Proposal for “Gracefully Decommission of NodeManager” (YARN-914)

Scenarios and Use Cases

One of important design goals for YARN is easier and more smoothly for operation and maintenance on a cluster with thousands of nodes. A major challenge for daily hadoop cluster operation is upgrade of software stacks on each node.
Some of software upgrades, e.g. YARN/HDFS components, happens in the hadoop layer and doesn’t involve the reboot of node or JVMs. These upgrades can be well addressed by Rolling Upgrade feature that get supported by YARN since 2.6 release based on feature of RM/NM work preserving. The NM daemon can be shutdown temporarily for upgrade and bring back without losing running containers and states. However, there are also other cases that software upgrades happen below the layer of hadoop and have to involve the reboot of nodes (e.g. OS upgrade) or relaunch of JVMs (e.g. upgrade of JDK or JRE).

For these scenarios, operation flow today is to decommission these nodes first, then do recommission after finish of upgrades which has many outstanding issues:

- Regular running containers get lost and will have to be reassigned with new attempts (waste of time and resources)
- AM container (even closed to be finished) could get lost and reassigned to other node with increase of App attempts. In some cases, this reassignment can happen again and again that potentially cause app failure.
- Without any notification and timeout mechanism, AM running on a decommissioning node can hardly find ways to migrate its own states to a new node that already upgraded. AM has to pay more price to recovery its state from somewhere to survival from regular upgrades but not failure casually.
- Current operation flow (based on node decommission) restraint the number of nodes being upgraded at the same time if user want their applications can keep running which cause the upgrade of large cluster seems like an endless process.

Thus, we need a new mechanism that could decommission nodes in a more “graceful” way.

Key Requirements

- NodeManager can be decommissioned after running app/containers against the node get finished rather than immediately. We call the node in stage of “DECOMMISSIONING” before it get decommissioned finally.
- A new CLI is provided to wait nodes’ new decommission process which accept a timeout as input. After timeout, it will call today’s forcefully decommission on nodes still in “DECOMMISSIONING”(or simply all nodes on decommission list). In addition, we can notify AM about the plan of decommissioning and the timeout for AM to decide whether to do anything about it or if their containers on that node will complete within that time naturally.
- If CLI get interrupted, then it won’t keep track of timeout to forcefully decommissioned left nodes. However, nodes in “DECOMMISSIONING” will still get terminated later (after running apps get finished) except admin call recommission CLI on these nodes explicitly.

- When NM in “DECOMMISSIONING”

➔ Containers can keep running to the end, NodeManager’s services (like shuffle) still serve as a normal node, but no new containers get allocated on this NM.

➔ NodeManager will be terminated (decommissioned) if no live containers and no running apps consume any services on this NM.

➔ AMs who has running containers on this NM will get noticed with preemption framework (with timeout), especially AM itself run against on this node. AM is supposed to handle properly for this situation later.

➔ NM in the state of “DECOMMISSIONING” can be rollback to “RUNNING” state by recommission on this node.

➔ Before NMs get decommissioned, the timeout can be updated to shorter or longer. e.g. admin can terminate the CLI and resubmit it with a different timeout value.

- (Optional) Notify admin (in log or UI) that NMs are decommissioned gracefully (with all related apps completed) or get timeout that help admin to set better timeout value for future.

- The command line to “gracefully” decommission nodes keep almost the same as the old one as “yarn rmadmin -refreshNodes” with a new option “-g” (gracefully or gently), no new configuration get added at this moment.

