered using Oracle Database functionality.

Java Messaging 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, JMS Sharded Queues provide significant improvements in performance, manageability and scalability in Oracle Database 12c, especially for high-concurrency architectures including Oracle Exadata Database Machines and Oracle RAC.

This technical white paper discusses the architecture and performance of JMS sharded queues and provides guidelines of when and how to use JMS sharded queues. For an overview of Advanced Queuing in general, please refer to “Oracle Database 12c: Advanced Queuing.”

Architecture

A sharded queue is a single logical queue that is transparently divided into multiple independent, physical queues through system 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.

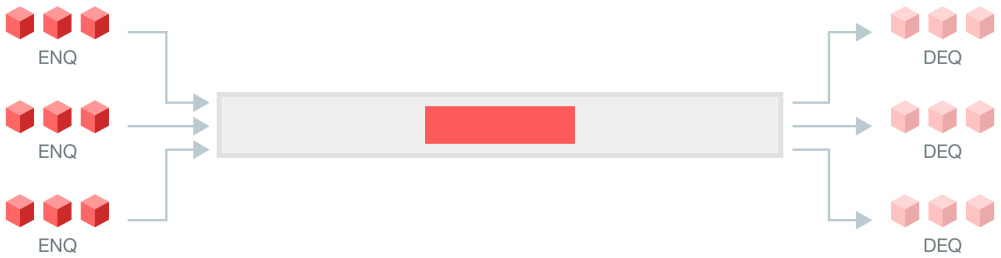

FIGURE 1 Diagram of a single logical queue.

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 JMS enqueues or dequeues, the sharded queue architecture is more appropriate. JMS 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 JMS Sharded Queue architecture. While logically presenting a single queue to the JMS developer, the queue is transparently divided into shards. One or more shards are allocated to each instance on a RAC cluster. Each enqueuing 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. JMS Sharded Queues use bulk operations such as partition truncates when appropriate to optimize performance.

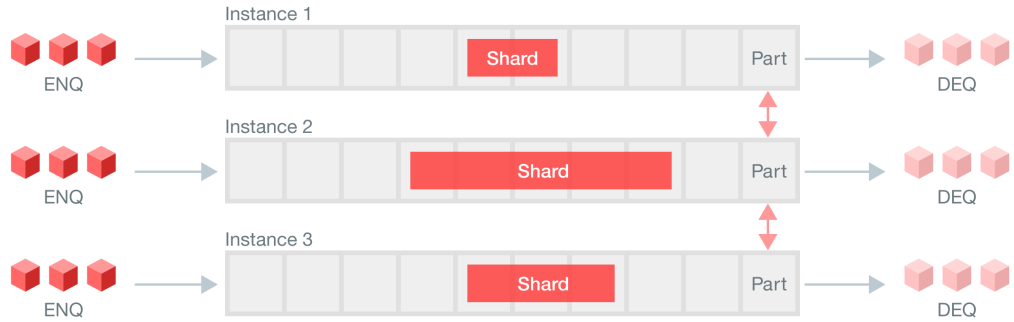

FIGURE 2 JMS Sharded Queue architecture.

Instead of maintaining a global index or IOT, a JMS 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 enqueuers, 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 Oracle JMS 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.

JMS Sharded Queues support a streaming interface to reduce the memory requirement when dealing with large messages. Large message playloads 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.

JMS 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 enqueuing instance, but in certain situations, Oracle will efficiently forward messages across instances for dequeuing on another instance. For example, if a JMS Sharded Queue has a single enqueuing session on one Oracle RAC instance and a single dequeuing session on another instance, then JMS Sharded Queues will forward messages between the Oracle RAC instances. The forwarding of messages is done asynchronously to the enqueuing transaction to improve performance. Dequeuers may get an ORA-25228 if they are connected to an instance whose shards have no messages.

The optimizations described above allow JMS Sharded Queues to improve the performance of queuing on a RAC database as well as a single-instance database significantly. Furthermore, since a JMS 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 improved performance over unsharded queues, especially on a RAC database. To illustrate this improvement, we perform a simple benchmark test where the number of enqueuer threads and dequeuer threads are incrementally increased for a JMS Sharded Queue on an Oracle Exadata X4-2 Database Machine until performance tails off. Each message is persistent and has a 512-byte payload. Ten enqueue or dequeue operations were in each transaction.

