8

Oracle Transactional Event Queues and Advanced Queuing Performance and Scalability

These topics discuss performance and scalability issues relating to Transactional Event Queues (TxEventQ) and Advanced Queuing (AQ).

Transactional Event Queues

A transactional event queue (TxEventQ) increases enqueue-dequeue throughput, especially across Oracle Real Application Clusters (Oracle RAC) instances, because messages from different enqueue sessions are allowed to be dequeued in parallel. Each event stream of the queue is ordered based on enqueue time within a session and ordering across event streams is best-effort. TxEventQs automatically manage table partitions so that enqueuers and dequeuers do not contend among themselves. In addition, TxEventQs use an in-memory message cache to optimize performance and reduce the disk and CPU overhead of enqueues and dequeues.

The advantages and tradeoffs of TxEventQs include the following:

- TxEventQs provide scalability of a single queue on Oracle RAC, especially in the case where each subscriber has multiple dequeuers on each instance.
- Oracle Real Application Clusters (Oracle RAC)s trades off increased memory usage to obtain performance.

This section contains the following topics:

- Transactional Event Queues Tuning

Transactional Event Queues and the Message Cache

TxEventQs introduce a special purpose message cache which lets you trade off SGA usage for increased throughput, reduced latency, and increased concurrency. When combined with partitioning, the message cache reduces the need for some queries, DML operations, and indexes. The message cache is most effective when all dequeuers keep up with enqueuers and when the message cache is big enough to store messages (including payloads) for each TxEventQ's enqueuers and dequeuers. The message cache uses the Streams pool. If TxEventQs share the Streams pool on the same instance as Streams replication functionality, you can use `DBMS_AQADM` procedures such as `SET_MIN_STREAMS_POOL` and `SET_MAX_STREAMS_POOL` to fine tune the allocation of Streams Pool memory.

See Also:

Oracle Database PL/SQL Packages and Types Reference for information

Transactional Event Queues and Enqueuing / Dequeuing Messages

To improve throughput and reduce overhead and latency, enqueues and dequeues are optimized to use the message cache, the rules engine, and background processing when possible. For example,

- TxEventQs take advantage of new rules engine improvements
- a message that has its payload in the message cache does not have to be re-read from disk during a dequeue
- dequeue by correlation id or other JMS properties can often be evaluated without accessing the disk
- partition operations on TxEventQs implements efficient bulk processing.

Transactional Event Queues and Native JMS Support

TxEventQs have native support for:

- Non-Durable Subscribers
- JMS payloads
- Priorities

TxEventQs support both persistent and nonpersistent messages. Nonpersistent messages are stored in memory inside the message cache and are not stored on disk. As a result, nonpersistent messages are lost upon instance crash or shutdown.

TxEventQs natively support two kinds of subscribers to meet the JMS requirements:

- Non-durable subscribers: These subscribers receive messages on their chosen topic, only if the messages are published while the subscriber is active. This subscription is not sharable among different sessions.
- Durable subscribers: These subscribers receive all the messages published on a topic, including those that are published while the subscriber is inactive. Multiple database sessions can share the same subscription.

TxEventQs do not use ADTs to store the JMS payload. The JMS message is stored in scalar columns of the database. JMS message types such as `TEXT`, `BYTES`, `MAP`, `STREAM` and `OBJECT` store the JMS payload in scalar `TEXT`/`RAW` or `CLOB`/`BLOB` columns in the queue table depending on payload size and type. The JMS message properties are stored in a `CLOB` (SecureFile) column in the queue table with attribute access functions defined for the user defined properties. The payload and user properties are pickled into `RAW`, `VARCHAR2` or Secure File columns instead of being stored as an ADT. JMS Header properties and JMS provider information are stored in their own scalar columns.

TxEventQs support integer priority values ranging between 0 (lowest priority) and 9 (highest priority), with the default being priority 4, as defined by the JMS standard.

Transactional Event Queues and Partitioning

TxEventQs automatically manage the underlying partitioned tables used for the queue table. Such partition management may occur in the foreground or the background. Each event stream provides session-level ordering of enqueued messages. Each enqueuing session is assigned an event stream. Each event stream is composed of a series of event stream partitions. Each event stream partition maps to a single partition. Messages are automatically assigned to a table partition upon enqueue.

New partitions are automatically created as needed, as when the queue table needs to grow when dequeuers do not keep up with enqueuers. Partitions are truncated and reused when all messages in the partition are dequeued and no longer needed. The message cache automatically loads messages from partitions into memory as required by dequeuers. Global indexes should not be created on the partitioned table underlying a TxEventQ. Local indexes are not typically recommended on the partitioned table either. If such indexes are desired and result in performance degradation, then AQ queues should be considered.

Transactional Event Queues and Oracle Real Application Clusters (Oracle RAC)

TxEventQs automatically provides enqueue session ordering while avoiding cross-instance communication when possible. Sometimes cross instance communication is required. For example, if a TxEventQ has a single enqueuing session on one Oracle RAC instance and a single dequeuing session on another instance, then TxEventQs will forward messages between the Oracle RAC instances. The forwarding of messages is non-atomic to the enqueuing transaction to improve performance. Dequeuers may get an ORA-25228 if they are connected to an instance that has no messages in its event streams.

In most cases, consider having multiple dequeuers for each subscriber or single consumer queue on each Oracle RAC instance to improve throughput and reduce cross-instance overhead. An exception to this guideline is when you are using dequeue selectors that specify a single message. If you want to dequeue a message from a TxEventQ by its message identifier in an Oracle RAC database, then you have to connect to the instance that is assigned dequeue ownership for the event stream containing the message. Otherwise, the message will not be available for dequeue to the dequeue session. If all dequeues are performed at a single instance, then messages will be automatically forwarded to this instance. Hence, for a single-consumer TxEventQ that extensively dequeues by message ID, consider having all dequeue sessions for the TxEventQ connect to a single instance. Similarly, for a multiconsumer TxEventQ that extensively dequeues by message ID, consider having all dequeue sessions for each subscriber connect to a single instance. Services can be used to simplify connecting dequeue sessions to a particular instance.

Transactional Event Queues and Message Retention

Starting from Oracle Database Release 21c, message retention is supported by TxEventQ . AQ queue already has this feature.

Message retention is the time for which a message is retained in the TxEventQ after being enqueued or dequeued as desired. The default is 0, which means that the message will be removed as soon as possible after it is dequeued by all of its subscribers. This helps users to retain the messages in the queue even after they are processed.

Applications can specify retention time while creating a TxEventQ. Applications can change the retention time and its type as needed after creation of the TxEventQ.

TxEventQ supports only dequeue time based retention. A event stream partition stores a set of messages. The event stream partition will be removed from the queue when the highest dequeue time for any message-subscriber pair in that event stream partition plus retention time is over. This scheme will ensure consumption of messages before retention comes in play. This is also the only retention policy available in AQ queues.

Transactional Event Queues and Seekable Subscribers

A seek operation for a subscriber can be for all event streams (queue level seek) or a set of specific event streams of choice (event stream level seek).

All the dequeue calls after a seek operation would dequeue messages from the seek point onwards. All the messages below seek point will never be dequeued or browsed by the subscriber unless the subscriber seeks back again.

Seek Granularity

A subscriber can perform seek in all event streams or a set of event streams of choice in the queue. The choice of the message to seek to can be explicitly specified in the seek operation or can be deduced from the inputs of seek operation.

Following are the different types of seek option inputs.

- Seek to end – With this seek option, the subscriber is not interested in existing messages. The subscriber will be able to dequeue only newly enqueued messages after the seek operation. This is the default behavior when a new subscriber is created.
- Seek to start - With this seek option, the subscriber is interested in existing messages including "retained". The subscriber will also be able to dequeue newly enqueued messages after the seek operation.
- Seek to a specific time - With this seek option, the subscriber is interested in existing messages including "retained" with enqueue time higher than input time. Seek stops if start or end is reached.
- Seek to a specific message – With this seek option, the subscriber is interested in existing messages including "retained" from the input message onwards. The input in this case is a specific message id so this seek always a event stream level seek. A separate unique message id per event stream is specified in the input for all the event streams on which seek needs to be performed.

Message Ordering

A seek action can break message ordering as it results in out of order dequeues. If the new seek point is a message which is not the first message of an enqueue transaction or an enqueue session and messages from new seek point onwards are dequeued, then the application can get only some messages of the enqueue session or enqueue transaction as remaining messages were enqueued before the new seek point.

It is application's responsibility to either choose correct seek point or be tolerant to such behavior.

Transactional Event Queues Restrictions

The following Oracle Database features are not currently supported for TxEventQs:

- Transaction grouping
- Anonymous posting for subscriber notification and OCI callback notification are not supported. PL/SQL callback notification is supported.
- Messaging Gateway
- Oracle extensions for JMS such as JMS propagation and remote subscribers
- Multiple queues per queue table. TxEventQs are created via the `CREATE_TRANSACTIONAL_EVENT_QUEUE` interface.
- Ordering other than message priority followed by enqueue time (as specified in the JMS standard)
- The JDBC thick (OCI) driver.
- Propagation between TxEventQ and AQ queues
- Message transformations

Transactional Event Queues Tuning

TxEventQs perform best under the following conditions:

- Dequeuers for each subscriber are located on each instance
- Subscribers keep up with the enqueuers. Consider having multiple dequeuers for each subscriber on each Oracle RAC instance

The message cache is most effective when dequeuers keep up with enqueuers and where the cache is big enough to store messages (including payloads) for each TxEventQ's enqueuers and dequeuers. When using TxEventQs, Oracle requires that you do one of the following:

- Setting parameter `STREAMS_POOL_SIZE`

  This parameter controls the size of shared memory available to the Oracle Database for the TxEventQ message cache. If unspecified, up to 10% of the shared pool size may be allocated for the Streams pool.

