Hadoop Apache Yarn

(Yet Another Resource Negotiator)

Introducción

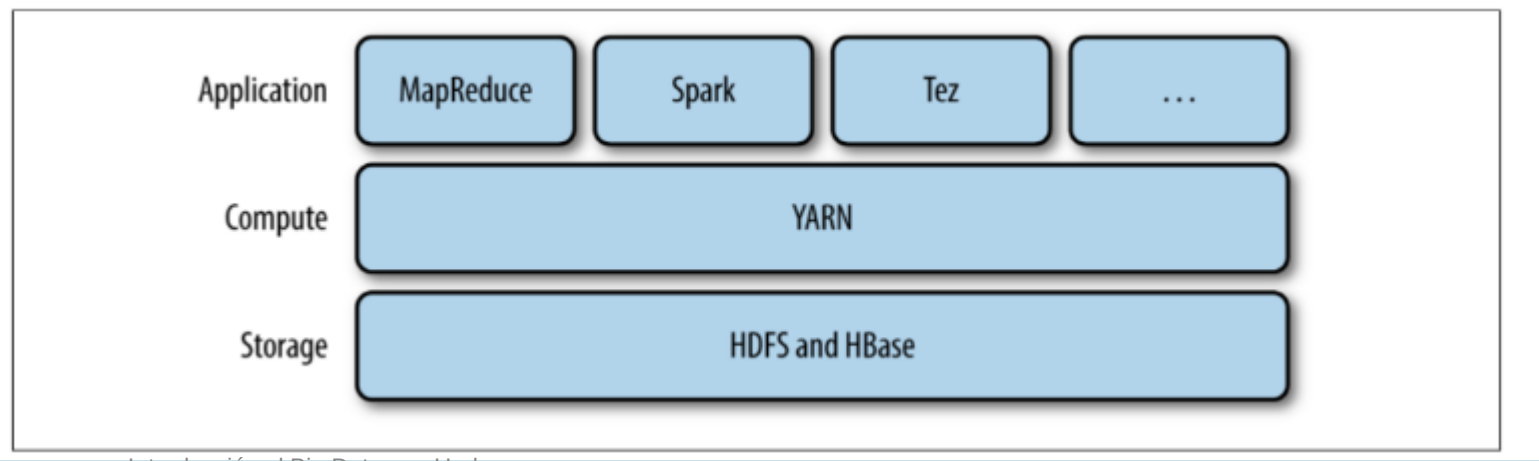
Fundamentos

Introducción

- Gestor de recursos del cluster de Hadoop
- Aparece en la versión 2.0 para mejorar el rendimiento de MapReduce
- Ofrece API’s para trabajar con los recursos del cluster
- No se utiliza directamente por el código del usuario

Introducción

- El código de usuario se escribe sobre frameworks de desarrollo (MapReduce y Spark)
- Ofrecen capa de aplicación para otras herramientas (Pig, Hive…)