As seen in the two log-log graphs below, sharded queues scale linearly with no regression as more instances are added until there are 32 concurrent enqueuer and dequeuer processes. In Oracle Database Release 12.1.0.2, the bottleneck for sharded queues was a combination of transaction commit overhead and partition maintenance overhead. For a sharded queue, the dequeuers were able to keep up with the equivalent number of enqueuers for the JMS Queue up to 76K dequeues per second. In contrast, unsharded queues do not show a large increase of dequeue throughput as more threads or instances are added.

An unsharded 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 unsharded queue. When possible, unsharded queues should leverage instance affinity on RAC and distribute the load across RAC nodes by using multiple unsharded queues.

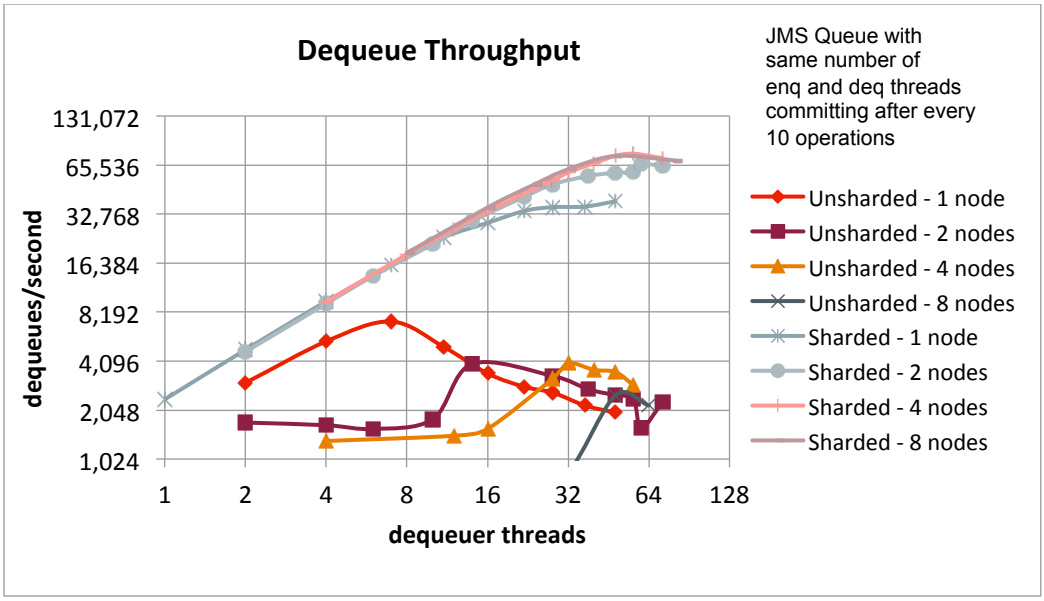

Figure 3. As the number of enqueuer and dequeuer threads is increased, dequeue throughput of a sharded queue outperforms an unsharded queue.

In the same test, enqueue throughput on the JMS Queue behaved similarly to the dequeue counterparts. A 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.