  Oracle's Automatic Shared Memory Management feature manages the size of the Streams pool when the `SGA_TARGET` initialization parameter is set to a nonzero value. If the `STREAMS_POOL_SIZE` initialization parameter also is set to a nonzero value, then Automatic Shared Memory Management uses this value as a minimum for the Streams pool.

  If the `STREAMS_POOL_SIZE` initialization parameter is set to a nonzero value, and the `SGA_TARGET` parameter is set to `0` (zero), then the Streams pool size is the value specified by the `STREAMS_POOL_SIZE` parameter, in bytes.

  If both the `STREAMS_POOL_SIZE` and the `SGA_TARGET` initialization parameters are set to `0` (zero), then, by default, the first use of the Streams pool in a database transfers an amount of memory equal to 10% of the shared pool from the buffer cache to the Streams pool.

> **See Also:**
>
> – `DBMS_AQADM.set_min_streams_pool()` and `DBMS_AQADM.set_max_streams_pool( )` in *Oracle Database PL/SQL Packages and Types Reference* for a finer grained control over `STREAMS_POOL` sharing with Streams processing.

- Turning on SGA autotuning

  Oracle will automatically allocate the appropriate amount of memory from the SGA for the Streams pool, based on Streams pool usage as well as usage of other components that use the SGA. Examples of such other components are buffer cache and library cache. If `STREAMS_POOL_SIZE` is specified, it is used as the lower bound.

- Manually tuning TxEventQ queues

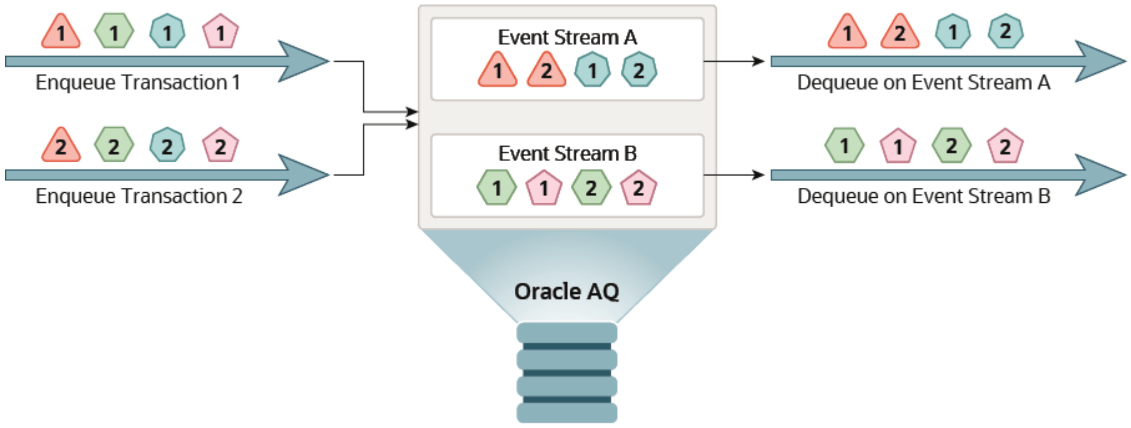
  TxEventQs can be tuned by allocating `STREAMS_POOL` memory for the message cache. The view `GV$AQ_MESSAGE_CACHE_ADVICE` provides advice on how much `STREAMS_POOL` should be allocated for TxEventQs based on a snapshot of the current messaging load. During periods of high load, select the columns `INST_ID`, `SIZE_FOR_ESTIMATE`, and `ESTD_SIZE_TYPE`. `ESTD_SIZE_TYPE` is one of three values: `MINIMUM`, `PREFERRED`, or `MAXIMUM`. Find the maximum value of `SIZE_FOR_ESTIMATE` across Oracle RAC instances for each of the `ESTD_SIZE_TYPE` values. It is highly recommended that `STREAMS_POOL` be set at least to the `MINIMUM` recommendation to provide any message cache performance gains. There is little additional performance gains to setting `STREAMS_POOL` greater than the `MAXIMUM` recommendation value. Setting `STREAMS_POOL` to the `PREFERRED` recommendation tries to provide a reasonable space-performance tradeoff. If the `MAXIMUM` size recommendation is much greater than the `PREFERRED` recommendation, then check that the TxEventQ has no orphaned subscribers, or whether more dequeuers should be added to the instance, so that dequeuers can keep up with the enqueue load. `STREAMS_POOL` tuning should be done over multiple periods of high load and whenever messaging load characteristics change.

User Event Streaming

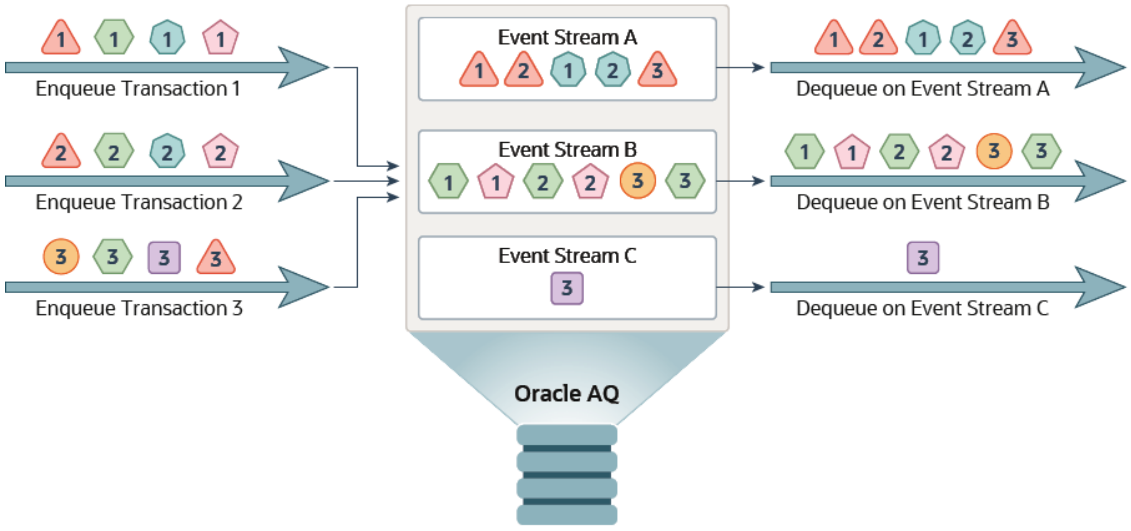
An application can decide the way messages are event streamed in the TxEventQ. In such cases, the application explicitly specifies to enqueue a message in a specific event stream.

For example, assume the application has four types of messages with different keys named red, green, blue, and pink. Each enqueue session can enqueue any of those messages in a transaction. Event Stream A is set to store the red and blue messages. Event Stream B is set to store the green and pink messages. Also, each event stream is set to have only one active dequeue session for a single-consumer queue or JMS Queue. Similarly, each event stream is set to have only one dequeue session per subscriber for a multi-consumer queue or JMS Topic. That dequeue session will stick to that event stream for the dequeuer session’s lifetime.

In the following examples, enqueue transactions are performing enqueues in parallel.



Applications can add new event streams at run time. Applications can also add new types of messages at run time by adding new keys. For example, two new types are introduced with keys orange and purple, and a third Event Stream C is added. Event Stream B is set to store the orange messages. Event Stream C is set to store the purple messages.



In an Oracle RAC database, an event stream is always owned by an instance. Initially, the event stream is owned by the instance where the first message is enqueued in that event stream. The owner instance of the event stream may change when database instances are shut down.

With user event streaming, a user can attempt to enqueue messages in an event stream which is not owned by the instance in which the session is running. In such cases, a cross instance enqueue is triggered. To support cross instance enqueues, the enqueue requests received at other instances are forwarded to the `OWNER INSTANCE` of the event stream over the cluster interconnect for the Oracle RAC database. The `REMOTE_LISTENER` parameter in listener.ora must also be set to enable forwarding of cross instance enqueue requests to the correct instance. Internally, TxEventQ queues on an Oracle RAC database may use database links between instances. Definer's rights PL/SQL packages that perform cross instance enqueues in TxEventQ queues on an Oracle RAC database must grant `INHERIT REMOTE PRIVILEGES` to users of the package.

Example 8-1 Setting REMOTE_LISTENER Parameter for Cross Instance Enqueues

In order to enable cross instance enqueue for TxEventQ, the `REMOTE_LISTENER` parameter must be set for each instance to have listener addresses for every other instance available in Oracle RAC.

