Work Preserving RM Restart

Motivation

YARN-128 added the ability to preserve RM application state in a pluggable persistent store and enabled the RM to refresh that state upon restarting. This allowed YARN to continue to function across RM restarts in a user transparent manner. After refreshing its state from the store, the RM would restart all the application masters (AMs) and kill all running containers so as to start from a clean state w.r.t. the dynamic container state of the cluster. As the next step to that effort, in work-preserving RM restart, we plan to refresh the dynamic container state of the cluster from the node managers (NMs) after RM restart. Restarting AM's and killing containers will not be required.

Logical flow of Work-Preserving RM Restart

1) Upon restart, the RM refreshes the application state from the store.
2) The internal state machines of previous applications/application-attempts will be transitioned to RUNNING state. ApplicationMasterService will be updated to consider these applications in its known applications set. The scheduler will be updated with these applications as the running applications.
3) At this point the RM would open its RPC servers for incoming connections (this is what currently happens).
4) When an NM heartbeat's with the RM it will be sent a RESYNC response (this is what currently happens). On receiving a RESYNC the NM will re-register with the RM (this is what currently happens). In this registration, along with the NM resource information, the NM will send information about all containers on the NM to the RM. During registration the RM will exchange a new shared key with the NM. In a secure cluster, the RPC would have already authenticated the NM via Kerberos.
5) Upon receiving a registration from an NM, the RM will add that node as usual with the additional container information being sent to the scheduler/liveliness monitor. The scheduler is expected to add the node and re-populate its state with the running container information. E.g. the scheduler would associate running containers with the application attempts and do necessary book-keeping (in the same manner as assigning those containers in the normal case).
6) When a known AM registers/unregisters with the ApplicationMasterService, it is allowed to do so like normal.
7) When a known AM heartbeats with the ApplicationMasterService then the RM sends it a RESYNC response (this is what currently happens) and accepts future heartbeats from that AM starting with sequence number 0. Upon receiving a RESYNC response, the AM will heartbeat to the RM with the full set of outstanding resource requests. This will inform the RM scheduler about all pending requests from the AM.
8) An AM is expected to block on the RM for register/heartbeat/unregister. If an AM crashes while the RM is restarting then the NM will report that completed container to the RM upon NM RESYNC. The AppAttempt state machine will handle the AM container completion like it normally does.

9) RM may have lost some completed container statuses that had been reported by the NM but not pulled by the AM before the RM restarted. The NM may be changed to discard completed container statuses after being instructed to do so by the RM via the NM heartbeat. The RM could send the discard instruction after the AM has pulled the information from the RM. NM will send all outstanding container completion statuses to the RM during RESYNC. This may result in the AM getting a duplicate completed container status if the RM had restarted before it could instruct the NM to discard pulled container requests. Given that neither solution is perfect, we may choose to not make additional changes. The AM can always get container status directly from the NM.

It is expected that the NM and AM synchronization can occur concurrently. The main issues would be whether the scheduler can successfully do the book-keeping. To the scheduler, this situation is similar to a running cluster losing all nodes and then recovering them again and so should be somewhat orthogonal to RM restart.

Overall work items

1) RESYNC on AM-RM protocol. Complexity – low.
2) RESYNC on NM-RM protocol. Complexity – low.
3) Startup changes in RM to transition applications to running state and update the necessary services. Complexity – medium.
4) Changing Capacity/Fair/Fifo schedulers to support re-populating scheduler state from NM/AM RESYNC information. Complexity – high.