SLAs in YARN: predictable resource allocation

Carlo Curino, Chris Douglas, Subru Krishnan, Sriram Rao

Motivation

Currently the YARN RM focuses on job throughput and use instantaneous notion of capacity/fairness for resource sharing. Consequently, applications see substantial variability in when their resource demands will be met as it varies with the load on the cluster during its execution, making it difficult to manage jobs with deadlines or dependencies. To support goals such as deadlines or pipelined execution (DAGs), tenants and operators collaborate on the configuration of mechanisms that guide resource allocations to future, globally favorable conditions. Job priorities, static queue hierarchies, over-provisioning, and frequent manual intervention provide a loose approximation of tenants' intent. Hence to address the issue of providing self-service SLA on a multi-tenant or cloud setting, we need to provide a mechanism for guaranteeing resource allocation.

The key intuition is to allow users to specify their resource needs and time/gang/dependency constraints ahead of time, and provide early feedback about whether the system can fulfill this request or not.

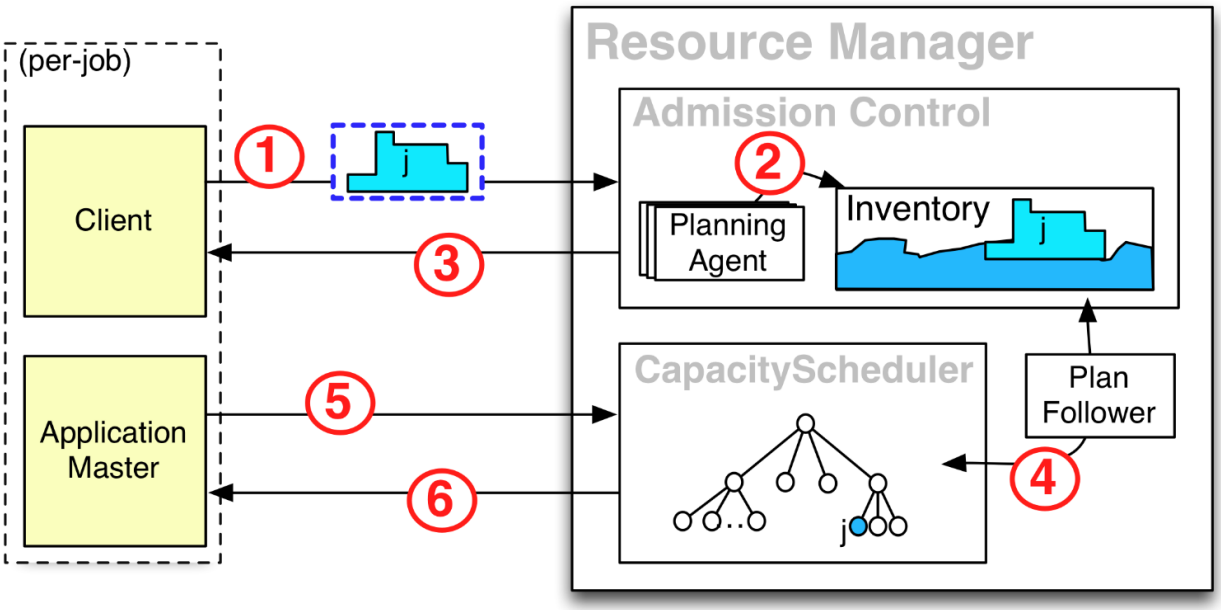
High Level Design

Presently the notion of resource allocation be it capacity or fairness is instantaneous. The crux of our approach is to enable users to reserve resource over time to execute their job (or pipeline of jobs).

This provides the RM with an understanding of future resource demand, and exposes jobs time and resource constraints, hence enabling the RM to look ahead and plan resource allocation over time. The plan is akin to a time-dependent allocation of capacity to jobs/pipelines.

The design involves projecting resource demand over time onto dynamic queues in the existing scheduler henceforth referred to as session queues. This achieves the twin benefits of allowing it to run side-by-side existing setups and also leveraging all the effort that has been put into the schedulers.