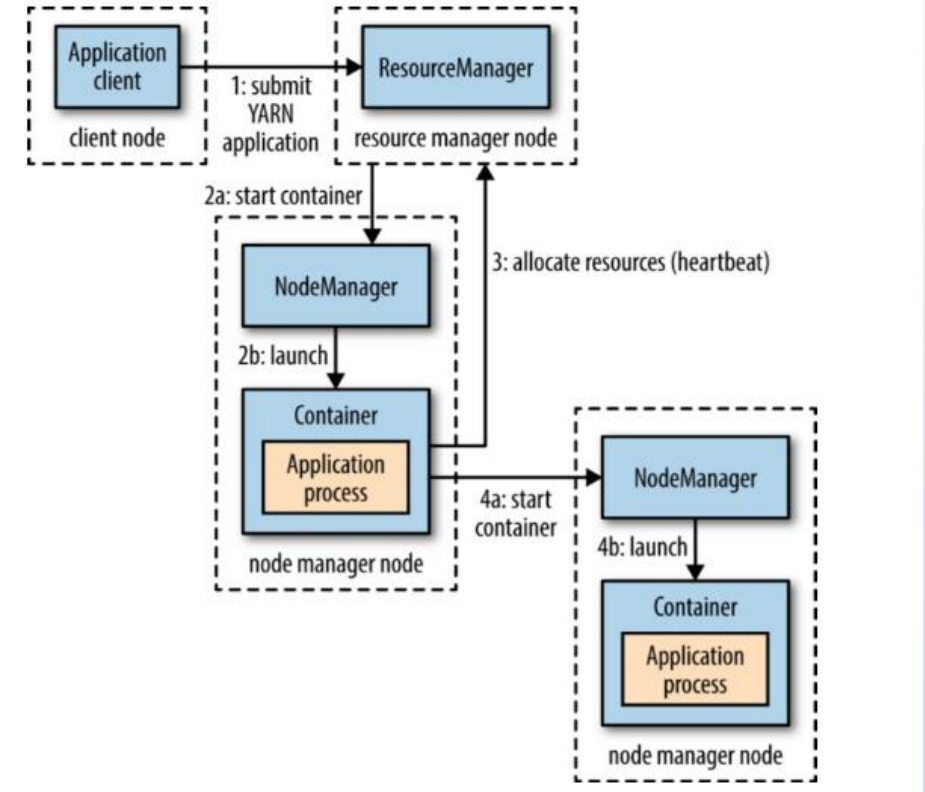
Introducción

- Utiliza dos daemon para ofrecer sus servicios
  - Administrador de recursos
    - Uno por cluster
    - Gestionar el uso de recursos
  - Administrador de nodos
    - Ejecuta en todos los nodos
    - Monitorizar los contenedores
- Contenedor → ejecuta un trabajo con recursos concretos

YARN - EJECución

Proceso de ejecución y petición de recursos

Ejecución


Application client
client node
1: submit YARN application
ResourceManager
resource manager node
2a: start container
NodeManager
2b: launch
Container
Application process
node manager node
3: allocate resources (heartbeat)
4a: start container
NodeManager
4b: launch
Container
Application process
node manager node

