Transforming Resource Distribution in Apache Hadoop YARN: Enhancing Flexibility for the Cloud

Community Over Code 2023

SPEAKERS

Who are we?

Benjamin Teke

https://github.com/brumi1024
https://www.linkedin.com/in/benjamin-teke-56aa2b159/

Tamas Domok

https://github.com/tomicooler/
https://www.linkedin.com/in/tomicooler/

AGENDA

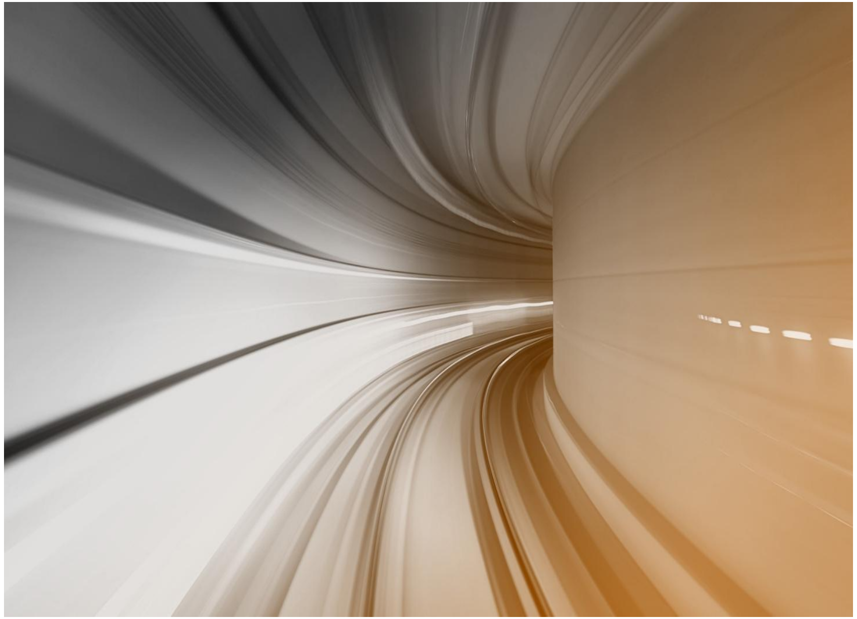
- What is YARN?
- Capacity Scheduler’s impacted features
- The way to flexible queue mode
- Capacity calculation changes

