Reporte de Evaluación de las Tecnologías Big Data: Hadoop vs. Athena

Autor: Mattias Morales

Recurso crucial: https://github.com/MattiasMR/EV02_advdb

1. Contextualización del Estudio Comparativo

Para este certamen, tuve que realizar un estudio entre dos tecnologías de big data, Hadoop y Athena. Se utilizó un dataset de películas para poder generar consultas y poder evaluar ambas tecnologías. Los criterios de evaluación considerados fueron: tiempos de respuesta, escalabilidad, tolerancia a fallos, requisitos mínimos, capacidad máxima de procesamiento de datos, y soporte para lenguajes de programación. Se enfatizó la necesidad de realizar procesos de análisis complejos, más allá de simples conteos, y asegurar la reproducibilidad de los resultados.

Condiciones del Entorno de Prueba:

- Utilización de NodeJS para Athena: se adjuntan guías de instalación y un ejemplo.
- Utilización de NodeJS o Python en el caso de Hadoop: se adjunta ejemplo con NodeJS.
- Va a ser necesario realizar adecuaciones en el dataset para la ejecución de las pruebas, estas adecuaciones deben ser realizadas por ud. Y deben quedar registradas.

2. Proceso de desarrollo y metodología

2.1 Preprocesamiento del Dataset

Comencé revisando el dataset de películas. Usé un Jupyter Notebook con la librería Pandas para realizar un análisis exploratorio inicial. Este análisis me permitió comprender la estructura de los datos, identificar ciertas columnas interesantes para posibles queries, lo cual quedó todo registrado en el archivo "preProcesamientoDatos.ipynb".

Una vez ya conocía las columnas, con la generación empezamos a hacer un listado de queries tomadas, así que revisando cuáles ya se habían tomado, y mis columnas de interés, fui armando las 5 queries que escogí, de esa manera me podía asegurar que sean complejas y originales.

Después hice un filtrado del dataset, manteniendo únicamente las columnas interesantes para las queries que ya había definido. Luego hice una conversión de tipos de datos y normalización de las fechas al formato YYYY-MM-DD para facilitar su procesamiento.

Finalmente, el dataset pre procesado se exportó en dos formatos: un archivo CSV (codificación UTF-8) destinado a Hadoop y un archivo PARQUET para Amazon Athena. La elección del formato PARQUET para Athena se basó en las buenas prácticas recomendadas para la optimización de tiempos y costos. Es importante destacar que él parseo de campos JSON (como production_companies y genres) se realizó en la etapa de mapeo en Hadoop, y no en el preprocesamiento con Pandas.

2.2 Implementación en Hadoop (MapReduce)

Una vez preparado el dataset, el archivo CSV fue transferido vía SSH a una instancia EC2 (Ubuntu 22.04 LTS noble) con Hadoop v3.3.6 y OpenJDK 8 preinstalados. Utilicé la metodología MapReduce. Ese método consta de un mapper.py encargado de transformar los datos de entrada en pares clave-valor, para que luego un reducer, específico para cada una de las 5 consultas (reducerMC.py, reducerROI.py, reducerDEBUT.py, reducerHG.py, reducerSTAB.py), agrupe los datos por clave y realice las operaciones de agregación correspondientes.

2.3 Implementación en AWS Athena

Para AWS Athena, el archivo PARQUET pre procesado lo subí en un bucket de S3 (s3://ev02-mattiasmorales/input/movies_metadata_final.parquet). Todo Athena lo hice mediante un servicio con NodeJS utilizando los SDKs de AWS. Ocupé el cliente llamado athenaClient.js para ejecutar queries, creé la base de datos ev02 y creé la tabla ev02.movies con setupAthena.js, y expressTest.js para exponer endpoints de mis queries para poder realizar tests.

Entornos y Versiones:

- Hadoop: 3.3.6
- OpenJDK 8.
- JAR streaming: ./hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar
- Athena: database=ev02, table=ev02.movies, resultados: s3://ev02-mattiasmorales/results/
- Node.js: v18.20.8
- Paquetes Node.js:
    - @aws-sdk/client-athena ^3.864.0
    - @aws-sdk/client-s3 ^3.864.0
    - express ^5.1.0
- Archivos clave (Hadoop):
    - movies_metadata_final.csv
    - mapper.py
    - reducerMC.py
    - reducerROI.py
    - reducerDEBUT.py
    - reducerHG.py
    - reducerSTAB.py
- Archivos clave (Athena):
    - movies_metadata_final.parquet
    - athenaClient.js
    - setupAthena.js
    - queries.js

3. Definición de las Consultas (Queries)

Se generaron 5 procesos de análisis complejos, asegurando su originalidad:

- q1 – Concentración de mercado por década (Top-5 productoras por revenue):

Identificación de las cinco compañías productoras con mayores ingresos (revenue) por cada década.

