Experimentation of message passing system in Oracle, for the CASTOR Project

Student :
Archit Gupta

Supervisor
Eric Cano

CERN Openlab
15 August, 2012

oTN-2012-01 openlab Summer Student Report

Experimentation of message passing systems in Oracle, for the CASTOR project

Archit Gupta
Eric Cano
15 August, 2012
Version 1
Distribution: **Public**

CONTENT

ABSTRACT

The project's objective was the development of logging procedures inside the PL/SQL context in the database, similar to the ones in the DLF. The log messages will be posted in message queues using Oracle's Advanced Queues and object representation. A simple C++ bridge daemon would then forward the log messages inside the DLF system, allowing a full logging of the events in castor's flow control. This functionality has already been implemented with other means before the start of this project, which has been scaled down to study implementation of Oracle Advanced queues, for potential use in Castor implementation in the future.

INTRODUCTION

The CERN Advanced Storage Manager (CASTOR, http://cern.ch/castor) is a scalable, hierarchical storage system used for managing LHC data on disk and tape. CASTOR currently holds over 58 PB of data in more than 300 million files. Data is persistently stored in 7 enterprise tape libraries currently holding 45,000 tapes and over 100 leading-edge tape drives. The tape data storage is growing by up to 20-30 Petabytes / year.

The core of the CASTOR data flow management is built around an Oracle database, using relational data on flat tables (non-object) and stored procedures. As the system matures, more logic can be consolidated inside the database as stored procedures. Currently, all the castor system logging is generated outside of the database, from C++ daemons, and sent into the Distributed Logging Facility (DLF). This reduces the scale of the logic we can push in the database, as the developers and operators are blind of what happens there without logs.

Advanced queues implements the message queuing functionality natively inside the database and leverages its easy manageability, high performance and security. *AQ supports point-to-point and publish/ subscribe queues, persistent and buffered messaging, and message ordering priorities that offer flexibility and powerful messaging functionality to applications*. [1]page4 We intend to check the functionality of PL/SQL procedural call to Oracle Streams Advanced Queues and try to measure the latency and test its functionality for the CASTOR.

MESSAGE QUEUING

Message queuing infrastructure enables information sharing and integration amongst different, possibly distributed, applications[1]. Producer applications send or enqueue messages into queue from which consumer applications receive or dequeue messages. *Producers and Consumers interact with the queue asynchronously and this 'decoupling' is the centrepiece of message queuing. A message stays in the queue until a consumer dequeues it or the message expires. A producer may stipulate a delay before the message is available to be consumed and a time after which the message expires. Likewise a consumer waits when trying to dequeue a message if no message is available. An agent program or application may act as both a producer and a consumer.*[1]

