Intra-queue preemption

prepared by Sunil G and Wangda Tan

Problem Statement

Capacity Scheduler provides a scheduling monitor framework which primarily focus on normalizing resource distribution (by preempting oversubscribed resources) anomalies across different queues. However another class of resource distribution anomaly also may occur within a queue, if applications were using resources above its concerned limit. This could lead to a resource shortage for high demanding apps and could lead to an under-performing cluster.

Requirement

High Level Overview

Many a time, applications running within a queue may starve as other applications from same queue may have oversubscribed resources. In such cases, we can look for below key attributes to solve such resource distribution anomalies.

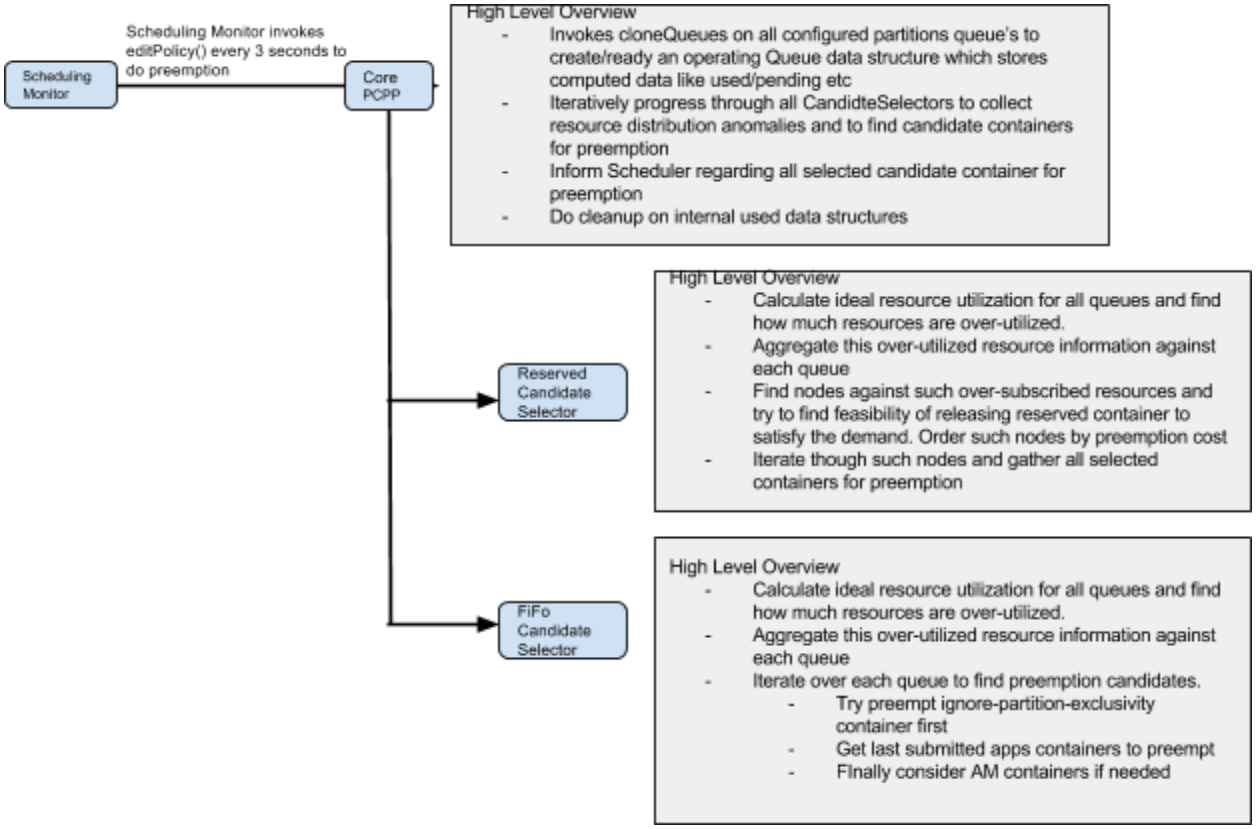
- Priority
- Fairness
- User limit

These factors will help to take the decision to preempt some low key resources for better resource distribution.

Existing Design

With the latest refactoring done in YARN-4822, ProportionalPreemptionPolicy has the support to have multiple CandidateSelector policies.

High level overview could be like below:



Additional Requirement specs

Though Intra-Queue preemption will be trying to look for anomalies within a queue, there could be few optimization and scenarios to consider for better completeness and usability. To explain this, I am introducing three categories of queue's here.