Figure 1 illustrates the sequence of operations:



1. The client will specify its session, i.e. resources it requires projected over the timelines to ensure SLA is satisfied. This is basically an extension of the existing resource request with the added dimension of time, for e.g.: I need 100 containers for 2 hours at any time before 3pm today.
2. The agent/admission control will try to find a suitable slot by reconciling against the cluster capacity and previously accepted sessions.
3. If it is able to find a slot that satisfies the SLA constraints of the client, it will return a session id. If not the client request will be rejected.
4. At the time when the session needs to be activated, the plan follower component will create an appropriately sized session queue, i.e. a dynamic queue with session id as the name in the scheduler. Overtime the plan follower can also increase/decrease size of queue according to the plan, and ultimately destroy queues when the corresponding session is expired.
5. The client/AM can then submit resource requests to the YARN just like it does today by specifying the session id as the queue name.
6. The scheduler issue containers as usual.

Enforcement of quotas is provided by means of pluggable policies, which allow to tradeoff flexibility of allocation with time-extended semantics for capacity. For example we will be able to say that a user or a group have rights in average to at most 25% of the cluster every 24h, but he/she can request 50% of the cluster for 12h and nothing afterwards. This provides the equivalent of a self-service API for time-bounded queues. Our design is compatible with existing queues, and it can be run side-by-side managing only a fraction of the overall capacity. As a byproduct, the design allows clients to skip the session creation step and directly submit to existing queues as is done today so that current behavior is maintained for clients that do not require SLA.

Implementation Details

The changes needed to achieve this are:

- Enhancing the RM to include a notion of ***inventory*** of cluster capacity by tracking future reservations. The inventory is a repository of all accepted user sessions that are currently active or will be in the future. The first version of the inventory will be an in-memory variant of interval trees that will allow efficient indexing based on time intervals. The inventory will be pluggable in order to support custom inventories in future. Sub-JIRA: YARN-1709.
- Every inventory will also support pluggable ***policies*** to prevent abuse. We will provide a simple policy in the first version that will enforce user capacity quota over time. Sub-JIRA: YARN-1711.
- Adding an additional client API for creating a session in order to reserve resources. This involves ***resource request*** to be enhanced to have a notion of lease duration and gang size as the mechanism enables support for gang scheduling which is not currently possible. Both these fields are optional and the default value for lease duration will be infinite and gang size would be 1 to maintain backward compatibility. Sub-JIRA: YARN-1708.
- A new admission control module called ***agent*** that handles a user's resource reservation requests by searching for allocations of the request in the cluster resource inventory. If a placement is found that does not violate previously accepted sessions, the reservation is accepted, and a session id is returned to the user. On the other hand, if the resource reservation cannot be satisfied by the current inventory either due to unavailability of resources or due to violation of the user quota, the reservation is rejected and an exception is thrown with information on the reason for rejection. Multiple jobs can be submitted to a session which is common in practice for running workflows. The initial version of the agent will be greedy that will perform a linear search on the inventory and return the first allocation it finds. The agent will also be pluggable to allow swapping in more optimal or specialized agents like a LP solver or one based on economic models in future. Sub-JIRA: YARN-1710.
- A new component called ***plan follower*** that runs periodically that synchronizes the inventory with the scheduler. This is done by creating a session queue for each currently active session, and sizing it based on the allocation in the inventory. In essence, our design provides predictable resource allocation to jobs by dynamically reconfiguring the scheduler queues. Sub-JIRA: YARN-1709.
- Enhance the ***capacity scheduler*** to efficiently perform dynamic creation/reconfiguration/destruction of queues. Sub-JIRA: YARN-1707.
- We will also provide simple policies to handle scenarios in which the cluster resources are reduced from when sessions have been accepted due to unforeseen scenarios like multiple machines or rack failure, and thus some of our promises must be violated.

The goal is to provide the complete set of mechanisms, and a set of initial policies, allowing for further evolution.

Failover

The inventory will be maintained in-memory for performance reasons to minimize the overhead of the system. To support the failover mechanism introduced in YARN-149, the inventory will be persisted to the RM state store asynchronously. The number of entries will be proportional to the number of jobs as a session encompasses either a single job or a pipeline of jobs which is aligns with the existing failover architecture. On failover the inventory will be recreated from the state store following which plan follower will perform a reset, i.e. delete all the session queues from the scheduler and re-sync from the inventory by creating session queues for currently active sessions. Post reset it will continue syncing periodically as usual.