Summary

More than one container is allocated on many NM heartbeats. Yet, the current scheduler allocates exactly one container per iteration over {queues, applications, priorities}. When there are many queues, applications, or priorities allocating only one container per iteration can needlessly increase the duration of the NM heartbeat.

In this JIRA, we propose bundling. That is, allow arbitrarily many containers to be allocated in a single iteration over {queues, applications and priorities}.

Design

While iterating, we propose to maintain a bundle which represents the best possible multi-container assignment to a node. The tasks in the bundle are updated as the scheduler's loop progresses. A new task can be concatenated to the bundle or replace some preexisting tasks in the bundle. The actual allocation is done after the loop finishes. The bundling logic allows for early termination of the loop. And has a flag that indicates whether rerunning the loop may be useful.

A bundler that subsumes current scheduling logic

Note that this subsumes the current scheduling logic. That is, the current scheduling logic is akin to (1) storing in the bundle the first task that satisfies all checks and (2) terminating the loop after (3) setting the flag that indicates the loop be retried.

Several more interesting bundling policies are possible. For example, the bundling logic can be a straight win in terms of computational overhead if multiple containers are picked in one iteration.

But first some constraints on bundlers.

Constraints on bundlers

Let a bundling opportunity denote an ask being considered for potential scheduling just because of the bundling logic. The opposite case when an ask would have been considered for scheduling without bundling is called a clean opportunity.

We have the following constraints for bundlers:

* To not affect delay scheduling logic, bundlers update scheduling opportunity counts only on a clean opportunity.
* To not affect reservations, bundlers should (a) create reservations only on clean opportunities and (b) allocate reserved containers as before.
* To not delay assignment of new AM containers, if an AM container can be allocated on a clean opportunity bundlers will early terminate the iteration and allocate to such AM container.
* Respect several existing checks such as
  + Queue and User resource limits
  + Node labels
* Relax some existing checks
  + Fairness across queues: currently, based on resorting queues upon every assignment. We call this instantaneous fairness. Instead, a bundler could offer bounded unfairness. For example, by using deficit round robin~(DRR) style accounting over multiple resources.
  + FiFo ordering: currently, applications within a queue are scheduled in FiFo order. Bundlers may relax this if needed.

A bundler that picks multiple containers per iteration

With these constraints, it is easy to see a bundling policy that does multi-container assignments.

While iterating, perform all checks as before. Satisfy reservations as before. When a container would have been allocated, that is some ask satisified all checks, add the container to the bundle instead. Defer allocation and keep iterating. Subsequently, do not make any changes to scheduling opportunity counters. When subsequent allocable containers are found, concatenate to the current bundle. Terminate when the node has no more resources to allocate.

A bundler that picks containers out-of-order to facilitate packing

Further bundler policies include: one that does packing. IE, picks containers that may not necessarily be the first discovered on the iteration if those containers are better at avoiding resource fragmentation.

Future work

This is an architectural change needed to facilitate some of the changes in Yarn-2745.

Our goal is to offer a bundler that supports multi-resource packing while also preferring shorter jobs (to lower average job completion time), offering bounded unfairness and supporting general DAGs of computation (packing + scheduling of Tez/Hive style jobs).

Credits

Robert Grandl

Chris Douglas

Srikanth Kandula