YARN-1011. Resource Oversubscription

Karthik Kambatla, Haibo Chen
May 30, 2017

Motivation

Yarn allocates resources based on the resource availability across nodes in the cluster. Today, a node's resource availability is calculated as its "capacity" (determined by yarn.nodemanager.resource.*) minus the resources currently allocated to containers on the node. A container's allocation is the upper limit of resources the container could use, and is based on the application's request.

In practice, users are conservative with resource estimates: (1) to avoid task failures or (2) because they are using higher-level abstractions (e.g. Hive) and are unaware of the resource requirements of generated MR/Tez/Spark jobs.

Proposal

Allocate the allocated-but-unused resources opportunistically to pending requests by using opportunistic containers (introduced in YARN-2882). This leads to oversubscribing the node: the node can accommodate allocations which in aggregate exceed the capacity of the node. The extent of oversubscription should be configurable.

When containers simultaneously grow in usage on a node that is sufficiently oversubscribed, the tasks in these containers could slow down severely and even fail. To avoid this, we propose preempting opportunistic containers if the node utilization crosses a configurable threshold.

Constraints

While oversubscribing helps with better cluster utilization, we need to keep the following constraints/goals in mind:

1. A worker node should not fall over due to oversubscription.
2. The execution of guaranteed containers should be minimally affected. This translates to (1) prompt launch of allocated guaranteed containers and (2) reasonable isolation of their resources (cpu and memory).
3. Cluster admins should be able to turn on oversubscription for apps, without the need for changes to the application. E.g. MR AM should not require changes.
4. Applications should be able to opt of opportunistic containers.
5. Nice to have: The oversubscribed resources should be distributed among applications in the same proportion as guaranteed resources.

Implementation

Identifying the opportunity

Opportunistic containers are to be allocated when a node is fully allocated but the resource utilization is under a configurable threshold (YARN-4512). We call this the over-allocation threshold (T_{alloc}), which is a vector of values between 0 and 1 where each dimension represents a type of resource. We propose to send this threshold to the RM on the node heartbeat along with the aggregate container usage information.

Configs to add (YANR-6670):

- yarn.nodemanager.overallocation.memory.allocation-threshold: Node memory utilization threshold upto which the scheduler allocates OPPORTUNISTIC containers.
- yarn.nodemanager.overallocation.cpu.allocation-threshold: Node CPU utilization threshold upto which the scheduler allocates OPPORTUNISTIC containers.

Scheduling opportunistic containers

We propose to allocate opportunistic containers for oversubscription per the queue policies. That is, allocate containers in the same proportion as the capacity/fairshare of the queue (YARN-1013 and YARN-1015).

Today, on node heartbeat, the scheduler iterates over pending containers in prioritized order and allocates as many guaranteed containers as it can. We propose supplementing this with a second iteration to allocate opportunistic containers. This way, we trigger opportunistic allocation only after fully allocating the node.

Opportunistic allocation on the node is bounded by T_{alloc}.

Launching Opportunistic Containers

Today, we already have the notion of opportunistic containers that are launched on the node if there is unallocated space. When oversubscription is enabled, we propose launching these containers even when there is *allocated-but-unused* space (YARN-6675). Hence, the condition for launching an opportunistic container becomes (1) unallocated space or (2) utilization < preemption-threshold (details on this threshold later).

The current approach of queuing opportunistic containers should continue to work well.

Avoiding Adverse Effects of Oversubscription

With oversubscription, opportunistic containers use resources already allocated to guaranteed containers. The scheduler allocates opportunistic resources based on reported resource usage.

However, resource usage of containers - both guaranteed and opportunistic - changes over time. If left unchecked, opportunistic containers can interfere with the execution of guaranteed containers. Depending on the type of resources, this can lead to container slow down (cpu, network etc.) or even failure (memory). In extreme cases of oversubscription, the node itself might fail.

To avoid that, we propose checks at multiple levels:

Preemption

Introduce a preemption-threshold ($T_{preempt}$), also a vector with each dimension representing a resource. The node preempts enough opportunistic containers to keep the utilization under this threshold (YARN-6672). For malleable resources like cpu, it might be okay for the node utilization to go over the preemption-threshold momentarily. We could consider introducing a second config that determines the number of subsequent checks to fail to trigger preemption. This is similar to our current procfs-based memory limit enforcement.

Note that preempting containers when resource usage goes over a threshold is a reactive approach. The effectiveness of this approach depends on how soon we realize the utilization is over the threshold. In our prototype, we find that computing aggregate container utilization by iterating over each containers' utilization (as determined by procfs parsing) can take upto ~10 seconds.

A higher gap between T_{alloc} and $T_{preempt}$ would lower the chances of oversubscription becoming untenable and hence the need to preempt opportunistic containers.

Cgroups (when using LCE)

When using LCE, cgroups provide a more active approach to enforcing resource usage.

- CPU (YARN-6673): In addition to setting `cpu.cfs_period_us` on a per-container basis, we could also set `cpu.shares` to 2 for opportunistic containers so they are run on a best-effort basis.
- Memory :
    - Soft limit (YARN-6674): When a node is under memory pressure, the system tries to limit the memory usage of processes to the soft limit. We could set the soft limits to '0' and allocated-value for OPPORTUNISTIC and GUARANTEED containers respectively.
    - Swap limit (YARN-6674): We could set the swappiness of OPPORTUNISTIC containers to 100 so the memory used by these processes can be aggressively swapped.
    - OOM control (YARN-6677): Cap the `hadoop-yarn` cgroup to the NM's memory capacity. If the aggregate usage goes over this capacity, all containers are paused and the NM is notified. The NM can then preempt enough opportunistic containers. The alternative approach of enforcing memory limits on a

per-container basis prevents containers from momentarily spiking over their limit, which is common in JVM-based containers.

In addition to enforcing limits, we could also use cgroups instead of procfs to monitor container resource usage (YARN-6668). E.g. `memory.usage`

OOM Killer (YARN-1014)

On Linux, the OOM Killer can be configured to favor preempting opportunistic containers when triggered.

Future Work

In the interest of putting together a usable version at the earliest, this JIRA aims to implement the simplest version of over-subscription. The following items are reserved for future work (in no order of priority):

1. Same node promotion of opportunistic containers: when resources become available on a node, consider promoting an opportunistic container before allocated new containers.
2. Support for non-Linux operating systems.
3. Sophisticated policies and heuristics to (1) pick applications more conducive to be run using OPPORTUNISTIC containers, (2) dynamic thresholds for the extent of over-subscription.
4. Cross node promotion of opportunistic containers.
5. Disk and network isolation using cgroups: need to be added as part of disk and network support (YARN-2139 and YARN-2140).