For example, consider an Oracle RAC cluster with 4 nodes. In such setup, `REMOTE_LISTSENER` at each node should have listener addresses of all four nodes of the Oracle RAC as follows.

```
REMOTE_LISTENER = (DESCRIPTION= (ADDRESS_LIST=
  (ADDRESS= (PROTOCOL=TCP) (PORT=<node1 PORT>) (HOST=<node1 IP>))
  (ADDRESS= (PROTOCOL=TCP) (PORT=<node2 PORT>) (HOST=<node2 IP>))
  (ADDRESS= (PROTOCOL=TCP) (PORT=<node3 PORT>) (HOST=<node3 IP>))
  (ADDRESS= (PROTOCOL=TCP) (PORT=<node4 PORT>) (HOST=<node4 IP>))
))
```

Alternatively, if TNS aliases are set with listener addresses of Oracle RAC nodes in `tnsnames.ora` as follows,

```
nd1=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(PORT=<node1 PORT>)(HOST=<node1 IP>)))
nd2=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(PORT=<node2 PORT>)(HOST=<node2 IP>)))
nd3=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(PORT=<node3 PORT>)(HOST=<node3 IP>)))
nd4=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(PORT=<node4 PORT>)(HOST=<node4 IP>)))
```

then, `REMOTE_LISTENER` can be set as:

```
REMOTE_LISTENER='nd1,nd2,nd3,nd4'
```

Once `REMOTE_LISTENER` is set correctly at each node, then a message with any key can be enqueued at any instance using the AQJMS client library and the JDBC thin driver in TxEventQ. In the example below, key `"RED"` can be enqueued at any Oracle RAC instance. It will be published in the correct event stream which stores all `"RED"` messages. Please note that it is advisable that a message publisher connects to the owner instance of the event stream to achieve high performance.

```
TopicConnectionFactory    tc_fact   = null;
TopicConnection           t_conn    = null;
TopicSession              jms_sess;
TopicPublisher            publisher1;
Topic                     shipped_orders;
int                       myport = 5521;

/* create connection and session */
tc_fact = AQjmsFactory.getTopicConnectionFactory(...);
t_conn = tc_fact.createTopicConnection();
jms_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);
shipped_orders = ((AQjmsSession )jms_sess).getTopic("OE", "Shipped_Orders_Topic");
publisher1 = jms_sess.createPublisher(shipped_orders);

/* Create TextMessage */
TextMessage     text_message = jms_sess.createTextMessage();

/* Set key as correlation */
text_message.setJMSCorrelationID("RED");

/* Publish  */
publisher1.publish(text_message);
```

Limitations of User Event Streaming

User event streaming has the following limitations:

- Cross instance enqueues are not enabled for PL/SQL enqueue calls.
- Cross instance enqueues are not enabled for array enqueues.

Cross instance enqueues can be done through Java and OCI clients.

See Also:

DBMS_AQADM in *Oracle Database PL/SQL Packages and Types Reference* for more information.

AQ Queues

This section includes the following topics:

- Persistent Messaging Performance Overview for Queues
- Persistent Messaging Basic Tuning Tips
- Propagation Tuning Tips
- Buffered Messaging Tuning

Persistent Messaging Basic Tuning Tips

Oracle Database Advanced Queuing table layout is similar to a layout with ordinary database tables and indexes.

See Also:

Oracle Database Performance Tuning Guide for tuning recommendations

Memory Requirements

Streams pool size should be at least 20 MB for optimal multi-consumer dequeue performance in a non-Oracle RAC database.

Persistent queuing dequeue operations use the streams pool to optimize performance, especially under concurrency situations. This is, however, not a requirement and the code automatically switches to a less optimal code path.

TxEventQs introduces a message cache for optimal performance of high throughput messaging systems. Ideally the Streams pool size should be large enough to cache the expected backlog of messages in TxEventQs.

Using Storage Parameters

Storage parameters can be specified when creating a queue table using the `storage_clause` parameter.

Storage parameters are inherited by other IOTs and tables created with the queue table. The tablespace of the queue table should have sufficient space to accommodate data from all the objects associated with the queue table. With retention specified, the history table and, also the queue table can grow to be quite big.

Oracle recommends you use automatic segment-space management (ASSM). Otherwise `initrans`, freelists and freelist groups must be tuned for AQ performance under high concurrency.

Increasing `PCTFREE` will reduce the number of messages in a queue table/IOT block. This will reduce block level contention when there is concurrency.

Storage parameters specified at queue table creation are shared by the queue table, IOTs and indexes. These may be individually altered by an online redefinition using `DBMS_REDEFINITION`.

I/O Configuration

Because Oracle Database Advanced Queuing is very I/O intensive, you will usually need to tune I/O to remove any bottlenecks.

See Also:

"I/O Configuration and Design" in *Oracle Database Performance Tuning Guide*

Running Enqueue and Dequeue Processes Concurrently in a Single Queue Table

Some environments must process messages in a constant flow, requiring that enqueue and dequeue processes run concurrently. If the message delivery system has only one queue table and one queue, then all processes must work on the same segment area at the same time. This precludes reasonable performance levels when delivering a high number of messages.

The best number for concurrent processes depends on available system resources. For example, on a four-CPU system, it is reasonable to start with two concurrent enqueue and two concurrent dequeue processes. If the system cannot deliver the wanted number of messages, then use several subscribers for load balancing rather than increasing the number of processes.

Tune the enqueue and dequeue rates on the queue so that in the common case the queue size remains small and bounded. A queue that grows and shrinks considerably will have indexes and IOTs that are out of balance, which will affect performance.

With multi-consumer queues, using several subscribers for load balancing rather than increasing the number of processes will reduce contention. Multiple queue tables may be used garnering horizontal scalability.

For information about tuning TxEventQs refer to Transactional Event Queues Tuning.

Running Enqueue and Dequeue Processes Serially in a Single Queue Table

When enqueue and dequeue processes are running serially, contention on the same data segment is lower than in the case of concurrent processes. The total time taken to deliver messages by the system, however, is longer than when they run concurrently.

Increasing the number of processes helps both enqueuing and dequeuing. The message throughput rate may be higher for enqueuers than for dequeuers when the number of processes is increased, especially with single consumer queues. Dequeue processes on multi-consumer queues scale much better.

Creating Indexes on a Queue Table

Creating an index on a queue table is useful if you meet these conditions.

- Dequeue using correlation ID

  An index created on the column `corr_id` of the underlying queue table `AQ$_`*`QueueTableName`* expedites dequeues.
- Dequeue using a condition

  This is like adding the condition to the where-clause for the `SELECT` on the underlying queue table. An index on *`QueueTableName`* expedites performance on this `SELECT` statement.

Other Tips for Queues

These are some other persistent messaging basic tuning tips.

- Ensure that statistics are being gathered so that the optimal query plans for retrieving messages are being chosen. By default, queue tables are locked out from automatic gathering of statistics. The recommended use is to gather statistics with a representative queue message load and lock them.
- The queue table indexes and IOTs are automatically coalesced by AQ background processes. However, they must continue to be monitored and coalesced if needed. With automatic space segment management (ASSM), an online shrink operation may be used for the same purpose. A well balanced index reduces queue monitor CPU consumption, and ensures optimal enqueue-dequeue performance.
- Ensure that there are enough queue monitor processes running to perform the background tasks. The queue monitor must also be running for other crucial background activity. Multiple `qmn` processes share the load; make sure that there are enough of them. These are auto-tuned, but can be forced to a minimum number, if needed.
- It is recommended that dequeue with a wait time is only used with dedicated server processes. In a shared server environment, the shared server process is dedicated to the dequeue operation for the duration of the call, including the wait time. The presence of many such processes can cause severe performance and scalability problems and can result in deadlocking the shared server processes.
- Long running dequeue transactions worsen dequeue contention on the queue, and must be avoided.
- Batching multiple dequeue operations on multi-consumer queues into a single transaction gives best throughput.

- Use `NEXT` as navigation mode, if not using message priorities. This offers the same semantics but improved performance.
- Use the `REMOVE_NODATA` dequeue mode if dequeuing in `BROWSE` mode followed by a `REMOVE`.

Propagation Tuning Tips

Propagation can be considered a special kind of dequeue operation with an additional `INSERT` at the remote (or local) queue table. Propagation from a single schedule is not parallelized across multiple job queue processes. Rather, they are load balanced.Propagation can be considered a special kind of dequeue operation with an additional `INSERT` at the remote (or local) queue table. Propagation from a single schedule is not parallelized across multiple job queue processes. Rather, they are load balanced.

For better scalability, configure the number of propagation schedules according to the available system resources (CPUs).

Propagation rates from transactional and nontransactional (default) queue tables vary to some extent because Oracle Database Advanced Queuing determines the batching size for nontransactional queues, whereas for transactional queues, batch size is mainly determined by the user application.

Optimized propagation happens in batches. If the remote queue is in a different database, then Oracle Database Advanced Queuing uses a sequencing algorithm to avoid the need for a two-phase commit. When a message must be sent to multiple queues in the same destination, it is sent multiple times. If the message must be sent to multiple consumers in the same queue at the destination, then it is sent only once.

Buffered Messaging Tuning

Buffered messaging operations in a Oracle Real Application Clusters environment will be fastest on the `OWNER_INSTANCE` of the queue.