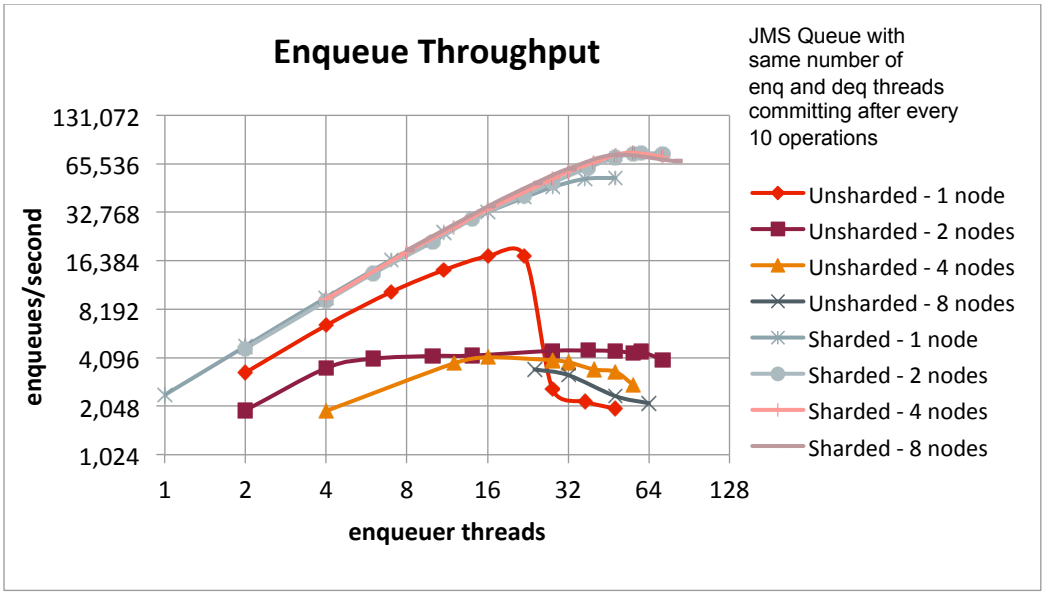

Figure 4. On the same test as in Figure 3, enqueue throughput of a sharded queue scales well across RAC nodes.

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

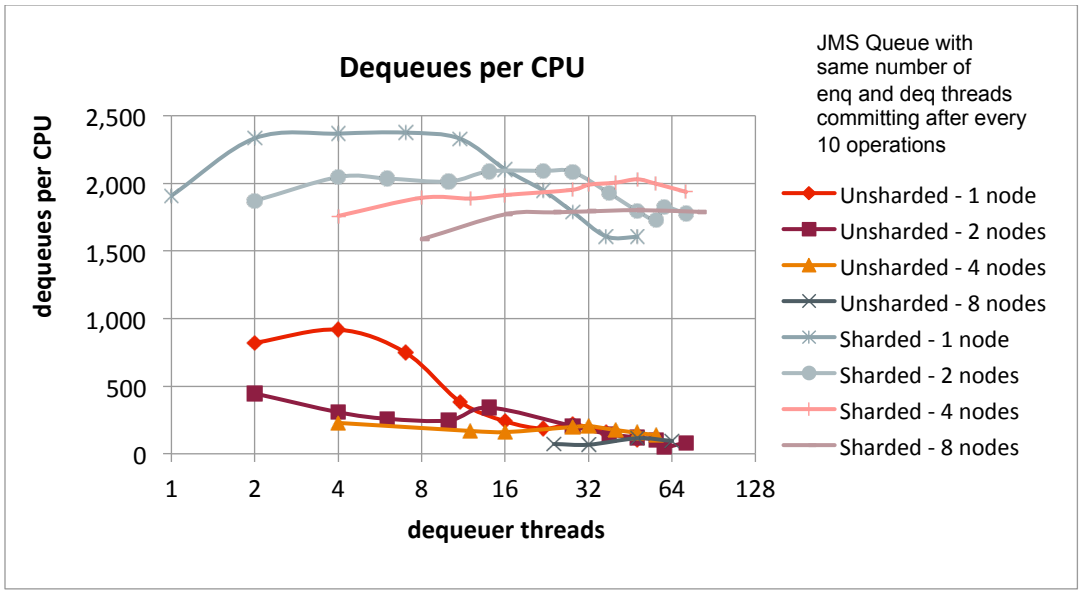

Figure 5. Sharded queues could support over four times the dequeue operations per CPU than an unsharded queue at higher loads.

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 a 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, sharded queues perform even better with a limit near 100K dequeues per second. Dequeue processes again keeps 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 12.1.0.2, the bottleneck for a single sharded queue is due to partition management of its underlying queue table.

