2nd April 2015 The Number of Maximum Attempts of an Yarn Application in Hadoop Two

A hadoop job in Yarn is called an application. For example, a mapreduce job is one-to-one mapped to an Y application currently. If an application fails for a certain reason, Yarn will retry the application, i.e., launch a r application master to manage the lifecycle of the application. The number of maximum attempts is set by configuration "*yarn.resourcemanager.am.max-attempts*" with a default value ***2*** and it's a global setting for all applica masters.

```
  public static final String RM_AM_MAX_ATTEMPTS =
    RM_PREFIX + "am.max-attempts";
  public static final int DEFAULT_RM_AM_MAX_ATTEMPTS = 2;
```

However, each application can set its own maximum number as long as the individual number is less than the glo upper bound. Otherwise, the resourcemanager will override it. The ApplicationSubmissionContext class provide method to set this number.

```
public abstract class ApplicationSubmissionContext {

  /**
   * Set the number of max attempts of the application to be submitted. WARNING:
   * it should be no larger than the global number of max attempts in the Yarn
   * configuration.
   * @param maxAppAttempts the number of max attempts of the application
   * to be submitted.
   */
  @Public
  @Stable
  public abstract void setMaxAppAttempts(int maxAppAttempts);

}
```

The retry logic lives inside the class RMAppImpl, which is a representation of an Yarn application on resource mana side.

```
public class RMAppImpl implements RMApp, Recoverable {
  private final int maxAppAttempts;
  private boolean isNumAttemptsBeyondThreshold = false;

  public RMAppImpl(ApplicationId applicationId, RMContext rmContext,
      Configuration config, String name, String user, String queue,
      ApplicationSubmissionContext submissionContext, YarnScheduler scheduler,
      ApplicationMasterService masterService, long submitTime,
      String applicationType, Set applicationTags,
      ResourceRequest amReq) {

    int globalMaxAppAttempts = conf.getInt(YarnConfiguration.RM_AM_MAX_ATTEMPTS,
        YarnConfiguration.DEFAULT_RM_AM_MAX_ATTEMPTS);
    int individualMaxAppAttempts = submissionContext.getMaxAppAttempts();
    if (individualMaxAppAttempts <= 0 ||
        individualMaxAppAttempts > globalMaxAppAttempts) {
      this.maxAppAttempts = globalMaxAppAttempts;
      LOG.warn("The specific max attempts: " + individualMaxAppAttempts
```

```
          + " for application: " + applicationId.getId()
          + " is invalid, because it is out of the range [1, "
          + globalMaxAppAttempts + "]. Use the global max attempts instead.");
    } else {
      this.maxAppAttempts = individualMaxAppAttempts;
    }
    ...
  }
}
```

As we can see, the *globalMaxAppAttempts* is obtained from Yarn configuration file and the *maxAppAttempts* is obtai from ApplicationSubmissionContext and then compared with the *globalMaxAppAttempts*.

The number of actual failed application attempts is calculated by the following method to exclude AM preempt hardware failures or NM resync.

```
private int getNumFailedAppAttempts() {
  int completedAttempts = 0;
  long endTime = this.systemClock.getTime();
  // Do not count AM preemption, hardware failures or NM resync
  // as attempt failure.
  for (RMAppAttempt attempt : attempts.values()) {
    if (attempt.shouldCountTowardsMaxAttemptRetry()) {
      if (this.attemptFailuresValidityInterval <= 0
          || (attempt.getFinishTime() > endTime
              - this.attemptFailuresValidityInterval)) {
        completedAttempts++;
      }
    }
  }
  return completedAttempts;
}
```

where *shouldCountTowardsMaxAttemptRetry()* is defined in class RMAppAttemptImpl as follows.

```
public boolean shouldCountTowardsMaxAttemptRetry() {
  try {
    this.readLock.lock();
    int exitStatus = getAMContainerExitStatus();
    return !(exitStatus == ContainerExitStatus.PREEMPTED
        || exitStatus == ContainerExitStatus.ABORTED
        || exitStatus == ContainerExitStatus.DISKS_FAILED
        || exitStatus == ContainerExitStatus.KILLED_BY_RESOURCEMANAGER);
  } finally {
    this.readLock.unlock();
  }
}
```

Then the variable *maxAppAttempts* is checked in the state transition AttemptFailedTransition to set the *isNumAttemptsBeyondThreshold*.

```
  private static final class AttemptFailedTransition implements
      MultipleArcTransition {

    private final RMAppState initialState;

    @Override
    public RMAppState transition(RMAppImpl app, RMAppEvent event) {
      int numberOfFailure = app.getNumFailedAppAttempts();
      LOG.info("The number of failed attempts"
          + (app.attemptFailuresValidityInterval > 0 ? " in previous "
              + app.attemptFailuresValidityInterval + " milliseconds " : " ")
          + "is " + numberOfFailure + ". The max attempts is "
          + app.maxAppAttempts);
      if (!app.submissionContext.getUnmanagedAM()
          && numberOfFailure < app.maxAppAttempts) {
        if (initialState.equals(RMAppState.KILLING)) {
          // If this is not last attempt, app should be killed instead of
          // launching a new attempt
          app.rememberTargetTransitionsAndStoreState(event,
            new AppKilledTransition(), RMAppState.KILLED, RMAppState.KILLED);
          return RMAppState.FINAL_SAVING;
        }

        boolean transferStateFromPreviousAttempt;
        RMAppFailedAttemptEvent failedEvent = (RMAppFailedAttemptEvent) event;
        transferStateFromPreviousAttempt =
            failedEvent.getTransferStateFromPreviousAttempt();

        RMAppAttempt oldAttempt = app.currentAttempt;
        app.createAndStartNewAttempt(transferStateFromPreviousAttempt);
        // Transfer the state from the previous attempt to the current attempt.
        // Note that the previous failed attempt may still be collecting the
        // container events from the scheduler and update its data structures
        // before the new attempt is created. We always transferState for
        // finished containers so that they can be acked to NM,
        // but when pulling finished container we will check this flag again.
        ((RMAppAttemptImpl) app.currentAttempt)
          .transferStateFromPreviousAttempt(oldAttempt);
        return initialState;
      } else {
        if (numberOfFailure >= app.maxAppAttempts) {
          app.isNumAttemptsBeyondThreshold = true;
        }
        app.rememberTargetTransitionsAndStoreState(event,
          new AttemptFailedFinalStateSavedTransition(), RMAppState.FAILED,
          RMAppState.FAILED);
        return RMAppState.FINAL_SAVING;
      }
    }
  }
```

As shown above, if *numberOfFailure >= maxAppAttempts*, the application state transfers to ***FAILED*** and no more retri