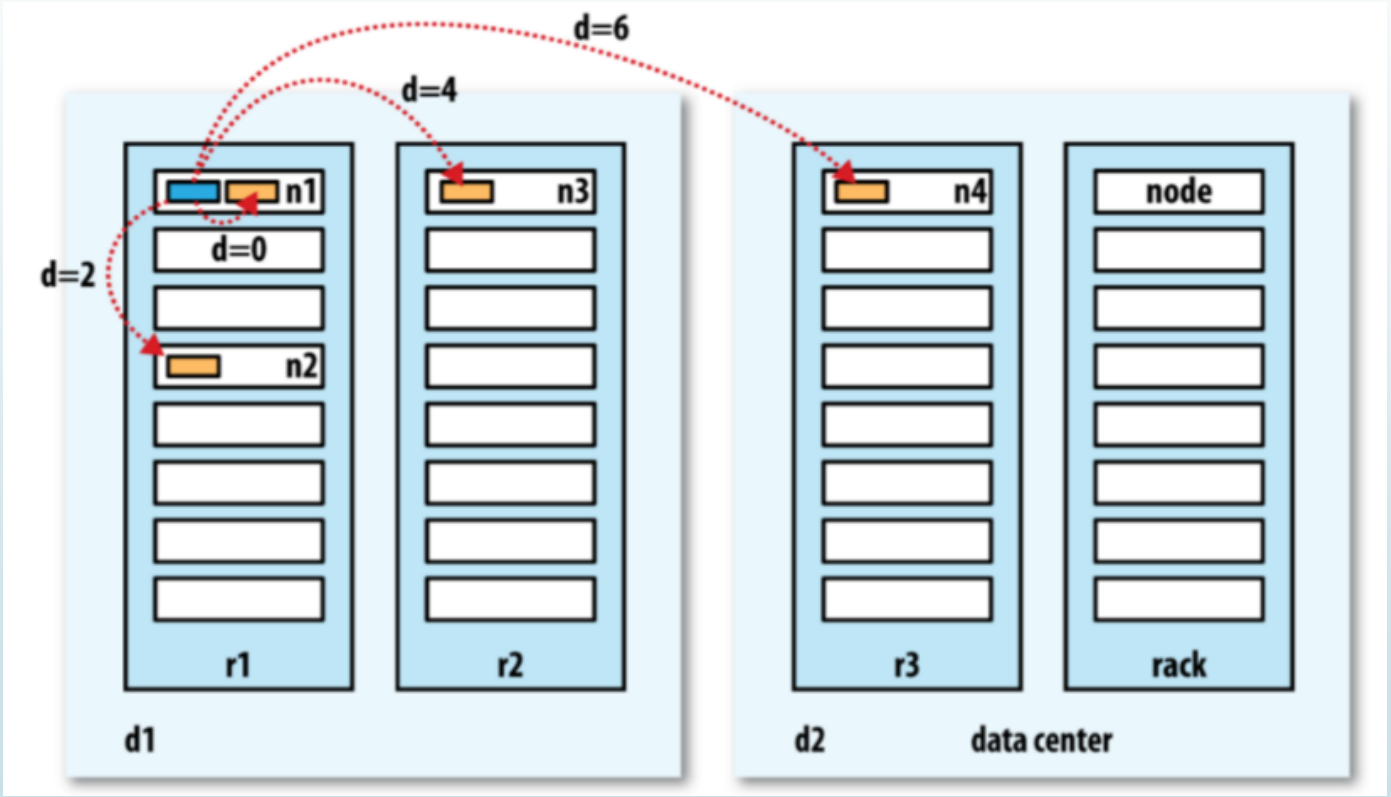
YARN Petición de recursos

- Es un modelo flexible
- Se puede indicar los recursos requeridos para cada contenedor (CPU, memoria)
- También puede establecer limitaciones a la ubicación
- La ubicación se considera crítica para el rendimiento al utilizar el ancho de banda efectivamente

YARN Recursos

- Con alto volumen de datos, la transferencia de datos puede ser limitador
- Hadoop calcula el ancho de banda por aproximación
- Calcula la distancia entre sus nodos representando la red en un árbol
- La distancia es el número de nodos a su antecesor común más cercano
  - Procesos en el mismo nodo
  - Diferentes nodos en el mismo rack
  - Nodos en diferentes rack en el mismo nodo

Yarn cálculo distancia



Yarn – Limitaciones de ubicación

- Es necesario para mejorar el rendimiento
- Especificar ubicación del contenedor
- Ubicación relajada→ cuando no se puede poner en el nodo
  - Por ejemplo, si no puede en el nodo, lo intenta en el mismo rack o si no en el mismo cluster

Yarn ubicación

- Las aplicaciones pueden hacer la petición en cualquier momento
- Spark lo hace al principio, reservando todos los contenedores
- MapRedude lo hace sobre la marcha cuando hace la tarea de mapa y luego cuando tiene que hacer la tarea de Reducción

Yarn Programación

Establecer los diferentes sistemas para ejecutar los trabajos

Yarn Tiempo de vida

- Las aplicaciones pueden ser de segundos o meses
- Clasificación por como se organizar los trabajos
  - Una aplicación por trabajo de usuario (MapReduce)
  - Una aplicación por flujo de trabajo (Spark)
  - Una aplicación de larga duración que se comparte entre usuarios (Impala)

Yarn programación

- En situación ideal las peticiones siempre se atienden
- En la realidad, recursos limitados → tiempos de espera
- YARN tienen varias maneras de establecer recursos
  - FIFO
  - Capacidad
  - Programación Justa (Fair)

FIFO (first in first out)

- Trabajos en cola en orden de llegada
- Es la más simple y no necesita configuración
- No encaja en cluster compartidos con trabajos largos y cortos