- q2 – ROI vs. decil de presupuesto por década:

Análisis de la relación entre el Retorno de Inversión (ROI) y los deciles de presupuesto (budget) para cada década.

- q3 – "Efecto debut" de productoras:

Comparación del ROI y la popularidad (popularity) de la primera película de una productora frente a las películas posteriores.

- q4 – "Hidden gems" por década (top 10% en vote_average, bottom 20% en popularidad):

Identificación de películas con alta calificación (vote_average) pero baja popularidad en cada década.

- q5 – Estabilidad de calificación según número de géneros:

Evaluación de la correlación entre la cantidad de géneros asignados a una película y la estabilidad de su calificación.

4. Comparaciones

4.1. Tiempo promedio de las queries:

Query	Athena	Hadoop
q1	4.38 s	8.30 s
q2	3.20 s	9.24 s
q3	3.14 s	10.23 s
q4	3.36 s	9.32 s
q5	3.40 s	8.86 s

4.2. Athena vs Hadoop

Criterio	Athena	Hadoop
Tiempos de respuesta	Más rápidos en general (Parquet)	Más lentos en este setup local
Escalabilidad	Serverless; escala por particiones y formatos columnares	Escala horizontal, depende del cluster y su tuning
Tolerancia a fallos	Media muy estricto con el tipo de datos	Alta - bien flexible sobre todo con la compatibilidad con el data science de Python
Requisitos mínimos	Cuenta AWS, S3, permisos	Instalar y gestionar Java, Hadoop, HDFS
Máximo de datos	Alto; limita costo/tiempo antes que la herramienta	Alto; limitado por recursos del cluster
Soporte lenguajes	SQL; orquestación con SDKs (Node, etc.)	Mappers/Reducers en varios lenguajes (Node, Python, etc.)

5. Conclusiones

Athena me rindió mejor en términos de tiempo y facilidad de probar una vez todo ya está listo. Pero el proceso de desarrollo fue altamente más tedioso que el de Hadoop por el único hecho de la diferencia de la compatibilidad de idiomas, la facilidad de hacerlo en Python, sobre todo con mis experiencias previas de análisis de datos lo hizo mucho más fácil de trabajar y procesar el dataset con Hadoop, aunque cabe destacar que en el caso de Athena, utilizar SQL no fue un limitante para nada.

Si tuviese que escoger, me quedaría con Hadoop para hacer cualquier tipo de estudio.

6. Reproducibilidad de los Resultados

> https://github.com/MattiasMR/EV02_advdb

Para la reproducción de los resultados obtenidos en este estudio, se proporcionan las siguientes instrucciones:

Requisitos Previos

> Tener Node.js versión 18.20.8 instalado.

El proyecto no ha sido probado con otras versiones.

- Configurar credenciales AWS en `~/.aws/credentials`

```
region=us-east-1
```

6.1 Hadoop (MapReduce)

Infraestructura requerida:

- Instancia EC2: `t3.micro`
- OS: Ubuntu Server 24.04 LTS x64
- Almacenamiento: 1x8 GiB gp3

Mapper y Reducers:

```
mapper.py

reducerMC.py
reducerROI.py
reducerDEBUT.py
reducerHG.py
reducerSTAB.py
```

Salidas:

```
./hadoop/mc
./hadoop/roi
./hadoop/debut
./hadoop/hg
./hadoop/stab.
```

6.1.0. **Instalar Hadoop**

```
sudo apt update -y && sudo apt-get upgrade -y
sudo apt install openjdk-8-jdk
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-
3.3.6.tar.gz
tar -xvzf hadoop-3.3.6.tar.gz
mv hadoop-3.3.6 hadoop
vim .bashrc
```

Te vas al final del archivo y pegas lo siguiente:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_HOME=$HOME/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

Guardas el archivo

Y terminas con:

```
source .bashrc
```

6.1.1. **Clonar el proyecto**

```
git init
git remote add origin https://github.com/MattiasMR/EV02_advdb.git
git config core.sparseCheckout true
echo "ApacheHadoop/*" > .git/info/sparse-checkout
git pull origin main
cd ApacheHadoop/
```

6.1.2. **Crear directorio y mover el dataset**

```
hdfs dfs -mkdir -p $HADOOP_HOME/input
hdfs dfs -put movies_metadata_final.csv $HADOOP_HOME/input/
```

