Dec.
2015

Application Master in-place upgrade

MARCO RABOZZI, GIOVANNI MATTEO FUMAROLA, SARVESH SAKALANAGA,
KONSTANTINOS KARANASOS, ARUN SURESH

Requirements

A core requirement for supporting long-running services on top of YARN is the possibility to upgrade the Application Master (AM) owning the service without losing the work done within its containers. This allows to deploy bug-fixes and new versions of the application master without incurring in long service downtimes.
More specifically, we should be able to provide the following functionalities:

- The user should be able to send an upgrade request for a running application
- The user should be able to monitor the upgrade status of the application
- The system should not kill the application containers during the upgrade
- The system should implement a rollback mechanism in case of update failures

Design Overview

Currently, YARN supports the capability of restarting an Application Master (AM) without killing the containers of the current application attempt. Once the AM restarts and re-registers to the Resource Manager (RM), the AM receives the list of running containers and Node Manager (NM) tokens from the previous attempt (YARN-1489). In order to support the in-place upgrade of the AM, we leverage the current work done for the work-preserving Application Master restart.

The AM in-place upgrade requires a new API for the **ApplicationClientProtocol** that allows the user to specify the ID of the application to upgrade and an **ApplicationUpgradeContext** that includes:

1. The resources required by the new AM container (e.g.: memory)
2. The new **ContainerLaunchContext** for the AM (e.g.: local resources, environment variables, start commands…)

Once the RM receives the request, it should locate the currently running application specified by the ID, stop the execution of the corresponding AM and start the execution of the new AM without killing the containers of the application. Once the new AM registers with the RM it will be provided with the list of containers and NM tokens owned by the previous instance of the application master. The start of the new AM is assigned to a new attempt that should not count towards the number of failing attempts for the application.

During the upgrade, the new AM could fail to start or fail after a short time period. In this situation, the RM should rollback and restart the previous application submission context in a new attempt. On the other hand, if the new AM remains alive for a certain time period (specified as a global configuration parameter) the upgrade is committed and a new submission context is stored for the application.

To keep track of upgrades and rollbacks, we add a new version number field to the attempt. The version of the first attempt of a newly submitted application is set to 0 by default. Once an upgrade is requested, the new attempt is assigned a version number that is the maximum value of the versions of the previous attempts increased by 1. In case of failures during the upgrade, a new attempt is started with a version number equal to the one of the attempt that was running before the upgrade request.

To allow the user to see the status of the upgrade, we add the information of the version attempt within the **ApplicationAttemptReport** and the UPGRADE states in the **ApplicationReport** that are returned to the client by means of **ApplicationClientProtocol**.

Implementation details

AM upgrade

In order to support the in-place upgrade functionality, we need to add new states within the state machines of the application (figure 1) and the application attempt (figure 2).
The following paragraphs give a detailed description of the upgrade steps for an application that was submitted with the **keepContainersAccrossApplicationAttempts** parameter set to true (see also figure 3 for a graphical representation).

Request for AM upgrade

1. The client sends an application upgrade request to the resource manager. Upon receival, the RM verifies the validity of the request and checks that the AM is managed by the RM. If the request is valid and the current application is running, the RM asks the state store to save the **ApplicationUpgradeContext** in the application data. During this step, the application state is changed from **RUNNING** to **UPGRADING_SAVING**.

Killing Old AM

2. Once the state store confirms that the upgrade information is saved (via the **APP_UPDATE_SAVED** event), the application state is changed to **CURRENT_VERSION_KILLING**. In this state, a **KILL_VERSION** event is sent to the current running attempt. The attempt goes into **FINAL_SAVING** and, after the state store confirms that the information is saved, the attempt moves to the final **KILLED_VERSION** state. During this transition, a request to the NM is sent to kill the current AM and the application state machine is notified via the **VERSION_KILLED** event.
3. The NM, after having received the command from the RM, sends a kill signal to the container of the AM.