YARN

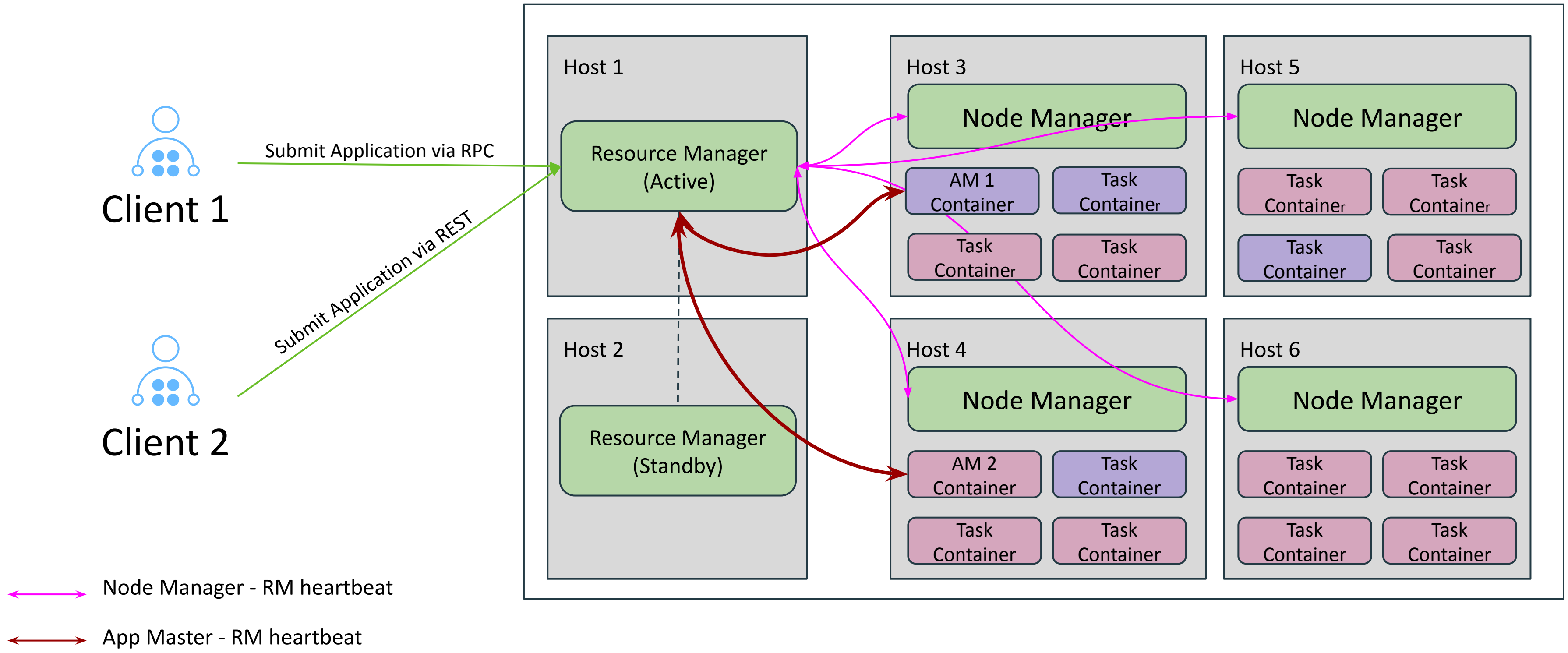
A quick overview

YARN stands for “Yet Another Resource Negotiator”

- Hosts:
  - Resource Manager: Manages resource allocations on a cluster.
  - Node Manager: Responsible for task execution on every single node.
- Application:
  - Application Master: Job lifecycle, fulfills resource needs. Works with NM by monitoring task execution.
  - Container: Bundle of resources for a task.



YARN ARCHITECTURE



SCHEDULERS IN YARN

YARN has three different schedulers:

