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 within an allowed time (minutes in usual), not suddenly. We call it “decommission in progress” before it get decommissioned finally.

- When decommission in progress
➔ NodeManager will be terminated (decommissioned) after a timeout or ahead if no live containers and no running apps consume any services on this NM.
➔ Containers can keep running until timeout or finished after “gracefully” decommissioned.
➔ NodeManager’s services (like shuffle) serve as a normal node, but no new container gets assigned on this NM.
➔ AMs who has running containers on this NM will get noticed, especially AM itself run against on this node. AM is supposed to handle properly for this situation later.
➔ NM in the statue of “decommission in progress” can be rollback to normal nodes before decommissioned
➔ Before NM decommissioned, the timeout can be updated to shorter or longer

- 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.

- Node decommission command line keep the same as old one to keep compatibility. However, adding two configurations: “gracefully_decommission_enabled” and “gracefully_decommission_timeout”. By default it set to false, and the behavior is exactly the same as before.

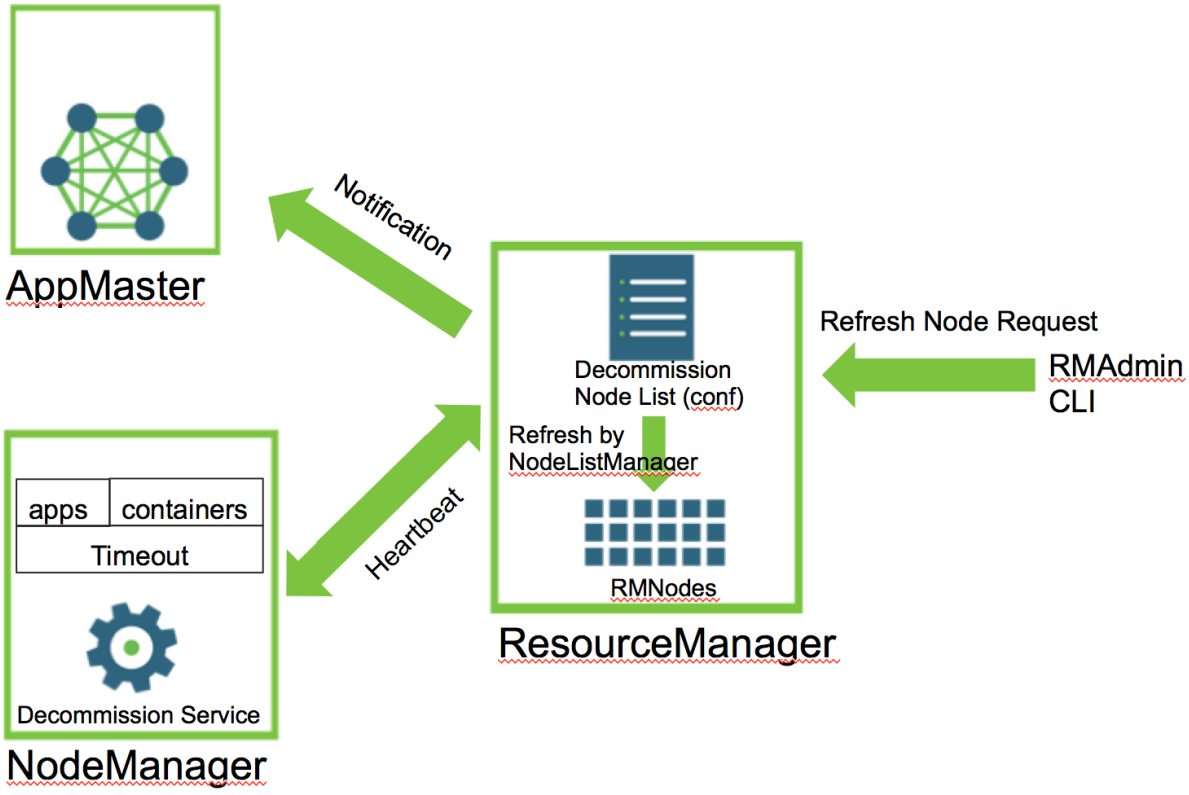
- UI (and CLI output) changes for listing NM’s status, a new state of “decommission_in_progress” will be added.

