Tema 2: Introducción a Hadoop

Tomás Fernández Pena

Máster en Tecnologías de Análisis de Datos Masivos: Big Data

Universidade de Santiago de Compostela

Tecnologías de Computación para Datos Masivos

citius.usc.es

Índice

Índice

Hadoop

Implementación open-source de MapReduce

- Procesamiento de enormes cantidades de datos en grandes clusters de hardware barato (commodity clusters)
  - Escala: petabytes de datos en miles de nodos

Características de Hadoop

Tres partes

- Almacenamiento distribuido: HDFS
- Planificación de tareas y negociación de recursos: YARN
- Procesamiento distribuido: MapReduce

Ventajas

- Bajo coste: clusters baratos o cloud
- Facilidad de uso
- Tolerancia a fallos

Introducción a Hadoop Instalación Introducción a HDFS YARN y MapReduce Ejemplo de programa MapReduce

Características de Hadoop

Tres partes

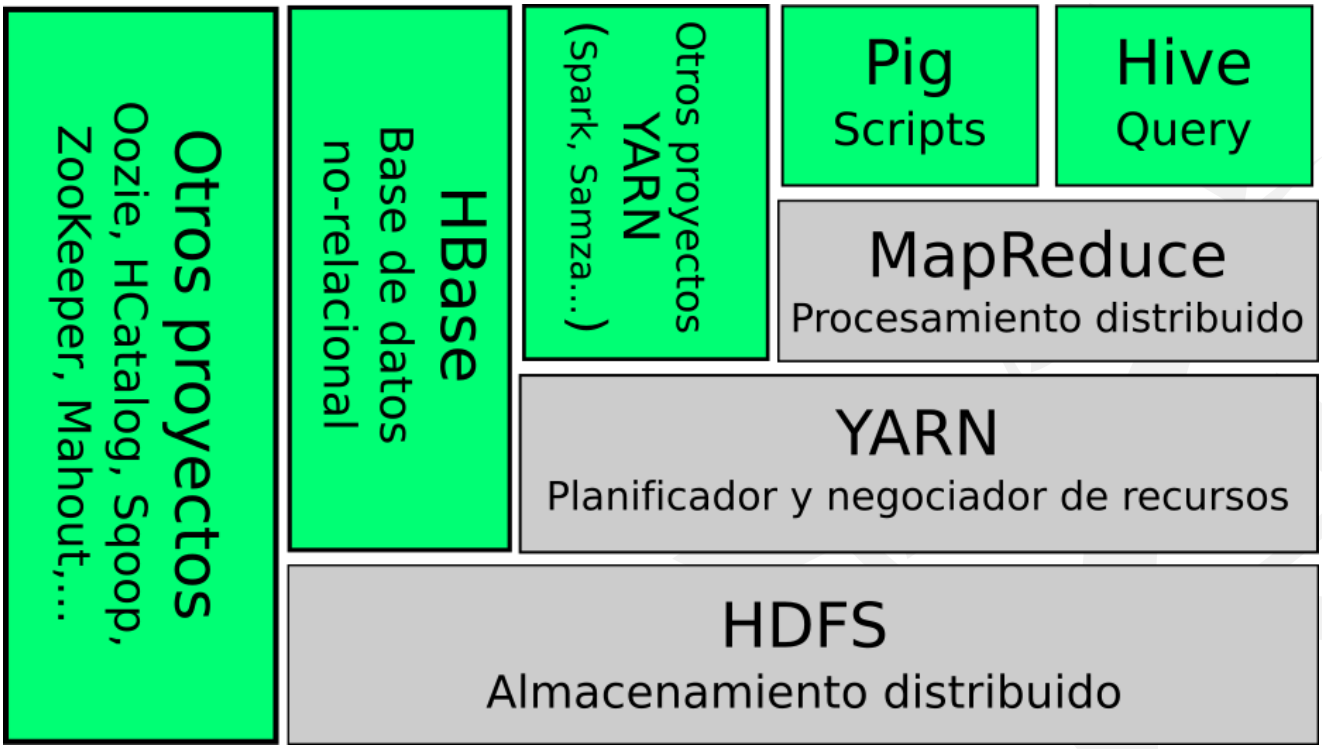
- Almacenamiento distribuido: HDFS
- Planificación de tareas y negociación de recursos: YARN
- Procesamiento distribuido: MapReduce