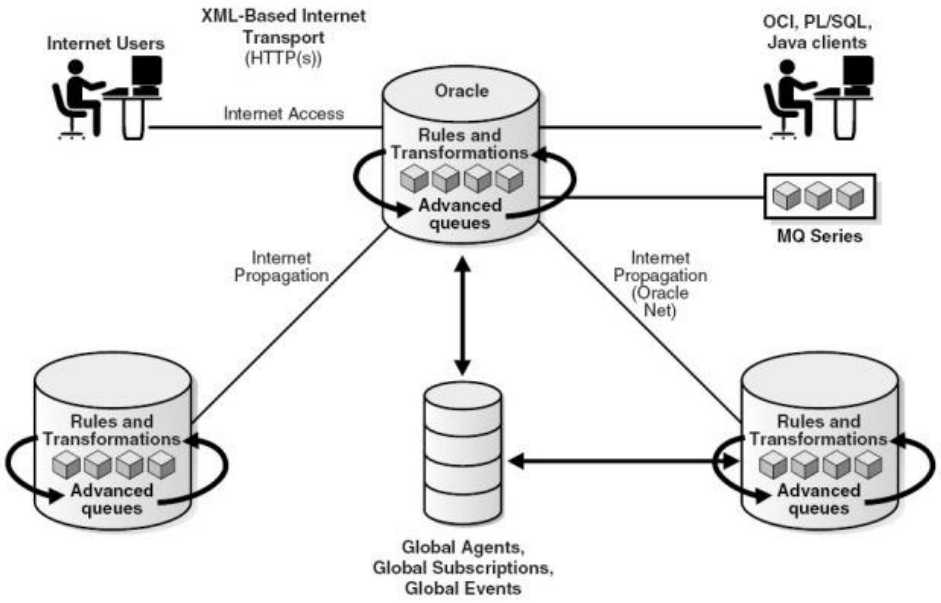
The propagation of messages is like the producers enqueue the messages into the message queue and the subscribed consumers dequeues them, both the enqueue and dequeue processes can go on side by side at the same time they can also be one at the time, i.e., when the producers stops finishing the enqueuing the messages he sends a message to all the subscribed consumers stating that they can now dequeue the messages, this type of operation is usually done when the application needs to be processed one at a time. Though both enqueue and dequeue operations can go on simultaneously. Messages which are there in the queue for more than the time allotted to them for the dequeuing operation are automatically expired and they are no longer available for the dequeue operation.

ORACLE STREAMS ADVANCED QUEUING

Oracle Streams Advanced Queuing provides database-integrated message queuing functionality. It is built on top of oracle streams and leverages the functions of Oracle Database so that messages can be stored persistently, propagated between queues on different computers and databases, and transmitted using Oracle Net Services and HTTP(S).

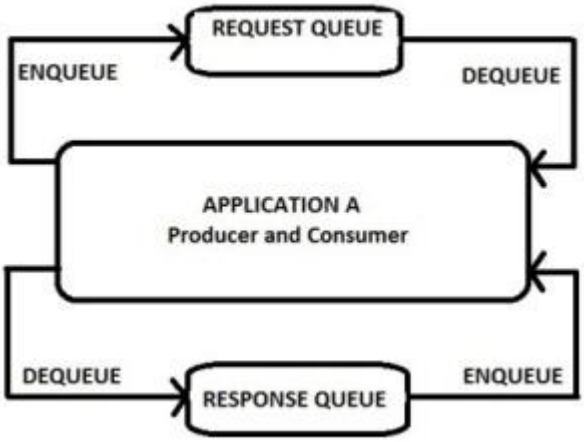
Because Oracle Streams Advanced Queuing is implemented in database tables, all operational benefits of high availability, scalability, and reliability are also applicable to queue data. Messages are queued using standard SQL. This means that you can use the SQL to access the messages properties, the message history and the payload. It can be possible that there are more than one producer and/or more than one consumer, if this is the case than according to the priority set in the dequeue table the consumers are able to dequeue the message though producers can enqueue at any time.

Enqueued messages are said to be propagated when they are reproduced on another queue, which can be in the same database or in a remote database. [2] Page 1-2



Source of Image: Oracle® Streams, Advanced Queuing User's Guide

An application A enqueues a request into the request queue. In a different transaction, the application A dequeues and processes the request. Application A enqueue the result in the response queue, and in yet another transaction, Application A dequeues it.



A message producer can submit a list of recipients at the time a message is enqueued. This allows for a unique set of recipients for each message in the queue. The recipient list associates with the message overrides the subscriber list associated with the queue, if there is one. The recipients need not be in the subscriber list. However, recipients can be selected from among the subscribers.

