Capacity Scheduler: Improve delay scheduling mechanism

Authors: Wangda Tan, Vinod Kumar Vavilapalli with inputs from Jason Lowe, Ram Venkatesh
Last Modified: Sep 18, 2015

Background

<Skip this part if you're already familiar with Capacity Scheduler delay scheduling implementation.>

What is delay scheduling?

- Data is spread across different nodes/racks in a data center.
- Performance varies if a process reads data node-local(from same node) / rack-local(from same rack) / off-switch.

- When placing containers in the cluster, the scheduler would like to wait for some time to achieve better locality, this can shorten job execution time AND use fewer cluster resources.

More to read

Please refer to Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling for more details.

In YARN today, implementation of Capacity Scheduler is similar to algorithm #2 (count-based) and implementation of Fair Scheduler is similar to algorithm #3 (time-based).

Jason Lowe has a very good explanation about why Capacity Scheduler chose count-based delay instead of time-based delay, see link.

Concepts

Missed-opportunity

Initially, it's 0.
In each NM heartbeat, if an app gets a chance to allocate a container on the node, but it ***prefers*** to wait for some time to get better locality, the scheduler increases missed-opportunity for priority of this application by 1.
When the app gets a node-local/rack-local allocation, scheduler resets the missed-opportunity to 0.

Node-locality-delay

When an app's missed-opportunity >= node-locality-delay, the scheduler will start giving the application containers on the same rack of requested host.

Rack-locality-delay

When an app's missed-opportunity >= rack-locality-delay, the scheduler will start giving the application containers on an arbitrary host.

Issues with current Capacity Scheduler implementation

1) Waiting time to allocate each container highly depends on cluster availability

Currently, app can only increase missed-opportunity when a node has available resource AND it gets traversed by a scheduler. There're lots of possibilities that an app doesn't get traversed by a scheduler, for example:

A cluster has 2 racks (rack1/2), each rack has 40 nodes. Node-locality-delay=40. An application prefers rack1. Node-heartbeat-interval=1s.
Assume there are 2 nodes available on rack1, **delay to allocate one container** = 40 sec.
If there are 20 nodes available on rack1, **delay to allocate one container** = 2 sec.

2) It could violate scheduling policies (Fifo/Priority/Fair)

Assume a cluster is highly utilized, an app (app1) has higher priority, it wants locality. And there's another app (app2) has lower priority, but it doesn't care about locality. When node heartbeats with available resource, app1 decides to wait, so app2 gets the available slot. This should be considered as a bug that we need to fix.

The same problem could happen when we use FIFO/Fair queue policies.

Another problem similar to this is related to preemption: when preemption policy preempts some resources from queue-A for queue-B (queue-A is over-satisfied and queue-B is under-satisfied). But queue-B is waiting for the node-locality-delay so queue-A will get resources back. In next round, preemption policy could preempt this resources again from queue-A.

Proposal

Refine-able container placement + Per-container-delay might be able to solve the problem.

Refine-able container-placement

When a node has free resources, the head-of-line (HOL) application can use the resources, but it prefers to wait for some time for better locality. Instead of skipping the node, it will allocate an RMContainer on the node and set the state to ALLOCATED_WAITING.
ALLOCATED_WAITING is a state before ALLOCATED, but it is not exposed to entities outside of ResourceManager.

After the container is allocated, if there's another node that heartbeats in and it also has available resources, HOL application has a chance to move the ALLOCATED_WAITING containers to the new node for better locality (or other purposes).

Per-container delay

In this proposal, delay is a part of container instead of global variable of application, Application can say: “for each allocated container, if it’s a rack-local/off-switch I’d like to wait for X secs for better locality”.

The benefit of per-container delay is the delay of allocating each container doesn’t relate to the number of nodes available in the cluster. In the above example, assume that the per-container delay is 2 sec, no matter there’re 2 nodes or 200 nodes available in a cluster, delay to allocate one container is 2 sec.

To avoid application set a very high delay (such as 10 min), we shall have a global max-container-delay to cap the delay to avoid resource wastage.

