December 21, 2017

YARN – The Capacity Scheduler

By Joseph Niemiec

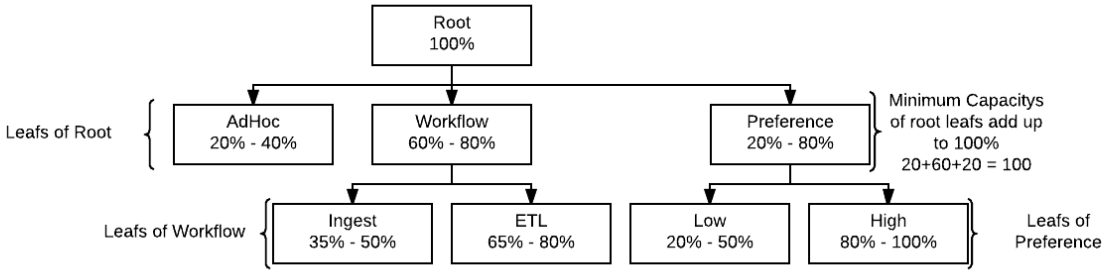
Apache Yarn

Understanding the basic functions of the YARN Capacity Scheduler is a concept I deal with typically across all kinds of deployments. While Capacity Management has many facets from sharing, chargeback, and forecasting the focus of this blog will be on the primary features available for platform operators to use. In addition to the basic features some gotcha's will be reviewed that are commonly ran into when designing or utilizing the queues.

Capacity and Hierarchical Design

YARN defines a minimum allocation and a maximum allocation for the resources it is scheduling for: Memory and/or Cores today. Each server running a worker for YARN has a NodeManager that is providing an allocation of resources which could be memory and/or cores that can be used for scheduling. This aggregate of resources from all Node Managers is provided as the 'root' of all resources the capacity scheduler has available.

The fundamental basics of the Capacity Scheduler are around how queues are laid out and resources are allocated to them. Queues are laid out in a hierarchical design with the topmost parent being the 'root' of the cluster queues, from here leaf (child) queues can be assigned from the root, or branches which can have leafs on themselves. Capacity is assigned to these queues as min and max percentages of the parent in the hierarchy. The minimum capacity is the amount of resources the queue should expect to have available to it if everything is running maxed out on the cluster. The maximum capacity is an elastic like capacity that allows queues to make use of resources which are not being used to fill minimum capacity demand in other queues.



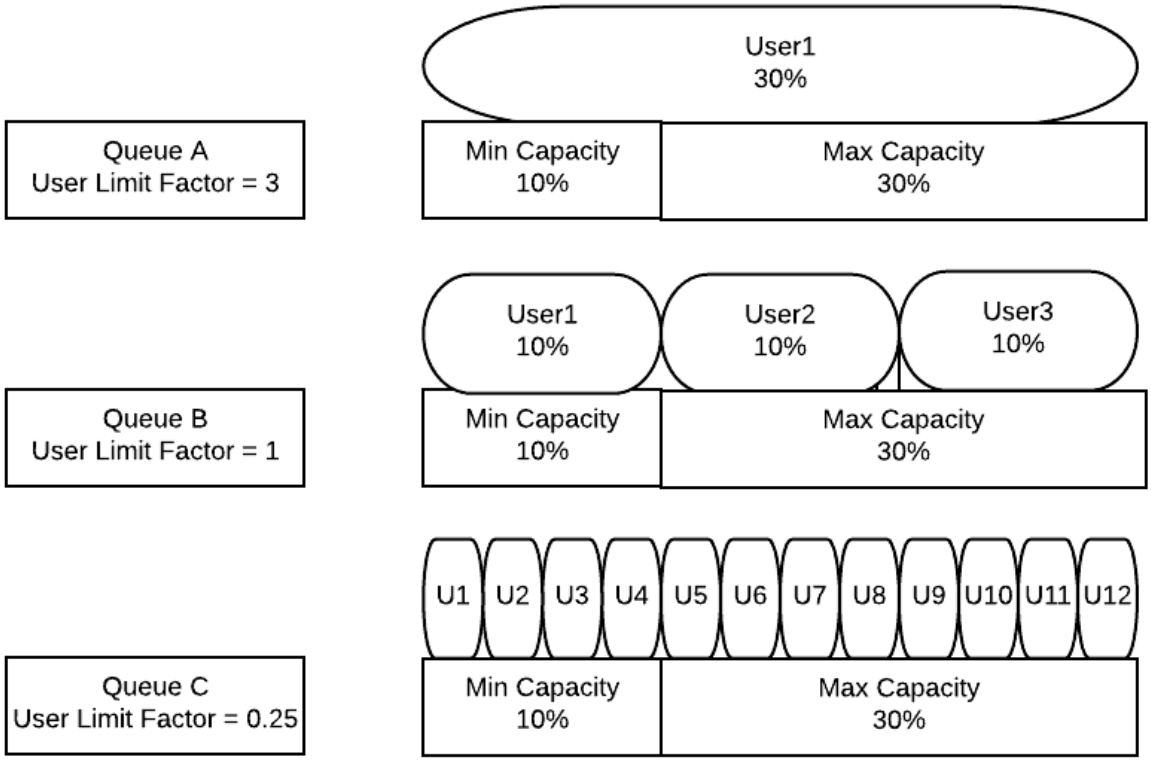
Children Queues like in the figure above inherit the resources of their parent queue. For example, with the Preference branch the Low leaf queue gets 20% of the Preference 20% minimum capacity while the High lead gets 80% of the 20% minimum capacity. Minimum Capacity always has to add up to 100% for all the leafs under a parent.