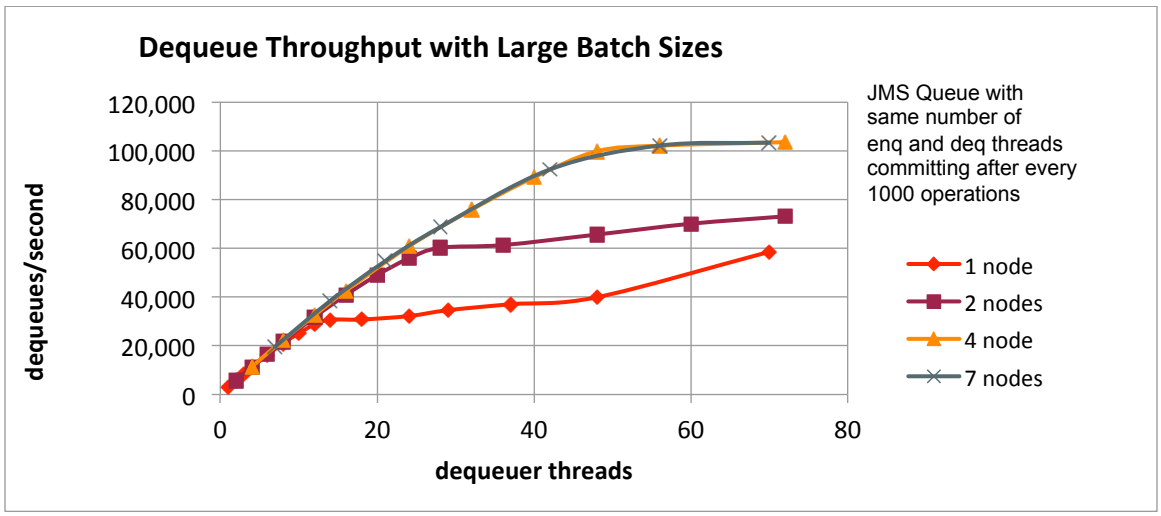

Figure 6. A sharded queue supports over 100K dequeues per second with large commit batch sizes.

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 enqueuer and dequeuer threads and their associated sessions are spread evenly over multiple instances. Similarly, dequeuer threads are spread evenly across subscribers. For example, if we have 4 instances and 4 enqueuers and 10 subscribers, one enqueuer is connected to each RAC node and each node has 10 dequeuer threads, with each of the threads for a different subscriber.

With 10 subscribers for each enqueued message, sharded queues achieved approximately 100K dequeues per second. Again, the bottleneck was a combination of commit overhead and partition management overhead. Unsharded 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. Sharded queues used less than half the CPU consumed by unsharded queues.

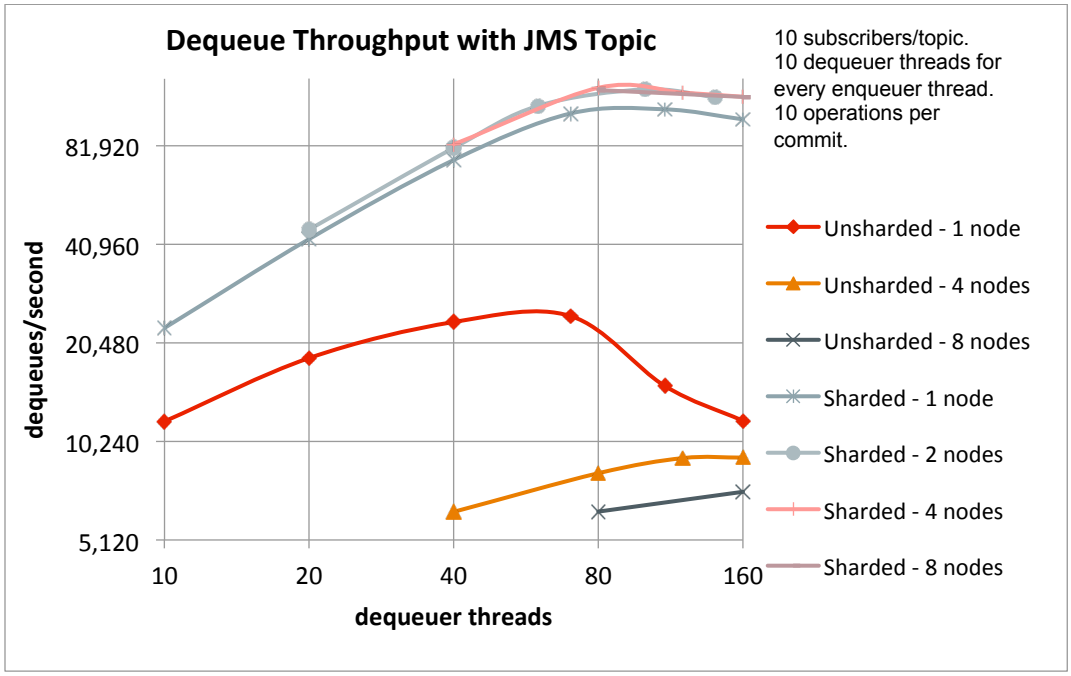

Figure 7. With 10 subscribers for each message, sharded queues achieved approximately 100K dequeues per second.