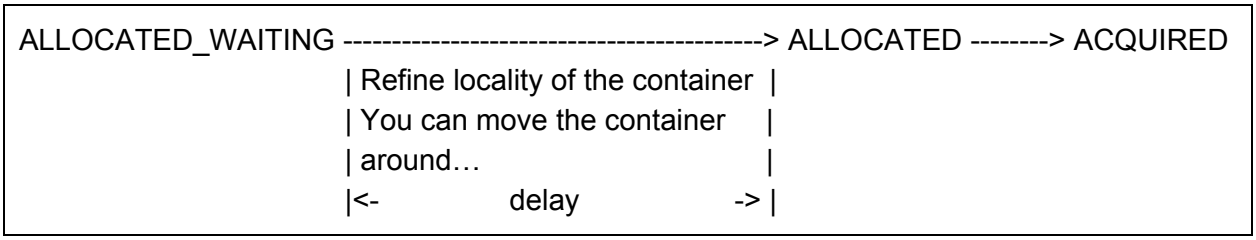
Issue if we use container reservation.
Currently, a reserved container will block the node allocation, so it is possible a higher-priority application can easily block a whole cluster.
Comparing to ALLOCATING_WAITING approach, an app can reserve only one container per node, but it can allocate node-available-resource/resource-per-container containers using the ALLOCATING_WAITING approach.

Addressed issues with this proposal

- Waiting time to allocate each container is no longer related to cluster availability.
- Scheduling policies will be always respected. No matter we enable fairness / priority / FIFO ordering policy, scheduler will make sure HOL application can get served first.

Implementation:

(Must) Add ALLOCATED_WAITING state to RMContainer



When RMContainer is in ALLOCATED_WAITING state, scheduler can move it around, and AM won’t aware of allocation of the container. With this, we can avoid facing issues such as container being moved after the container acquired/launched by AM already. Once it exceeds the delay, it becomes ALLOCATED state, so AM will receive it in next heartbeat.

(Must) Make the delay scheduling logic pluggable

To make sure it's a backward compatible change, we need to make the delay scheduling logic pluggable, admin can choose to use previous per-application delay or currently per-container delay.

(Optional) Be able to specify delay per-application / per-resource-request

A cluster can run different kinds of workloads, some are sensitive to container placement, some are not, we can improve this by setting different locality-delay for different apps (or even different resource requests)

Things to discuss

1. How to support both node-locality / rack-locality with the proposal?

In the scheduler side, depends on application's requirements, it can calculate the delay time in ALLOCATE_WAITING according to policy, and scheduler can update delay if the container gets updated.

For example, if an app wants 2 secs as minimum wait time for rack-local, and 10 secs as minimum wait time for off-switch. Assume a scheduler allocates an off-switch container at T1 (request is node-local), it sets the container's state to ALLOCATED_WAITING and set delay to 10 secs. At (T1 + 1.5), scheduler finds a rack-local slot that can move the container, it will update the delay to 0.5 sec (2 sec - 1.5 sec, max-wait-time-for-rack-local minus elapsed-time).

In the future, we can implement different policies to calculate and modify delay to support other delay scheduling cases, such as soft label (An application prefers GPU, but also accepts CPU, it will wait some time before switch to CPU), virtualization-layer locality support (YARN-19).

2. Time-based waiting / nodes-tick-based waiting

There are pros/cons of using time-based waiting / node-ticks-based waiting as container allocation delay.

Time-based waiting is more predictable, it clearly describes: when there's any resource available, I will wait at most X secs for achieving better locality. But it doesn't consider the time wasted to do GC or machine freeze due to CPU contention, etc.

Node-ticks-based waiting is more reliable to machine freezes / RM GC, etc.

In general we don't want to hard code which waiting approach will be used, it could be time-based / nodes-tick-based or a combination of them.

Plan:

- Do Proof-of-concept first
- If we can make changes under control, will open sub tickets and start development.

Appendix:

Alternative solutions

There are some alternative solutions, let me explain what's the issues of these solutions:

Scale down delay for allocation based on cluster availability

When there are fewer resources available in a cluster, an application shouldn't be picky. For example, a cluster with 2 nodes available should wait 1/100 comparing to a cluster with 200 nodes available.

Issues: It cannot solve the "priority inversion issue", and can only solve "waiting time depends on availability" partially: If an app wants hard locality, such as only want resource on host[1-3, 7-8], it is very expensive to calculate availability of host[1-3, 7-8]. And it needs to consider other properties such as node partition, etc.