Resource Over-commitment Based on Opportunistic Container Preemption

Zian Chen, Vinod Vavilapalli, Wangda Tan
with input from: Clay Baenziger, Sunil Govindan

Motivation

In YARN, currently, we set a universal preemption monitoring interval for all the queues in the cluster. Having the same interval for all queues, capacity scheduler and Resource Manager(RM) can perform preemptions in a clean and elegant way.

However, this design cannot satisfy some scenarios. One of these scenarios is that we may have two queues which run the different type of applications, one queue contains long-running applications which should not preempt frequently while another queue may run interactive jobs which need to gather resources as soon as possible when new applications are submitted. To satisfy both of these types of application resource allocation request, we can not set universal preemption monitoring interval with long timeout intervals for them.

At the other end of the design spectrum, setting different timeout per queue are the leading alternative for offering low allocation latency and high resource allocation throughput. However, the calculation of preemptable resources per queue will become a hint since we need to predict how much resources preempted from overutilized queues will be assigned to underutilized queues. We cannot guarantee that the preempted resource will be assigned only to the demanding queue and not to any other queue since the status of resource utilization and preemption request keeps changing.

Apart from this, we also want to support over-commitment to further improve resource utilization within the cluster, since in most cases, the application which gets resource containers allocated does not fully utilize which leads to some of the allocated resource been unused inside containers. This dramatically decreases the overall resource utilization within the cluster.

Requirements

Based on the discussion above, we try to figure out a solution so that we could trigger preemption with more flexibility while improving resource utilization and decrease the amount of kill container operations. In this document, we achieve two main requirements in our design as explain below,

More aggressive preemption by setting shorter timeout

These requirements need us to deal with queue preemption frequency based on the type of applications running inside them.

To achieve this, we do the following,

- Still set universal preemption monitoring interval for all the queues within the cluster, but change the interval to be much shorter compared with the default settings of today.
- Instead of allowing applications to allocate resources with a mix of guaranteed and opportunistic containers, we allow newly submitted applications to only contain guaranteed containers.
- We change the preemption logic to, instead of killing containers, demote guaranteed containers into opportunistic ones, so that when there are new applications submitted, we can ensure that these containers can be launched by preempting opportunistic containers.

Support over-commitment in cluster resource utilization

These requirements need us to further improve the cluster resource utilization by using over-commitment. To achieve this, we enable opportunistic containers so that allocated-but-unused-space can be utilized by these containers after demoting from being guaranteed to opportunistic.

From NM's point of view, after containers demote from guaranteed to opportunistic, they can continue to use node-level resources as long as they are not preempted or killed by RM. The resources used by these opportunistic containers to further run their applications come from the actually unused allocated resource in each guaranteed containers located on the same node with the opportunistic containers. In this way, we can achieve over-commitment (JIRA: YARN-8178) with the minimum code changes to the current implementation and less impact to the current scheduler design.

Design

To address those challenges mentioned above, we will introduce opportunistic container based preemption for more aggressive preemption use cases. In this section, we will give the overall architecture design first, then give implementation details regarding our proposed changes.

Architecture Overview

The architecture of the system is shown in Figure 1. Our system is designed to change the current scheduler logic to allocate all the containers as guaranteed in the first place when new applications are submitted into queues. As time goes on, when the available resources inside the queues are insufficient for allocating more guaranteed containers, RM will check preemption when preemption monitoring interval reached and pick whatever guaranteed container inside overutilized queues as preemptable, then instead of directly killing them after *max-kill-wait-timeout*, RM will demote guaranteed containers (which are marked as preemptable) from guaranteed to opportunistic so that new guaranteed containers from pending applications can be allocated immediately.