Starting new AM

4. The application state machine, after having received the **VERSION_KILLED** event, moves to **CURRENT_VERSION_KILLED** state. During the transition, the RM asks the node manager (not necessarily the same node manager where the previous AM was running) to start an AM using the **ApplicationUpgradeSubmissionContext** provided by the client. From the RM point of view, a new attempt is created with a new version number (computed as described in the previous section).
5. The NM starts the application master.
6. Once the AM is started, it sends a request for registration to the RM.
7. The RM registers the new AM and gives to the AM the list of containers already in use and the NM tokens (as in YARN-1489) . The new attempt sends an **ATTEMPT_REGISTERED** event and the state of the application is changed to **NEW_VERSION_RUNNING**.

Upgrade finalization

8. If the new attempt does not fail before the configured time period, the RM confirms the upgrade, change the application state to **UPGRADE_SAVING_SUCCESS** and asks the state store to:
    a. replace the current submission context of the application with the new one.
    b. remove the **ApplicationUpgradeContext** data.

9. Once the state store confirms that the information is saved with the **APP_UPGRADE_SUCCESS_SAVED** event, the application state is changed to **RUNNING** and the upgrade is completed.

If the application was submitted with **keepContainersAccrossApplicationAttempts** parameter set to false, the RM will kill the previous running containers during the upgrade. This is done to maintain consistency across different versions of the application, so that, in case of failures during the upgrade process, it is possible to rollback to a previous version that may not be able to handle already running containers.

Upgrade rollback and failures handling

During the upgrade process (i.e.: when the application is in one of the newly introduced state) an attempt could issue an **ATTEMPT_FAILED** event for several reasons such as: fail requested by the user, unable to launch AM container, AM container expired and so on. If we receive an **ATTEMPT_FAILED** event when the application is in the **UPGRADE_SAVING** or in the **CURRENT_VERSION_KILLING** state, it means that the currently running AM has failed before we were able to kill it. In this case, if we have reached the maximum number of failure attempts, the application state is changed directly to **FINAL_SAVING** and the application lifecycle ends. Otherwise, if the maximum number of failures has not been reached yet, the application state is changed to **UPGRADE_SAVING_FAILED**. During the transition the RM asks the state store to remove the **ApplicationUpgradeContext** data and to reconfirm the old application submission context. Once the state store confirms that the data are saved (by means of the **APP_UPGRADE_FAILED_SAVED** event), the application starts a new attempt with the old submission context and version number.

On the other hand, when we encounter a failure during the states: **CURRENT_VERSION_KILLED**, **NEW_VERSION_RUNNING** and **UPGRADE_SAVING_SUCCESS**, it means that the new version of the AM has failed. Since in these states the upgrade has not been confirmed yet, we restore the previous application data by moving the application state to **UPGRADE_SAVING_FAILED** (as done in the previous scenario). Once the state store confirms that the data are saved, the application resume from state **ACCEPTED** and an attempt with the old submission context and version number is started (rollback). Notice that in this case, neither the failed attempt running the new AM, nor the attempt issued for rollback are considered as failing attempts. This is important for long-running service since the number of maximum failed attempts is limited.

Counting of failed attempts

As stated in the previous section, the upgrade/rollback attempts should not count towards the number of failed attempts. More formally, an attempt should not count as a failure if the subsequent attempt has a different version number. Indeed, if the version number of the subsequent attempt is greater than the current one, it means that the current attempt has been killed to start a new version of the application. On the other hand, if the subsequent attempt has a smaller version number, it means that the current attempt is a tentative upgrade that failed and did rollback.