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

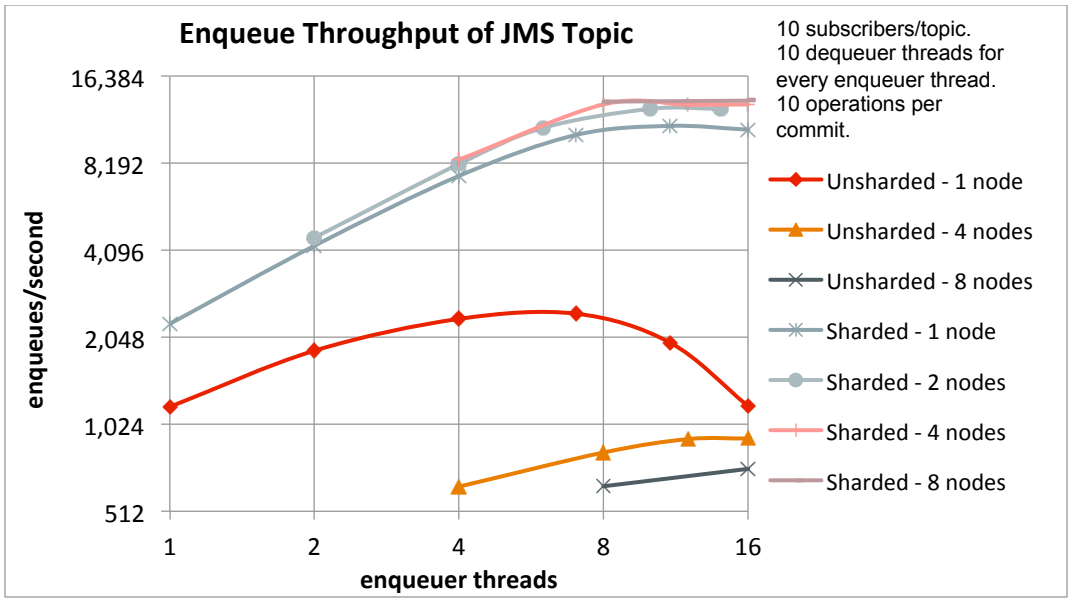

Figure 8. Sharded queue enqueue performance did not degrade as more nodes were added.

With larger commit batch sizes to eliminate the commit overhead bottleneck not experienced in typical application environments, 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.