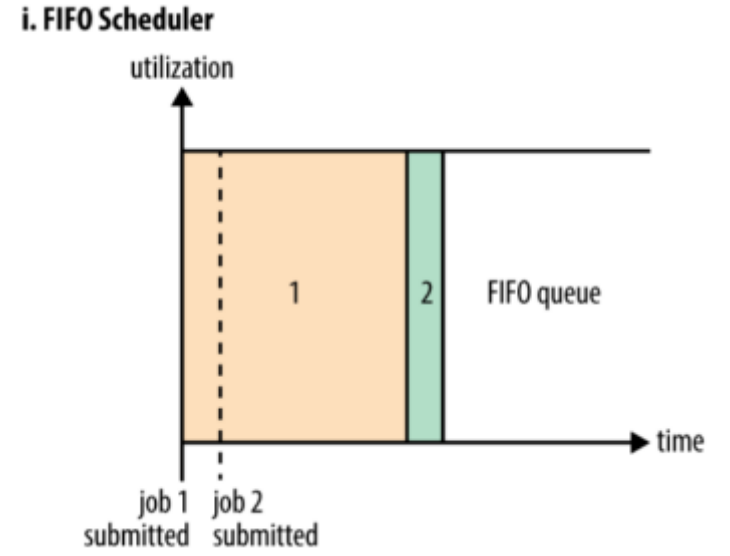
YARN Capacidad

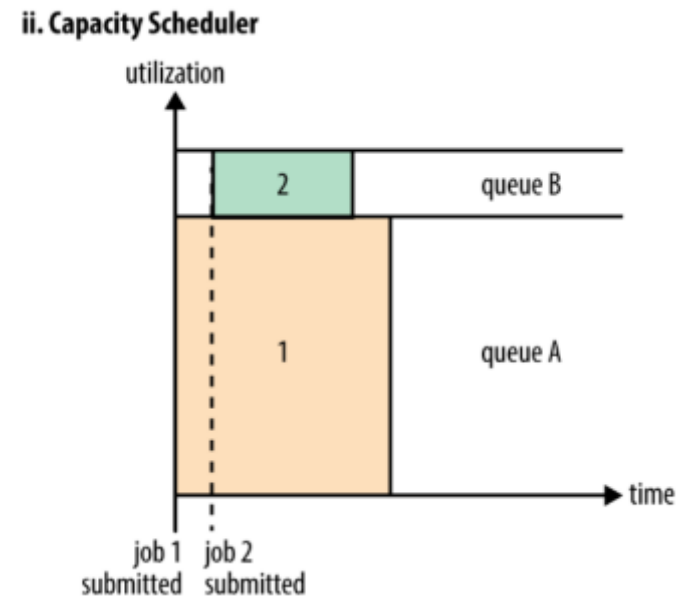
- Dispone de una cola específica para trabajos cortos
- Globalmente el rendimiento es menor
- Trabajos largos acaban más tarde que con FIFO

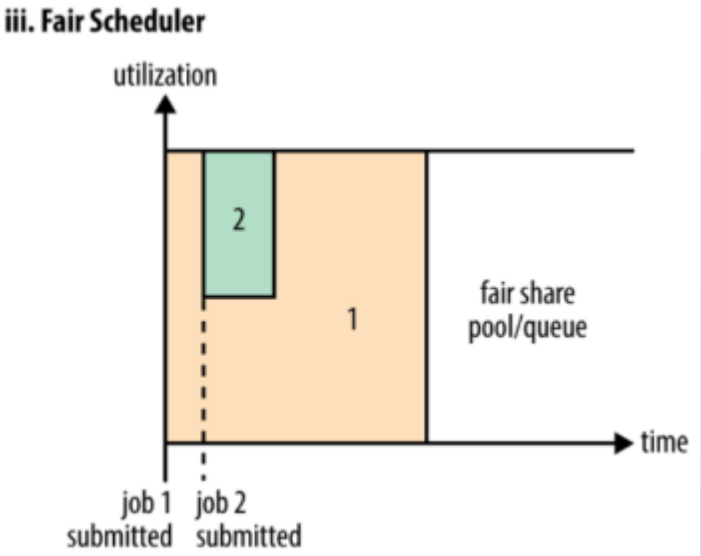
YARN Programación FAir

- No se reserva capacidad
- Dinámicamente reparte los recursos
- Hay tiempo de demora en la asignación
- Ejecución de trabajos grandes y pequeños menor que los anteriores

