Intra-Queue Preemption Use Cases

1. Ensure each user in a queue is guaranteed its appropriate `minimum-user-limit-percent`

The `minimum-user-limit-percent` property is used by the capacity scheduler to allow a set number of users to run apps in a queue at the same time and to guarantee resources will be assigned fairly to those users. However, if one or more users begin running containers in a queue that consume resources for a long period of time, new apps launched by a new user can be starved of its guaranteed resources.

1.1. When one (or more) user(s) are below their `minimun-user-limit-percent` and one (or more) user(s) are above their `minimum-user-limit-percent`, resources will be preempted after a configurable time period from the user(s) which are above their `minimum-user-limit-percent`.

1.1.1. Example:

Queue Name	Queue Guaranteed Resources	Queue Maximum Resources	`minimum-user-limit-percent`	`user-limit-factor`
`root`	100	100	N/A	N/A
`QUEUE1`	100	100	0.5 (guarantees that 2 different users can use 50% of Queue1 at the same time)	1.0 (allows one user to use all of QUEUE1)

1.1.1.1. `User1` starts `App1` which consumes 100 resources in `QUEUE1`. `App1` has launched long-running containers which hold on to their resources for a very long time and will not release them in a timely manner. The time it takes to preempt these containers is governed by the intra-queue config for natural termination.

Queue Name	User Name	App Name	Resources Consumed by User	Resources Guaranteed to User by `minimum-user-limit-percent`	Pending Resource Requests for User
`QUEUE1`	`User1`	`App1`	100	50	0

1.1.1.2. User2 starts `App2` and is asking for 50 resources

Queue	User Name	App	Resources	Resources Guaranteed	Pending

Name		Name	Consumed by User	to User by minimum-user-limit-percent	Resource Requests for User
QUEUE1	User1	App1	100	50	0
QUEUE1	User2	App2	0	50	50

1.1.1.3. The user-limit-percent-intra-queue-preemption policy selects 50 resources from App1 to be preempted. The scheduler will reassigned them to App2.

Queue Name	User Name	App Name	Resources Consumed by User	Resources Guaranteed to User by minimum-user-limit-percent	Pending Resource Requests for User
QUEUE1	User1	App1	50	50	50
QUEUE1	User2	App2	50	50	0

1.2. When two (or more) users are below their minimum-user-limit-factor, neither will be preempted in favor of the other.

1.2.1. Example:

Queue Name	Queue Guaranteed Resources	Queue Maximum Resources	minimum-user-limit-percent	user-limit-factor	Is Queue Preemptable?
root	200	200	N/A	N/A	No
QUEUE0	100	200	1.0 (one user is guaranteed all of QUEUE0)	2.0 (one user can consume double QUEUE0's resources)	No
QUEUE1	100	200	0.5 (guarantees that 2 different users can use 50% of Queue1 at the same time)	1.0 (allows one user to use all of QUEUE1)	Yes

1.2.1.1. User0 starts App0 which consumes 150 resources in QUEUE0. These are long-running containers that will not complete for a long time.

Queue Name	User Name	App Name	Resources Consumed by User	Resources Guaranteed to User by minimum-user-limit-percent	Pending Resource Requests by App
QUEUE0	User0	App0	150	100	0

QUEUE1	N/A	N/A	0	0	0

1.2.1.2. User1 starts App1 in QUEUE1 and consumes its minimum-user-limit-percent: 50 resources.

Queue Name	User Name	App Name	Resources Consumed by User	Resources Guaranteed to User by minimum-user-limit-percent	Pending Resource Requests by App
QUEUE0	User0	App0	150	100	0
QUEUE1	User1	App1	50	50	0

1.2.1.3. User2 starts App2 in QUEUE1, requesting 50 resources.

Queue Name	User Name	App Name	Resources Consumed by User	Resources Guaranteed to User by minimum-user-limit-percent	Pending Resource Requests by App
QUEUE0	User0	App0	150	100	0
QUEUE1	User1	App1	50	50	0
Queue1	User2	App2	0	50	50

1.2.1.4. App0 in QUEUE0 is consuming 50 extra resources and is not releasing them. But, since QUEUE0 is not preemptable, inter-queue preemption will not free the 50 resources from QUEUE0. In QUEUE1, User2 is guaranteed 50 resources, but User1 is also, and User1 is not going over its minimum-user-limit-percent, so User1 is not violating any policy. Therefore, the user-limit-percent-intra-queue-preemption policy does not preempt any resources from User1.

1.3. If all users in a queue are at or over their minimum-user-limit-percent, the user-limit-percent-preemption policy will not preempt resources.

1.3.1. Example:

Queue Name	Queue Guaranteed Resources	Queue Maximum Resources	minimum-user-limit-percent	user-limit-factor
root	200	200	N/A	N/A
QUEUE1	100	200	0.5 (guarantees that 2 different users can	2.0 (allows one user to use twice the

	use 50% of QUEUE1 at the same time)	resources of QUEUE1)

1.3.1.1. `User1` starts `App1` in `QUEUE1` which consumes 150 resources. `App1` has launched long-running containers which hold on to their resources for a very long time and will not release them in a timely manner.