AQ COMPONENTS

The four main components of AQ are:[1]page 4

1. ***Message*** *– A message consists of message content, or payload, which can be specified using typed or raw data and message attributes or control information.*

2. ***Message Queue*** *– Messages are stored in queues and these queues act as "postal boxes" where different applications can look for "mail" in the form of the messages. Thus when one application wants to contact certain applications for certain tasks, it can leave messages in these queues, and the receiving applications will be able to find these massages for processing. AQ supports enqueue, dequeue and propagation operations where the queue type is an abstract datatype. A queue is persisted in the database using one or more database tables where messages in a queue correspond to rows in the underlying table.*

3. ***Message Interface*** *– AQ can seamlessly with the existing applications through support for popular standards. AQ messages can be created, queried, propagated and consumed using popular programming interfaces (API) such as PL/SQL, C/C++, Java and Visual Basic. AQ provides support for the Java Message Service(JMS) API that allows Java applications to utilize the message queuing functionality*

4. ***Message handling*** *– Messages can be routed according to the data in the message payload or attributes. AQ also support rules based message routing where complex rules can be created by combining payload based and attributes-based rules. Additionally message-transformation can be applied to messages to reformat data and delivered automatically to target applications or subscribers*

ORACLE STREAM AQ USING PL/SQL PROCEDURES USING A DATABASE TABLE FOR ENQUEUING AND DEQUEUING MESSAGES

Oracle Streams is an information sharing feature that provides replication, message queuing, data warehouse loading, and event notification. The PL/SQL packages DBMS_AQADM and DBMS_AQ support access to Oracle Stream Advanced Queuing administrative and operational functions using the native Oracle Stream Advanced Queuing interface. The basic steps to create a PL/SQL procedural call to Advanced Queues are as follows:

1. **Creating a queue table** - A database table where queues are stored. Each queue contains default exception queue.

```
DBMS_AQADM.CREATE_QUEUE_TABLE
        ( queue_table                => 'objmsgs80_qtab',
          queue_payload_type         => 'message_typ',
          multiple_consumers         => TRUE );
```

2. **Create a queue -** A queue is the abstract storage unit used by a messaging system to store messages.

```
DBMS_AQADM.CREATE_QUEUE
        (queue_name                  => 'MSG_QUEUE',
         queue_table                 => 'objmsgs80_qtab');
```

3. **Start a queue –** a queue needed to be started in order to receive the messages send by the sender.

```
DBMS_AQADM.START_QUEUE
        (queue_name                  => 'MSG_QUEUE');
```

4. **Adding Subscriber –** a subscriber is an agent authorized by a queue administrator to retrieve messages from a queue. An application can enqueue messages to a specific list of recipients or to the default list of subscribers, and the total no of subscriber must be 1024 or less.

```
DBMS_AQADM.ADD_SUBSCRIBER
        ( queue_name                 => 'msg_queue',
          subscriber                 => sys.aq$_agent( 'recipient', null, null ) );
```

5. **Register to receive messages asynchronously**

```
DBMS_AQ.REGISTER
        ( sys.aq$_reg_info_list(
          sys.aq$_reg_info('MSG_QUEUE:RECIPIENT',
                           DBMS_AQ.NAMESPACE_AQ,
                           'plsql://dequeue_msg',
                           HEXTORAW('FF')) ) ,
         1 );
```

6. **Enqueue a message –** The messages that need to be send are enqueued in the queue .

```
//creating a procedure for enqueue
CREATE OR REPLACE PROCEDURE "ARGUPTA"."ENQUEUE_MSG" ( p_msg in varchar2)
as
        message message_typ;
        enqueue_options dbms_aq.enqueue_options_t;
        message_properties dbms_aq.message_properties_t;
        message_handle RAW(16);
BEGIN
        message := message_typ(0, 'NORMAL MESSAGE', p_msg);
        dbms_aq.enqueue(queue_name        => 'msg_queue',
                payload                   => message,
                enqueue_options           => enqueue_options,
                message_properties        => message_properties,
                msgid                     => message_handle);
END;

//enqueueing a message

        enqueue_msg ('this is '|| a ||' enqueued message,'|| ' enqueued at ' ||
                        to_char(current_timestamp, 'hh24:mi:ss.ff'));
```