Persistent Messaging Performance Overview for Queues

When persistent messages are enqueued, they are stored in database tables. The performance characteristics of queue operations on persistent messages are similar to underlying database operations.

The code path of an enqueue operation is comparable to `SELECT` and `INSERT` into a multicolumn queue table with three index-organized tables. The code path of a dequeue operation is comparable to a `SELECT` operation on the multi-column table and a `DELETE` operation on the dequeue index-organized table. In many scenarios, for example when Oracle RAC is not used and there is adequate streams pool memory, the dequeue operation is optimized and is comparable to a `SELECT` operation on a multi-column table.

Note:

Performance is not affected by the number of queues in a table.

Queues and Oracle Real Application Clusters

Oracle Real Application Clusters (Oracle RAC) can be used to ensure highly available access to queue data.

The entry and exit points of a queue, commonly called its tail and head respectively, can be extreme hot spots. Because Oracle RAC may not scale well in the presence of hot spots, limit usual access to a queue from one instance only. If an instance failure occurs, then messages managed by the failed instance can be processed immediately by one of the remaining instances. If AQ queues are experiencing hot spots, then consider using TxEventQs instead.

You can associate Oracle RAC instance affinities with 8.1-compatible queue tables. If you are using `q1` and `q2` in different instances, then you can use `ALTER_QUEUE_TABLE` or `CREATE_QUEUE_TABLE` on the queue table and set `primary_instance` to the appropriate `instance_id`.

See Also:

- Creating a Queue Table
- Altering a Queue Table
- Transactional Event Queues and Oracle Real Application Clusters (Oracle RAC)

Oracle Database Advanced Queuing in a Shared Server Environment

Queue operation scalability is similar to the underlying database operation scalability.

If a dequeue operation with wait option is applied, then it does not return until it is successful or the wait period has expired. In a shared server environment, the shared server process is dedicated to the dequeue operation for the duration of the call, including the wait time. The presence of many such processes can cause severe performance and scalability problems and can result in deadlocking the shared server processes. For this reason, Oracle recommends that dequeue requests with wait option be applied using dedicated server processes. This restriction is not enforced.

See Also:

"DEQUEUE_OPTIONS_T Type" in *Oracle Database PL/SQL Packages and Types Reference* for more information on the wait option

Performance Views

Oracle provides these views to monitor system performance and troubleshooting.

- V$AQ_MESSAGE_CACHE_STAT: Memory Management for Sharded Queues
- V$AQ_SHARDED_SUBSCRIBER_STAT: Sharded Queue Subscriber Statistics
- V$AQ_MESSAGE_CACHE_ADVICE: Simulated Metrics
- V$AQ_REMOTE_DEQUEUE_AFFINITY: Dequeue Affinity Instance List

- V$PERSISTENT_QUEUES: All Active Persistent Queues in the Instance
- V$PERSISTENT_SUBSCRIBERS: All Active Subscribers of the Persistent Queues in the Instance
- V$PERSISTENT_PUBLISHERS: All Active Publishers of the Persistent Queues in the Instance
- V$BUFFERED_QUEUES: All Buffered Queues in the Instance.
- V$BUFFERED_SUBSCRIBERS: Subscribers for All Buffered Queues in the Instance
- V$BUFFERED_PUBLISHERS: All Buffered Publishers in the Instance
- V$PERSISTENT_QMN_CACHE: Performance Statistics on Background Tasks for Persistent Queues
- V$AQ: Number of Messages in Different States in Database
- V$AQ_BACKGROUND_COORDINATOR: Performance Statistics for AQ's Master Background Coordinator Process (AQPC)
- V$AQ_JOB_COORDINATOR: Performance Statistics per Coordinator
- V$AQ_NONDUR_REGISTRATIONS: Non-Durable Registrations
- V$AQ_SERVER_POOL: Performance Statistics for all Servers
- V$AQ_CROSS_INSTANCE_JOBS: Cross Process Jobs Description
- V$AQ_NONDUR_REGISTRATIONS: Non-Durable Registrations
- V$AQ_NOTIFICATION_CLIENTS: Secure OCI Client Connections
- V$AQ_SUBSCRIBER_LOAD: Durable Subscribers
- V$AQ_NONDUR_SUBSCRIBER: Non-Durable Subscribers
- V$AQ_NONDUR_SUBSCRIBER_LWM: LWM of Non Durable Subscriber
- V$AQ_MESSAGE_CACHE: Performance Statistics

These views are integrated with the Automatic Workload Repository (AWR). Users can generate a report based on two AWR snapshots to compute enqueue rate, dequeue rate, and other statistics per queue/subscriber.

Migrating from AQ to TxEventQ

- Flowchart: Migration from AQ to TxEventQ
- Example Walkthrough
- Steps to Migrate from AQ to TxEventQ
- Overview of How Migration Functions
- Limitations and Workarounds

Transactional Event Queues (TxEventQ) is the next generation messaging system by Oracle that offers many benefits and features over Advanced Queuing (AQ), like scalability, performance, key-based sharding, and Kafka compatibility with a Java client. The online migration tool offers user-friendly interfaces designed to streamline the process of migrating from AQ to TxEventQ. Its intuitive user experience simplifies even the most intricate scenarios, sparing users the need to delve into the complexities of AQ objects. The tool incorporates built-in compatibility diagnostics between AQ and TxEventQ. This diagnosis involves the validation of AQ metadata and data, enabling interactive solutions to any encountered issues.

Furthermore, the tool supplies straightforward procedures to address fallback and recovery scenarios in the event of any potential failures.

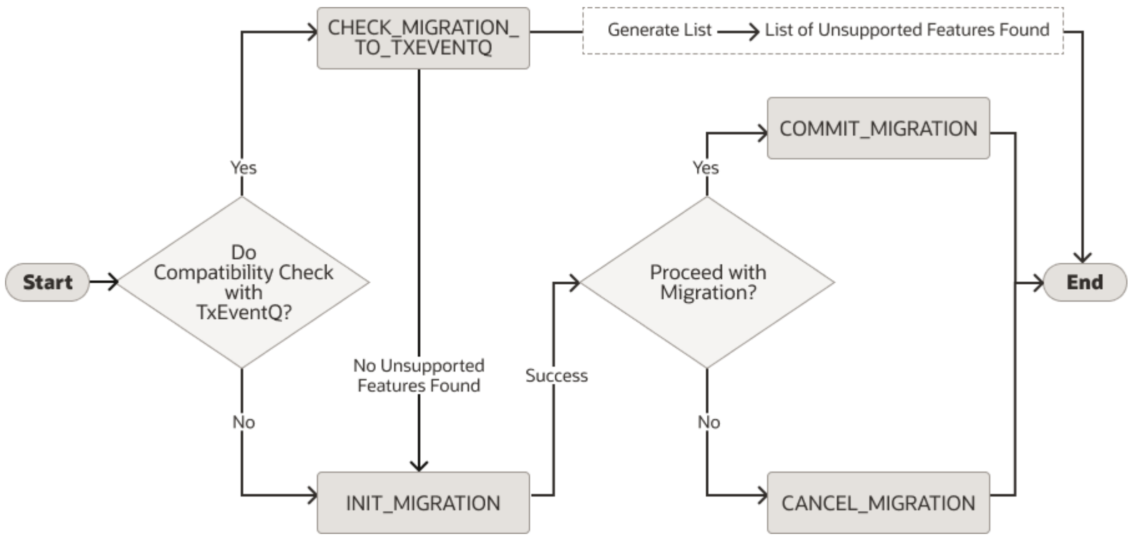
Upgrading AQ to TxEventQ is not a standard procedure like typical upgrades due to substantial feature disparities, significant data representation changes, and other factors. Manually migrating entails system downtime and a deep understanding of the internal mechanisms of both AQ and TxEventQ. The intricacies of handling internal data from various versions of AQ or TxEventQ, alongside existing messages, can present significant challenges. Moreover, rewriting applications to integrate TxEventQ can be expensive. To address these complexities, Oracle Database 23ai introduces the PL/SQL package `DBMS_AQMIGTOOL`, designed to automate the various steps involved in migration. This package offers multiple migration modes, including `AUTOMATIC`, `INTERACTIVE`, `OFFLINE`, and `ONLY_DEFINITION`, providing flexibility to align with specific requirements.

The migration tool interface provides the following functionalities:

- Inspect AQ definition and data, and report unsupported features.
- Approve or prevent migration based on detected unsupported features.
- Initiate migration with modes: `AUTOMATIC`, `INTERACTIVE`, `OFFLINE`, `ONLY_DEFINITION`.
- Offer migration commit or AQ fallback via the user-initiated procedure in the migration interface.
- Keep track of messages that are not currently in the `PROCESSED` state in both the AQ and TxEventQ.
- Migration history record of any queue.
- Provide an option to purge AQ messages if they don't want to consume old messages.
- The migration procedures support rolling upgrades. The migration procedures also support Oracle GoldenGate replicas(OGG) replication.