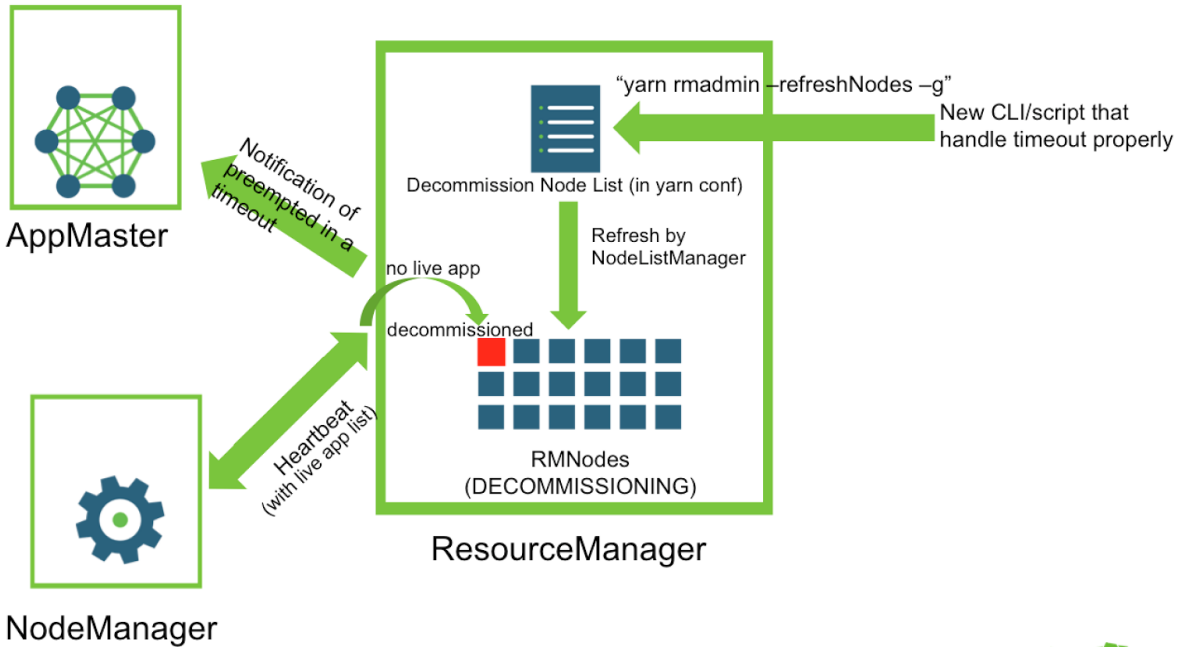
- UI (and CLI output) changes for listing NM’s status, a new state of “DECOMMISSIONING” will be added.

- Works well with other cases, e.g. NM being restart with work preserving (Rolling Upgrade) or RM in transition between active and standby (RM HA). Nodes in stage of “DECOMMISSIONING” need to get stored in persistent storage to survival from RM failed over (tracked by YARN-2567).

Design

1. Architecture

The overall architecture for this feature is as following:



When triggered by user’s call on new decommission CLI, it will call RMAdmin CLI to notify ResourceManager to decommission nodes gracefully. NodeListManager running inside RM will load node list that specified in a configuration file (specified in “yarn.resourcemanager.nodes.exclude-path”) and send out events to notify related RMNodes (state machines within RM to represent NM) that they are being decommissioned. The states of these nodes will be marked as **DECOMMISSIONING** and their available resources will be deduced from YARN cluster (by calling APIs provided by YARN-291 for node’s resource adjust in runtime) and get updated every time when container get finished. This state change only happens in RM side, and NM doesn’t need to aware any of it.

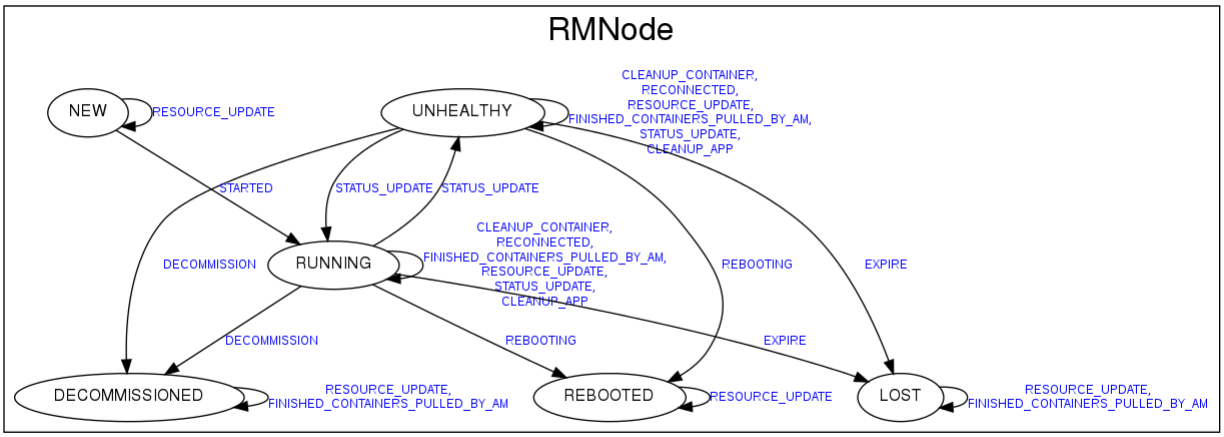
Then, whenever next heartbeat comes from these NMs, the regular resource tracker service in RM will check that if live app list is empty from heartbeat messages. If so,

RMNode will get updated to decommissioned state and send back shutdown response to these NMs (just like what decommission do today).

AM can get notified on decommissioning of Node. Container preemption framework could be used here and a timeout can be passed so AM can decide what to do before the timeout no matter it is LRS or not.