Minimum User Percentage and User Limit Factor

Minimum User Percentage and User Limit Factor are ways to control how resources get assigned to users within the queues they are utilizing. The Min User Percentage is a soft limit on the smallest amount of resources a single user should get access to if they are requesting it. For example a min user percentage of 10% means that 10 users would each be getting 10% assuming they are all asking for it; this value is soft in the sense that if one of the users is asking for less we may place more users in queue.

User Limit Factor is a way to control the max amount of resources that a single user can consume. User Limit Factor is set as a multiple of the queues minimum capacity where a user limit factor of 1 means the user can consume the entire minimum capacity of the queue. If the user limit factor is greater than 1 it's possible for the user to grow into maximum capacity and if the value is set to less than 1, such as 0.5, a user will only be able to obtain half of the minimum capacity of the queue. Should you want a user to be able to also grow into the maximum capacity of a queue setting the value greater than 1 will allow the minimum capacity to be

overtaken by a user that many times.



A common design point that may initially be non-intuitive is creation of queues by workloads and not by applications and then using the user-limit-factor to prevent individual takeover of queues by a single user by using a value of less than 1.0. This model supports simpler operations by not allowing queue creation to spiral out of control by creating one per LoB but by creating queues by workloads to create predictable queue behaviors. Once a cluster is at full use and has applications waiting in line to run the lower user-limit-factors will be key to controlling resource sharing between tenants.

Initially this may not provide the cluster utilization at the start of their Hadoop platform journey due to using a smaller user limit to limit user resources, there are many approaches but one to consider is that initially it may be justified to allow a single tenant to take over a tiny cluster (say 10 nodes) but when expanding the platform footprint lower the user limit factor so that each tenant keeps the same amount of allocatable resources as before you added new nodes. This lets the user keep his initial experience, but be capped off from taking over the entire cluster as it grows making room for new users easily to get allocated capacity while not degrading the original user's experience.

Archetypes

Designing queue archetypes to describe effective behavior of the tenant in the queue provides a means to measure changes against to see if they align or deviate from the expectation. While by no means a complete list of workload behaviors the below is a good list to start from. Create your own definitions of the types of queues your organization needs as the types of patterns applications and end users are performing.

AD-HOC
: This is where random user queries, unknown, and new workloads may be ran, there is no expectation as to resource allocation behaviors but can work as a good place to initially run applications to get a feel for per application tuning needs.

PREFERENCE
: These are applications that should get resources before and retain them longer. This could be for many reasons such as catch up applications, emergency runs, or other operational needs.

