Proposal of Time Extended Fair Scheduling for YARN

Wei Shao@Rocket Fuel Inc

Last modified: June 19 2015

Introduction

This proposal talks about the issues of YARN fair scheduling policy, and tries to solve them by YARN-3806 and the new policy time extended fair scheduling.

YARN Fair Scheduling Overview

(Note: Readers who are already familiar with YARN fair scheduling can skip this section)

Fair scheduling maximizes the minimum allocation received by the application queue in the cluster. Assuming each queue has enough demand, this policy gives each queue a share of the resources proportional to its fair weight.

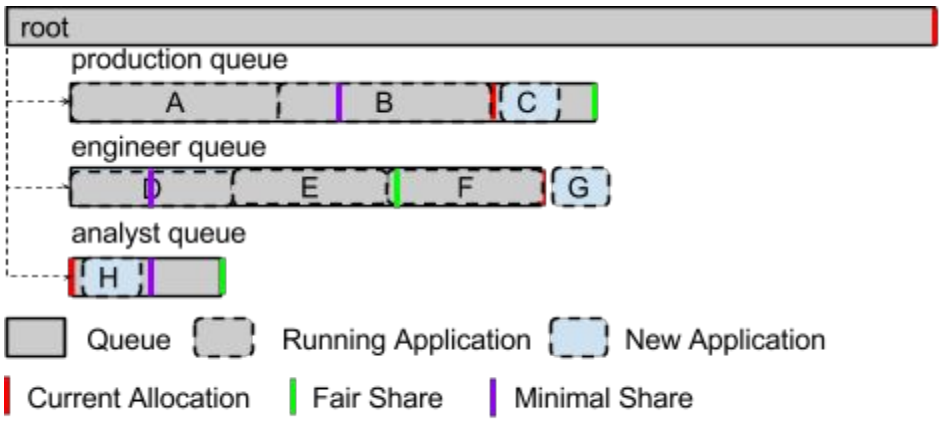

Figure 1

Figure 1 is an simplified instance of fair scheduling model. The cluster has 1000G RAM (For simplification, CPU is not considered here). There are three queues in the cluster, production, engineer and analyst. For production queue, current allocation is 450G, fair share is 600G (fair share is calculated based on fair weight, minimal share and so on), and minimal share is 250G. For engineer queue, current allocation is 550G, fair share is 300G, and minimal share is 50G. For analyst queue, current allocation is 0, fair share is 100G, and minimal share is 50G.

Assuming the cluster is fully utilized, by fair scheduling policy, when a queue has new demands - it accepts a new application which requires resources or one of its running application asks for additional resource:

1. If current allocation is below minimal share, the queue will get resource immediately (or after configured preemption timeout) by preemption to satisfy new demands. For example, the new application H of analyst queue. And in this case, the resource in engineer queue will be preempted since its current allocation is above fair share.

2. If current allocation is above minimal share and below fair share, the queue has the priority to get next available resource to satisfy new demands. For example, the new application C of production queue. Available resources could come from the application which release containers or from new servers added to the cluster, and so on.

3. if current allocation is above fair share, the queue has lower priority than the queues in second case to get available resource. The queue can only get resource after all demands of queues in second case are satisfied. For example, the new application G of engineer queue.

4. The queue will always allocate resource to the the application with least share. For example, when production queue obtains an available resource, application C has highest priority to get the resource, since application C has least share among A, B and C.

Issues of YARN Fair Scheduling

YARN fair scheduling enforces static weight based fairness among the queues. However, it is not fair from other perspectives illustrated below:

1. It is not fair for applications entering queue in different situations: When the cluster is fully utilized, if the current allocation of the queue is below minimal share, the new application can get the resources immediately by preemption. If the resource allocation of the queue is above minimal share and below fair share, the new application has priority to get next available resource, but the allocation rate is unpredictable and certainly slower than the first case. Finally, if the resource allocation of the queue is above fair share, the new application can get resources only after all demands of other queues whose allocations are below fair share are satisfied, it is even slower.
   This issue can be solved by application level configuration which is supported by generic scheduling framework proposed in YARN-3806. In detail, if applications in same queue have positive minimal share, preemption will help to enforce fairness among applications.

2. It is not fair for users: The fair scheduling base unit is application, if user A submit more applications than user B, user A can get high resource share than user B which is unfair from perspective of users. So the user can try to divide workload into multiple applications or simply submit duplicate applications in order to get application done

sooner. Currently, one of our cluster is heavily loaded, some engineers complains that their ad hoc hive query applications often get killed or progress slow, so they sometimes try to submit multiple same applications.
This issue can be solved by fair scheduling configured with one user per queue enforcement which is supported by generic scheduling framework proposed in YARN-3806.

3. It is not fair if resource allocation time is considered: YARN fair scheduler only enforces instant fairness. It will try to even the resource allocations for each application at every time instant. So if user A submits applications more frequently than user B in a long time period, user A can get higher resource share than user B which is unfair. For example, user A may submits one application once an hour, user B may submit one application once a day. When two applications of users A and B are running at the same time, fair scheduler will try to even the resource share of applications of A and B, although user A may already be allocated many resources before. So users could submit applications more frequently than necessary to get higher share of cluster.
This issue can be solved by time extended fair scheduling policy proposed here.

