Better Queue Management in YARN

Configuration-based Queue Management

Table of Contents

Problems

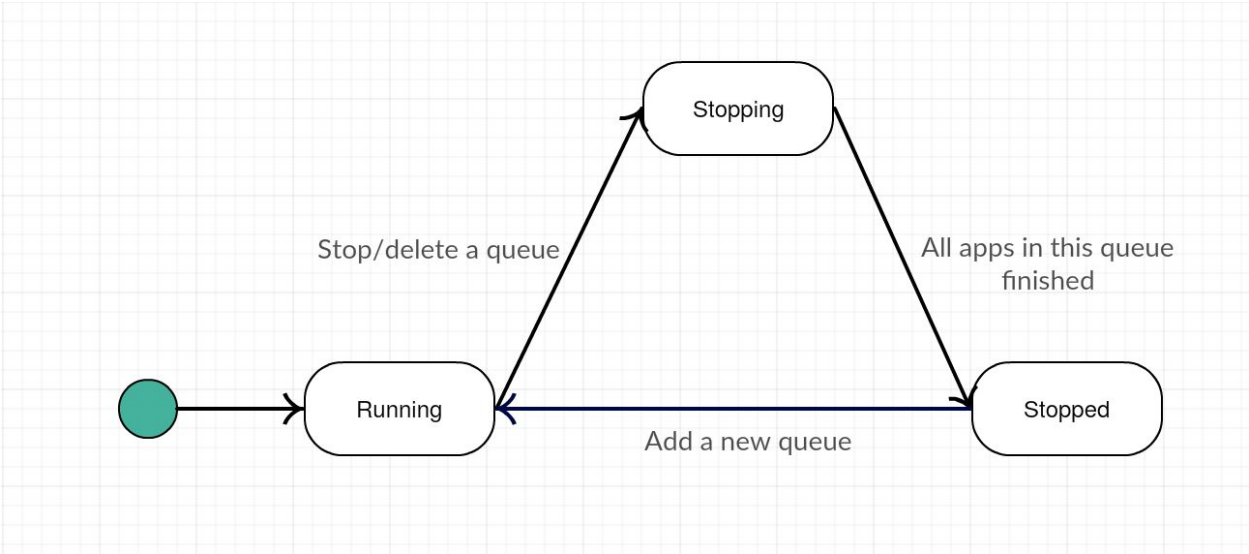
In today's approach (configuration-based) on queue-management in YARN, we still have several places that need to be improved:

- It is possible today to add or modify queues without restarting the ResourceManager, via a refresh-queues feature for the CapacityScheduler/FairScheduler. But for deleting a queue, we have to restart the ResourceManager.
- When a queue is STOPPED, resources allocated to the queue can be handled better. Currently, they'll only be used if the other queues are explicitly setup to go over their capacity.

In this design doc, we will focus on the configuration-based queue management, and how we will improve this approach to solve the existing issues.

Queue State-machine

Each queue will have 3 states: {Running, Stopping, Stopped}. We will add a simple state-machine to represent the queue. The basic flow will look like the following:



By default (or when adding a new queue), the initial state of the queue will be ***Running***.

1. In the *Running* State:
    a. When we STOP/DELETE a queue, we would transit the state from *Running* state to *Stopping* state for the queue as well as all its Children queues.
    b. This means that one can never delete a *Running* queue directly. It will have to first go into a stopping state.
2. In the *Stopping* state: We have to wait for all the applications in the queue to finish and at the same time we would reject any new applications being submitted to this queue. When all the applications in the queue finish, we would move the state from *Stopping* to *Stopped*.
3. A queue can added any time, but the added queue would inherit the same state as its parent.

Also, we must maintain the consistent state for all the queues in a hierarchy:

Parent Queue State	Children Queues State
Stopped	Stopped
Stopping	Stopping/Stopped
Running	Running/Stopping/Stopped

Configuration-based Queue Management

In order to manage the queue, the admin user needs to edit configuration files and then issue a refresh command. This action is very error-prone, and we would completely erase all in-memory information about this queue. So, it is difficult for the users to figure out what is wrong if we make a mistake when we modify the queue. For instance, we have the following queue hierarchy

```
<property>
   <name>yarn.scheduler.capacity.root.queues</name>
   <value>a1,a2</value>
</property>
```

Assume we want to delete queue a2, the admin user needs to remove a2 from the configuration file:

```
<property>
   <name>yarn.scheduler.capacity.root.queues</name>
   <value>a1</value>
</property>
```

then call rmadmin refreshQueues. After that, the queue: a2 would be erased completely.

But if the admin user makes a typo:

```
<property>
   <name>yarn.scheduler.capacity.root.queues</name>
   <value>a</value>
</property>
```

What will happen here is that we accidently remove queue:a1, queue:a2 and add a new queue:a. This small typo could bring a huge negative impact to the cluster.

To avoid this issue, we could instead ask the admins to modify the queue state and let scheduler itself to figure out the changes.

For example, to delete queue: root.a2, we should modify the queue state to

```
<property>
   <name>yarn.scheduler.capacity.root.a2.state</name>
```

```
    <value>DELETED</value>
</property>
```

As a bonus, the users would have a full picture of queue changes and in the config file, we allow multiple DELETE queue with same name exists, but we require unique queue name for running queues.

For the resources of the deleted/stopped queue, users should explicitly distribute them away to its siblings.

Overall Work Items

1. Synchronize the state of the parent queue and its child queues
2. Add state machine implementation for Queues
3. Enhancements to STOP queue handling
4. Support for deleting queues without requiring a RM restart