- FIFO Scheduler
- Fair Scheduler
- Capacity Scheduler

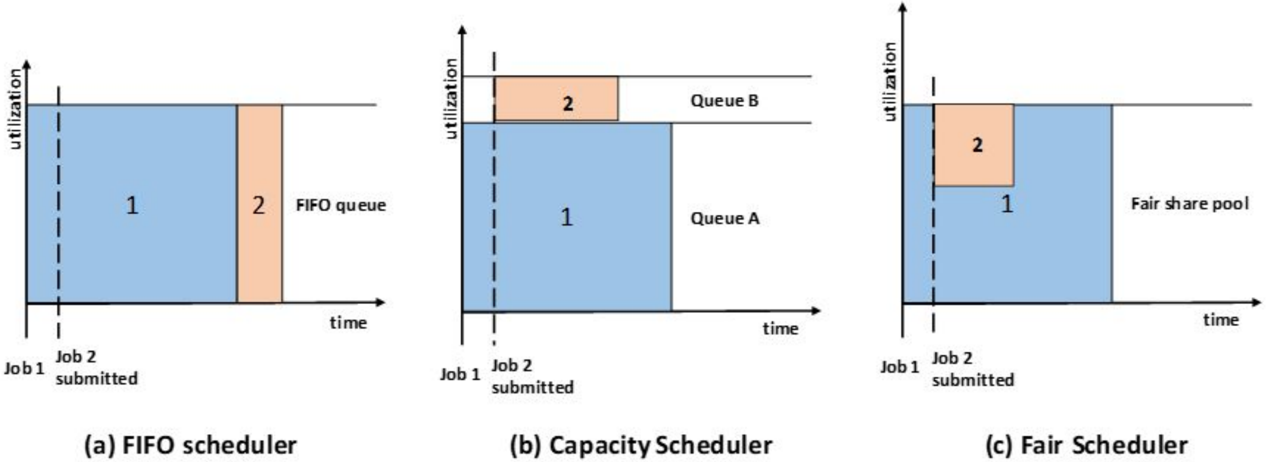


Figure 1: YARN Schedulers' cluster utilization vs. time

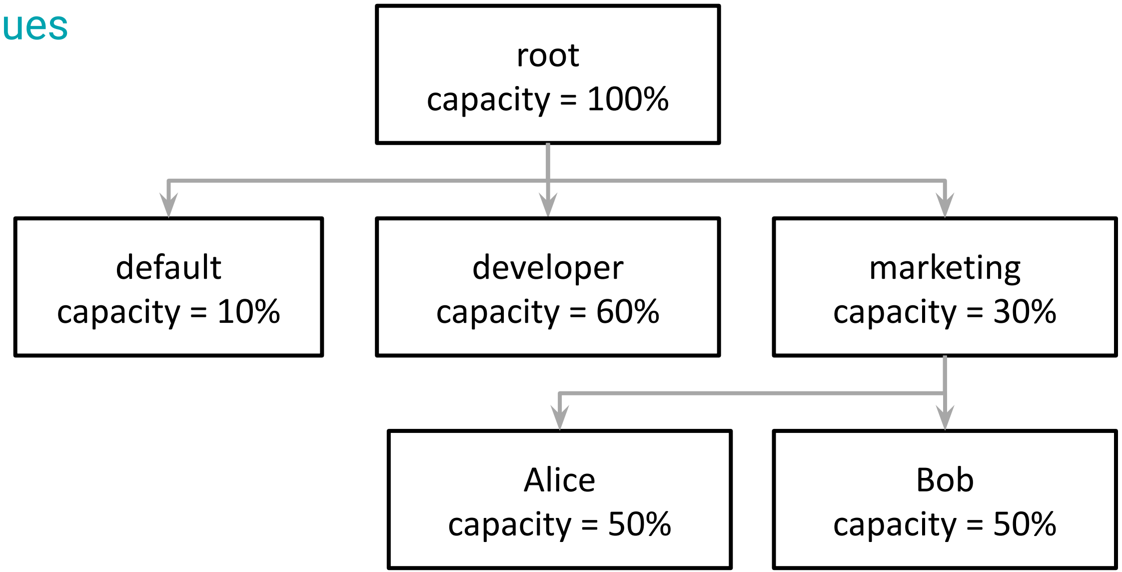
CAPACITY SCHEDULER (CS)

- Designed to support multi-tenancy in a cluster
- Hierarchical queues
- Capacity guarantees
  - Elasticity by using any excess resources
- Supported resource types
  - Memory
  - Dominant Resource Fairness (memory, CPU, FPGA, GPU, etc)

QUEUES

Applications are organized in queues

- Queues should be arranged into a hierarchy
  - All queues descend from “root”
  - Placement engine enables automatic application placement
  - By default all apps are submitted to a single queue named “default”
- Configurable limits
  - Capacity
  - Maximum Capacity
- All of the resources must be distributed



CONFIGURING QUEUE CAPACITIES

Legacy queue mode

- Percentage / Relative mode:
  - root.queuename.capacity=50

- Absolute mode:
  - root.queuename.capacity=[memory=4096, vcores=2]

APPLICATION PLACEMENT

- Applications can be placed automatically based on different rules
    - The rules are evaluated from top to bottom
- Matching can be based on:
    - Submitter's username
    - Submitter's group
    - Application name
- Policies define the actual placement
    - For example: the application can be placed to a queue named after the submitter's user name, group name or simply the name of the application
- If the rule doesn't apply the fallback option is executed
    - Fallback options include:
        - Skip this rule
        - Reject the placement
        - Place to "default" queue

CLOUDERA

AUTO QUEUE CREATION

Percentage/Absolute mode

- A parent that has the auto-creation enabled becomes a Managed Parent
- CS can create leaf queues automatically
  - Static queue
  - Dynamic queue

- Dynamic queues can be configured via templates
  - capacity.root.parent1.leaf-queue-template.<queue-property>
- The sum of capacities between siblings rule is relaxed with dynamic queues
- Unused dynamic queues are automatically set to zero capacity

INTRODUCING FLEXIBLE AUTO QUEUE CREATION

Weight mode

- Shortcomings of legacy auto queue creation:
  - Only leaf queues can be dynamically created

  - It’s not possible to create static queues under a Managed Parent
    - Every dynamic queue under a parent is created based on one template, so it has the same configuration

- Reason for these shortcomings:
  - Rigidity of capacity configuration

CLOUDERA

WEIGHT MODE

Legacy queue mode

- Separate distribution mode
- Describes the amount of resources in relation to sibling queues
- Internally it is translated back to percentage mode
- Mixing modes:
  - Percentage + weight possible, but not under the same parent
  - Absolute + weights not possible

CONFIGURING QUEUE CAPACITIES

Legacy queue mode

- Percentage / Relative mode:
  - root.queuename.capacity=50
- Absolute mode:
  - root.queuename.capacity=[memory=4096, vcores=2]
- Weight mode:
  - root.queuename.capacity=1.0w

CLOUDERA

Percentage

root.default.capacity=12.5

root.test1.capacity=50

root.test2.capacity=37.5

root.test1.test1_1.capacity=12.5

root.test1.test1_2.capacity=12.5

root.test1.test1_3.capacity=75

Weight

root.default.capacity=2w

root.test1.capacity=8w

root.test2.capacity=6w

root.test1.test1_1.capacity=1w

root.test1.test1_2.capacity=1w

root.test1.test1_3.capacity=6w

Absolute

root.default.capacity=
[memory=16384, vcores=4]

root.test1.capacity=
[memory=65536, vcores=16]

root.test2.capacity=
[memory=49152, vcores=12]

root.test1.test1_1.capacity=
[memory=8192, vcores=2]

root.test1.test1_2.capacity=
[memory=8192, vcores=2]