MACHINE LEARNING
: Machine Learning applications can be typically characterized by their long run times and large or intensive resource requirements. Termination of a task for some machine learning workloads can have long running impacts by greatly extending the duration

DASHBOARDING
: Low concurrency (refresh rate) but high number of queries per day. Dashboards need to be refreshed in an expected time but are very predictable workloads

EXPLORATION
: Exploration users have a need for low latency queries and need throughput to churn through very large datasets. Resources will likely be held for use the entire time the user is exploring to provide an interactive experience

BATCH WORKFLOW
: Designed to provide general computing needs for transformation and batch workloads. Setup is concerned most frequently with throughput of applications not individual application latency

ALWAYS ON
: Applications that are always running with no concept of completion. Applications that keep resources provisioned while waiting for new work to arrive. Slider deployed instances such as LLAP

Container Churn

Churn within a queue is best described as a constant existing and starting of new containers. This behavior is very important to have a well behaving cluster were

queues are quickly rebalanced to their minimum capacity and to balance capacity of the queue between its users fairly. The anti-pattern to churn is the long lived container that allocates itself and never releases as it can prevent proper rebalancing of resources in some cases it can completely block other applications from launching or queues from getting their capacity back. When not using preemption if a queue was to grow into elastic space but never release its containers the elastic space will never be given back to queues that have been guaranteed it. If a application running in your queues has long lived containers make special note and consider ways to mitigate its impact to other users with features like user-limit-factors, preemption, or even dedicated queues without elastic capacity.

Features & Behaviors of the Capacity Scheduler

CPU Scheduling (Dominant Resource Fairness)

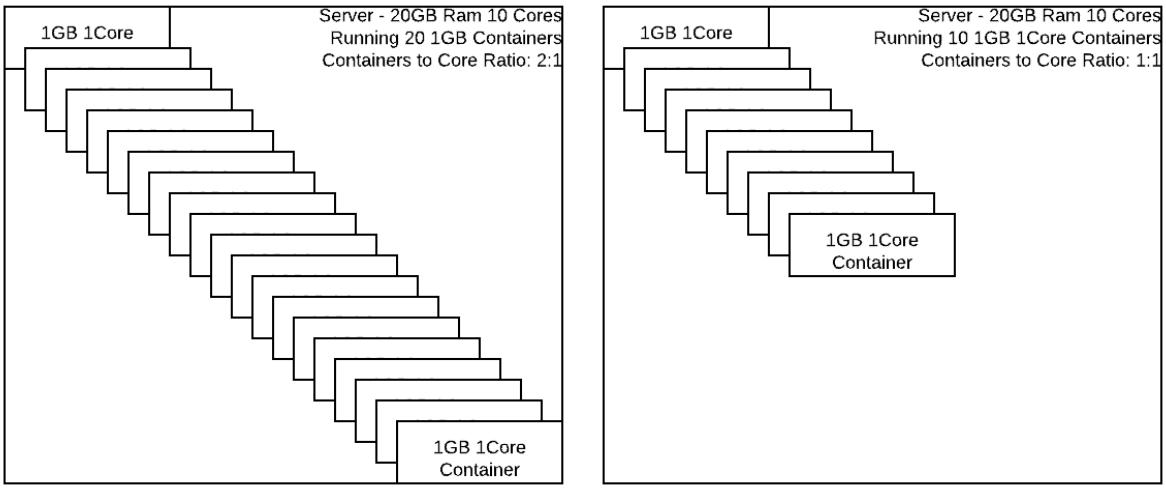
CPU Scheduling is not enabled by default and allows for the oversubscription of cores with no enforcement of use or preferred allocation taken into account. Many batch driven workloads may experience a throughput reduction if CPU scheduling is used, but may be required for strict SLA guarantees. A method called Dominant Resource Fairness (DRF) is used to decide what resource type to weight the utilization as: DRF takes the most used resource and treats you as using the highest percent for scheduling.

There are 2 primary parts to CPU Scheduling

1. Allocation and Placement
2. Enforcement

Allocation and placement is solved simply by enabling CPU Scheduling so that the Dominant Resource Fairness algorithm and VCores Node Manager's report begin getting used by the scheduler. This solves for some novel problems like an end user that used to schedule his Spark applications with 1 or 2 executors but 8 cores per, and then all other tasks that ran on these nodes due to the excess of available memory were impacted. By enabling simple CPU Scheduling other tasks would no longer be provisioned onto the servers were all the cores are being utilized and find other prefered locations for task placement in the cluster.

Enforcement is solved by utilizing CGroups. This allows for YARN to ensure that if a task asked for a single vcore that they cannot utilize more than what was requested. Sharing of vcores when not used can be enabled or a strict enforcement of only what was scheduled can be done. The Node Manager's can also be configured as to what the max amount of CPU use on the server they will allow all tasks to sum up to, this allows for cores to be guaranteed to OS functions.



