Adding Support for Recurring Reservations in YARN Reservation System

Sangeetha Abdu Jyothi, Subru Krishnan, Carlo Curino, Ishai Menache

Motivation

YARN-1051 introduced reservation mechanism for jobs[1] with deadlines. Currently, the YARN reservation system provides support for non-periodic jobs (i.e. single instance) only. However, periodic jobs (recurring instances with identical resource requirements separated by a constant time interval) constitute a significant fraction of jobs in today's cluster environments. Under the current reservation mechanism, a separate reservation is required for each instance of a periodic job. For example, consider an hourly job that arrives at the beginning of every hour and lasts for 15 min. Under the current system, a new reservation request has to be generated for every instance (24 requests per day). This leads to significant overhead in reservation time and storage. In addition, this mechanism fails to guarantee that all instances of a job would be accepted, leading to unpredictable performance for the user. To overcome this issue, we propose adding native support for periodic jobs within the reservation system.

Proposal

The high level proposal is to modify the internal implementation of the YARN reservation system to provide seamless support for periodic jobs

Implementation Details

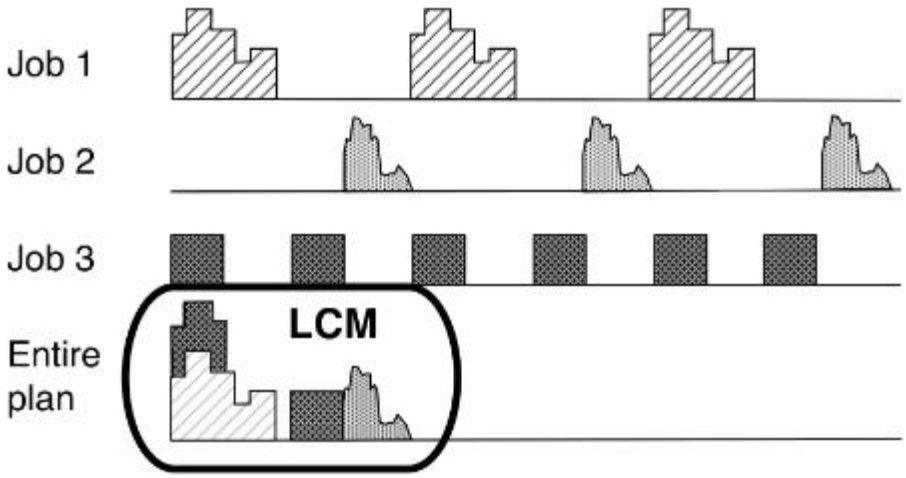
Periodicity support needs to be incorporated at all stages of the reservation system.

1) The API has to be extended to include a period associated with periodic reservation requests. The reservation is valid until it is cancelled explicitly.

2) *InMemoryPlan* will be extended to store periodic reservations in a separate data structure. It is inefficient to calculate and store a separate allocation for each instance of a periodic reservation. To address this challenge, we force the constraint that all instances associated with the same periodic reservations would have the same allocation across runs. Having a fixed offset for each periodic reservation produces a repeating pattern in the overall allocation of all periodic reservations. This also has the advantage of providing consistency

[1] we use jobs to represent both individual jobs as well as workflows

across job executions to the user. It is then sufficient to identify and store the smallest repeating unit which can accurately capture this recurring pattern in the set of all periodic reservations. The length of this unit will be the Least Common Multiple (LCM) of all periodic reservations as illustrated below. We propose to use PeriodicRLESparseResourceAllocation class to compactly store periodic jobs. PeriodicRLESparseResourceAllocation is a simple extension of RLESparseResourceAllocation with a time period. The time period is set to the LCM of all periodic reservations in the system.



3) Internal *Plan* APIs will transparently compose the sum of periodic and non-periodic reservations at any instant. *PlanFollower* will rely on these APIs to retrieve the resource usage at a given instant.

4) *ReservationAgent* will be updated to handle periodicity building on the optimizations that will be added as a part of YARN-4359.

5) *SharingPolicies* will be updated to handle periodic reservations.
    a) *NoOvercommitPolicy* will be modified to consider the maximum of sum of periodic and non-periodic allocations at all instances while placing a new job.
    b) *CapacityOverTimePolicy* for periodic reservations can be easily computed among periodic only reservations, enforcing COTP on mix periodic / non-periodic requires further analysis.

6) Integration with failover -- The implementation will be compatible with the failover mechanism added to *ReservationSystem* as part of YARN-2573.

Future work

1) When priority support (YARN-5211) is incorporated, priorities will be valid only within reservation classes -- with periodic reservations having strictly higher priority than non-periodic reservations.

2) Individual executions of a periodic job may be misaligned with respect to the reservation. When the progress of an instance differs from the reserved allocation due to delayed arrival of data or slow nodes in the cluster, it is imperative to modify the allocation to accommodate the requirements temporarily. In future, we would like to extend the system by adding renegotiation mechanisms for tackling temporary variations in individual instances.