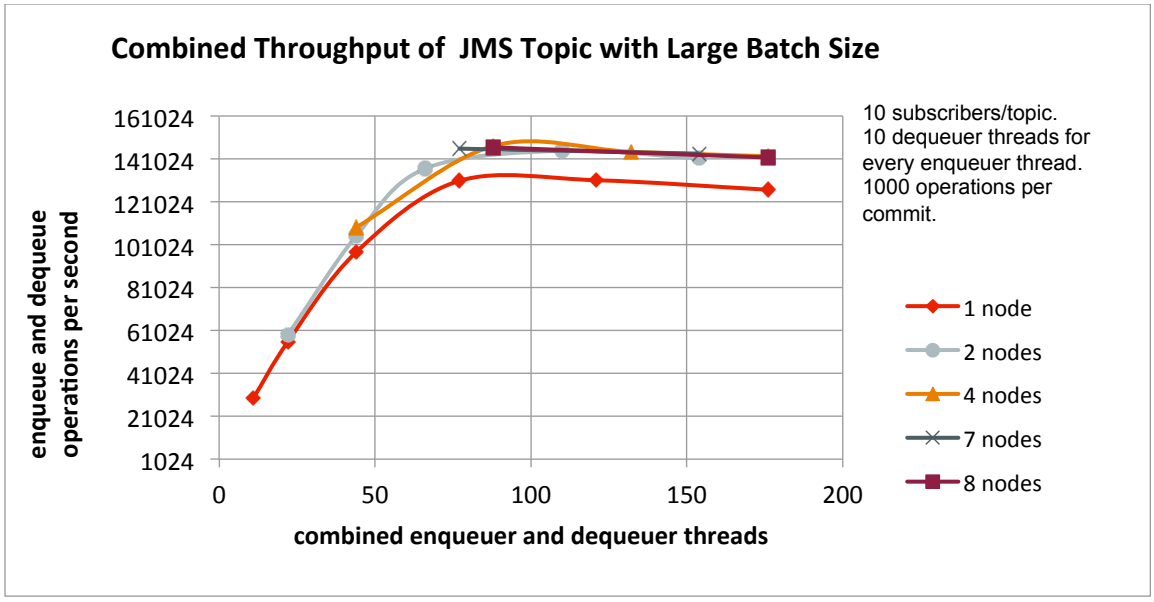

Figure 9: Sharded queues supported approximately 150k combined enqueue and dequeue operations per second or over a half billion operations an hour on a single JMS Topic.

As with JMS Queues, the bottleneck for JMS Topics was largely due to partition management of a single 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 sharded queue on a single JMS Topic.

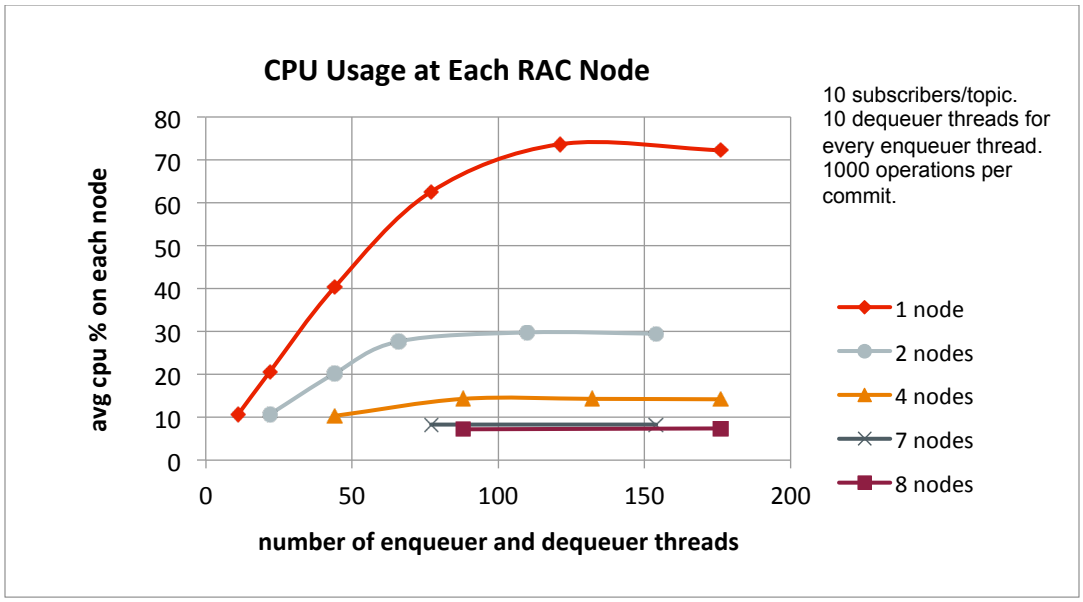

Figure 10. The CPU utilization on each of the 8 nodes on the X4-2 was less than 10% for the sharded queue.