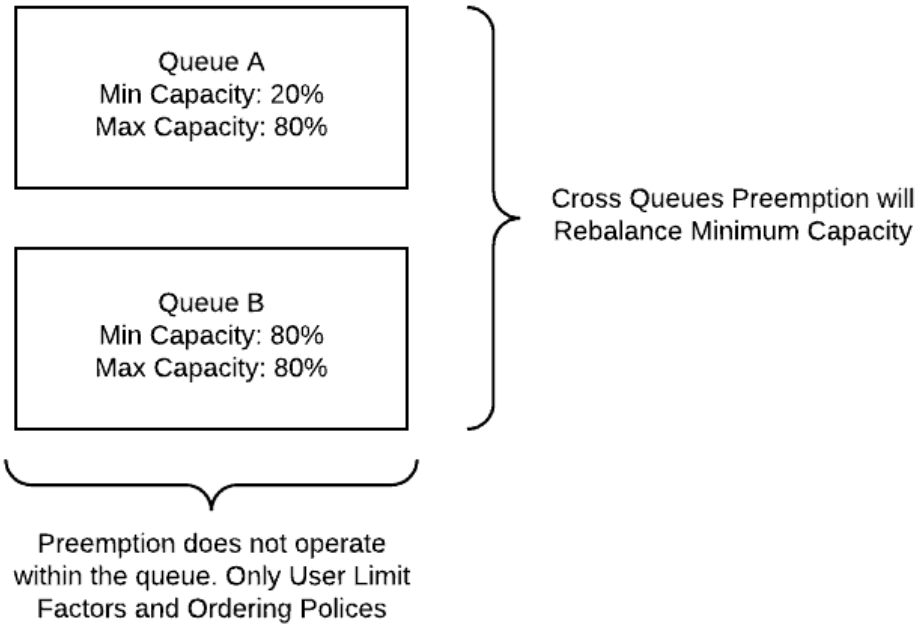
The above figure gives an idea of how many concurrent containers can change if limited to the smallest resource (typically CPU cores.) It's unlikely a true 1:1 Core to Container ratio is what will be required but this aspect of tuning is best left to monitoring historical system metrics and then increasing or decrease the number of NodeManager VCores available for scheduler to allow more or less containers on a given server group.

Preemption

When applications are making use of elastic capacity in their queues and another application comes along to demand back their minimum capacity (which is being used in another queue as elastic) traditionally the application would have to wait for the task to finish to get its resource allocations. With Preemption enabled resources in other queues can be reclaimed to provide the minimum capacity back to the queue that needs it. Preemption will try not to outright kill a application, and will take reducers last as they have to repeat more work then mappers if they have to re-run. From an ordering perspective Preemption looks at the youngest application and most over-subscribed ones first for task reclamation.

Preemption has some very specific behaviors and some of them don't function as expected for users. The most commonly expected behavior is for the queues to preempt within themselves to balance the resources over all users. That

assumption is wrong as preemption only works across queues today, resources that are mis-balanced within a queue between users needs to look into other ways to control this such as User Limit Factors, Improved Queue Churned, and Queue FIFO/FAIR Policies.



Another behavior is that Preemption will not preempt resources if it cannot provide enough to fulfill another resource allocation request. While typically this is not an issue with large cluster for small clusters with large maximum container sizes could run into a scenario were Preemption is not configured to reclaim a container of the largest possible size and therefore will do nothing at all. The main properties used to control this behavior is the Total Preemption Per Round and the Natural Termination Factor. Total Preemption Per Round is the percentage of resources on the cluster that can be preempted at once, the Natural Termination Factor is the percent of resources out of the total cluster (100%) requested that will be preempted up to the Total Preemption Per Round.