YArN Programación



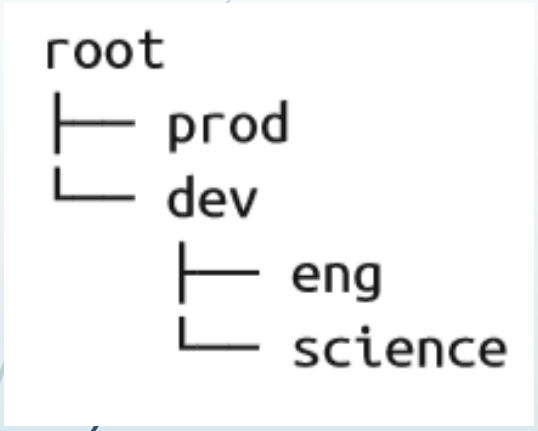


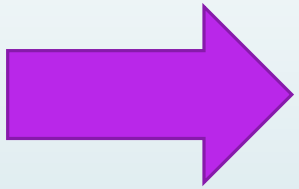


YArN Configuración

- La programación por capacidad permite establecer líneas con unos recursos preasignados
- Las colas pueden dividirse por jerarquías
- Dentro de cada cola → FIFO
- Elasticidad de las colas → Superar su capacidad si hay otra que no está en uso.

Yarn Configuración





```
<configuration>
  <property>
    <name>yarn.scheduler.capacity.root.queues</name>
    <value>prod,dev</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.queues</name>
    <value>eng,science</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.prod.capacity</name>
    <value>40</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.capacity</name>
    <value>60</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.maximum-capacity</name>
    <value>75</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.eng.capacity</name>
    <value>50</value>
  </property>
  <property>
    <name>yarn.scheduler.capacity.root.dev.science.capacity</name>
    <value>50</value>
  </property>
</configuration>
```

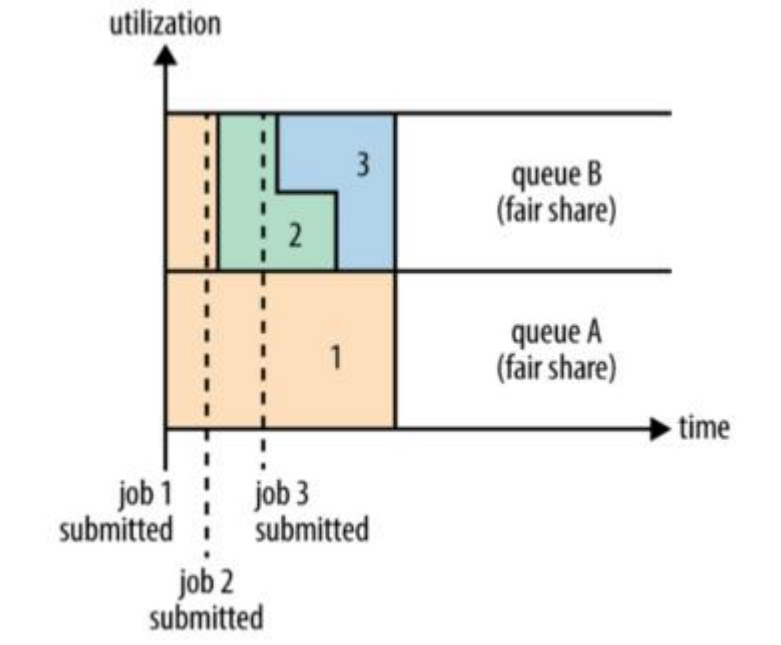
YARN Configuración

- Definir la cola depende del FrameWork
  - MapReduce→ mapreduce.job.queue.name
  - Si no existe la cola genera un error
  - Si no se especifica la cola se pone en la cola “default”

