Deadlock Detection And Cluster Protection

Abstract

This page is devoted to possible deadlocks in Ignite. Along with each deadlock type there also will be suggestions on how to resolve deadlock with minimal impact on running cluster. After all discussions we will have tickets filed to change Ignite and Web Console accordingly.

Possible Deadlocks in Ignite

Deadlocks with Cache Transactions

Description

Deadlocks of this type are possible if user locks 2 or more keys within 2 or more transactions in different orders (this does not apply to OPTIMISTIC SERIALIZABLE transactions as they are capable to detect deadlock and choose winning tx). Currently, Ignite can detect deadlocked transactions but this procedure is started only for transactions that have timeout set explicitly or default timeout in configuration set to value greater than 0.

Detection and Solution

Each NEAR node should periodicely (need new config property?) scan the list of local transactions and initiate the same procedure as we have now for timed out transactions. If deadlock found it should be reported to logs. Log record should contain: near nodes, transaction IDs, cache names, keys (limited to several tens of) involved in deadlock. User should have ability to configure default behavior - REPORT_ONLY, ROLLBACK (any more?) or manually rollback selected transaction through web console or Visor.

Report

If deadlock found it should be reported to logs. Log record should contain: near nodes, transaction IDs, cache names, keys (limited to several tens of) involved in deadlock.

Also there should be a screen in Web Console that will list all ongoing transactions in the cluster including the following info:

1. Near node
2. Start time
3. DHT nodes
4. Pending Locks (by request)

Web Console should provide ability to rollback any transaction via UI.

Hanging Transactions not Related to Deadlock

Description

This situation can occur if user explicitly markups the transaction (esp Pessimistic Repeatable Read) and, for example, calls remote service (which may be unresponsive) after acquiring some locks. All other transactions depending on the same keys will hang.

Detection and Solution

This most likely cannot be resolved automatically other than rollback TX by timeout and release all the locks acquired so far. Also such TXs can be rolled back from Web Console as described above.

If transaction has been rolled back on timeout or via UI then any further action in the transaction, e.g. lock acquisition or commit attempt should throw exception.

Report

Web Console should provide ability to rollback any transaction via UI.

Long running transaction should be reported to logs. Log record should contain: near nodes, transaction IDs, cache names, keys (limited to several tens of), etc (?)..

Also there should be a screen in Web Console that will list all ongoing transactions in the cluster including the info as above.

Java Level Deadlocks

Description

This situation occurs if user or Ignite comes to a Java-level deadlock due to a bug in code - reverse order synchronized(mux1) {synchronized (mux2) {} } sections, reverse order reentrant locks, etc.

Detection and Solution
This most likely cannot be resolved automatically and will require JVM restart.

We can implement periodical threaddumps analysis and detect the deadlock.

**Report**

Deadlock should be reported to the logs.

Web Console should fire an alert on java deadlock detection and display a warning on UI.

**Ignite Thread Pools Starvation**

**Description**

This situation can occur if user submits tasks that recursively submit more tasks and synchronously wait for results. Jobs arrive to worker nodes and are queued forever since there are no free threads in public pool since all threads are waiting for job results.

**Detection and Solution**

Task timeout can be set for tasks, so task gets canceled automatically.

Web Console should provide ability to cancel any task and job from UI.

**Report**

Timed out tasks and jobs should be reported on Web Console and reported to logs. We need to introduce new config property to set timeout for reported jobs.

Log record and Web Console should include:

- Master node ID
- Start time

**GC Pauses**

**Description**

When Ignite node suffers from GC pauses it is literally unresponsive for every other node in topology.

**Detection and Solution**

Very good solution with 2 native threads is described here [IGNITE-6171 - Native facility to control excessive GC pauses](https://issues.apache.org/jira/browse/IGNITE-6171) - RESOLVED

**Report**

Native threads should report GC pause to stdout and if possible to a logger instance. Of course, if policy is set to "kill the node" then output via log is not possible as native thread will stuck in safepoint and no killing and logging occur until safepoint is released.