The last misconception is that Preemption will rebalance max (elastic) capacity use for between queues that wish to grow into it. Maximum Capacity is given on a first come first serve basis only. If a single queue has taken over all the of the clusters capacity, and another application start in a queue that needs its minimum capacity back, only the minimum capacity will be preempted and all of the

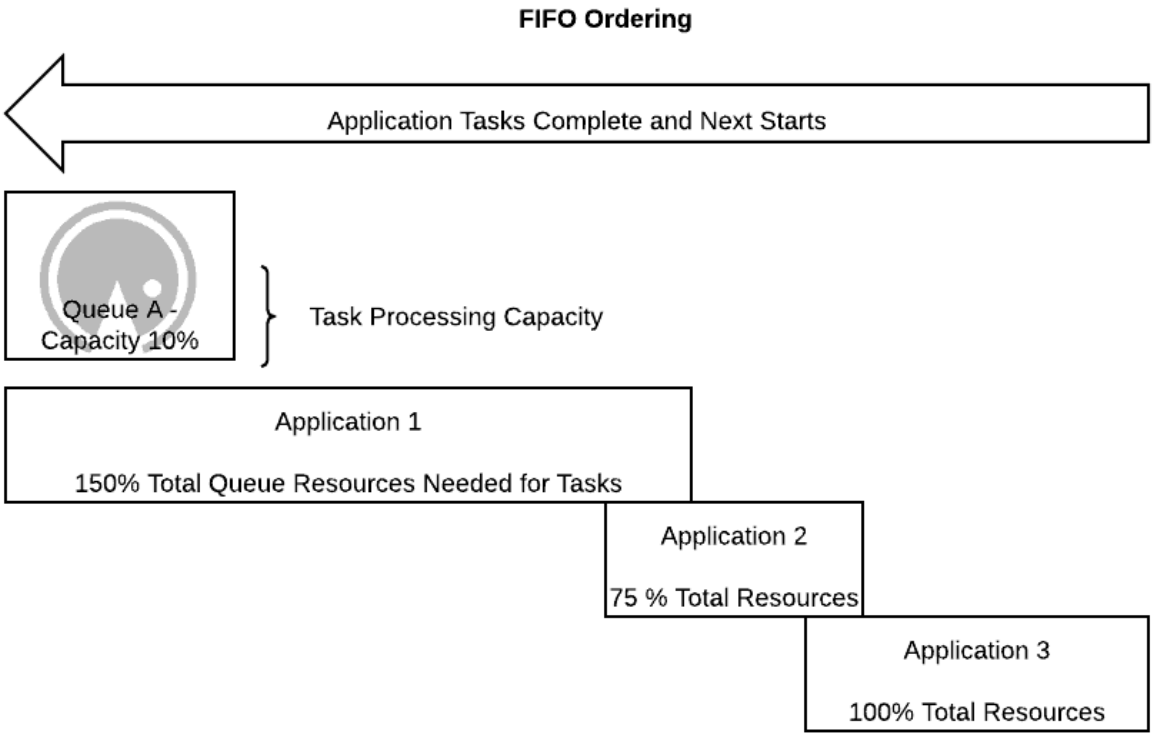
maximum capacity being used by the other queue will remain until the containers churn naturally.

For more details on configuring Preemption please see the blog here https://hortonworks.com/blog/better-slas-via-resource-preemption-in-yarns-capacityscheduler/

