Kuali Rice Performance

Overview

This page is intended to be a space for collaboration and sharing of solutions and workarounds for various Kuali Rice performance issues. This page should not serve as a substitute for jira, but attempts to provide a single point where people can learn about potential performance gotchas and how to avoid and/or address them.

For real-time discussion on Kuali Rice performance, you can visit the #kuali channel on irc.freenode.net

Workflow Engine Processing

`SELECT ... FOR UPDATE WAIT 3600` against `KREW_DOC_HDR_T`

Description

Use of `SELECT ... FOR UPDATE` against `KREW_DOC_HDR_T` is problematic, especially for transactions that lock more than one document. At least when using an Oracle database, deadlocks in this scenario appear to not be recoverable until the timeout has elapsed at least (which by default is 3600 seconds which is an hour). The reason that Rice performs this row-level locking is because workflow documents are oftentimes being processed or updated in a concurrent fashion (by more than one user or by more than one background processing thread). If no locking is performed, frequent OptimisticLockExceptions are thrown from OJB when the document is updated. The "serialization" of processing at the database level "solves" this but is very taxing on the database and leads to problematic deadlock situations.

Possible Solutions and Workarounds

- One possible workaround to alleviate some of the problems is to lower the timeout, this can be done by adding the following to rice-config.xml:
    - ```
      <param name="document.lock.timeout">900</param>
      ```
    - The example above changes the timeout to 900 seconds
- Another possible workaround is to disable the locking altogether and just live with the optimistic lock exceptions that are generated.
    - In order to disable this completely, you can make the following modification (note: this is untested!:

      org.kuali.rice.kew.routeheader.service.impl.RouteHeaderServiceImpl

      ```
      public void lockRouteHeader(String routeHeaderId, boolean wait) {
          // getRouteHeaderDAO().lockRouteHeader(routeHeaderId, wait);
          // LOG.debug("Successfully locked document [docId=" + routeHeaderId +
      "]");
      }
      ```
    - As can be seen in the above example, this effectively disables the code that performs the document locking
- To eliminate OptimisticLockingExceptions, it may be possible to remove the locking defined in `OJB-repository-kew.xml` from the `VER_NBR` column on `KREW_DOC_HDR_T`. Typically the only data that actually gets updated on that table are the various timestamps, document status, and (rarely) the title.
    - Note however that this is untested and could potentially result in data corruption to records in the KREW_DOC_HDR_T table.
- Ultimately, the Rice team should look at a solution for workflow document processing that does not require a pessimistic lock to be placed on the `KREW_DOC_HDR_T` row.

Notes/Comments

- Indiana University
    - So far this hasn't been a debilitating problem for us, but I believe that it has contributed to some of our production issues we've encountered in the past.
    - We have previously reduced our lock timeout to 900 seconds which seemed to help. However we still occasionally get reports of row-lock contention on our database from this statement.
    - The locks usually clear up eventually but sometimes they takes awhile.
    - When we have a lot of contention on the database at a given time, this tends to drive CPU up on the database which can eventually cause the database to crash if it has too much load against it.
- Cornell University
    - Attempted to disable the locking, but reintroduced the locking due to end user impact with OptimisticLockExceptions.

Direct access to Kuali Rice database from client application

Description

Direct database access to certain tables is part of the way in which embedded KEW is implemented. As additional client applications come online, the number of connections being established to the database grow which requires planning and resource allocation on the database side.
      ```

Possible Solutions and Workarounds

- TODO

Notes/Comments

- TODO

Action List

Caching Behavior and deadlocks in KREW_USR_OPTN_T

Description

Action List caching behavior is controlled by inserting and deleting records from the KREW_USR_OPTN_T table. This can cause a lot of contention and occasional deadlocks against this table. Additionally, overtime this table can grow very large with lots of entries in there with the option id starting with "REFRESH_ACTION_LIST".