Ventajas

- Bajo coste: clusters baratos o cloud
- Facilidad de uso
- Tolerancia a fallos

CiTIUS Introducción a Hadoop, TCDM 2/40

Hadoop



Índice

Instalación

Instalación relativamente simple: aplicación Java

- Paquete fuente: hadoop.apache.org/releases.html
- Sistemas preconfigurados proporcionados por empresas como Cloudera/Hortonworks

Modos de funcionamiento:

- Standalone: todo en un nodo, para pruebas
- Pseudodistribuido: funciona como una instalación completa, pero en un solo nodo
- Totalmente distribuido

Introducción a Hadoop Instalación Introducción a HDFS YARN y MapReduce Ejemplo de programa MapReduce

Instalación

Instalación relativamente simple: aplicación Java

- Paquete fuente: hadoop.apache.org/releases.html
- Sistemas preconfigurados proporcionados por empresas como Cloudera/Hortonworks

Modos de funcionamiento:

- Standalone: todo en un nodo, para pruebas
- Pseudodistribuido: funciona como una instalación completa, pero en un solo nodo
- Totalmente distribuido

CiTIUS Introducción a Hadoop, TCDM 4/40

Ficheros de configuración

Principales ficheros de configuración:

- `core-site.xml`: parámetros de configuración general
- `hdfs-site.xml`: configuración del HDFS
- `yarn-site.xml`: configuración de YARN
- `mapred-site.xml`: configuración del MapReduce

Algunos parámetros generales

Fichero `core-site.xml`

- `fs.defaultFS`: nombre del sistema de ficheros a usar (HDFS u otro), por defecto `file:///`
- `hadoop.tmp.dir`: directorio base para otros directorios temporales, valor por defecto `/tmp/hadoop-${user.name}`
- `hadoop.security.authentication`: indica el tipo de autenticación, puede ser `simple` (sin autenticación) o `kerberos`, por defecto `simple`
- `hadoop.security.authorization`: indica si está activada la autorización a nivel de servicio, por defecto `false`

Índice

HDFS: *Hadoop Distributed File System*

Hadoop puede acceder a diferentes tipos de filesystems (local, HDFS, KFS, S3,. . .)

- Se recomienda HDFS: *Hadoop Distributed File System*

HDFS: Ventajas

- Diseñado para almacenar ficheros muy grandes en *commodity hardware*
- Elevado ancho de banda
- Fiabilidad mediante replicacion

HDFS: Inconvenientes

- Elevada latencia
- Poco eficiente con muchos ficheros pequeños
- Modificaciones siempre al final de los ficheros
- No permite múltiples escritores (modelo *single-writer, multiple-readers*

HDFS: *Hadoop Distributed File System*

Hadoop puede acceder a diferentes tipos de filesystems (local, HDFS, KFS, S3,. . .)

- Se recomienda HDFS: *Hadoop Distributed File System*

HDFS: Ventajas

- Diseñado para almacenar ficheros muy grandes en *commodity hardware*
- Elevado ancho de banda
- Fiabilidad mediante replicacion

HDFS: Inconvenientes

- Elevada latencia
- Poco eficiente con muchos ficheros pequeños
- Modificaciones siempre al final de los ficheros
- No permite múltiples escritores (modelo *single-writer, multiple-readers*

Introducción a Hadoop Instalación Introducción a HDFS YARN y MapReduce Ejemplo de programa MapReduce

HDFS: *Hadoop Distributed File System*

Hadoop puede acceder a diferentes tipos de filesystems (local, HDFS, KFS, S3,. . .)

- Se recomienda HDFS: *Hadoop Distributed File System*

HDFS: Ventajas

- Diseñado para almacenar ficheros muy grandes en *commodity hardware*
- Elevado ancho de banda
- Fiabilidad mediante replicacion

HDFS: Inconvenientes

- Elevada latencia
- Poco eficiente con muchos ficheros pequeños
- Modificaciones siempre al final de los ficheros
- No permite múltiples escritores (modelo *single-writer, multiple-readers*

CiTIUS Introducción a Hadoop, TCDM 7/40

Conceptos de HDFS

Namenode

Mantiene la información (metadatos) de los ficheros y bloques que residen en el HDFS

Datanodes

Mantienen los bloques de datos

- No tienen idea sobre los ficheros

Conceptos de HDFS (cont.)