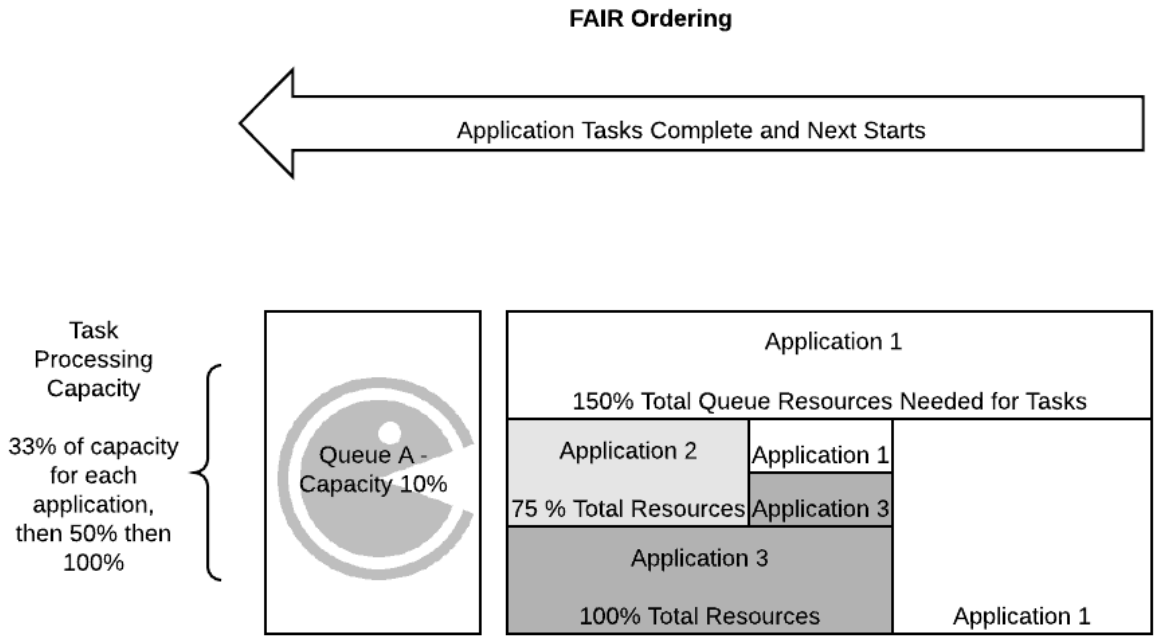
Queue Ordering Policies

There are currently two types of ordering policies the Capacity Scheduler supports today: FIFO and FAIR. The default that queues start with is FIFO which in my experience is not the behavior that clients expect from their queues. By configuring between FIFO and FAIR at the Queue Leaf level you can create behaviors that lead to throughput driven processing or shared fair processing between applications running. One important thing to know about ordering policies is that they operated at the application level in the queue and do not care about which user owns the applications.

With the FIFO policy applications are evaluated in order of oldest to youngest for resource allocation. If an application has outstanding resource requests they are immediately fulfilled first come first serve. The result of this is that should an application have enough outstanding requests that they would consume the entire queue multiple times before completing they will block out other applications from starting while they are first allocated resources as the oldest application. It is this very behavior that comes most unexpected to users and creates the most discontent as a users can even block their own applications with their own applications!



That behavior is easily solved for today by using the FAIR ordering policy. When using the FAIR ordering policy on a leaf queue applications are evaluated for resource allocation requests by first the application using the least resources first and most last. This way new applications entering the queue who have no resources for processing will be first asked for their required allocations to get started. Once all applications in the queue have resources they get balanced fairly between all users asking for them.



It's important to note that this behavior only occurs if you have good container churn in your queue. Because Preemption does not exist inside of a queue resources cannot be forcefully redistributed within it and the FAIR ordering policy only is concerned with new allocations of resources not current ones; what does this mean? That if the queue is currently utilized with tasks which never complete or run for long periods of time without allowing for container churn to occur in the queue will hold the resources and still prevent applications from executing.

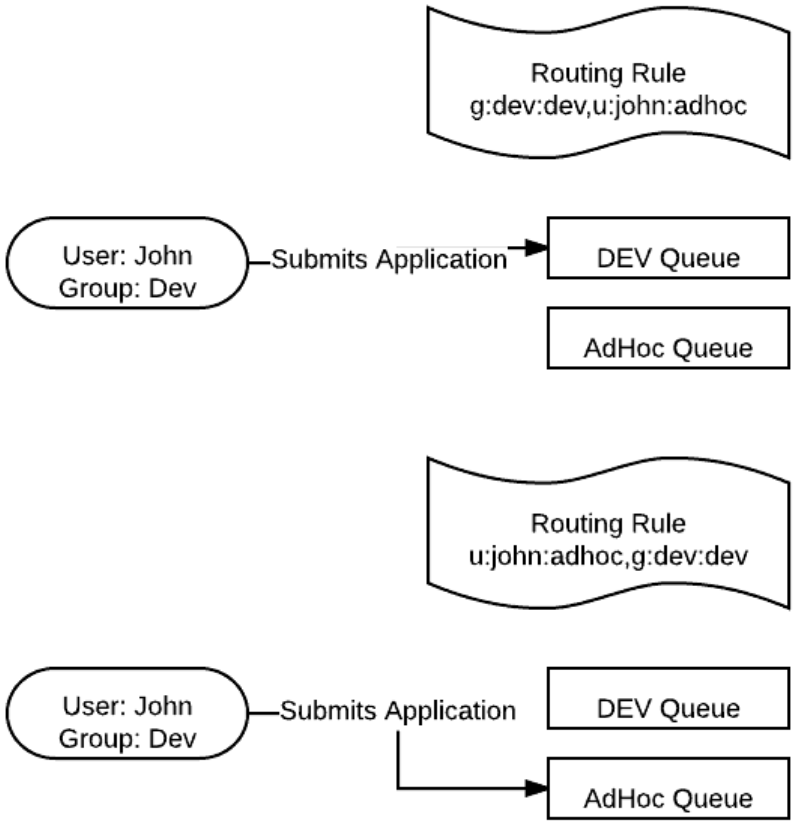
Username and Application Driven Calculations