- Works well with corner cases, e.g. NM being restart with work preserving (Rolling Upgrade) or RM in transition between active and standby (RM HA).

Design

1. Architecture

The overall architecture for this feature is as following:



When trigged by user's call on RMAdmin CLI on ResourceManager to decommission nodes, 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 decommission_in_progress and their resources will be deduced from YARN cluster (by calling APIs provided by YARN-291 for node's resource adjust in runtime).

Then, whenever next heartbeat comes from these NMs, the new state as a message will update back to NMs with timeout value. When receiving this new state and value, NM will have a new service – decommission service to handle this "decommission_in_progress" situation. The new service will track the states of running containers and running apps (that launched containers on this NM before) as well as the time interval since decommission time. When all running containers/apps get completed or hit to timeout, this decommission service will send a heartbeat to notify RM that this node is already decommissioned and shutdown NM afterwards.

In the mean time, AM can pull the states of related NMs that in decommissioning (may leverage existing preemption framework) to get understand potentially resource change.

2. More Details

New state and state transition event for RMNode

A new state of “decommission_in_progress” will be added and get transition from “running” state trigged by a new event call “gracefully_decommission”. This new state can be transit to state of “decommissioned” if receiving heartbeat from NM that all apps get completed or hit to timeout. Or it can back to “running” if user decides to cancel previous decommission (by calling refresh node to re-commission on the same node).

New Decommission Services within NodeManager

A new daemon thread within NodeManager will get launched when receiving heartbeat back from RM with hearing “decommission in progress”. Like existing DeletionServices within NM, Decommission Services will track the status of apps that used to run against this NM before. If all related apps are completed (success or failed) or get timeout there, the decommission service will update NM state as “decommission_complete” or “decommission_timeout” into a shared context within NM and update to RM in next heartbeat.

New request and response messages for NM-RM heartbeat

Previously, we only have two node action messages - “SHUTDOWN” and “Resync” that RM notify NM to do some action. Now we need to add a new action - “DECOMMISSON_IN_PROGRESS” to notify NM to launch decommission services and start tracking time and status of apps.
In addition, when decommission service decided to execute decommission, the heartbeat request from NM with new messages or states will be sent to RM to notify this node get “decommission_complete” or “decommission_timeout”. Receiving these new messages, the RM will return “shutdown” node action message immediately to shutdown NM daemon, like current decommission process.

New API between AM and RM

When decommission process get started, we need to find way to notify AM that potential resource change on specific nodes. New API could be added between AM and RM, via ApplicationMasterProtocol or AMRMClient. (or via existing preemption framework?)

New configurations

Basically, two configurations will be added here:
“yarn.resourcemanager.nodes.gracefully-decommission.enable” to enable or disable gracefully decommission feature, which default to be false.

“yarn.resourcemanager.nodes.gracefully-decommission.timeout” is specified the timeout for decommission in progress, default to be 600 secs (?)

New UI page or contents (TBD)

3. Corner Cases and Open Questions

RM in failed over (with HA enabled) when gracefully decommission is just triggered. We should make sure the new active RM can carry on the action forward (how to keep sync for decommissioned node list between active and standby RM?)

NM restart with work preserving happens before or after NM being decommissioned (not finished yet). What behavior is expected here?

AM could get notified that undergoing NM is being decommissioned, how AM deal with it (like migrate its states)?

With containers of long running services, the timeout may not help but only delay the upgrade/reboot process. Shall we skip it and decommission directly in this case?

Another possibility is to track decommission timeout in RM side, instead of NM side - a new decommission services proposed above. Which way is better?