Bloques

Por defecto 128 MB, tamaño configurable por fichero

- bloques pequeños aumentan el paralelismo (un bloque por Map)
- bloques más grandes reducen la carga del NameNode

Replicados a través del cluster

- Por defecto, 3 réplicas (configurable por fichero)

Backup/Checkpoint node

Mantiene backups y checkpoints del NameNode

- debería ejecutarse en un sistema con características similares al NameNode

HDFS: propiedades configurables (I)

Múltiples propiedades configurables (fichero `hdfs-site.xml`)

- `dfs.namenode.name.dir`: lista (separada por comas) de directorios donde el NameNode guarda sus metadatos (una copia en cada directorio), por defecto `file://${hadoop.tmp.dir}/dfs/name`
- `dfs.datanode.data.dir`: lista (separada por comas) de directorios donde los datanodes guarda los bloques de datos (cada bloque en sólo uno de los directorios), por defecto `file://${hadoop.tmp.dir}/dfs/data`
- `dfs.namenode.backup.address`: dirección y puerto de Backup node (por defecto, `0.0.0.0:50100`)

HDFS: propiedades configurables (II)

- `dfs.blocksize`: tamaño de bloque para nuevos ficheros, por defecto 128MB
- `dfs.replication`: nº de réplicas por bloque, por defecto 3
- `dfs.replication.max`: máximo nº de réplicas permitido por bloque, por defecto 512
- `dfs.namenode.replication.min`: mínimo nº de réplicas permitido por bloque, por defecto 1

Interfaz con HDFS

Varias interfaces:

1. Interfaz en línea de comandos: comando `hdfs dfs`
2. Interfaz web
3. Interfaz Java

Interfaz en línea de comandos:

- Permite cargar, descargar y acceder a los ficheros HDFS desde línea de comandos
- Ayuda: `hdfs dfs -help`

Más información: hadoop.apache.org/docs/stable3/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html, hadoop.apache.org/docs/stable3/hadoop-project-dist/hadoop-common/FileSystemShell.html

Introducción a Hadoop Instalación Introducción a HDFS YARN y MapReduce Ejemplo de programa MapReduce

Interfaz con HDFS

Varias interfaces:

1. Interfaz en línea de comandos: comando `hdfs dfs`
2. Interfaz web
3. Interfaz Java

Interfaz en línea de comandos:

- Permite cargar, descargar y acceder a los ficheros HDFS desde línea de comandos
- Ayuda: `hdfs dfs -help`

Más información: hadoop.apache.org/docs/stable3/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html, hadoop.apache.org/docs/stable3/hadoop-project-dist/hadoop-common/FileSystemShell.html

CiTIUS Introducción a Hadoop, TCDM 12/40

Índice

YARN: *Yet Another Resource Negociator*

Se encarga de la gestión de recursos y job-scheduling/monitorización usando tres demonios:

- *Resource manager* (RM): planificador general
- *Node managers* (NM): monitorización, uno por nodo
- *Application masters* (AM): gestión de aplicaciones, uno por aplicación

Permite que diferentes tipos de aplicaciones (no solo MapReduce) se ejecuten en el cluster

- Las aplicaciones se despliegan en contenedores (YARN JVMs)
- En Hadoop v3 se pueden usar Dockers

