Work-preserving restart of ApplicationMasters

```
By: Vinod Kumar Vavilapalli & Jian He
Last edited: December 30 2013
```

The ‘what’

In YARN, ApplicationMasters (AMs) run on compute nodes just like every other container (ignoring unmanaged AMs for this context). AMs can crash, exit, or shut-down for various reasons. Today if an AM goes down,

- ResourceManager (RM) kills all the containers of that ApplicationAttempt. This includes
  - Any containers that are currently running on NodeManagers (NMs)
  - Outstanding requests from previous AM.
- RM automatically starts a new ApplicationAttempt on the same or another compute node

There are different classes of applications that wish to deal with a AM-restart event in different ways. MapReduce-like applications can live with losing presently running containers, though not losing work is a desired feature. But for long running YARN services with long-lived containers, we don’t want the AM to be a single point of failure that brings the whole service down.

Problems to solve

When an AM goes down and restarts

- Depending on app’s desire, we should or shouldn’t kill the containers from the previous ‘generation’.
- RM does not persist any information about resource-requests and their allocation. These are mapped to an ApplicationAttempt and when a new AM is launched, RM forgets about previously outstanding requests for scalable recovery. The new ApplicationAttempt needs to be aware of an expected

consistent behavior for outstanding requests across AM restarts.

- Containers may finish in the interim while an AM crashes and the subsequent AM is launched. These notifications are again bound to an ApplicationAttempt, so are not tracked in the RM and are completely lost.
- The new ApplicationAttempt doesn't know where the containers started by the previous ApplicationAttempt are running.
- In case of apps like MapReduce where containers need to communicate directly with AMs, the old running-containers don't know where the new ApplicationMaster is running and how to reach it (service addresses).
- Clients also need to learn about the new AM. Some of it is already implemented for the MapReduce framework - the MR JobClient intelligently redirects between the old AM, RM and the new RM with help from the RM. This solution needs to be made generic.

We need to address these issues to enable work-preserving AM restart. Some of these issues can be resolved at the application level, but there are others which need common solution in YARN for all the possible apps.

The 'how'

Today containers are bound to an ApplicationAttempt. This is the natural binding. For work-preserving AM restart, we will

- **Rebind** the containers to a new ApplicationAttempt when an app restarts.
- **Running containers** - A new AM will be informed of any containers running in the system. This involves AMRMProtocol changes - addition of new fields for sending this information to the AMs.
    - This only covers basic container information - ContainerID, NodeID (host:port), Node Http address, resource size, priority etc.
    - If containers have more information to share with the AMs (say a RPC server inside the container where AM pushes work to), that information is not available at the RM and so is not in the scope of this document.

As need arises, this can be implemented by having a meta-data service running in the NodeManager for the local-containers).

- **Previously outstanding containers** - allocated, acquired, reserved - will be cancelled/killed and AMs have to resubmit them.
    - Arguably, this is lost work (scheduling effort) but remembering them has implications on RM scalability.
    - An alternate solution to inform the app of previously allocated containers *only* if the RM *has not* restarted in the mean-while is possible. But this is inconsistent view for the apps as they'll not be sure when they will get notifications about outstanding containers and when they may not.
- **Finished containers**: RM will notify the new AM of any finished-in-the-interim containers.
- Some kind of a **registry-service** for containers to learn about their new AM's location. We can either
    - Make available a library for containers to find out the new AM's location:
        - Use existing YARNClient to directly ask the ResourceManager
        - For scalability, we can have a NM-level local service that in turn talks to the RM (pull or push) together with a cache of AM locations.
        - Have an alternate scalable solution outside core YARN, say via ZooKeeper
    - or continue the status-quo. Neither force nor obstruct application-level logic to learn about new AMs, say via ZooKeeper.

More details

- For **rebinding** of containers to a new AM, inside the ResourceManager, we need to transfer containers from ApplicationAttempt 1 to ApplicationAttempt

2. This involves efforts to

    - reroute any events destined to the container
    - reroute any events originating from a running container (FINISH etc) to the previous ApplicationAttempt.

- **Finished containers**: There is a problem with notification of finished containers. It exists even with the Work-preserving RM restart effort too. RM doesn't remember any container-state and if it crashes before an AM is sent the corresponding information, neither a current AM beyond RM-restart nor a restarted-AM have any way of getting notified about those containers. This is a general problem about reliable-notification.

    - One simple solution is to make NM remember finished-containers till RM acknowledges. RM shouldn't ack the NM till AM receives them.

- **Cluster changes**: A similar notification problem exists w.r.t node-addition and node-removal and the corresponding message to the restarted AMs. This has been an issue with non work-preserving AM restart too though.

- **Rebinding on the NM**: NMs only deal with applications and not AMs, so would not need to do any rebinding of the containers when an AM crashes.

- **RM restart**: Because RM doesn't remember containers and their allocations, any rebinding that happens is forgotten. This problem exists even after we have a work-preserving RM restart, the fundamental problem being that we don't record the rebinding. Let's say an app gets restarted, a rebinding happens and then RM goes down and comes back up. Nodes resync back with the RM, report running containers. Now routing these events to the correct AppAttempt needs to be done. Two solutions

    - Store the rebinding. Not scalable.
    - Always send notifications about containers to the last alive AM. Seems reasonable, but are there any issues?

- **ApplicationHistory**: Because of the rebinding, all tools that depend on the AppAttempt-to-container hierarchy will be broken. One such example is Application-History. When any history of the rebound containers is recorded,

it should be such that they are tracked against the new ApplicationAttempt.

- **Security**: Given that outstanding requests are cancelled, container-tokens are newly issued against containers requested by a new AM. NMTokens are also manageable similar to non-work preserving AM restart.