Flowchart: Migration from AQ to TxEventQ

Figure 8-1 Flowchart: Migration from AQ to TxEventQ


CHECK_MIGRATION_TO_TXEVENTQ
Generate List
List of Unsupported Features Found
COMMIT_MIGRATION
Yes
Yes
Start
Do Compatibility Check with TxEventQ?
Proceed with Migration?
End
No Unsupported Features Found
Success
No
No
INIT_MIGRATION
CANCEL_MIGRATION

Example Walkthrough

Let's go through a few hypothetical scenarios to enhance our understanding of how to use `DBMS_AQMIGTOOL` procedures.

Scenario 1

Consider a scenario involving a user named `aquser` with an AQ named `JSON_QUEUE`. This AQ stores customer events in JSON payload format. If the user wishes to determine whether any features used by AQ are unsupported in TxEventQ, then they can run the `DBMS_AQMIGTOOL.CHECK_MIGRATION_TO_TXEVENTQ` procedure.

```
SQL> DECLARE
  2    migration_report sys.TxEventQ_MigReport_Array := sys.TxEventQ_MigReport_Array();
  3  BEGIN
  4    DBMS_AQMIGTOOL.CHECK_MIGRATION_TO_TXEVENTQ('aquser', 'JSON_QUEUE',
migration_report);
  5    dbms_output.put_line('Migration report unsupported events count: ' ||
migration_report.COUNT);
  6
  7  END;
  8  /
Migration report unsupported events count: 0

PL/SQL procedure successfully completed.
```

After examining `JSON_QUEUE`, it is evident that no unsupported features are present. Consequently, the user can proceed with migration by initiating the `DBMS_AQMIGTOOL.INIT_MIGRATION` procedure.

```
SQL> EXECUTE DBMS_AQMIGTOOL.INIT_MIGRATION(cqschema => 'aquser', cqname => 'JSON_QUEUE');

PL/SQL procedure successfully completed.

SQL> SELECT name || ' -> ' || sharded AS "QueueName -> Sharded" FROM user_queues where
queue_type = 'NORMAL_QUEUE';

QueueName -> Sharded
--------------------------------------------------------------------------------
JSON_QUEUE -> FALSE
JSON_QUEUE_M -> TRUE
```

The migration process begins by creating an interim TxEventQ `JSON_QUEUE_M` (with the default suffix `M`) with the same configuration as AQ. Upon executing this procedure, the user can carry out data manipulation operations such as enqueue and dequeue.

> **Note:**
>
> No adjustments are required in the enqueue and dequeue calls; the workload can proceed unchanged.

Users can verify successful evaluation by querying in the `USER_TXEVENTQ_MIGRATION_STATUS` view.

```
SQL> SELECT event FROM USER_TXEVENTQ_MIGRATION_STATUS WHERE source_queue_name =
'JSON_QUEUE' AND event = 'Init_Migration';
```

```
EVENT
--------------------------------------------------------------------------------
Init_Migration
```

Pre-requisite requirement for `DBMS_AQMIGTOOL.COMMIT_MIGRATION` is to have empty AQ (all messages are either in the `PROCESSED` or `EXPIRED` state). The user can utilize the `DBMS_AQMIGTOOL.CHECK_MIGRATED_MESSAGES` procedure to monitor messages within the AQ and interim TxEventQ.

```
SQL> DECLARE
      migrated_q_msg_cnt    number := 0;
      aq_msg_cnt            number := 0;
      BEGIN
          DBMS_AQMIGTOOL.CHECK_MIGRATED_MESSAGES(
                       cqschema                   => 'aquser',
                       cqname                     => 'JSON_QUEUE',
                       txeventq_migrated_message  => migrated_q_msg_cnt,
                       cq_pending_messages        => aq_msg_cnt);
         dbms_output.put_line('AQ ready state message count: ' || aq_msg_cnt);
         dbms_output.put_line('Migrated TxEventQ message count: ' ||
migrated_q_msg_cnt);
      END;
      /
AQ ready state message count: 1000
Migrated TxEventQ message count: 3500

PL/SQL procedure successfully completed.
```

Once the `READY` state message count in AQ reaches zero, the user has the option to complete the migration by executing the `DBMS_AQMIGTOOL.COMMIT_MIGRATION` procedure.

```
SQL> EXECUTE DBMS_AQMIGTOOL.COMMIT_MIGRATION(cqschema => 'aquser', cqname =>
'JSON_QUEUE');

PL/SQL procedure successfully completed.

SQL> SELECT name || ' -> ' || sharded AS "QueueName -> Sharded" FROM user_queues where
queue_type = 'NORMAL_QUEUE';

QueueName -> Sharded
--------------------------------------------------------------------------------
JSON_QUEUE -> TRUE
```

Following the successful migration, the `JSON_QUEUE` has been transformed from an AQ to a TxEventQ. The user can seamlessly proceed with their everyday operations as usual on `JSON_QUEUE`.

Users can verify successful evaluation by querying in the `USER_TXEVENTQ_MIGRATION_STATUS` view.

```
SQL> SELECT event FROM USER_TXEVENTQ_MIGRATION_STATUS WHERE source_queue_name =
'JSON_QUEUE' AND event = 'Commit_Migration';

EVENT
--------------------------------------------------------------------------------
Commit_Migration
```

Scenario 2

Let's consider the same example outlined in Scenario 1, continuing until the successful execution of `DBMS_AQMIGTOOL.INIT_MIGRATION`.

In certain situations, after executing a workload, it's possible that unsupported features may be detected. For example, if a user configured a transformation in enqueue operation. In such cases, the user can use the `DBMS_AQMIGTOOL.CHECK_STATUS` procedure to determine if the migration process can be successfully finalized or needs an application change to clear the error.

```
SQL> DECLARE
     2        mig_STATUS    VARCHAR2(128);
     3        mig_comments  VARCHAR2(1024);
     4    BEGIN
     5        DBMS_AQMIGTOOL.CHECK_STATUS(
     6                   cqschema                 => 'aquser',
     7                   cqname                   => 'JSON_QUEUE',
     8                   status                   => mig_STATUS,
     9                   migration_comment        => mig_comments);
    10       dbms_output.put_line('Migration Status: ' || mig_STATUS);
    11       dbms_output.put_line('Migration comments: ' || mig_comments);
    12    END;
    13    /
Migration Status: Compatibility Error: Transformation in Enq Unsupported Feature
Migration comments: Unsupported parameter in Enqueue

PL/SQL procedure successfully completed.
```

Additionally, the user has the option to examine the list of captured unsupported features by querying the `USER_TXEVENTQ_MIGRATION_STATUS` view.

```
SQL> SELECT event FROM USER_TXEVENTQ_MIGRATION_STATUS WHERE source_queue_name =
'JSON_QUEUE' AND event_status = 2;

EVENT
--------------------------------------------------------------------------------
Transformation in Enq
```

If the user later decides to cancel the migration process without any message loss, they can use the `DBMS_AQMIGTOOL.RESTORE` option in the `DBMS_AQMIGTOOL.CANCEL_MIGRATION` procedure, which is set by default.

```
SQL> SELECT count(*) FROM JSON_QUEUE_TABLE WHERE q_name = 'JSON_QUEUE';

  COUNT(*)
----------
      1000

1 row selected.


SQL> EXECUTE DBMS_AQMIGTOOL.CANCEL_MIGRATION(cqschema => 'aquser', cqname =>
'JSON_QUEUE', cancelmode => DBMS_AQMIGTOOL.RESTORE);

PL/SQL procedure successfully completed.
```

```
SQL> SELECT name || ' -> ' || sharded AS "QueueName -> Sharded" FROM user_queues WHERE
queue_type = 'NORMAL_QUEUE';

QueueName -> Sharded
--------------------------------------------------------------------------------
JSON_QUEUE -> FALSE


SQL> SELECT count(*) FROM JSON_QUEUE_TABLE WHERE q_name = 'JSON_QUEUE';

  COUNT(*)
----------
      4500

1 row selected.
```

As evident, the message counts within the queue table following the execution of `DBMS_AQMIGTOOL.CANCEL_MIGRATION` reflects the initial count (`1000` messages) and the messages restored from TxEventQ (`3500` messages).

Steps to migrate from AQ to TxEventQ

To migrate from AQ to TxEventQ, perform the following steps:

1. Identify if source AQ is compatible with migration.

   Start by determining whether the source AQ is suitable for migration to TxEventQ. The procedure below enables the identification of any features utilized by AQ that are not supported in TxEventQ.

   ```
   DBMS_AQMIGTOOL.CHECK_MIGRATION_TO_TXEVENTQ(cqschema, cqname, migration_report,
   checkmode)
   ```

   The `DBMS_AQMIGTOOL.CURRENT` option for the `checkmode` parameter only examines current queue metadata and existing messages. However, future queue operations won't be examined. Furthermore, like transformations, particular features don't rely on persistent queue data, which cannot be identified without runtime enqueue/dequeue operations. To comprehensively identify all unsupported features, it's advisable to use the `DBMS_AQMIGTOOL.ENABLE_EVALUATION` option in the `checkmode` parameter and subsequently run your user application.

   The `migration_report` will generate no data if the output is compatible. Once it is confirmed that there is no incompatibility, the user can proceed with `DBMS_AQMIGTOOL.INIT_MIGRATION`. Users should not initiate migration APIs if any of the features listed under Limitations is used in their application. Most of these are not very commonly used.