YARN: *Yet Another Resource Negociator*

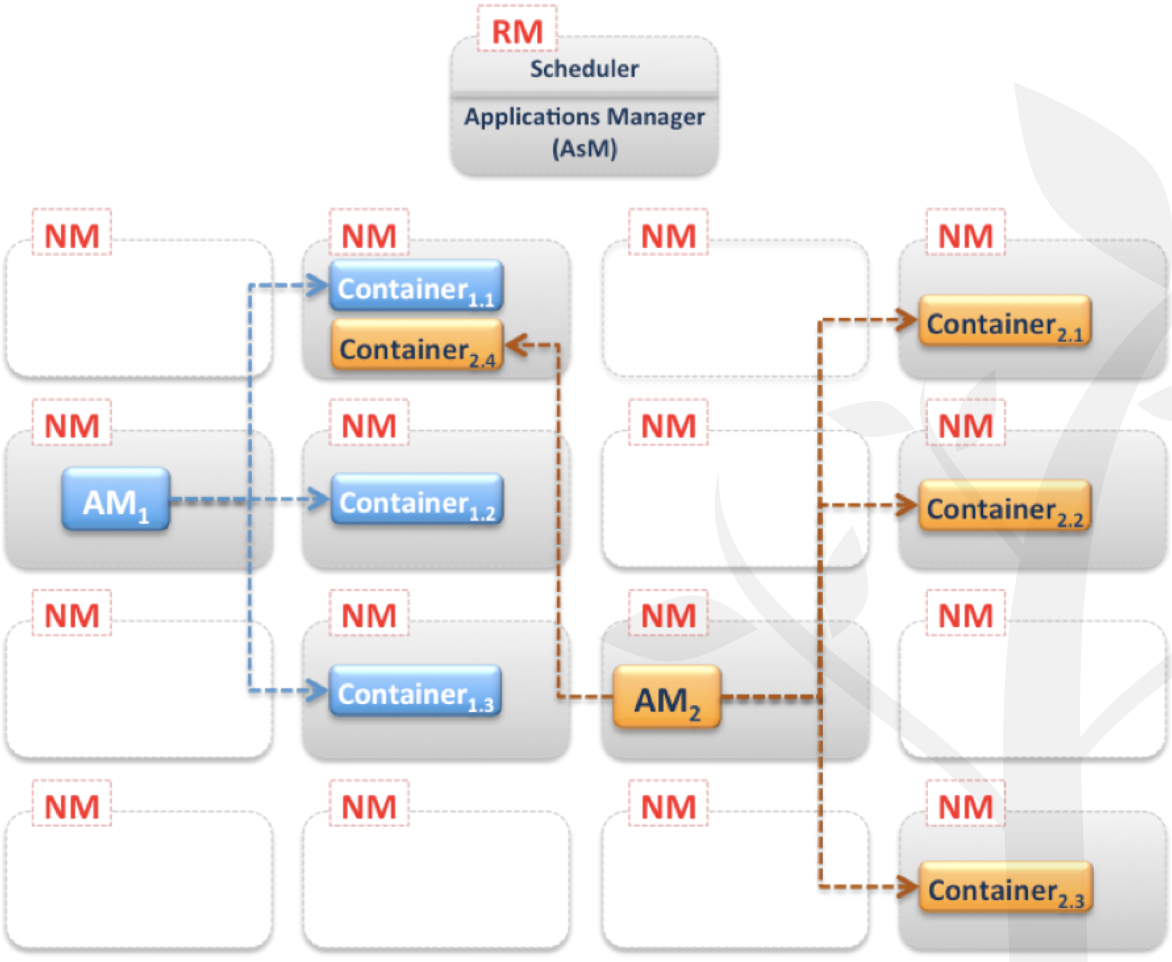
Se encarga de la gestión de recursos y job-scheduling/monitorización usando tres demonios:

- *Resource manager* (RM): planificador general
- *Node managers* (NM): monitorización, uno por nodo
- *Application masters* (AM): gestión de aplicaciones, uno por aplicación

Permite que diferentes tipos de aplicaciones (no solo MapReduce) se ejecuten en el cluster

- Las aplicaciones se despliegan en contenedores (YARN JVMs)
- En Hadoop v3 se pueden usar Dockers

Arquitectura YARN



Demonios YARN (I)

Resource manager

- arbitra los recursos entre las aplicaciones en el sistema
- demonio global, obtiene datos del estado del cluster de los node managers
- dos componentes:
  - Scheduler: planifica aplicaciones en base a sus requerimientos de recusos
  - Applications Manager: acepta trabajos, negocia contenedores y gestiona fallos de los Application Masters