7. **Dequeuing a message from the queue -** The message is dequeued from the specified queue to be used by the application for further processing.

```
CREATE OR REPLACE PROCEDURE "ARGUPTA"."DEQUEUE_MSG" (  context   raw,
                reginfo      sys.aq$_reg_info,
                descr        sys.aq$_descriptor,
                payload      raw,
                payload      number)
AS
                dequeue_options      dbms_aq.dequeue_options_t;
                message_properties   dbms_aq.message_properties_t;
                message_handle       RAW(16);
                message              message_typ;

BEGIN
               dequeue_options.msgid              := descr.msg_id;
               dequeue_options.consumer_name      := descr.consumer_name;
DBMS_AQ.DEQUEUE(queue_name                        => descr.queue_name,
                        dequeue_options           => dequeue_options,
                        message_properties        => message_properties,
```

```
                        payload                 => message,
                        msgid                   => message_handle);
    FOR a IN 1..10
    LOOP
   //dequeuing a messge from the message_table
              insert into message_table (time, msg) values
                   ( current_timestamp, 'this message is dequeued!' );

    END LOOP;
    COMMIT;
END DEQUEUE_MSG;
```

8. **Removing a subscriber –** all references to the subscriber in existing messages are removed as part of the operation. It is not an error to remove the subscriber even when the messages are available for dequeue by the consumer. These messages are automatically made unavailable for dequeue after the remove subscriber operation.

9. **Stopping a queue –** This operation disables enqueueing, dequeuing, or both on a specified queue.

10. **Dropping a queue –** This operation drops an existing queue. All the queue data is deleted as part of the drop operation.

11. **Dropping a queue table –** This operation drops an existing queue table. We must stop and drop all the queues in a queue table before the queue table can be dropped.

```
 CREATE OR REPLACE PROCEDURE "ARGUPTA"."REMOVE" (recipient in sys.aq$_agent)
 AS
         queue_name      varchar2(15 char);
         queue_table     varchar2(15 char);
         subscriber      sys.aq$_agent;

BEGIN

         DBMS_AQADM.REMOVE_SUBSCRIBER(
                  queue_name      => 'msg_queue',
                   subscriber     => recipient);

         DBMS_AQADM.STOP_QUEUE
                  (queue_name     => 'MSG_QUEUE');

         DBMS_AQADM.DROP_QUEUE(
                  queue_name      => 'msg_queue');

         DBMS_AQADM.DROP_QUEUE_TABLE(
                  queue_table     =>'objmsgs80_qtab');
END;

/
```

MEASURMENT OF LATENCY

For the purpose of measuring the latency of the program we have enqueued 100 messages into the message queue and following results were obtained.

- Start time: 15:14:09.254172000
- Finish time: 15:14:09.296493000
- Hence the total time taken for propagation of 100 messages: 0.04232100 seconds
- Minimum time taken : 0.000251 seconds
- Maximum time taken: 0.000638 seconds
- And the average time for propagation of a single message: 0.00042321 seconds

CONCLUSION & FUTURE WORK

From the project I have been able to enqueue and dequeue a message into a message queue using PL/SQL procedural call into Oracle Streams Advanced Queues. The average time for propagation of a single message was found to be 0.00042321 seconds

The castor data flow management is very close to an OLTP system, where requests generate sub-requests for their completion in response to external events. Those master-slave requests relationships could be consolidated into objects using class inheritance and message passing as a mean of communication inside the database to signal the events. I would also work towards enqueuing and dequeuing the messages using Oracle C++ Call Interface (OCCI).

REFERENCES

1. Oracle Database 11g: Advanced Queuing
   An Oracle white Paper, November 2010

2. Oracle® Streams
   Advanced Queuing User's Guide, 11*g* Release 2 (11.2):**E11013-04**

3. Oracle® C++ Call Interface
   Programmer's Guide, 11*g* Release 2 (11.2): **E10764-02**