2. Start migration and create an interim TxEventQ.

   To begin the migration process, start by creating an interim TxEventQ using this method:

   ```
   DBMS_AQMIGTOOL.INIT_MIGRATION(cqschema, cqname, txeventqschema, txeventqname,
   mig_mode, ordering, suffix)
   ```

   Through the execution of `DBMS_AQMIGTOOL.INIT_MIGRATION`, compatibility checks are conducted using both AQ definitions and data. If any unsupported features are detected, an exception will be raised. In such instances, users have the opportunity to modify their applications, address the identified incompatibilities (refer to the Limitations and Workarounds section for guidance), and subsequently reattempt the execution of `INIT_MIGRATION`.

If found compatible, this procedure initiates the migration process and creates an interim TxEventQ with the same configuration of the AQ, encompassing queue properties, payload types, and subscriber data. It is important to note that following the execution of this procedure, all the administrative operations on the AQ are restricted.

3. Finalize migration either by committing or canceling migration.

   Concluding the migration process involves two options: committing or canceling the migration.

   - In cases where all messages from AQ have been consumed (indicating that all messages are in a `PROCESSED` state) and no incompatibilities found, the user can execute the following to complete the migration:

     ```
     DBMS_AQMIGTOOL.COMMIT_MIGRATION(cqschema, cqname, ignore_warning)
     ```

     This procedure drops the AQ and renames the interim TxEventQ to the AQ's name. Additionally, the TxEventQ is enabled for all administrative operations, completing the migration process.

   - Suppose a user gets any compatibility error after `DBMS_AQMIGTOOL.INIT_MIGRATION`, that operation will be blocked, and the user will be asked to review compatibility issues. Unsupported features will be recorded in the `USER_TXEVENTQ_MIGRATION_STATUS` view. Users facing such issues have the option to suitably modify their application to address the incompatibility or invoke the procedure to restore the AQ.

     ```
     DBMS_AQMIGTOOL.CANCEL_MIGRATION(cqschema, cqname, cancelmode)
     ```

     Utilizing `DBMS_AQMIGTOOL.CANCEL_MIGRATION`, users can restore TxEventQ messages and their states back to AQ based on the specified `cancelmode`. This process ensures no data loss. It's important to note that executing `CANCEL_MIGRATION` may lead to extended downtime due to the movement of messages and their states from TxEventQ back to AQ.

   In summary, this final migration stage offers flexibility, allowing users to choose between commitment and restoration based on their specific needs and circumstances.

Overview of how Migration Functions

The DBMS_AQMIGTOOL package helps with the seamless migration from AQ to TxEventQ.

- To start the migration process, `DBMS_AQMIGTOOL.INIT_MIGRATION` sets up an interim TxEventQ copying the AQ's configuration, including payload type, rules, subscribers, privileges, PLSQL notifications, and more. It restricts AQ from administrative changes to maintain TxEventQ configuration integrity until the migration is completed or canceled.
- During migration, the new workload gradually transitions to interim TxEventQ. New enqueue requests go to interim TxEventQ; dequeue requests check AQ first, then interim TxEventQ, progressively emptying AQ. The message ordering is maintained as per `INIT_MIGRATION`'s `ordering` configuration.
- To complete migration, `DBMS_AQMIGTOOL.COMMIT_MIGRATION` drops AQ and renames interim TxEventQ to AQ's name, ensuring application compatibility.
- To cancel migration, `DBMS_AQMIGTOOL.CANCEL_MIGRATION` in `RESTORE` mode moves messages and their states from interim TxEventQ to AQ to prevent data loss, then drops interim TxEventQ.
- For capturing unsupported features, `DBMS_AQMIGTOOL.CHECK_MIGRATION_TO_TXEVENTQ` examines AQ's metadata and messages to capture unsupported features. The

`DBMS_AQMIGTOOL.ENABLE_EVALUATION` option captures runtime-specific unsupported features.

Limitations and Workarounds

Before starting the migration process, it is recommended that the user review the list of the following unsupported features.

Table 8-1 Unsupported Features and Workarounds

Name	Feature Type	Description	Workaround
retry delay See Also: MESSAGE_PROPERTIES_T.	Queue-level	Specifies the delay in seconds before a message is available for dequeue after an application rollback	Set `retry_delay` to zero using `DBMS_AQADM.ALTER_QUEUE`.
transformation	Message-level	Allows message transformation before enqueueing or after the subscriber dequeue the message	Move the transformation to the application layer.
listen See Also: Listen Procedures.	others	Listens on one or more queues for a list of agents/subscribers	Implement single queue listening with dequeue browse.
invalid priority value (valid range - `0` to `9`)	Message-level	TxEventQ only allows priority values between `0` to `9`. The default priority value for AQ is `1`, and for TxEventQ is `4`. Also, AQ interprets the lower value as the higher priority, But TxEventQ interprets the opposite, that is, the higher value is interpreted as the higher priority.	Ensure the application adheres to a valid range (`0` - `9`). Also, keep in mind the difference in interpretation of the priority value of AQ and TxEventQ.

If propagation is already scheduled on AQ before migration, it will result in an incompatibility error. To address this, users can unschedule the propagation using `DBMS_AQADM.UNSCHEDULE_PROPAGATION`. After migration, the propagation can be scheduled on TxEventQ using `DBMS_AQADM.SCHEDULE_PROPAGATION`.

Table 8-2 Unsupported Features without Workarounds

Name	Feature Type
message grouping (transactional grouping) See Also: • CREATE_QUEUE_TABLE Procedure • MESSAGE_PROPERTIES_T Type	Queue-level
sequence deviation and relative msgid See Also: ENQUEUE_OPTIONS_T Type	Message-level
recipient list See Also: MESSAGE_PROPERTIES_T Type	Message-level

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_AQMIGTOOL`

Monitoring TxEventQ with Prometheus/Grafana

Some of the advantages of having a real-time monitoring framework for a high throughput messaging system are as follows.

- Know overall messaging system health at a glance and be able to adjust resources up or down with how heavy or light the messaging work load is.
- Monitor high level key performance indicators: enqueue rates, dequeue rates, queue depth, etc.
- Find the messaging bottlenecks due to the database load or the system load, by monitoring CPU load, memory utilization, and the database wait class from messaging activity.
- Check the health condition of each queue to quickly identify under-performing ones easily.
- Access messaging metrics from anywhere, enabling developers to monitor any overheads from applications and debug message related issues.
- Respond quickly by setting alerts when something goes wrong with the feature in Grafana.

See Also:

Monitoring Transactional Event Queues for more information.

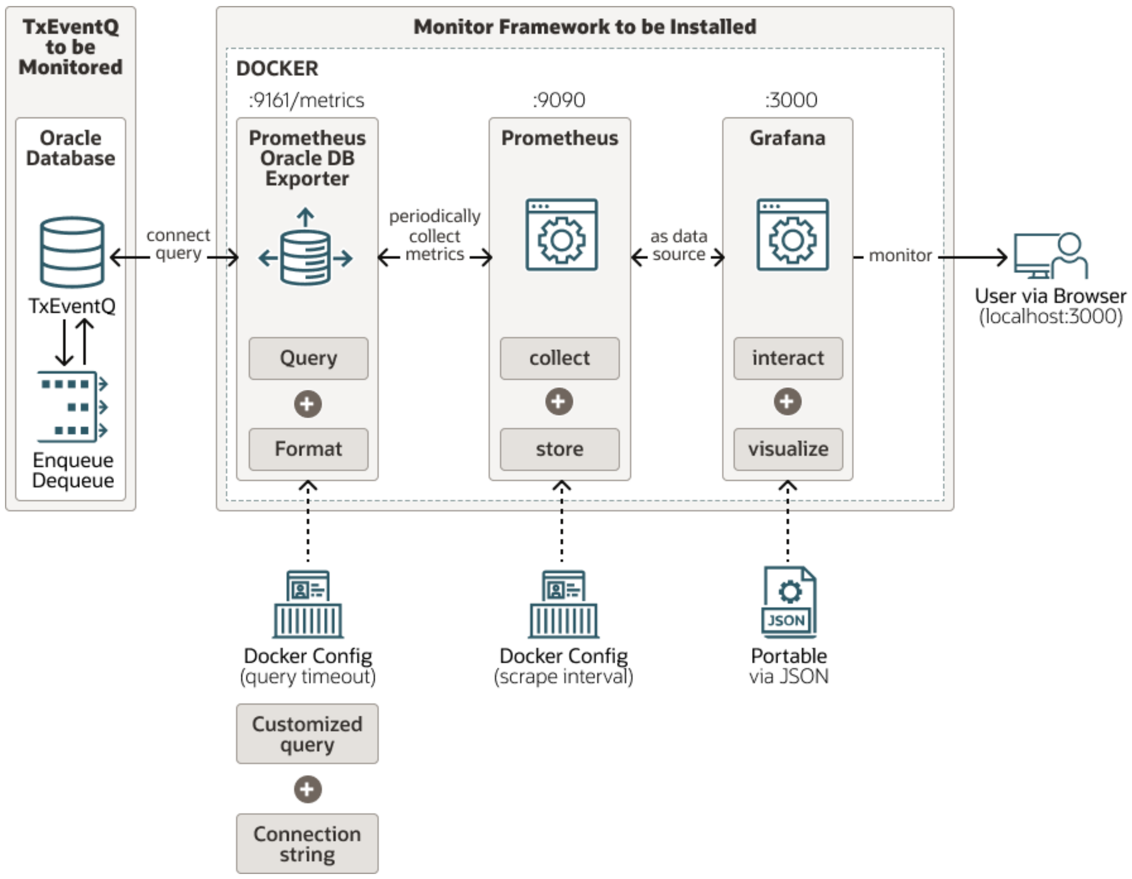
Monitoring Data Flow and UI Framework Setup

The TxEventQ monitor system consists of three independent open-source components. A Docker container is used to help manage all environments, services, and dependencies on the machine where the monitoring framework is installed.