Node managers

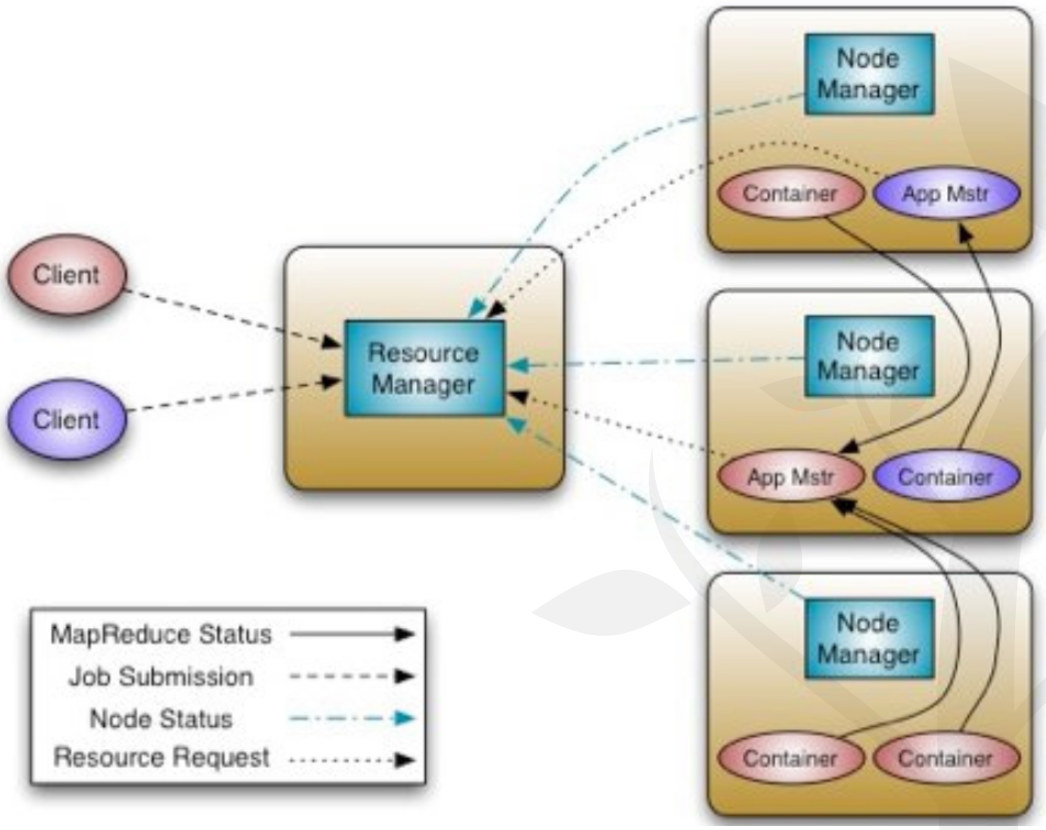
- uno por nodo
- monitorizan los recursos del cluster

Demonios YARN (II)

Application masters

- uno por aplicación, se encarga de gestionar el ciclo de vida de la aplicación
- solicita recursos (**contenedores**) al Resource manager y ejecuta la aplicación en esos contenedores
  - en una aplicación Mapeduce en un contenedor se ejecutan tareas Map o Reduce
  - el AM se ejecuta en su propio contenedor
- trabaja con los Node managers para ejecutar y monitorizar las tareas

Elementos de control YARN



Fuente: A. Murthy, V. Vavilapalli, “Apache Hadoop YARN”, Addison-Wesley, marzo 2014

YARN: propiedades configurables (I)

Múltiples propiedades configurables (fichero `yarn-site.xml`)

- `yarn.resourcemanager.hostname`: el host ejecutando el ResourceManager
- `yarn.scheduler.maximum-allocation-vcores`, `yarn.scheduler.minimum-allocation-vcores`: nº máximo y mínimo de cores virtuales (threads) que pueden ser concedidos a un contenedor
- `yarn.scheduler.maximum-allocation-mb`, `yarn.scheduler.minimum-allocation-mb`: memoria máxima y mínima que puede ser concedida a un contenedor (la memoria solicitada se redondea a un múltiplo del mínimo)

YARN: propiedades configurables (II)

- `yarn.nodemanager.aux-services`: lista de servicios auxiliares que deben implementar los NodeManagers (uno de ellos, el barajado MapReduce)
- `yarn.nodemanager.resource.memory-mb`: cantidad de memoria que puede reservarse para contenedores YARN en un nodo (si -1 se determina automáticamente, si la detección está habilitada)

Comando yarn

Permite lanzar y gestionar trabajos en YARN:

- `yarn jar`: ejecuta un fichero jar
- `yarn application`: información sobre las aplicaciones ejecutándose en YARN
- `yarn container`: información sobre los contenedores
- `yarn node`: información sobre los nodos
- `yarn top`: información sobre el uso del cluster
- `yarn rmadmin`: comandos para la administración del cluster

Más información: hadoop.apache.org/docs/stable3/hadoop-yarn/hadoop-yarn-site/YarnCommands.html

Mapreduce en Hadoop

Hadoop incorpora una implementación de MapReduce

- Programable en Java
- Uso de otros lenguajes mediante sockets (C++) o Streaming (Python, Ruby, etc.)