Possible Solutions and Workarounds

- For Rice versions in the range 1.0.2 through 1.0.3.1, applying the changesets for KULRICE-5011 should:
  - fix the deadlocking issue and
  - prevent the table from growing large due to "REFRESH_ACTION_LIST" entries:
- Here are those changesets:
  - http://fisheye.kuali.org/changelog/rice?cs=16234
  - http://fisheye.kuali.org/changelog/rice?cs=16235
- To see if you might be affected by the growing `KREW_USR_OPTN_T` table, run the following query:
  - `select PRNCPL_ID, count(*) cnt from KREW_USR_OPTN_T where PRSN_OPTN_ID like 'RELOAD_ACTION_LIST%' group by PRNCPL_ID order by cnt desc`
  - If the above query returns a large "cnt" for certain users (like in the thousands or 10's of thousands) you are likely being affected by this issue, see next bullet for possible workaround
- A solution that some institutions have employed to deal with the growing of this table is to run a nightly job which essentially does the following:
  - `DELETE FROM KREW_USR_OPTN_T WHERE PRSN_OPTN_ID LIKE 'REFRESH_ACTION_LIST%'`

Notes/Comments

- It was an attempted fix to the large table size issue in Rice 1.0.2 that introduced the database deadlocking bug. See KULRICE-4037 and http://fisheye.kuali.org/changelog/rice?cs=12365 for that change.

Large Action Lists

Description

Very large actions lists are not handled well in many cases, resulting in large amounts of memory usage. This is in particular a problem when modifying a group in KIM by either adding or removing somewhere which will trigger a large number of action items and action requests to be loaded into memory for processing. Large action lists can happen when people are the recipient of a large number of FYI requests and don't clear them frequently enough. One of the main issues here is the use of the ORM to load these for processing since it will cache them in memory for the life of the transaction, even once the object loaded by the ORM goes out of local scope in the code which is processing the data.

An example of problematic code in Rice which loads the entire action list into memory can be found below, it is code from the `WorkgroupMembershipChangeProcessor` class:

```
    private void updateActionListForUserRemovedFromGroup(String principalId, String
groupId) {
  List<String> allGroupsToCheck =
KIMServiceLocator.getIdentityManagementService().getParentGroupIds(groupId);
        allGroupsToCheck.add(0, groupId);
        Collection<ActionItem> actionItems =
getActionListService().findByPrincipalId(principalId);
  for (Iterator<ActionItem> itemIt = actionItems.iterator(); itemIt.hasNext();) {
         ActionItem item = itemIt.next();
         if (item.isWorkgroupItem()) {
          for (String groupIdToCheck : allGroupsToCheck) {
           if (item.getGroupId().equals(groupIdToCheck)) {
            getActionListService().deleteActionItem(item);
           }
          }
         }
        }
    }
```

Possible Solutions and Workarounds

- The Rice team should probably find and rewrite this type of code which loads an entire action list so that it either clears the persistence broker cache between iterations or rewrite it use direct JDBC (the later would probably be significantly more performant).
- Failing that, another temporary workaround is to ensure that all user's action lists remain relatively small through manual intervention.
    - To find the size of all action lists with the largest returned first, the following query can be executed: `select COUNT(*) c, PRNCPL_ID from KREW_ACTN_ITM_T group by PRNCPL_ID order by c DESC`

Notes/Comments

- TODO...

Remote `CustomActionListAttribute` callbacks from Kuali Rice Standalone Server

Description

TODO...

Possible Solutions and Workarounds

- TODO...

Notes/Comments

- TODO...

Finding primary delegates when loading action list

Description

Every time a user loads their action list a list of the primary delegates for the user is created. To do this it is essentially running this query:

```
select * from KREW_ACTN_ITM_T where DLGN_TYP = 'P' and (DLGN_PRNCPL_ID = '<current
user's principal ID>' or DLGN_GRP_ID in (<list of group IDs the user is a member of>))
```

This query can be very slow and causes the action list to load slower every time it is accessed.

Possible Solutions and Workarounds

```
create index KREW_ACTN_ITM_TI# on KREW_ACTN_ITM_T (DLGN_TYP, DLGN_PRNCPL_ID,
DLGN_GRP_ID);
```

Notes/Comments

- Indiana University
  - Before creating this index this query would take well over 10 seconds to run and running an explain plan on the query in Oracle for a sample user reported the cost of the query to be 5117. This could be even worse depending on the number of action items in the table and the number of groups the user belongs to.
  - After creating this index the query runs in 10ms and the explain plan for the same query reports the cost to be 25.

Document Search

Remote `SecurityAttribute` callbacks from Kuali Rice Standalone Server for document search result filtering

Description

Document search does security filtering on each row in the document search result set that is returned. Document search supports the ability to create a custom `SecurityAttribute` which can be invoked remotely. The problem is that this gets invoked for **every single row** returned from the document search which uses this security attribute. This means each individual row results in a remote callback into a client application which is a large amount of network and bus traffic that could really be accomplished more easily with a single call.