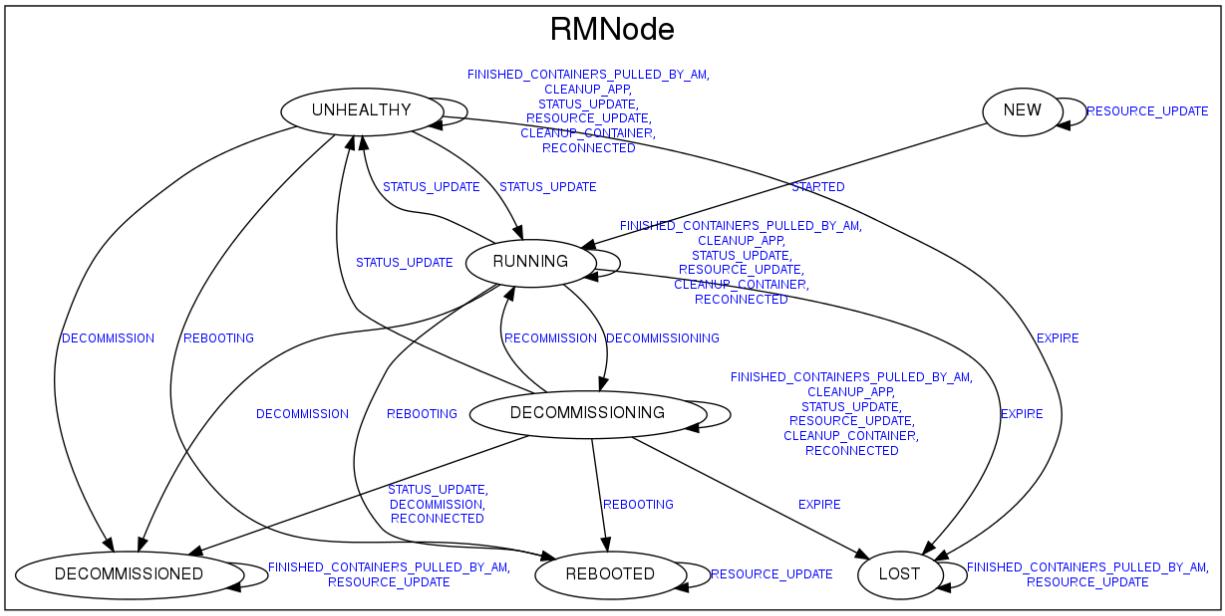
2. More Details

New state and state transition event for RMNode
A new state of “DECOMMISSIONING” will be added and can transition from “running” state triggered by a new event - “decommissioning”.
This new state can be transit to state of “decommissioned” when Resource_Update if no running apps on this NM or NM reconnect after restart. Or it received DECOMMISSIONED event (after timeout from CLI).
In addition, it can back to “running” if user decides to cancel previous decommission by calling recommission on the same node.
The reaction to other events is similar to RUNNING state.
The diagram of state machine of RMNode will be updated,
From:



ResourceManager

To:



ResourceManager

Resource update from RMNode decommission/recommission

When node transit from RUNNING to DECOMMISSIONING, it will send RMNodeResourceUpdateEvent to update its available resource to 0 (and record the previous value somewhere for possible rollback later). The used resource will keep update when containers get finished on this NM. However, the available resource will consistently be 0 (recorded available resource will get updated) because there is no free resource for new container get launched. if recommission is triggered, it recover available resource back from recorded available resource with another RMNodeResourceUpdateEvent.

AM notification for containers on DECOMMISSIONING node

YARN preemption framework will be used to notify AM that some containers will be preempted after a timeout.

New parameter for "-refreshNodes" in RMAdmin CLI

When the "refreshNodes" with an additional parameter like "-g" (means gracefully or gently) get triggered in RMAdmin CLI, it will call a new method call regreshNodesGracefully() in NodeListManager which will send the new RMNode event - DECOMMISSIONING to related RMNodes to make them transite to new state of "DECOMMISSIONING".

Persist new RMNode state of DECOMMISSIONING

RMNode in state of “DECOMMISSIONING” need to get persisted to across RM failed over (see more details in YARN-2567).

New CLI/script for monitoring gracefully decommission process with timeout

New configurations
No.

New UI page or contents
(JIRA comments from Jason) As for the UI changes, initial thought is that decommissioning nodes should still show up in the active nodes list since they are still running containers. A separate decommissioning tab to filter for those nodes would be nice, although I suppose users can also just use the jquery table to sort/search for nodes in that state from the active nodes list if it's too crowded to add yet another node state tab (or maybe get rid of some effectively dead tabs like the reboot state tab).

3. BREAK DOWN WORK ITEMs

Plan to file several sub JIRAs as below:
New state and state transition event for RMNode gracefully decommission

Resource update properly, include to make RMNode keep track of old resource for possible rollback and keep available resource to 0 and used resource updated when container finished

Notify AM with containers (on decommissioning node) could be preempted after timeout

New parameter for RMAdmin CLI

New CLI/script for monitor gracefully decommission process with timeout

Add diagnostic info for long running services

Minimum UI changes