- Oracle DB Exporter: A Prometheus exporter for Oracle Database, which connects to the database, queries metrics, and formats metrics into Prometheus-like metrics.
- Prometheus: A monitor system and time-series database, which manages metrics collecting from Oracle DB Exporter in time-series-fashion.
- Grafana: An analytics and interactive visualization platform, which specifies Prometheus as data source.

TxEventQ Monitor System consists of three services including Prometheus Oracle DB Exporter, Prometheus, and Grafana. The system is designed to run with Docker, which lets user use the system as a lightweight, portable, self-sufficient container, which can run virtually anywhere. Exporter is the connector to Oracle DB and formats the query results to Prometheus-like metrics. Prometheus is a time-series database and periodically controls Exporter to query and collect/store metrics. Grafana uses Prometheus as a data source to show the metrics and visually. Grafana is a user-interface with charting and computation built-in. The whole services is configured, managed and handled by Docker-compose.

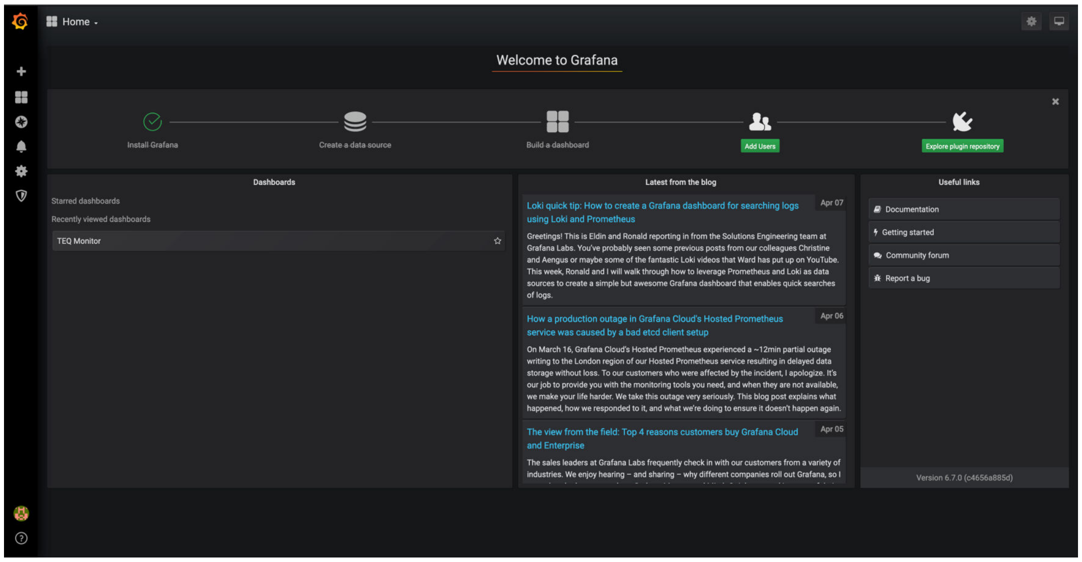
Figure 8-2 Monitoring Transaction Event Queue



To monitor the TxEventQ dashboards using Grafana, perform the following steps.

1. Login to the Grafana dashboard using admin user name and password. The **Welcome Page** is displayed.

Figure 8-3 Welcome Page



Home

Welcome to Grafana

Dashboards

Starred dashboards

Recently viewed dashboards

TEQ Monitor

Latest from the blog

Loki quick tip: How to create a Grafana dashboard for searching logs using Loki and Prometheus — Apr 07

Greetings! This is Eldin and Ronald reporting in from the Solutions Engineering team at Grafana Labs. You've probably seen some previous posts from our colleagues Christine and Aengus or maybe some of the fantastic Loki videos that Ward has put up on YouTube. This week, Ronald and I will walk through how to leverage Prometheus and Loki as data sources to create a simple but awesome Grafana dashboard that enables quick searches of logs.

How a production outage in Grafana Cloud's Hosted Prometheus service was caused by a bad etcd client setup — Apr 06

On March 16, Grafana Cloud's Hosted Prometheus experienced a ~12min partial outage writing to the London region of our Hosted Prometheus service resulting in delayed data storage without loss. To our customers who were affected by the incident, I apologize. It's our job to provide you with the monitoring tools you need, and when they are not available, we make your life harder. We take this outage very seriously. This blog post explains what happened, how we responded to it, and what we're doing to ensure it doesn't happen again.

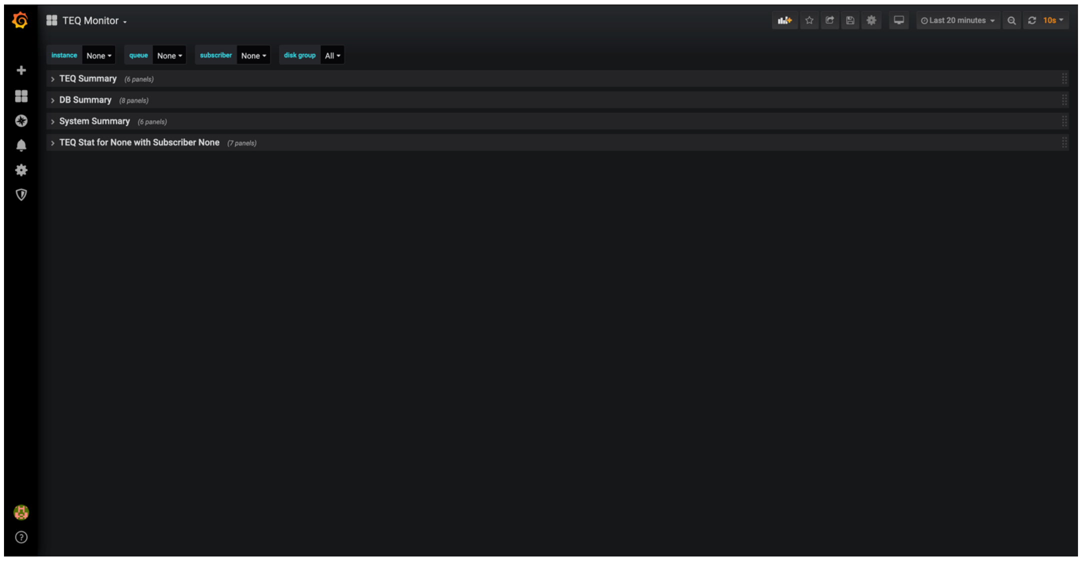
The view from the field: Top 4 reasons customers buy Grafana Cloud and Enterprise — Apr 05

The sales leaders at Grafana Labs frequently check in with our customers from a variety of industries. We enjoy hearing – and sharing – why different companies roll out Grafana, so I

Useful links

Version 6.7.0 (c4656a885d)

2. Click **TxEventQ Monitor** on the Welcome Page. Once Grafana is setup, the metrics are presented in four selections, and the top level selections are for an instance, queue, subscriber and disk group. The four selections are as follows:
   - Summary across all TxEventQs
   - Database metrics summary
   - System metrics summary
   - Subscriber summary for each TxEventQ



TEQ Monitor

instance None · queue None · subscriber None · disk group All

› TEQ Summary (6 panels)

› DB Summary (8 panels)

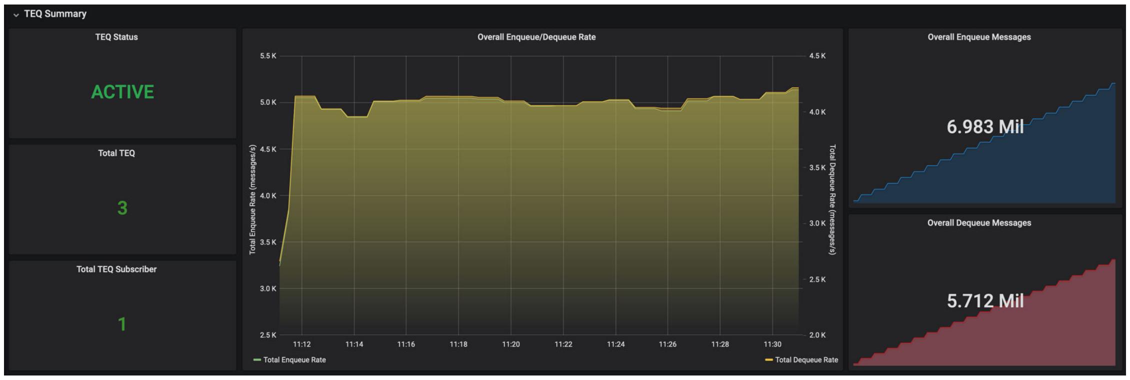
› System Summary (6 panels)