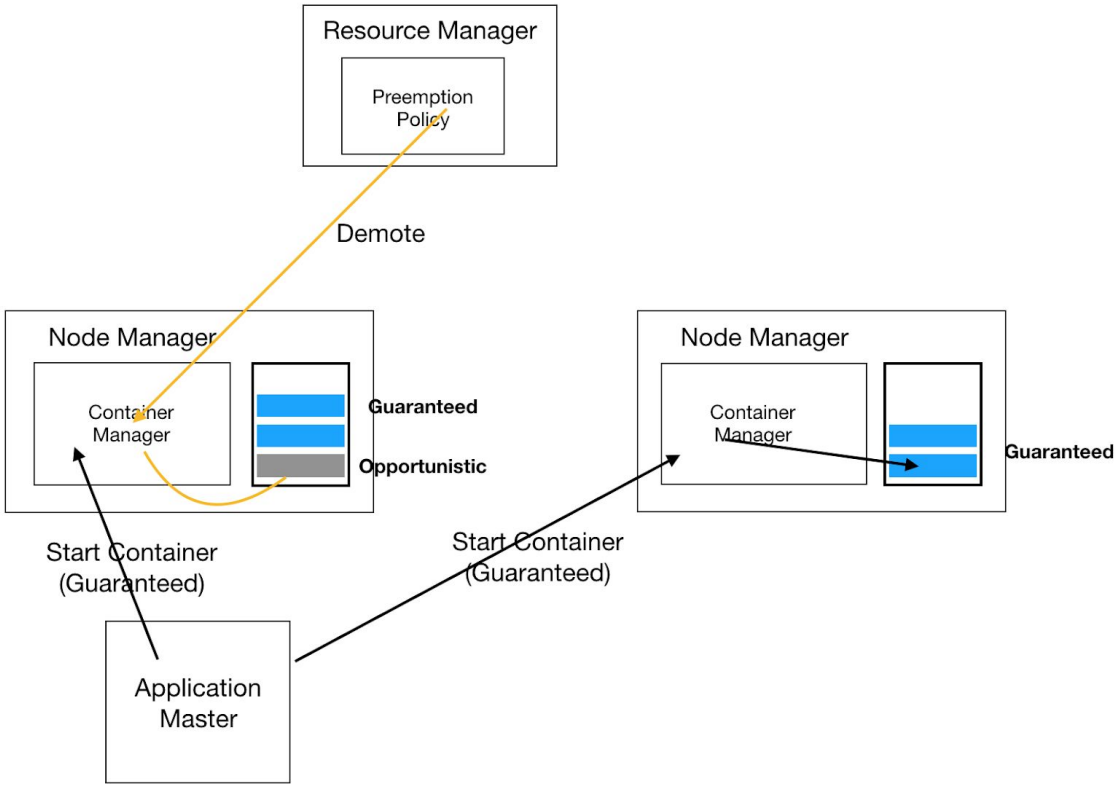

Figure 1 Architecture

Implementation Details

In this section we give the implementation details of the system design we proposed above. Each subsection corresponds to an sub JIRA of the JIRA YARN-8178.

Change preemption logic to demote guaranteed container to opportunistic rather than kill it

Currently, opportunistic-containers feature is disabled by default. When enabled, RM/AM will allocate a mix of guaranteed and opportunistic container whenever an application comes in. In our design, we are trying to allocate all containers as guaranteed-containers in the first place, and let high demanding queue A "borrow" resources from resource sufficient queues B. When queue B have application submitted in and need the "borrowed" resource back, we then mark

whose containers who took queue B's resource, and demote them from guaranteed-container to opportunistic. The whole process is depicted in Figure 2.

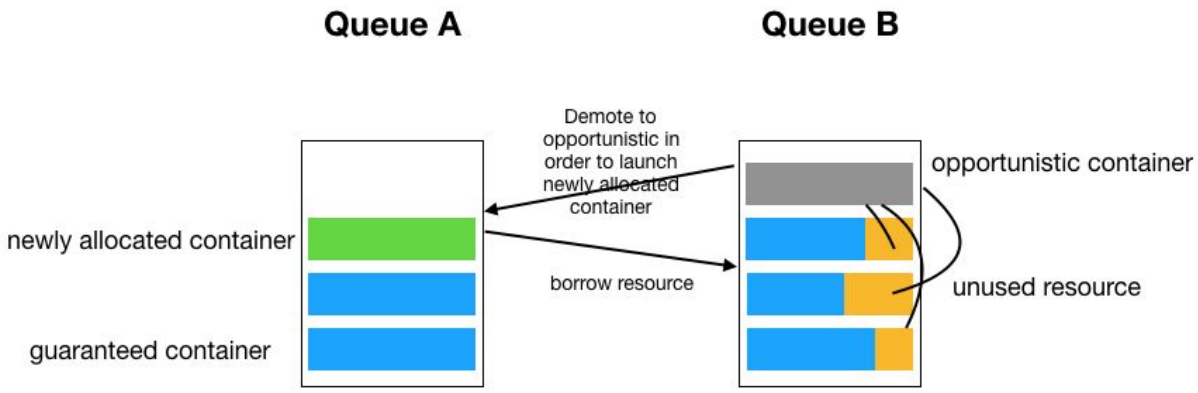

Figure 2 opportunistic container based preemption

Make more aggressive preemption interval

This defines how Capacity Scheduler can perform aggressive preemptions when opportunistic container is enabled. We do this by setting smaller timeout / preemption monitor interval: from 3 seconds to less than 1 seconds. Once the container demoted (with very short latency), guaranteed container can be allocated/launched if we reclaim the right resources.