Mapreduce en Hadoop

Múltiples propiedades configurables (fichero `mapred-site.xml`)

- `yarn.app.mapreduce.am.resource.cpu-vcores`: cores virtuales usados por el AM
- `yarn.app.mapreduce.am.resource.mb`: cantidad de memoria requerida para el AM
- `yarn.app.mapreduce.am.command-opts`: opciones Java para el AM
- `mapreduce.{map,reduce}.cpu.vcores`: cores solicitados al scheduler para cada tarea map/reduce
- `mapreduce.{map,reduce}.memory.mb`: memoria solicitada al scheduler para cada tarea map/reduce
- `mapreduce.{map,reduce}.java.opts`: opciones Java para los contenedores

Comando mapred

Permite gestionar trabajos MapReduce:

- `mapred job`: interactúa con trabajos MapReduce
- `mapred archive`: crea archivos .har (más información en la **Hadoop Archives Guide**)
- `mapred distcp`: copia recursiva entre clusters Hadoop (más información en la **Hadoop DistCp Guide**)

Más información: hadoop.apache.org/docs/stable3/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapredCommands.html

Parámetros de configuración de YARN y MapReduce

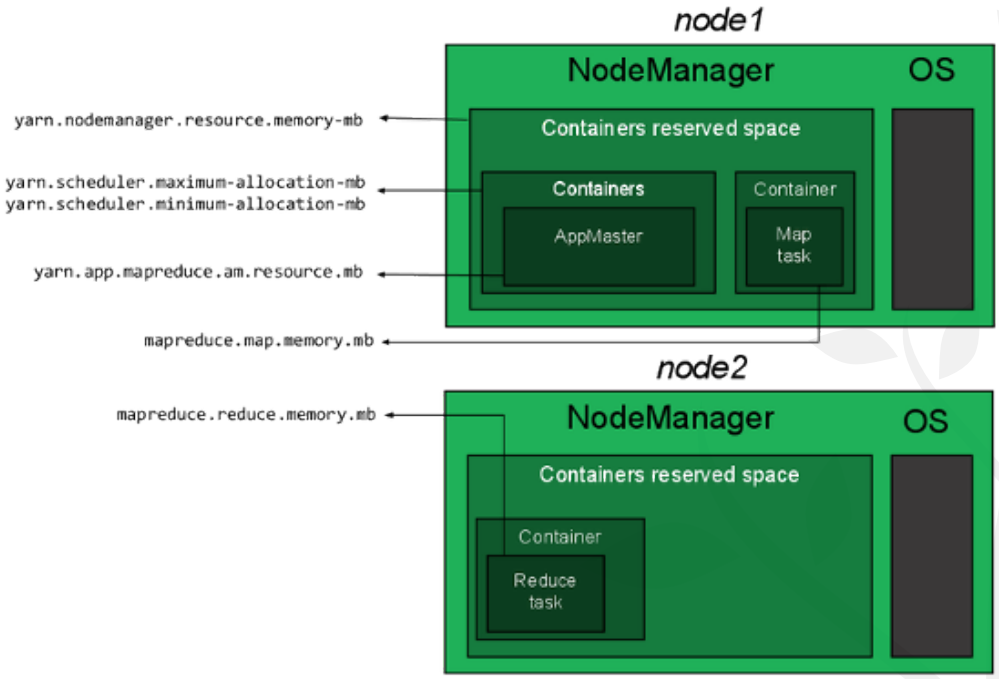
Necesitamos balancear el uso de RAM, cores y discos

- Ajustar los parámetros de Hadoop al hardware disponible

Los parámetros más sensibles son los referidos a la memoria

- yarn.scheduler.maximum-allocation-mb, yarn.scheduler.minimum-allocation-mb, yarn.nodemanager.resource.memory-mb
- yarn.app.mapreduce.am.resource.mb, yarn.app.mapreduce.am.command-opts, mapreduce.map.memory.mb, mapreduce.reduce.memory.mb, mapreduce.map.java.opts, mapreduce.reduce.java.opts

Memoria en YARN y MapReduce



Hadoop puede seleccionar el valor yarn.nodemanager.resource.memory-mb de forma automática.