In summary, with fair scheduling, users won't get benefit by lying about their resource demands, and the cluster resources are utilized effectively since they are users' actual demands. On the other hand, if scheduling is unfair, users could submit more applications than necessary. And if every users do so, the cluster will get overloaded, and resources are underutilized since they aren't users' actual demands. Overall, users will focus on how to better utilize resources instead of manipulating scheduler if scheduling is fair, and the cluster will be utilized more effectively.

Time Extended Fair Scheduling

(Note: please read YARN-3806 before continuing, time extended fair scheduling policy is extended on generic scheduling framework proposed there.)

Time extended fair scheduling policy is proposed to solve third issue mentioned above. It enforces time extended fairness among users. For example, if two users share the cluster weekly, each user's fair share is half of the cluster per week. At a particular week, if the first user has used the whole cluster for first half of the week, then in second half of the week, second user will always have priority to use cluster resources since the first user has used up its fair share of the cluster already.

YARN-3806 proposed a generic interface that particular scheduling policy should implement. To introduce time extended fair scheduling, we just talk about its own scheduling configurations and the implementation of generic interface.

Scheduling Configurations/Parameters: There are three configuration items and one additional parameter for each sub queue.

1. fairShare or fairWeight. FairShare is used in the calculation of accumulatedAllocation which determines the user's priority to get free resources. If fairWeight is configured, the corresponding fairShare is dynamic and will be calculated at runtime. Suppose the parent queue use fair scheduling policy, then follow constraints should be satisfied.

$$\forall\, i, j \in configurations\ of\ children\ who\ has\ fairWeight\ configured, \quad \frac{i.fairShare}{i.fairWeight} = \frac{j.fairShare}{j.fairWeight}$$

   and

$$queue.configurations.fairShare \geq \sum_{child\ who\ has\ static\ fairShare} child.configurations.fairShare + \sum_{child\ who\ has\ dynamic\ fairShare} child.configurations.fairShare$$

2. timePeriod and startTime. Each cycle start at

$$startTime + timePeriod * n, \quad n \in \mathbb{N}$$

   For example, one user can specify timePeriod to one week and startTime to June 14 2015.

3. maximalInstantShare. The resources allocation cap enforced at any time instant for the user.

4. accumulatedAllocation. The resource usage in each cycle. The formula is:

$$subQueue.configurations.accumulatedAllocation = \frac{\sum_{each\ completed\ container\ allocated\ by\ the\ sub\ queue} container.resources * container.runningTime}{subQueue.configurations.fairShare * subQueue.configurations.timePeriod}$$

   And it will be reset to 0 at the start of every cycle. The user with smallest accumulatedAllocation has highest priority to get free resource. But the scheduling policy won't guarantee the users whose accumulatedAllocation <= 1 won't be preempted.

Implementation of Generic Scheduling Interface: We only talk about implementations of functions with non trivial implementations.

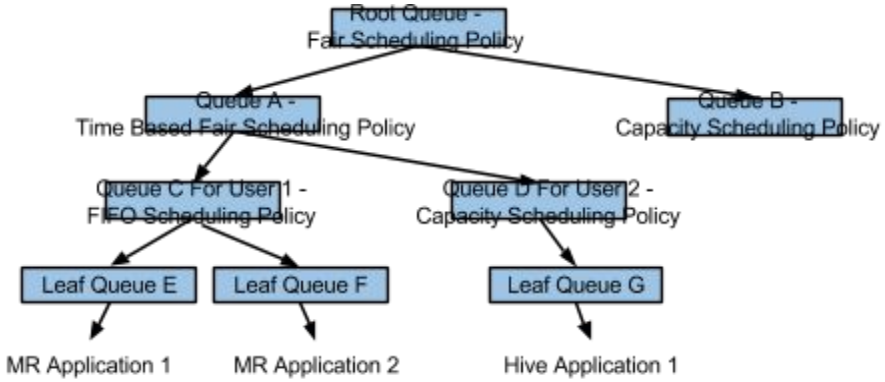

Figure 2

1. dispatchApplications(). In figure 2, suppose MR application 2 submitted by user 1 and hive application 1 submitted by user 2 are in pendingApplicationList of queue A, If dispatchApplications() of queue A selects MR application 2 to run, it will create leaf queue F for the application under queue C which is already created for user 1 before. If dispatchApplications() selects hive application 1 to run, it will create queue D for user 2 first and then create leaf queue G for the application under queue D. The user queue can use different scheduling policies to manage its applications. And it won't be destroyed until the end of the cycle, configuration items like accumulatedAllocation should be maintained in entire cycle.

2. updateCapacities(). Preemption demands for all children are 0. However, this doesn't mean the users cannot get resources by preemption. For example, in figure 2, queue A uses time extended fair scheduling. Since root queue use fair scheduling, queue A has configuration minimalShare. So if allocation of queue A is below minimalShare, it can ask root queue to preempt resources from neighbor queue B, and distribute these preempted resources to users. If capacity < allocation for parent queue, resources of any user could be preempted. Distribution should maximize:

   $$\min_{userQueue} userQueue.schedulingConfigs.accumulatedAllocation$$

   Also, maximalInstantShare should be respected, the user whose current allocation reaches maximalInstantShare don't need additional capacity.

Future Work

Write prototype to evaluate proposed scheduling policy.

Related Efforts

YARN-3806 Proposal of Generic Scheduling Framework for YARN
YARN-3807 Proposal of Guaranteed Capacity Scheduling for YARN