6.1.3. **Limpiar directorios para inicio limpio**

```
hdfs dfs -rm -r $HADOOP_HOME/mc 2>/dev/null
hdfs dfs -rm -r $HADOOP_HOME/roi 2>/dev/null
hdfs dfs -rm -r $HADOOP_HOME/debut 2>/dev/null
hdfs dfs -rm -r $HADOOP_HOME/hg 2>/dev/null
hdfs dfs -rm -r $HADOOP_HOME/stab 2>/dev/null
```

6.1.4. **Correr las queries**

```
# 1) MC 8.3s
hadoop jar "$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar"
\
  -input $HADOOP_HOME/input/movies_metadata_final.csv \
  -output $HADOOP_HOME/mc \
  -mapper "python3 mapper.py" \
  -reducer "python3 reducerMC.py"

# 2) ROI 9.239s
hadoop jar "$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar"
\
  -input $HADOOP_HOME/input/movies_metadata_final.csv \
  -output $HADOOP_HOME/roi \
  -mapper "python3 mapper.py" \
  -reducer "python3 reducerROI.py"

# 3) DEBUT 10.228s
hadoop jar "$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar"
\
  -input $HADOOP_HOME/input/movies_metadata_final.csv \
  -output $HADOOP_HOME/debut \
  -mapper "python3 mapper.py" \
  -reducer "python3 reducerDEBUT.py"

# 4) Joyitas escondidas 9.319s
hadoop jar "$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar"
\
  -input $HADOOP_HOME/input/movies_metadata_final.csv \
  -output $HADOOP_HOME/hg \
  -mapper "python3 mapper.py" \
  -reducer "python3 reducerHG.py" \
  -cmdenv MIN_VOTES=10

# 5) Estabilidad 8.859s
hadoop jar "$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar"
\
  -input $HADOOP_HOME/input/movies_metadata_final.csv \
  -output $HADOOP_HOME/stab \
```

```
  -mapper "python3 mapper.py" \
  -reducer "python3 reducerSTAB.py"
```

6.1.5. **Visualizar los outputs de las queries**

```
# Década - Etiqueta - % Suma de participaciones de las top 5 productoras -
Etiqueta - Total
# Década - Ranking - ID Productora - Nombre Productora - Revenue de la
productora - % de participación
hdfs dfs -cat $HADOOP_HOME/mc/part-00000 | tr '\t' '|' | column -s '|' -t


# Década - Decil - Promedio ROI (revenue-budget)/budget - cantidad de
películas
hdfs dfs -cat $HADOOP_HOME/roi/part-00000  | tr '\t' '|' | column -s '|' -t


#
idProductora|nombreProductora|fechaEstreno|debutROI|debutPopularidad|noDebu
tPromedioROI|noDebutPromedioPopularidad|nPeliculasNoDebut|deltaROI|deltaPop
ularidad
hdfs dfs -cat $HADOOP_HOME/debut/part-00000 | tr '\t' '|' | column -s '|' -
t | head


# Década - idPelicula - titulo - promedioVotos - popularidad - nVotos
hdfs dfs -cat $HADOOP_HOME/hg/part-00000  | tr '\t' '|' | column -s '|' -t


# década - bucket de géneros - promedioVotos - std de promedio de votos -
nPeliculas
hdfs dfs -cat $HADOOP_HOME/stab/part-00000 | tr '\t' '|' | column -s '|' -t
```

6.2 Athena (NodeJS)

Infraestructura requerida:

- Bucket S3: Uso general, importante recordar el nombre ya que es la ruta.

Ej. ev02-mattiasmorales

> s3://ev02-mattiasmorales

6.2.1. **Clonar el proyecto**

```
git init
git remote add origin https://github.com/MattiasMR/EV02_advdb.git
git config core.sparseCheckout true
echo "Athena/*" > .git/info/sparse-checkout
```

```
git pull origin main
cd Athena/
```

6.2.2. **Setup de Athena y Servidor con Endpoints**

```
npm install
# reemplazar testathenareplicacion con el nombre de su bucket
npm run setup testathenareplicacion && npm run serve
```

6.2.3. **Probar endpoints**

```
curl http://localhost:3000/q/q1 2>/dev/null | head -3
curl http://localhost:3000/q/q2 2>/dev/null | head -3
curl http://localhost:3000/q/q3 2>/dev/null | head -3
curl http://localhost:3000/q/q4 2>/dev/null | head -3
curl http://localhost:3000/q/q5 2>/dev/null | head -3
```