At every invocation of *editScheduler()*, it first figures out those overutilized queues which consume underutilized queue resource. Calculate the amount of resources underutilized queue needed to run its pending applications and then dispatch these resource request into those overutilized queues in a rate fair manner. Where the rate is the percentage of the guaranteed capacity each queue consumed from underutilized queue. Based on this assignment it determines which queue it should perform preemption on and select a set of containers from each application that should demote from guaranteed to opportunistic.

Maintain rate of opportunistic containers based on utilization of current containers

When we mark some containers from guaranteed to opportunistic, we still allow these containers to continue use resources and running their applications, the only different after marking it into opportunistic is that, we "tell" those opportunistic container the resources they are using will not be guaranteed, we will kill them immediately to free up resources in order to launch new guaranteed containers if we get new applications been submit into these queues

where those opportunistic containers located. So the problem we are facing now is where are these resource which can be used by these containers after they are been demoted from guaranteed to opportunistic? The logic here is we use cgroup to monitor the current actual utilization of resources of current guaranteed containers in these queues and calculate the difference of these container's actual utilization and their allocated resources in RM point of view. these unused allocated resource will be used to support further execution of those opportunistic containers.

Based on the discussion of resource utilization, we take different types of resources into account, discuss as below,

- **CPU & Memory** collect cpu/memory usage of the node and add ResourceUtilization as part of NodeReport and NodeInfo has been done by YARN-1011 . We will reuse most of the effort implemented in this Umbrella JIRA.

- **Other resource types (GPU, FPGA, etc)** These resource type need to be supported as part of ResourceUtilization report to RM which haven't been done yet. We will extend YARN-3534 to collect these resource type utilization metrics and support reasonable resource isolation to ensure the execution of guaranteed containers to be minimally affected

Add a configurable strategy to force kill O containers after configured delays

Since our current solution is not considering promote container from opportunistic to guaranteed. We may encounter a large amount of opportunistic containers as time goes by. The main drawback of maintaining too many opportunistic containers is maintain too many applications as running state without make their current consuming resource "GUARANTEED" cause these opportunistic containers might be killed unexpectedly at any time. We address these issue like below,

- Instead of maintaining these containers as opportunistic constantly until next possible preemption happens, we force kill these opportunistic containers after configured delays. This delays can be set as a configuration property inside yarn-site.xml named as " yarn.resourcemanager.monitor.capacity.preemption.force_kill_opportunistic_container_timeout"
- Time in milliseconds between the starting time when a container is demoted from guaranteed to opportunistic and the final time when the preemption-policy forcefully kills the container. By default, preemption-policy will wait for 30 minutes before a forceful-kill of containers. Administrators can adjust this parameter to a smaller value if he/she wants to reclaim resource back faster or set it to a higher value if he/she wants more gradual reclamation of resources.

Implementation Comparison to YARN-1011

Since YARN-1011 is also support resource over-subscription[1] based on opportunistic container, we would like to compare with the implementation of YARN-1011 and our design here to show the major difference of these two solutions.

Although both our YARN-1011 and our design are using opportunistic container to support over-subscription in order to further improve resource utilization with the cluster, the original intention of these two solutions are distinct which makes the implementation differ from each other. The major difference are shown below,

	YARN-1011	Our Solution
Trigger point	**Trigger by AM resource request:** over-subscription is triggered by application master sending resource request to RM. RM will need to get resource utilization report from NMs to make accurate decision for allocating opportunistic containers	**Triggered by preemption:** we only need to change the current preemption logic to demote guaranteed container to opportunistic when preemption timeout reached and resource is insufficient to launch guaranteed containers.
Container Allocation Strategy	**Allocate mix of guaranteed and opportunistic container:** when new application is been submitted to RM, RM will evaluate the amount of containers this application needs and allocate a mix of **Guaranteed and Opportunistic containers** based on the resource utilization metrics.	**Allocate guaranteed container only:** in our design, all the application submitted to the queue will get its required resource by guaranteed container only.

Since our design is focus on using preemption to achieve over-commitment, we will bring relatively less implementation changes to the current code base. Which will provide better backward compatibility.

In fact, our design is not aiming at replace YARN-1011, these two solutions are complementary. Since YARN-1011 supports resource utilization reports to RM, this information will help our design make more accurate decision on determine which guaranteed containers can be preempted to achieve better fairness among these task queues within the cluster. In the interest of putting together a usable version to address more aggressive preemption and over-commitment, this JIRA aims to implement the simplest version of over-commitment.

References

[1] https://issues.apache.org/jira/secure/attachment/12874299/yarn-1011-design-v3.pdf