Fuente: https://docs.deistercloud.com/Technology.50/Hadoop/Hadoop cluster.20.xml

Estimación de los valores

No existe una fórmula mágica para determinar los mejores valores

- Diferentes ajustes para diferentes cargas de trabajo
- Aproximaciones heurísticas como la presentada por Hortonworks
- Parte de:
  - Memoria disponible por nodo
  - Número de cores por nodo
  - Número de discos por nodo

Memoria disponible por nodo

Memoria total menos la reservada para el sistema

- La reservada será un porcentaje de la total
- Una aproximación es la de la tabla

Memoria total por nodo	Memoria para el sistema
< 8GB	1 GB
8GB - 16 GB	2 GB
24 GB	4 GB
48 GB	6 GB
64 GB - 72 GB	8 GB
96 GB	12 GB
128 GB	24 GB
> 128 GB	MemTotal/8

Número de contenedores por nodo

Función de la memoria disponible, nº de cores y nº de discos:

$$\begin{aligned} \text{Ncontenedores} = \min(&2 \times \text{Ncores}, \\ &1.8 \times \text{Ndiscos}, \\ &\text{RAMdisponible}/\text{TamañoMínimoContenedor}) \end{aligned}$$

Memoria por contenedor

La memoria mínima por contenedor va a depender:

- Del la memoria total del nodo y la memoria disponible
- Del número de contenedores por nodo

Memoria total por nodo	Tamaño Mínimo por Contenedor
< 4GB	256 MB
4 GB - 8 GB	512 MB
8 GB - 24 GB	1024 MB
> 24 GB	2048 MB

RAMporcontenedor = max(TamañoMínimoContenedor,
RAMdisponible/Ncontenedores)

Valores de los parámetros

Parámetro	Valor
yarn.nodemanager.resource.memory-mb	Ncontenedores × RAMporcontenedor
yarn.scheduler.minimum-allocation-mb	RAMporcontenedor
yarn.scheduler.maximum-allocation-mb	Ncontenedores × RAMporcontenedor
mapreduce.map.memory.mb	RAMporcontenedor
mapreduce.reduce.memory.mb	2 × RAMporcontenedor
mapreduce.map.java.opts	0.8 × RAMporcontenedor
mapreduce.reduce.java.opts	0.8 × 2 × RAMporcontenedor
yarn.app.mapreduce.am.resource.mb	2 × RAMporcontenedor
yarn.app.mapreduce.am.command-opts	0.8 × 2 × RAMporcontenedor

Ejemplo

Cada nodo del cluster tiene: 12 cores, 48 GB RAM y 12 discos

- RAMdisponible = 48 GB − 6 GB = 42 GB
- TamañoMínimoContenedor = 2048 MB = 2 GB
- Ncontenedores = min(2×12, 1.8×12, 42/2) = 21
- RAMporcontenedor = max(2, 42/21) = 2 GB

Ejemplo: valores de los parámetros

Parámetro	Valor
yarn.nodemanager.resource.memory-mb	43008
yarn.scheduler.minimum-allocation-mb	2048
yarn.scheduler.maximum-allocation-mb	43008
mapreduce.map.memory.mb	2048
mapreduce.reduce.memory.mb	4096
mapreduce.map.java.opts	-Xmx1638m
mapreduce.reduce.java.opts	-Xmx3276m
yarn.app.mapreduce.am.resource.mb	4096
yarn.app.mapreduce.am.command-opts	-Xmx3276m

Memoria virtual/física

Propiedades en el yarn-site.xml:

- `yarn.nodemanager.{vmem,pmem}-check-enabled`: si **true**, se chequea el uso de la memoria virtual/física
- `yarn.nodemanager.vmem-pmem-ratio`: ratio memoria virtual/física que pueden usar los contenedores

El NodeManager puede chequear el uso de la memoria virtual/física del contenedor, matándolo si:

- Su memoria física excede “mapreduce.{map,reduce}.memory.mb”
- Su memoria virtual excede “yarn.nodemanager.vmem-pmem-ratio” veces el valor “mapreduce.{map,reduce}.memory.mb”

Índice

Ejemplo MapReduce: WordCount

El programa WordCount es el ejemplo canónico de MapReduce

- Veremos una implementación muy simple

Definimos tres clases Java

- Una clase para la operación Map (`WordCountMapper`)
- Una clase para la operación Reduce (`WordCountReducer`)
- Una clase de control, para inicializar y lanzar el trabajo MapReduce (`WordCountDriver`)

Introducción a Hadoop Instalación Introducción a HDFS YARN y MapReduce Ejemplo de programa MapReduce