Possible Solutions and Workarounds

- TODO...

Notes/Comments

- TODO...

Document Search becomes slow as `KREW_DOC_HDR_T` and `KREW_DOC_HDR_EXT_*_T` grow larger

Description

TODO...

Possible Solutions and Workarounds

- TODO...

Notes/Comments

- TODO...

Re-resolving Workflow Requests

Rule Change Re-Resolution

Description

When a change is made to a routing rule, it will trigger a requeue of any documents with pending action requests that could be affected by that rule change. This can often result in a massive amount of documents being requeued depending on the number of outstanding documents.

Role Membership Change Re-Resolution

Description

When a change is made to the membership of a role, it will trigger a requeue of any documents with pending action requests that could be affected by that rule change. This can often result in a massive amount of documents being requeued and processed depending on the number of outstanding documents.

Kuali Nervous System

`SessionDocumentService` causes poor performance when working with large documents

Description

See "Disabling of the SessionDocumentService" at the bottom of this page: UC Davis Production Environment Details

Kuali Service Bus

Quartz Deadlocks

Description

When an asynchronous message get sent on the bus and fails, it is scheduled for retry in Quartz. When many thousand messages fail around the same time (which can happen when an application is down), this causes a large amount of database contention on KRSB_QRTZ_LOCKS table because of it's use of SELECT ... FOR UPDATE.

Possible Solutions and Workarounds

- This problem currently occurs when using quartz 1.6.x. There are newer versions of quartz available (there is now a 2.x version). It's not clear whether this would actually fix the problem or not as there is nothing in the release notes for quartz indicating that it would, but it would be worth a shot.
- If the above solution does not work, we may want to consider the possibility of crafting a better-optimized solution which does not use quartz or SELECT ... FOR UPDATE

Notes/Comments

- Indiana University
    - In the past this has brought down our production database instance when we get a large number of cache messages failing at once.
    - It is fairly easy for us to reproduce this problem in a test environment by leaving some bad OSCacheNotification service endpoints in the registry and then doing something like ingesting some rules or document types which triggers a flurry of messages to be sent. If they all fail around the same time, this will typically lock up and crash our oracle database.
    - We recently looked at performing a quartz upgrade ourselves locally but 2.0 has impacting changes and requires Spring 3.x in order to allow for Rice to configure it the way it does currently (through Spring).

Too Much Cache Flushing Activity

Description

All Rice client apps that are running embedded KEW or embedded KIM services publish an `OSCacheNotificationService` which essentially receives messages whenever cachable data is updated. This works well in general, but there are a few problematic scenarios:

1. When a document type is updated, it sends out a flush messages for each document type in the hierarchy. It should really only need to send one to each cache endpoint and combine all relevant information into a single message. This can result in hundreds of messages sent to each individual endpoint.
2. Whenever a rule is updated, it sends out a flush message for that rule for each document type in it's hierarchy. As with document types, it should be possible to send all related information in a single message. This can result in hundreds of messages sent to each individual cache endpoint.
3. In many cases, certain clients might not even care about certain pieces of information. For example, the Kuali Financial System application is (probably) never going to load Kuali Coeus rules. However, if a KC routing rule is modified, KFS is still sent a message to let it know it can flush that rule from it's cache.
    - The same is also true for roles in KIM, which is a bit more of a gray area since cross-application integration there is more likely, but the cache flush behavior still results in messages to applications that really don't care about them.

Misbehaving Clients

Constant Updates

Description

Some applications are using rice in ways that cause frequent (and hopefully unnecessary) database writes. This causes any caching solutions to be ineffective since the cache is always dirty.

One example is the way Kuali Coeus creates many KIM Roles on the fly, which results in lots of cache flush messages and effectively eliminates caching of roles in KIM (since the KIM cache flush is very course-grained).

Possible Solutions and Workarounds

- TODO...

Notes/Comments

- TODO...

OJB (Object Relational Bridge)

Persistence Broker Pool settings not allowing enough brokers in the pool to accommodate connection pool settings.

TODO

XAPool and JOTM

XAPool does not recover properly from a connection drop from the database server.

TODO