› TEQ Stat for None with Subscriber None (7 panels)

3. Click on each summary to view information about the summary.

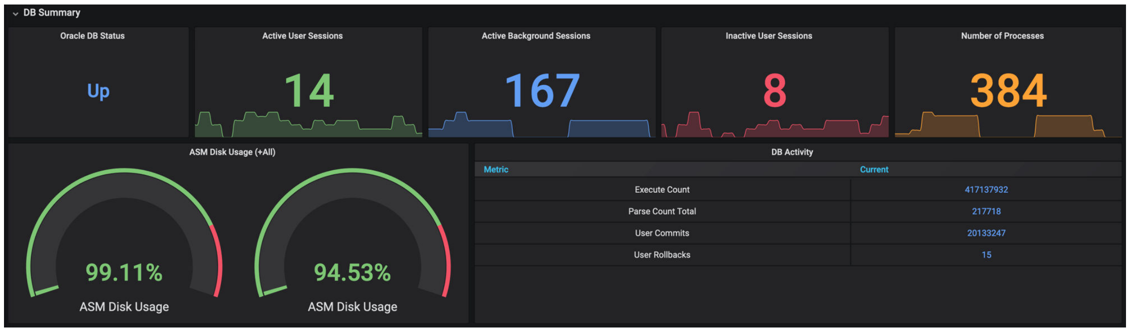
The following figures shows the dashboards of TxEventQ Summary, DB Summary, Database Wait Class Latency, and System Summary respectively.

The TxEventQ Summary dashboard shows overall aggregated TxEventQ stats including status, number of queues, number of subscribers, enqueue/dequeue rate and number of messages



The Database Summary dashboard shows overall DB performance and stats.

Figure 8-4   Database Summary



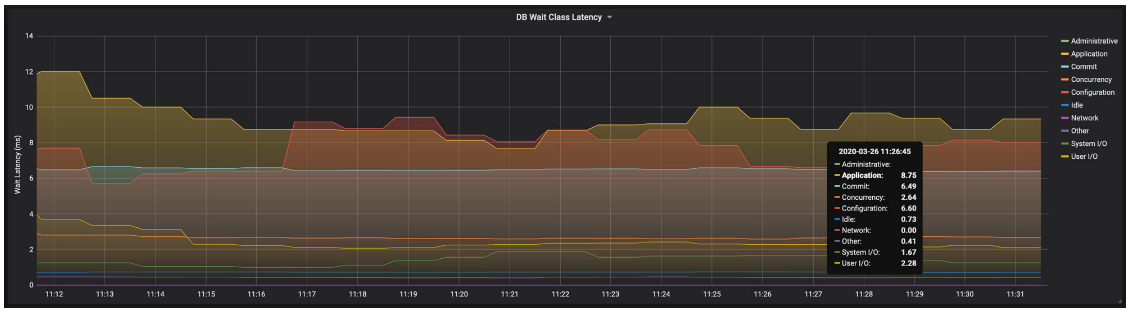
Metric	Current
Execute Count	417137932
Parse Count Total	217718
User Commits	20133247
User Rollbacks	15

The screen tiles are as follows.

- Oracle DB Status – Up or Down
- Active User Sessions – number of user sessions active
- Active Background sessions – number of background sessions active
- Inactive user sessions – Number of inactive user sessions
- Number of processes – Number of database processes
- ASM Disk Usage – Percent of disk free for each disk volume
- DB Activity – SQL activity for the number of execute counts, parse count total, user commits, user rollbacks.

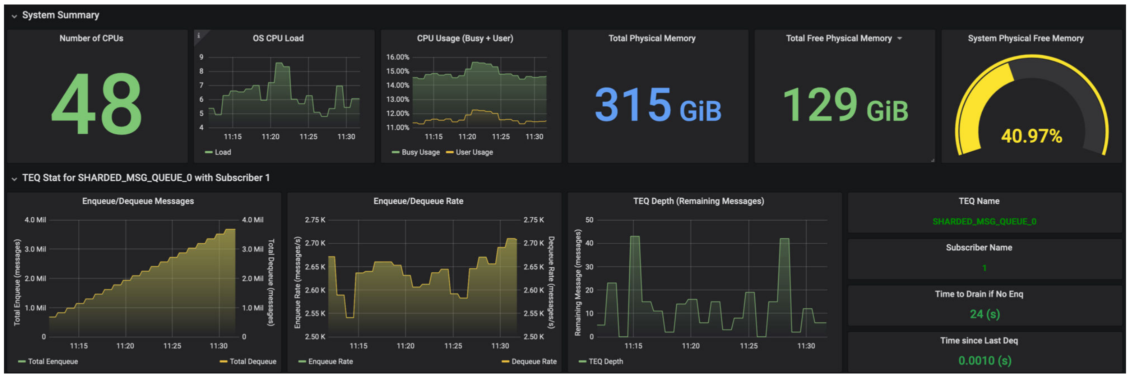
The database wait class latencies are shown in the DB Wait Class Latency dashboard. Wait class latency is the wait class events latency in milliseconds in the database and can be used to guide overhead analysis through a more detailed AWR report analysis.

Figure 8-5 Database Wait Class Latency



The System Summary dashboard shows system level metrics and also the queue level metrics. It reflects the overall performance and status of the system running Oracle DB based on CPU utilization and memory used.

Figure 8-6 System Summary



System Level Statistics

- Number of CPUs – Total number of CPUs on the system
- OS CPU Load - The percentage of CPU capability currently used by all System and User processes
- CPU Usage: % of CPU busy (for all processes) and % of CPU busy for user processes
- Total Physical Memory: Total memory on the system, one instance in case of an Oracle RAC database
- Total Free Physical Memory: Total amount of free memory on the instance
- System Physical Memory free: % of free physical memory

TxEventQ Queue Level Stats

It displays the statistics of one specific queue, which the user can select from the drop-down menu including rate, total messages, queue depth, estimated time to consume and time since last dequeue.

- Enqueue/Dequeue Messages: Number of messages enqueued; number of messages dequeued

- Enqueue/Dequeue rate: Number of messages per second that are enqueued and dequeued
- TxEventQ Depth – Remaining messages in the queue
- TxEventQ Name - Name of the queue
- Subscriber Name – Name of the subscriber
- Time to drain if no enq – Estimate of time to drain the queue if there are no new enqueues
- Time since last dequeue – Time elapsed since the last dequeue on the queue

Key Metrics Measured

This section provides a little more detail on the metrics seen in the previous section and how to get these from the Grafana screen. The drop-down menu options are at the level of a: database instance, queue, and a subscriber. AQ/TxEventQ Summary metrics and Database metrics are for the database instance the user selects in the drop-down menu.

- AQ/TxEventQ Summary Metrics
  - TxEventQ Status: if TxEventQs are running or not
  - Total Number of TxEventQs: the number of TxEventQs running
  - Total TxEventQ Subscribers: the total number of subscribers for all TxEventQs
  - Overall Enq/Deq Rates: aggregate enq/deq rates for all TxEventQs
  - Overall Enqueued Messages: total enqueued messages for the entire queue system
  - Overall Dequeued Messages: total dequeued messages for the entire queue system
- Database Summary Metrics
  - Oracle DB Status: if Oracle DB is running or not.
  - Active User Sessions: the number of active user sessions
  - Active Background Sessions: the number of active background sessions
  - Inactive User Sessions: the number of inactive user sessions
  - Number of Processes: the number of Oracle processes running
  - ASM Disk Usage: Oracle Automatic Storage Management disk group memory usage (e.g. +DATA, +RECO)
  - DB Activity: the number of specific DB operations that occurred including execute count, user commits, parse count total, user rollbacks.
  - DB Wait Class Latency: average latency for DB wait class in ms including administrative, application, commit, concurrency, configuration, idle, network, other, system I/O, user I/O
- System Summary Metrics
  - Number of CPUs: the number of CPU of the system running Oracle DB
  - OS CPU Load: current number of processes that are either running or in the ready state, waiting to be selected by the operating-system scheduler to run. On many platforms, this statistic reflects the average load over the past minute
  - CPU Usage (Busy + User): the CPU usage in percentage in real time including CPU in busy state or CPU in executing user code.

  - Total Physical Memory: total physical memory of the system.
  - Total Free Physical Memory: total free physical memory of the system.
  - System Free Physical Memory: the percentage of free memory in the system.
- Queue Level Metrics
  - Enq/Deq Messages: total messages enqueued/dequeued to/from the TxEventQ
  - Enq/Deq Rate: enqueue/dequeue rate for the TxEventQ
  - TxEventQ Depth: total messages remaining in the queue.
  - TxEventQ Name: the name of TxEventQ
  - Subscriber Name: the name of TxEventQ subscriber
  - Time to Drain if No Enq: total amount of time to consume all messages if there are no more messages enqueued on the TxEventQ
  - Time since Last Deq: time difference between current time and the time since the last dequeue operation on the TxEventQ

See Also:

Monitoring Transactional Event Queues for more information.