YARN Configuración FAIR

- Intenta asignar recursos para todas las aplicaciones
- Puede hacer reparto en la misma cola o entre colas
- Si una cola no está en uso toma los recursos
- Si entra un trabajo se ejecuta en la cola correspondiente
- Si hay más trabajos en la misma cola se reparte

Yarn Configuración



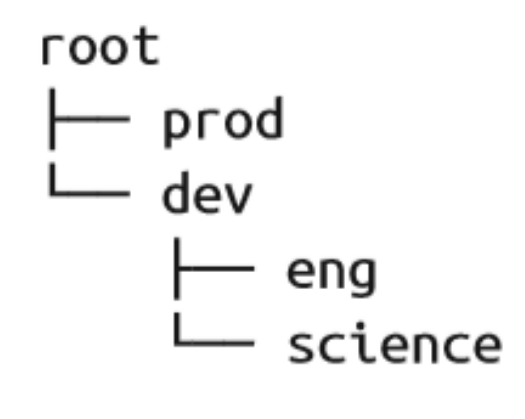
YARN configuración FAIR

- Normalmente se utiliza la programación por capacidad
- Modificar yarn-site.xml, yarn.resourcemanager.scheduler.class
- La configuración de la cola utiliza fair-Schedule.xml
- Si no hay archivo de configuración se crea una cola por usuario la primera vez que se ejecuta una aplicación
- Se puede establecer jerarquías

Yarn Configuración fair

- Reglas para establecer las colas
  - Específica: nombre de la cola
  - Usuario: el nombre de usuario de ejecución
  - Default: cuando no se ejecuta ninguan

YARN Configuración FAir




```
<allocations>
  <defaultQueueSchedulingPolicy>fair</defaultQueueSchedulingPolicy>

  <queue name="prod">
    <weight>40</weight>
    <schedulingPolicy>fifo</schedulingPolicy>
  </queue>

  <queue name="dev">
    <weight>60</weight>
    <queue name="eng" />
    <queue name="science" />
  </queue>

  <queuePlacementPolicy>
    <rule name="specified" create="false" />
    <rule name="primaryGroup" create="false" />
    <rule name="default" queue="dev.eng" />
  </queuePlacementPolicy>
</allocations>
```

Yarn configuración fair

- Cuando un trabajo se envía a una cola vacía en un cluster ocupado tiene que esperar.
- Preemption permite hacer el cálculo de cuando va a quedar libre
- Permite acabar con el procesamiento de un contenedor para liberar recursos
- Hay que configurar yarn.scheduler.fair.preemption a True y asignarles tiempos
  - DefaultMinSharePreemptionTimeOut
  - FairSharePreemtionTimeOut

YARN retrasos

Gestión de los retraso

Yarn Retrasos

- Cuando se asigna un trabajo es posible que esté ocupado → asigna a otro nodo
- Este proceso de asignación se puede retrasar para esperar que se libere
- Soportado por programación por capacidad y Fair

YARn REtrasos

- Cada nodo envía un pulso al administrador de recursos cada segundo
- Envía información sobre contenedores y recursos disponibles
- Propiedad DelayScheduling indica el número máximo de pulsos
  - yarn.scheduler.capacity.node-locality-delay
  - yarn.scheduler.fair.locality.threshold.nod

YARN Recurso Dominante

- Si hay un solo recurso la capacidad es fácil de calcular
- Cuando hay más de un recurso (memoria, cpu)
- YARN mira el recurso dominante para cada usuario

YarN DRF

- Un cluster con 100 CPU, 10 TB de memoria
- Aplicación A pide contenedores 2CPU, 300 GB
- Aplicación B pide contenedores 6 CPU y 100 GB
- A rquiere 2%,3% del cluster, memoria es dominante
- B require 6%,1%, cpu es dominante
- Como B es el doble en el recurso dominante, le asignaran la mitad que al A en una programación FAIR