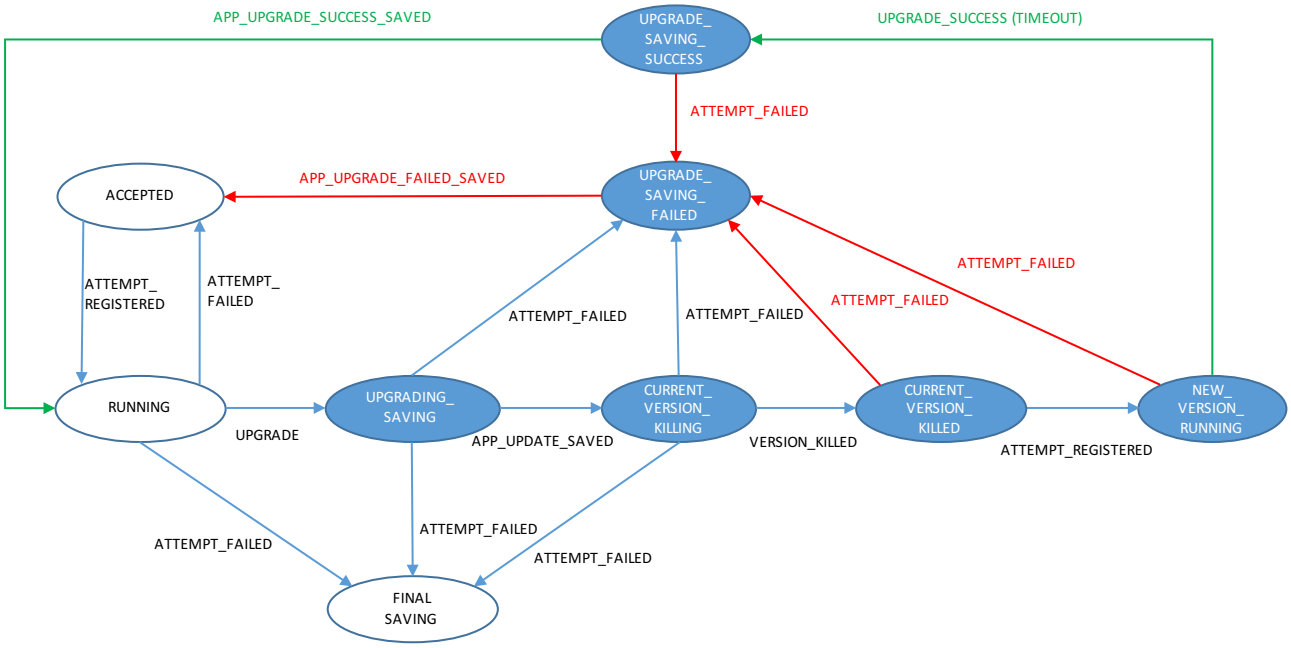

figure 1: *partial view of the application state machine (new states are shown in blue, green arrows represent successful upgrade transitions and red arrows rollback transitions)*

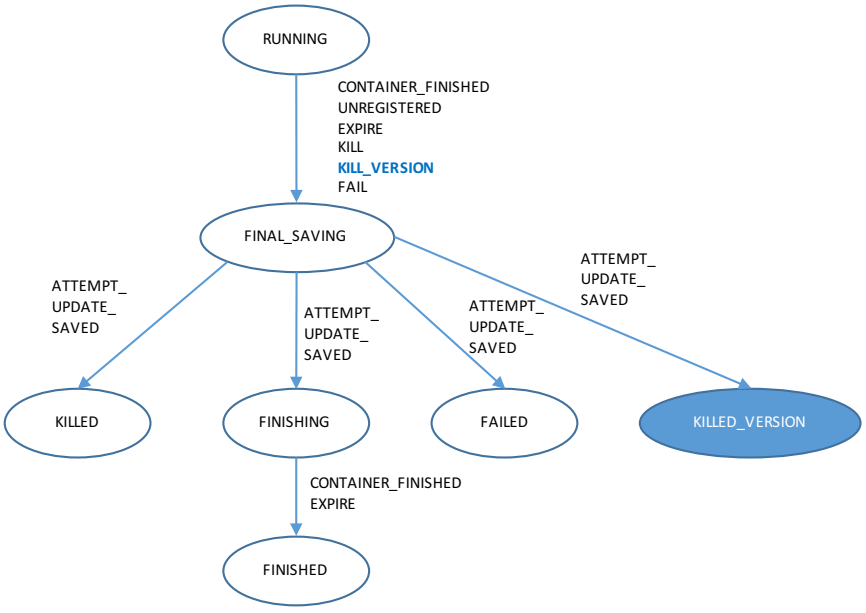

figure 2: *partial view of the application attempt state machine (new states and transitions are shown in blue)*