1. Over subscribed queue: A queue which has borrowed resources over its defined capacity. This is an over utilizing queue which may have high demand
2. Normal running queue: A queue which is running in its defined capacity. All its running applications are well under the configured capacity limit
3. Under served queue: A queue which does not have enough resource to serve its applications even though it has a defined capacity allocated.

Now for these types of queues, it could have pending applications demand.

- Over subscribed queue might have few starving applications w.r.t priority or user-limit. For example, there could be applications which are under user-limit but are not getting any more resources OR higher priority apps are waiting to finish lower priority apps container. If such lower priority apps are long running, high priority app will starve if there are no other resources in the cluster.
  Inter-queue preemption module could have marked few containers for preemption from this queue, hence intra-queue preemption framework can validate how much of such marked containers can help to serve starving application (user-limit or priority) in each round. “Selected Containers” will hereby indicate all containers which are marked for preemption by other CandidateSelector policies from Inter-queue preemption.
  - Selected containers will completely serve resource need from starving apps.
    For eg, an application is starving for resources in given queue. It’s possible that there could be some containers which are already marked in “Selected containers” are may be from a low priority application. This will help starved application to get its demand.
  - Selected containers only partially serves the need
    From the above example itself, starved application’s demand is served only partially by already Selected Containers. Remaining demands has to be found.
  - Selected containers are not at all serving the need from starving apps.
    From the above example, there are no other good containers pre-selected. Remaining demands has to be found.

New intra-queue preemption policy will try to do a better handling here as much as possible to resolve the resource distribution anomalies..

- Normal running queue which is untouched by preemption framework might have starving apps. This could be due to user-limit or priority and such apps could not get new resource as other apps running in same are over utilizing (fairness of resources across application).
- Under serving queue might be expecting resources from other queues, hence at this given round intra-queue preemption module will not take any action here provided there are resources available. (If unclaimable resources are available in queue due user-limit or priority, a minimum preemption factor can decide whether to act on such queues)

Configurations and considerations

- Provide a configuration to turn on/off intra-queue preemption along with the type of policy it is going to handle (priority, fairness, user-limit etc).
- When intra-queue preemption can happen: In some cases, we need intra-queue preemption when queue is under its guaranteed resource, and we also need to make sure no excessive preemption (like crossfire between apps) happens.

- Intra-queue preemption vs Inter-queue preemption. Intra-queue preemption should happen after inter-queue preemption. This could help to resolve the anomalies of resource distribution across queues before acting within queue.

Implementation Specification

Consider already selected inter-queue preemption containers.

By scanning through each partition and its associated queues (**TempQueuePerPartition**), we can understand how much resources are offered from each queue for preemption and also the selected container list. This can be used as a reference to avoid double calculations in intra-queue preemption round.

Compute Ideal Resource allocation to find under-served queues.

Based on similar ideal resource allocation calculation for all queues done by Inter-Queue preemption module, get the list of queues which starts from most under-served queue.

1. Recursively calculate ideal resource allocation for all queues to find under-served queues per partition
2. Store list of under-served queues per partition for further calculations.

Compute pending resource from most needed apps within a queue.

For each queue, compute ideal resource allocation for its running and pending applications. This derived value could be used to identify starving apps.

1. Iterate through each queues to find resource anomalies across its apps.
2. For each app (live and pending), calculate the “used”, “pending”, “guaranteed” resource information per partition level. **TempApplicationForQueue** could help to have this information stored. This app level information could be added as List of apps in **TempQueuePerPartition**.
3. ‘pending’ resource per partition will be calculated for all the apps and together store in a consolidated map (resourceToObtain) of pending resource to be collected per partition in one queue.
4. While doing this, we will ensure that certains apps will be skipped if it is already equal or more that its user-limit quota.

This map will be the entry point to select candidates from lower priority apps in next step.

Select ideal candidates for intra-queue preemption per priority.

1. Iterate through all queues one by one which is ordered from most underserved way.

2. For each queue, get all the apps from scheduler ordered by the preemption comparator, this will be help to get apps of lower priority first.
3. Give **resourceToObtain** map, we know how much resource need to be claimed per partition. So as we do in inter-queue preemption, sort all containers and get the lowest priority container first. Verify the partition it is associated with and iteratively deduct from **resourceToObtain** (per partition) for each container.
4. Finally when all apps are completed or resource demand is met, we can stop selecting containers. Also we ensure that these selected containers are not duplicated by inter-queue module earlier.
5. Return the selected container list to pcpp module for further preemption markings etc.

In this POC patch, we try to cover these much items in order which is mentioned above.