Calculations in the Capacity Scheduler look at two primary attributes when attempting to provide allocations: User Name and Application ID. When it comes to sharing resources between users in a queue aspects like Minimum User Percentage and User Limit Factor both look at the User Name itself; this can obviously cause some conflicting problems if you're using a service account for multiple users to run jobs as only 1 user will appear to the Capacity Scheduler. Within a Queue which application gets resource allocations are driven by the leaf queues ordering policies: FIFO or FAIR which only care about the application and not which user is running it. In FIFO resources are allocated first to the oldest application in the queue and only when it no longer requires any will the next application gets an allocation. With FAIR applications that are using the least about of resources are first asking if there are pending allocations for them and

fulfilled if so, if not the next application with least resources is checked; this helps to evenly share queues with applications that would normally consume them.

Default Queue Mapping

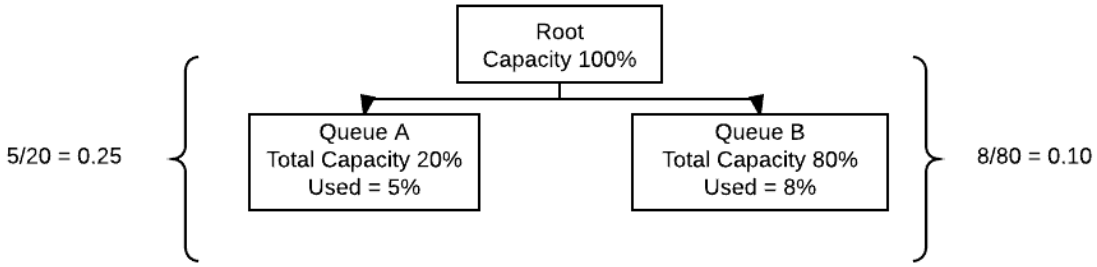
Normally to target a specific queue the user provides some configuration information telling the client tools what queue to request. But commonly users utilize tools that have a hard time passing configurations downstream to target specific queues. With Default Queue Mappings we can route an entity by its username or groups it belongs to into specific queues. Take note that the default queue routing configuration matches whichever routing attribute comes first. So if a group mapping is provided before the user mapping that matches the user he is routed to the queue for the group.



Priority

When resources are allocated over multiple queues the one with the lowest relative capacity gets resources first. If you're in a scenario where you want to have a high priority queue that receives resources before others then changing to

a higher priority is a simple way to do this. Today using queue priorities with LLAP and Tez allows for more interactive workloads as these queues can be assigned resources at a higher priority to reduce the end latency that an end user may experience.