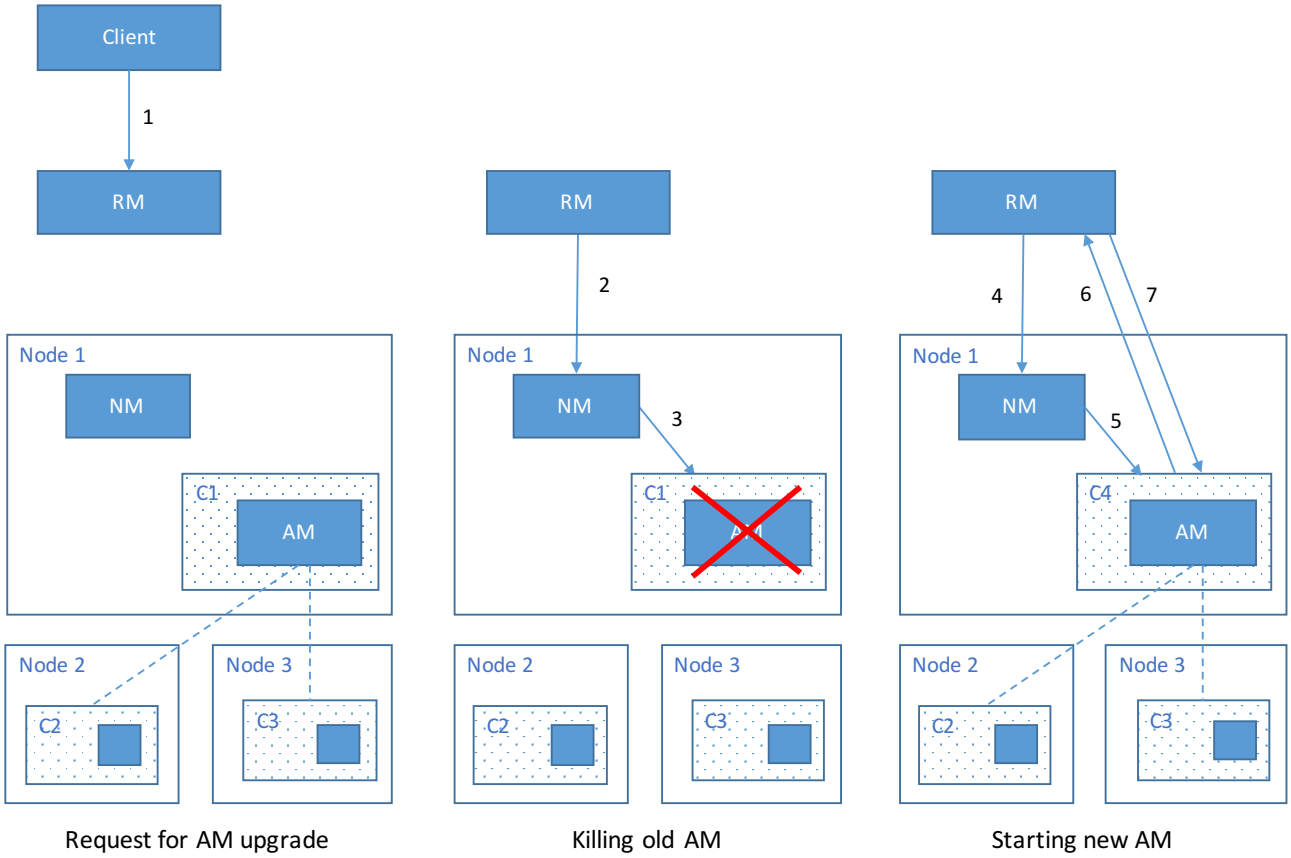

figure 3: *AM upgrade steps*

Future work

We are investigating the possibility to switch the order of the steps during the upgrade process to first start the new application attempt and then kill the old one in order to reduce the downtime during the upgrade. However, this would require to have multiple active attempts for the same application at the same time. Furthermore, we are also considering to add a new API to allow the client to switch between different (already submitted) versions of the application without the need to explicitly resend a submission context. These features will be targeted in different JIRAs.