Queue Name	User Name	App Name	Resources Consumed by User	Resources Guaranteed to User by `minimum-user-limit-percent`	Pending Resource Requests for User
QUEUE1	User1	App1	150	50	0

1.3.1.2. `User2` starts `App2` in `QUEUE1` and is asking for 100 resources. Since only 50 more resources are available in the cluster, those 50 are assigned to `App2`.

Queue Name	User Name	App Name	Resources Consumed by User	Resources Guaranteed to User by `minimum-user-limit-percent`	Pending Resource Requests for User
QUEUE1	User1	App1	150	50	0
QUEUE1	User2	App2	50	50	50

1.3.1.3. The user-limit-percent-intra-queue-preemption policy does not preempt any resources from `App1` even though `App1` is consuming 100 more resources than it is guaranteed because both users are at or above their `minimum-user-limit-percent`.

1.3.1.3.1. TBD: I could see this going the other way. That is, we may want the policy to preempt from `App1` until both users have an equal number of resources. The reason we may want that behavior is for consistency with the scheduler, because if I manually kill 50 resources from `App1`, the capacity scheduler will give them to `App2`, which will bring both of them to an equal number of 100 resources. However, I chose the above behavior because that minimizes the amount of lost work.

2. Ensure priority inversion doesn't occur between applications.

Applications can be assigned a YARN priority that orders application within a queue. The scheduler will use this priority to determine the order in which applications within a queue are assigned containers. However, a priority inversion can occur when a lower priority application has begun running, consumed resources in the queue with long-running containers, and then a higher priority wants to run in that queue.

2.1. When a lower priority app is consuming long-running resources, a higher priority app is requesting resources, and the queue cannot grow to accommodate the higher priority app's request, the priority-intra-queue-preemption policy will preempt resources from the lower priority app, after a configurable period of time.

2.1.1. Example:

Queue Name	Queue Guaranteed Resources	Queue Maximum Resources
root	100	100
QUEUE1	100	100

2.1.1.1. `User1` launches `App1` of priority 1 and consumes 100 resources for a long time.

Queue Name	User Name	App Name	Resources Consumed by User	Pending Resource Requests for User	App Priority
QUEUE1	`User1`	`App1`	100	0	1

2.1.1.2. `User1` launches `App2` of priority 2 and requests 100 resources.

Queue Name	User Name	App Name	Resources Consumed by User	Pending Resource Requests for User	App Priority
QUEUE1	`User1`	`App1`	100	0	1
QUEUE1	`User1`	`App2`	0	100	2

2.1.1.3. The priority-intra-queue-preemption policy preempts 100 non-AM containers from `App1` to be allocated to `App2`.

Queue Name	User Name	App Name	Resources Consumed by User	Pending Resource Requests for User	App Priority
QUEUE1	`User1`	`App1`	0	100	1
QUEUE1	`User1`	`App2`	100	0	2

2.1.1.3.1. TBD: Should there be some (possibly configurable) limit on the percent of containers preempted from `App1`? I guess that would be very complicated because the policy would have to remember that it had already preempted the max allowable, and keep track of that across preemption iterations. I guess that's not a good idea, but I still don't like the fact that we are losing all of the work done by the running containers from `App1`.

3. Interaction between the priority and minimum-user-limit-percent preemption policies.

3.1. If priority inversion occurs between apps owned by different users, the priority preemption policy will not preempt containers from the lower priority app if it would cause the lower priority app to go below the user's `minimum-user-limit-percent` guarantee.

3.1.1. Example:

Queue Name	Queue Guaranteed Resources	Queue Maximum Resources	Minimum-user-limit-percent
root	100	100	N/A
QUEUE1	100	100	0.5 (guarantees that 2 different users can use 50% of QUEUE1 at the same time)

3.1.1.1. `User1` launches `app1` of priority 1 that consumes 100 resources.

Queue Name	User Name	App Name	Resources Consumed by User	Pending Resource Requests for User	App Priority	mimim-user-limit-percent Guarantee
QUEUE1	User1	App1	100	0	1	50

3.1.1.2. `User2` launches `App2` at priority 2 that requests 90 resources.

Queue Name	User Name	App Name	Resources Consumed by User	Pending Resource Requests for User	App Priority	mimim-user-limit-percent Guarantee

QUEUE1	User1	App1	100	0	1	50
QUEUE1	User2	App2	0	90	2	50

3.1.1.3. The priority-preemption policy preempts 50 resources from `App1` and stops there because preempting more would cause `User1` to go below its `minimum-user-limit-percent` guarantee of 50.

Queue Name	User Name	App Name	Resources Consumed by User	Pending Resource Requests for User	App Priority	mimim-user-limit-percent Guarantee
QUEUE1	User1	App1	50	50	1	50
QUEUE1	User2	App2	50	40	2	50