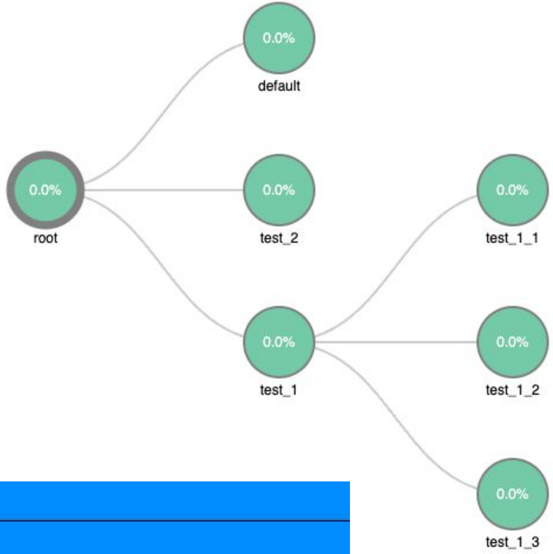
root.test1.test1_3.capacity=
[memory=49152, vcores=12]

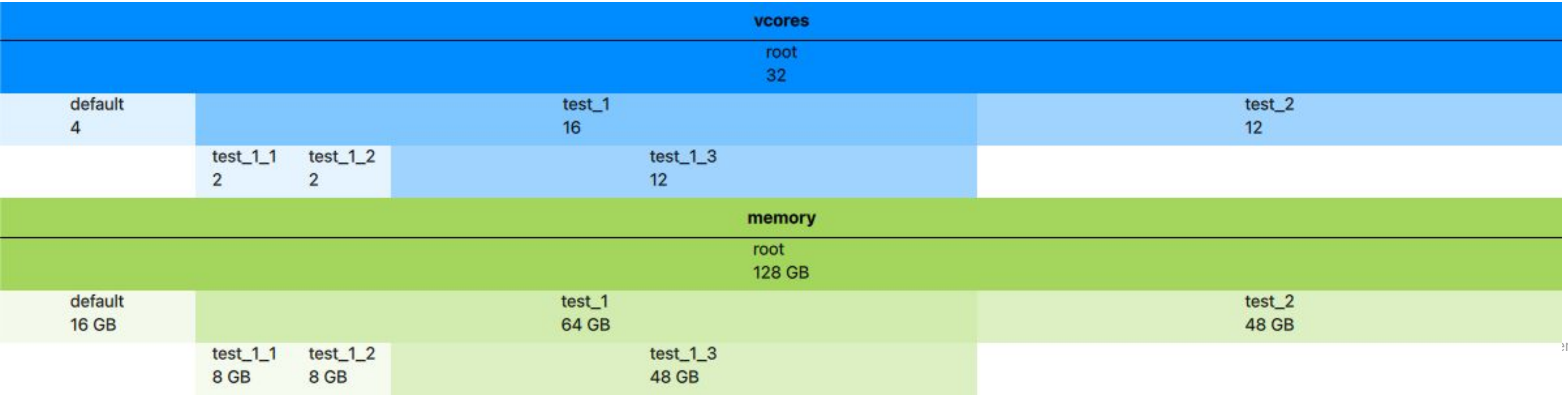
EXAMPLE QUEUE HIERARCHY

Let’s share the cluster resources (128 GB memory, 32 vcores)

root.queues=
  default, test1, test2

root.test1.queues=
  test1_1, test1_2, test1_3





FLEXIBLE AUTO QUEUE CREATION

Weight mode

- Flexible auto queue creation goes hand in hand with weight mode

- Any parent can have both dynamic and static parent and leaf child queues, the only restriction is that every child under that parent must be in weight mode
  - Auto creation enabled parent queues and dynamic leaf queues are no longer differentiated in the code

- Features:
  - Templating
  - Auto Queue Deletion
  - Configurable depth

MOTIVATION

The limitations of the legacy queue mode

- Not possible to mix capacity modes, e.g. queues in percentage mode under an absolute parent
- Not possible to set queue capacity in a mixed manner, e.g. 2048 MB memory and 10% CPU
- Two Auto Queue Creation modes
  - Legacy
  - Flexible

For resources like GPU/FPGA the absolute mode makes more sense than using weight or relative mode.

Some apps also needs a specific amount of resource.

=> AQC should support flexible queue capacities

INTRODUCING THE CAPACITY VECTOR

A new way to define capacities for queues

[memory=4096, vcores=10%, custom=2w]

Each resource can be specified with either an **absolute unit** or **percentage** or using **weight**. Any combination is allowed.

