Proyecto: Implementación y Ejecución de Aplicaciones MapReduce en Java

Objetivo general: Desarrollar, compilar, empaquetar y ejecutar dos programas fundamentales de MapReduce en Java dentro de un clúster Hadoop, y aprender a monitorizar la ejecución del *job*.

Fase 1: Configuración del Entorno y Preparación

1. **Configuración del *CLASSPATH***: Los estudiantes deben configurar el entorno para que Hadoop pueda acceder a las librerías de compilación de Java.

◦ Comando: export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar.

2. **Preparación del Workspace y Datos**:

◦ Crear un directorio de trabajo (ejemplo: Prácticas) y copiar los programas Java (ContarPalabras.java, AnalizarLog.java) a esta carpeta.

◦ Asegurarse de que los ficheros de entrada (por ejemplo, quijote.txt o un fichero *accesslog* para el Ejercicio 1, y log1.log para el Ejercicio 2) estén subidos en el HDFS, idealmente en la carpeta /practicas.

3. **Activación de Monitorización**: Es fundamental activar el servidor de historial de trabajos para poder ver los detalles de los *mappers* y *reducers* ejecutados en la consola web.

◦ Comando para iniciar el History Server: Mr. Job History. Daemon PSH start history Server (usando el *script* SHIELD en el directorio *sbin*).

◦ Verificar la disponibilidad de la página web de administración (e.g., nodo1:8088).

Fase 2: Ejercicio 1 - Contador de Palabras (ContarPalabras.java)

Este programa se utiliza para contar el número de veces que aparece cada palabra en un documento.

2.1. Desarrollo y Estructura (Análisis del código)

Los alumnos deben identificar las tres partes esenciales del programa ContarPalabras.java:

• **Mapper** (TokenizerMapper): Extiende Mapper<Object, Text, Text, IntWritable>. Su función map recibe cada línea del fichero de entrada, la tokeniza y emite pares (palabra, 1) por cada palabra que encuentra.

• **Reducer** (IntSumReducer): Extiende Reducer<Text,IntWritable,Text,IntWritable>. Su función reduce recibe todos los valores con la misma clave (la palabra) y suma las ocurrencias, devolviendo la clave y la suma total.

• **Clase Principal (main)**: Configura el *job*, establece las clases Mapper y Reducer, y define las rutas de entrada y salida.

2.2. Compilación y Empaquetado

1. **Compilación**: Compilar el programa usando el comando de Hadoop: hadoop com.sun.tools.javac.Main ContarPalabras.java.

◦ Resultado esperado: Se deben generar tres ficheros .class (uno para la clase principal, otro para el Mapper, y otro para el Reducer).

2. **Generación del JAR**: Crear un archivo JAR que contenga las clases compiladas (ejemplo: mi_libreria.jar o contarpalabras.jar): jar cf mi_libreria.jar Contar*.class.

2.3. Ejecución y Verificación

1. **Ejecución del Job**: Ejecutar la aplicación MapReduce, especificando el JAR, la clase principal, el path de entrada en HDFS (e.g., /practicas/quijote.txt), y la ruta de salida (e.g., /resultado3).

◦ Comando: hadoop jar mi_libreria.jar ContarPalabras /practicas/quijote.txt /resultado3.

2. **Monitorización**: Mientras el trabajo se ejecuta, los alumnos deben observar la conexión al Resource Manager y el progreso (porcentaje de map y reduce) en la consola y en la interfaz web de administración (8088).

◦ Deben verificar cuántos *mappers* y *reducers* fueron lanzados y en qué nodos se ejecutaron, utilizando la vista de *History*.

3. **Verificación de Salida**: Listar y visualizar el contenido del directorio de salida en HDFS para confirmar el conteo de palabras.

◦ Comandos: hdfs dfs -ls /resultado3 y hdfs dfs -cat /resultado3/part-r-00000.

Fase 3: Ejercicio 2 - Análisis de Logs (AnalizarLog.java)

Este programa se utiliza para analizar un fichero de log de una página web y determinar el tamaño máximo, la media y el mínimo de los ficheros solicitados.

3.1. Desarrollo y Estructura (Análisis del código)

• **Mapper** (AMapper): Utiliza un patrón de expresión regular (httplogPattern) para parsear las líneas de log, que tienen un formato específico (indicando la llamada y el tamaño devuelto, e.g., 200 6245). El *mapper* extrae el tamaño del fichero (size) usando el grupo 5 de la expresión regular y lo emite con una clave fija ("msgSize").

• **Reducer** (AReducer): Recibe todos los tamaños asociados a la clave "msgSize". Itera sobre los valores para calcular la suma total (tot), el conteo (count), el mínimo (min) y el máximo (max).

◦ La salida del *reducer* escribe los resultados finales: Mean, Max y Min.

3.2. Compilación y Empaquetado (Actualización del JAR)

1. **Compilación**: Compilar el nuevo programa: hadoop com.sun.tools.javac.Main AnalizarLog.java.

2. **Actualización del JAR**: Añadir las nuevas clases (Analizar*.class) al archivo JAR existente (mi_libreria.jar): jar uf mi_libreria.jar Analizar*.class.

3.3. Ejecución y Verificación

1. **Ejecución del Job**: Ejecutar el programa, usando el fichero de log de entrada (e.g., /practicas/log1.log) y un nuevo directorio de salida (e.g., /resultado_log).

◦ Comando: hadoop jar mi_libreria.jar AnalizarLog /practicas/log1.log /resultado_log.

2. **Verificación de Salida**: Visualizar el contenido del fichero de salida part-r-00000 para confirmar los valores calculados de Media, Máximo y Mínimo.

Salida de ejemplo esperada:

Mean 1150

Max 6823936

Min 0

Resumen Conceptual

Este proyecto ilustra cómo MapReduce utiliza la fase **Map** para procesar registros de entrada de forma independiente y extraer la información relevante (en el Contador de Palabras, las palabras y un conteo de 1; en el Analizador de Logs, el tamaño del fichero). La fase **Reduce** se encarga de recibir toda la información asociada a una clave (ya sea una palabra o la clave fija "msgSize") y realizar la agregación final (suma, media, mínimo, máximo). La compilación manual y la creación de un archivo JAR demuestran cómo se prepara un programa Java para su ejecución distribuida en el entorno Hadoop