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

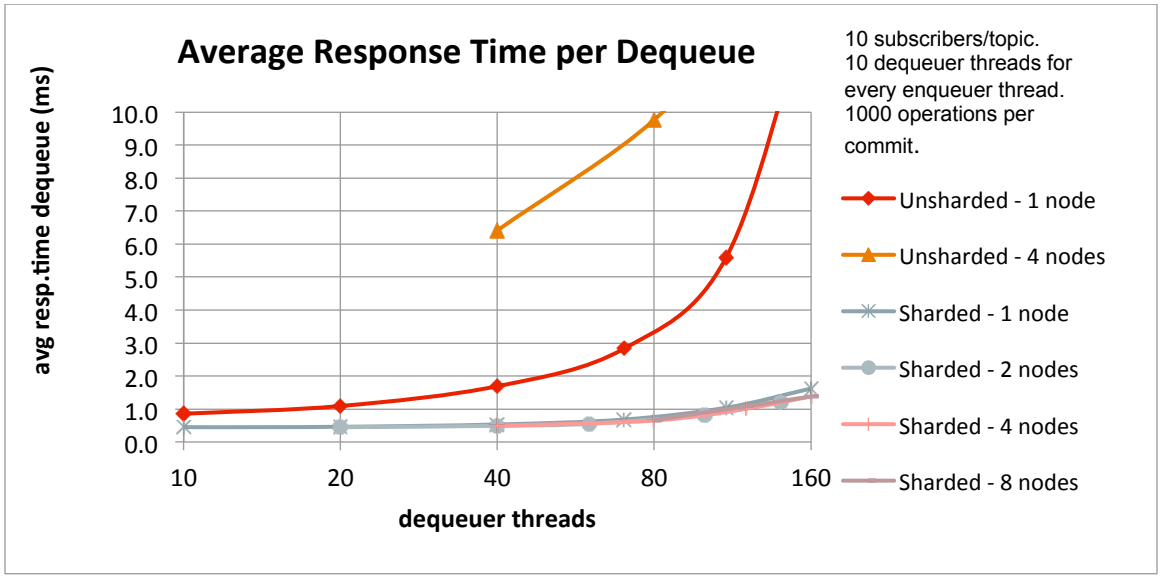

Figure 11. The response time for unsharded queues increased with more instances or with more dequeuers.

SYSTEM CONFIGURATION

The above performance tests were run on an Oracle Exadata X4-2 Database Machine. 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 caches and 94GB memory.

The tests ran Oracle Database Release 12.1.0.2. The configuration included up to eight RAC instances. JMS clients typically ran on one of 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

JMS Sharded Queues provide many advantages over unsharded AQ queues for high-end users of JMS. However, unsharded queues are a mature, popular feature of the Oracle Database. If unsharded queues satisfy your requirements, there is no compelling need to migrate to sharded queues. Furthermore, JMS sharded queues in Oracle Database Release 12.1 have many restrictions that do not apply to unsharded AQ queues. JMS Sharded Queues are aimed at the JMS standard and do not support PL/SQL interfaces for enqueue, dequeue, or notifications. Oracle-specific JMS extensions, such as array enqueue and dequeue, are not supported. JMS Sharded Queues must use the thin JDBC driver. Only unsharded queues support AQ features that are not part of the JMS standard including transactional grouping, exception queues, the messaging gateway, propagation, non-JMS ordering, and recipient lists. In 12.1, JMS applications that heavily depend upon dequeue by message ID should evaluate unsharded AQ queues.

CONVERTING TO JMS SHARDED QUEUES

Existing applications built upon the JMS standard should not require changes to use JMS Sharded Queues beyond using the 12.1 JMS and JDBC drivers. You simply need to recreate the queue when it is empty. The first step to convert an unsharded JMS queue to a sharded JMS queue is to stop the unsharded queue for enqueue operations using the DBMS_AQADM.STOP_QUEUE procedure. Then, drain the queue by dequeuing all the messages in the unsharded AQ queue. Next, stop the queue for dequeues, verify that the queue is empty, and drop the empty unsharded queue. Then, set the STREAMS_POOL_SIZE parameters. Finally, use DBMS_AQADM.CREATE_SHARDED_QUEUE() to create a sharded queue of the same name, start the queue to enable enqueues and dequeues, and grant any required privileges on the queue.

Sharded Queues introduce new or modified monitoring views or tables. Monitoring scripts based upon views or tables for unsharded 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. 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. Sharded queues perform well on a single-instance database as well as on RAC. Sharded queues have backup and recovery functionality like other database tables and support high availability via Oracle Data Guard. JMS sharded queues have set a new bar for database-backed messaging systems.

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

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

Hardware and Software, Engineered to Work Together

ORACLE DATABASE 12C: JMS SHARDED QUEUES
March 2015

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