- root.queuename.capacity=[memory=4096,vcores=2]
  - [memory=4096, vcores=2]
- root.queuename.capacity=50
  - [memory=50%, vcores=50%]
- root.queuename.capacity=1w
  - [memory=1w, vcores=1w]

EXAMPLE: MIXING MODES

Using different queue capacity modes than the parent's mode

yarn.scheduler.capacity.legacy-queue-mode.enabled=false

root.default.capacity=1w

root.test_1.capacity=[memory=65536, vcores=16]

root.test_2.capacity=75

root.test_1.test_1_1.capacity=50

root.test_1.test_1_2.capacity=1w

root.test_1.test_1_3.capacity=[memory=49152, vcores=12]

EXAMPLE: MIXED QUEUE CAPACITIES

Using different modes for the resource types

yarn.scheduler.capacity.legacy-queue-mode.enabled=false

root.default.capacity=[memory=1w, vcores=4]

root.test_1.capacity=[memory=65536, vcores=100%]

root.test_2.capacity=[memory=3w, vcores=12]

root.test_1.test_1_1.capacity=[memory=1w, vcores=1w]

root.test_1.test_1_2.capacity=[memory=50%, vcores=2]

root.test_1.test_1_3.capacity=[memory=49152, vcores=86%]

CLOUDERA

CALCULATION EXPLAINED

ResourceCalculationDriver; CapacityScheduler.updateClusterResource

calculate(Queue Configuration, Cluster Resource) => Effective Minimum and Maximum Resources

The calculation is done for each resource type **one at a time** (e.g. vcores, memory).

Precedence:

1. Absolute
2. Percentage
3. Weight

In a homogenous hierarchy where every capacity is set in either percentage or weight or absolute, the **capacity** (in percentage relative to its parent queue) of the queue can be calculated without knowing the available cluster resources.

This is not true if capacity modes can be mixed, because changing the available cluster resources results in different resource shares.

CLOUDERA

CALCULATION EXAMPLE

clusterResource=

[memory=16384, vcores=16, custom=100]

root.a.capacity=

[memory=4096, vcores=50%, custom=3w]

root.b.capacity=

[memory=30%, vcores=10, custom=5w]

root.c.capacity=

[memory=70%, vcores=1w, custom=50]

memory: root.a has absolute resource, root.b and root.c has percentage, therefore the percentage is calculated from the remaining resource (16384 - 4096)

vcores: root.b has absolute resource, root.a has percentage, then root.c has weight, which gets the remaining vcore

custom: root.c has absolute resource, then root.a and root.b has weight

CHANGES

and differences

The effective capacity, absoluteCapacity and derived properties like maximumApplications are calculated from the hierarchy between the resources.

Zero cluster resource means zero capacity, hence the maximumApplications defaults to the configured value.

Scheduler Rest Changes:

capacity, maxCapacity shows the configured values in legacy queue mode while effective in non-legacy queue mode. normalizedWeight is always 0 in non-legacy queue mode.


```
"queueCapacityVectorInfo" : {
  "configuredCapacityVector" :
"[memory-mb=3.0w,vcores=12.0]",
  "capacityVectorEntries" : [ {
    "resourceName" : "memory-mb",
    "resourceValue" : "3.0w"
  }, {
    "resourceName" : "vcores",
    "resourceValue" : "12.0"
  } ]
},
```

TESTING

How not to break everything?

- Feature flag
  - **yarn.scheduler.capacity.legacy-queue-mode.enabled=false**
- Ran the whole test set *(manually)* with both legacy and non-legacy queue mode
- Added easy to maintain characterization tests
  - TestRmWebServicesCapacitySched*
  - A simple git diff can reveal breaking changes
  - Added a test suite that runs with both queue mode automatically

DOMINANTRESOURCECALCULATOR

YARN-11507

ResourceCalculator abstraction:

- DefaultResourceCalculator: considers the memory when comparing two Resource objects.
- DominantResourceCalculator: considers the dominant ([memory=1024, vcores=2] < [memory=4096, vcores=1]) resource when comparing two Resource objects. (white paper)
- While it is useful in a multiuser environment, this should not affect the calculation of the queue properties (like capacity/absoluteCapacity). **Currently it does**, it's not a regression, just an observation we made.

CLOUDERA

References

- YARN-10888
- YARN – The Capacity Scheduler
- YARN Capacity Scheduler
- Flexible Queue Auto-Creation

@9quapaw

CLOUDERA

THANK YOU

CLOUDERA