YARN ResourceManager Automatic Failover

Motivation

ResourceManager (RM) is the central controlling entity in a YARN cluster for resource allocations. RM failure results in a disruption of allocation of new work in the cluster and prevents external clients from obtaining information about the cluster. Thus, high-availability (HA) of the RM is essential for high-availability of YARN. RM HA involves the following bodies of work

1) Preserving essential RM state that may be lost upon RM failure. A new RM instance can load this saved state and continue to provide services without additional user input. This is tracked via YARN-128.
2) Automatic failover of RM to enable safe and consistent transfer of control between RM instances. This is being tracked via YARN-149 and is the focus of this document. YARN-128 is a pre-requisite since safe transfer of stored state is an essential requirement for this.
3) Preserving ongoing work in the cluster. The active work in the cluster is a high volume, high churn data set and hence is not stored as part of the RM state. This work may be lost after the RM restarts. It is envisioned that the new RM instance can refresh its knowledge of the outstanding information about the cluster from the NodeManager (NM) service instances. Hence, RM failover may proceed without work-preservation. This is tracked via YARN-556.

High-Level Design

The main areas of work to enable RM failover are

1) Automatic server failover
2) Automatic client failover
3) Fencing of shared data

Automatic Server Failover

Automatic server failover involves leader election among RM server instances and transfer of control to the leader. This infrastructure has already been built in Hadoop Common HA packages and we will leverage that work for RM HA. The main changes envisioned are as follows

1) Adding the notion of Active and Standby service states (HAServiceState and allied artifacts) and addition of HAServiceProtocol implementation to the RM. An Active RM serves clients. HAServiceProtocol enables an external entity to inform the RM that it may attempt to become the active RM instance. Admins or failover controllers will use this protocol to perform failovers. We envision using existing ZKFailoverController (ZKFC) as the failover controller.
2) When HA is enabled, the RM always starts in a Standby state. For non-HA scenarios, the RM starts in Active state. All RM services are started in both states. External RPC services accept client requests for the active RM instance and reject client requests for the standby RM. They may additionally provide redirect information to indicate the current active RM instance. Having the standby RM provide clients with information about the active RM is envisioned as a scalable

and fast method for active RM discovery. YARN-128 already provides a universal RMProxy layer that can be used to manage client side re-direction and caching. It is expected that the standby RM internal state will receive no external stimulus and hence there will be no internal activity on the standby RM. Internal RM services are not expected to be HA aware. External facing RPC services may be HA aware but its not necessary as the first step.

3) Upon receiving the transition to active signal, the standby RM will try to optionally fence the RMStateStore. This fencing may also be performed by the admin or failover controller. It will then load the state from the store and kick-start its internal state machines. Finally, it will change its state to Active and start accepting client requests on its RPC services.
4) Upon receiving the transition to standby signal, the active RM will first change its state to Standby. This will make its RPC servers start rejecting client requests. It will then stop internal app related services, close the RMStateStore and then re-start the internal app related services so that they are clean and ready for the next transition to active.

Automatic Client Failover

Automatic client failover involves clients getting re-directed to the new active RM instance when the RM fails over. YARN-128 already adds a universal RMProxy layer in which related logic may be added. Different types of client failovers are possible

1) IP based failover. In this the admin/infrastructure transparently transfers the configured RM address to the new RM instance.
2) Pre-configured list of RM instances. This involves the clients iterating through a pre-configured list of RM instances in order to determine the active RM. This already exists in Hadoop Common.
3) Standby re-direction to Active. This involves the standby RM providing the client with information about the active RM and may be faster than pre-configured lists when there are more than 2 RM instances.

All of the above involve the RMProxy layer caching the last known active RM so that the failover cost is paid only once per failover.

Additional changes may be needed in YARNClient, AMRMClient and the NM to handle operations that failed because they got transferred to the new RM instance. E.g. after the YARNClient creates a new application, it waits for the application to be accepted by the RM. If the RM fails over while the YARNClient is polling for acceptance then the poll that goes to the new RM instance will fail since the RM does not store a new application until it has been accepted.

Fencing of shared data

Fencing of shared data involves preventing the old RM instance from editing the shared storage. The admin or failover controller must fence the old the RM before making a new instance active. Fencing may involve harsh measures like shutting down the RM node or killing the RM process. The old RM may be allowed to voluntarily give up control before resorting to such measures. If the shared storage can be fenced then that is the least invasive option. ZooKeeper based storage may be a good option for this since it allows fencing based on permissions to create and delete znodes. ZKFC provides facilities to enable all of the above options.

High Availability

The RM is expected to startup very fast because the state stored by the RM is limited to information about currently running and queued applications. It does not store fine grained task information and expects to refresh that information from NM’s (in work preserving mode). Since NM’s typically heartbeat within a few seconds (with the default being 1 second), the refresh period is expected to be fast too. The RPC servers will start accepting client requests as soon as the saved state has been loaded from the store. Hence, the NM refresh period will not be visible to clients. This leads us to expect minimal downtime during RM failover, thus making the RM highly available. If we discover that RM failover downtimes are much higher than expected then we will need further action but that is not expected at this point.