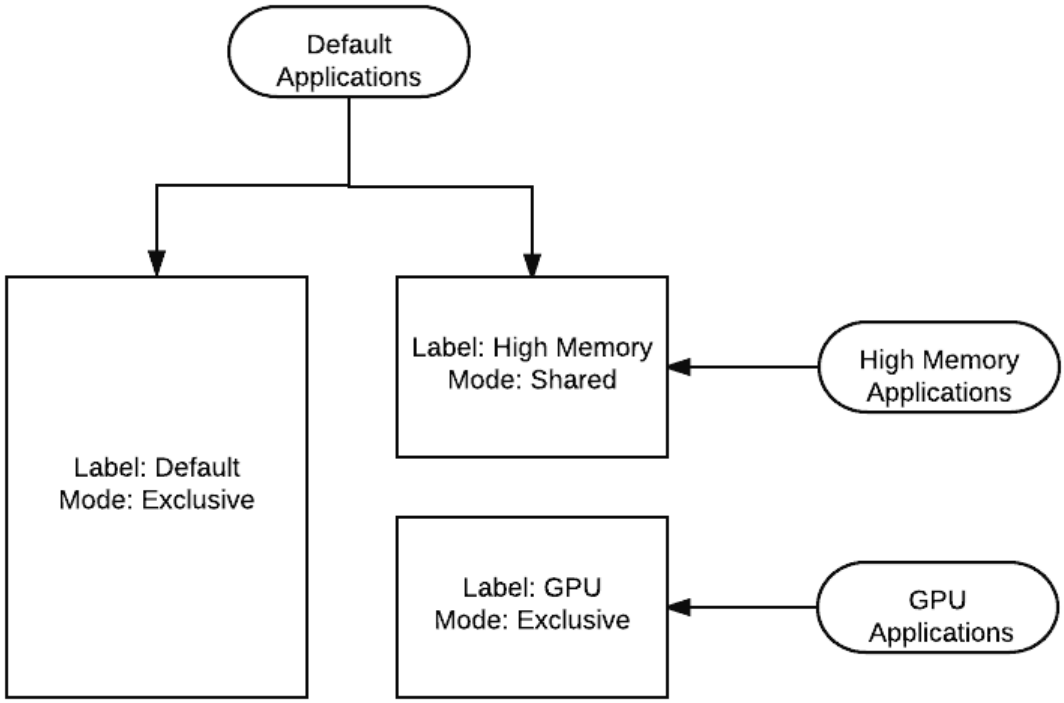


Above the two queues provide an example of how relative capacity is used unmodified by queue priorities. In this case even while Queue A is smaller than Queue B and while Queue B is using more absolute resources it is elected to continue receiving them first because its relative capacity is lower than that of Queue A. If the desired behavior required Queue A to always receive resource allocations first the Queue priority should be increased above that of Queue B. When assigning priority a higher value represents higher priority.

Labels

Labels are better describe as partitions of the cluster. All clusters start with a default label or partition that is exclusive in the sense that as new labeled partitions are added to the cluster they are not shared with the original default cluster partition. Labels be defined as exclusive or shared when created and a node can only have a single label assigned to it. More common uses of labels is for the targeting of GPU hardware in the cluster or to deploy licensed software against only a specific subset of the cluster. Today LLAP also uses Labels to leverage dedicated hosts for long running processes.

Shared labels allow applications in other labeled partitions such as the default cluster to grow into them and utilize the hardware if no specific applications are asking for the label. If an application comes along that targets the label specifically the other applications that were utilizing it will be preempted off the labeled node so the application needing it can make use of it. Exclusive labels are just that, exclusive and will not be shared by any others; Only applications specifically targeting labels will run solely on them. Access to labeled partitions is provided to leaf queues so users able to submit to them are able to target the label. If you wish to autoroute users to labels say creating a GPU queue that automatically uses the GPU label this is possible.



Queue Names

Queue leaf names have to be unique with the Capacity Scheduler. For example if you created a queue in the capacity scheduler as root.adhoc.dev dev will have to be unique as a leaf over all queue names and you cannot have a root.workflow.dev queue as it would no longer be unique. This is in line with how queues are specified for submission by only using the leaf name and not the entire composite queue name. Parents of leafs are never submitted to directly and have no need to be unique so you can have root.adhoc.dev and root.adhoc.qa without issue as both dev and qa are unique leaf names.

Limiting Applications per Queue

Spawning of a queue by launching many applications into it so that none can effectively complete can create bottlenecks and impact SLAs. At the very worst the entire queue becomes deadlocked and nothing is able to process without and admin physically killing jobs to free up resources for compute tasks. This is easily prevented by placing a limit on the total number of applications allowed to run in the leaf queue, alternatively it's possible to control the % of resources of the leaf that can be used by Application Masters. By default this value is typically rather

large over 10,000 applications (or 20% of leaf resources) and can be configured per leaf if needed otherwise the value is inherited from prior parent queues of the leaf.

Container Sizing

An unknown to many people utilizing the Capacity Scheduler is that containers are multiples in size of the minimum allocation. For example if your minimum scheduler mb of memory per container was 1gb and you requested a 4.5gb sized container the scheduler will round this request up to 5gb. With very high minimums this can create large resource wastage problems for example with a 4gb minimum if we requested 5gb we would be served 8gb providing us with 3 extra GB that we never even planned on using! When configuring and minimum and maximum container size the maximum should be evenly divisible by the minimum.