Ejemplo MapReduce: WordCount

El programa WordCount es el ejemplo canónico de MapReduce

- Veremos una implementación muy simple

Definimos tres clases Java

- Una clase para la operación Map (`WordCountMapper`)
- Una clase para la operación Reduce (`WordCountReducer`)
- Una clase de control, para inicializar y lanzar el trabajo MapReduce (`WordCountDriver`)

CiTIUS Introducción a Hadoop, TCDM 34/40

Mapper

```
public class WordCountMapper
  extends Mapper<LongWritable, Text, Text, IntWritable> {
  @Override
  public void map(LongWritable key, Text value, Context ctxt)
         throws IOException, InterruptedException {
      Matcher matcher = pat.matcher(value.toString());
      while (matcher.find()) {
          word.set(matcher.group().toLowerCase());
          ctxt.write(word, one);
      }
  }
  private Text word = new Text();
  private final static IntWritable one = new IntWritable(1);
  private Pattern pat =
          Pattern.compile("\\b[a-zA-Z\\u00C0-\\uFFFF]+\\b");
}
```


Reducer

```
public class WordCountReducer
  extends Reducer<Text, IntWritable, Text, IntWritable> {
  @Override
  public void reduce(Text key, Iterable<IntWritable> values,
     Context ctxt) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable value : values) {
          sum += value.get();
      }
      ctxt.write(key, new IntWritable(sum));
  }
}
```

Driver (I)

```
public class WordCountDriver
       extends Configured implements Tool {
  public int run(String[] arg0) throws Exception {
    if (arg0.length != 2) {
      System.err.printf("Usar: %s [ops] <entrada> <salida>\n",
            getClass().getSimpleName());
      ToolRunner.printGenericCommandUsage(System.err);
      return -1;
    }
    Configuration conf = getConf();
    Job job = Job.getInstance(conf);
    job.setJobName("Word Count");
    job.setJarByClass(getClass());
    FileInputFormat.addInputPath(job, new Path(arg0[0]));
    FileOutputFormat.setOutputPath(job, new Path(arg0[1]));
```


Driver (II)

```
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setNumReduceTasks(1);

    job.setMapperClass(WordCountMapper.class);
    job.setCombinerClass(WordCountReducer.class);
    job.setReducerClass(WordCountReducer.class);

    return (job.waitForCompletion(true) ? 0 : -1);
  }
  public static void main(String[] args) throws Exception {
   int exitCode = ToolRunner.run(new WordCountDriver(), args);
   System.exit(exitCode);
  }
}
```

Compilación y ejecución

Aspectos a tener en cuenta:

1. La nueva API (desde 0.20.0) se encuentra en `org.apache.hadoop.mapreduce` (la antigua en `org.apache.hadoop.mapred`)
2. Preferiblemente, crear un jar y ejecutarlo con:

```
yarn jar fichero.jar [opciones]
```

   - Para gestionar las aplicaciones, utilizad:
     - en general, la opción `application` del comando `yarn` (`yarn application -help` para ver las opciones)
     - para trabajos MapReduce, la opción `job` del comando `mapred` (`mapred job -help` para ver las opciones)
   - Más información en
     - hadoop.apache.org/docs/stable3/hadoop-yarn/hadoop-yarn-site/YarnCommands.html
     - hadoop.apache.org/docs/stable3/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapredCommands.html

Alternativas a Java

Hadoop Streaming

- API que permite crear códigos map-reduce en otros lenguajes
- Utiliza streams Unix como interfaz entre Hadoop y el código
- Permite usar cualquier lenguaje que pueda leer de la entrada estándard y escribir en la salida estándard (Python, Ruby, etc.)

Hadoop Pipes

- Interfaz C++ a Hadoop MapReduce
- Usa sockets como canal de comunicación entre el NodeManager y el proceso C++ que ejecuta el map o el reduce

Alternativas a Java

Hadoop Streaming

- API que permite crear códigos map-reduce en otros lenguajes
- Utiliza streams Unix como interfaz entre Hadoop y el código
- Permite usar cualquier lenguaje que pueda leer de la entrada estándard y escribir en la salida estándard (Python, Ruby, etc.)

Hadoop Pipes

- Interfaz C++ a Hadoop MapReduce
- Usa sockets como canal de comunicación entre el NodeManager y el proceso C++ que ejecuta el map o el reduce