Hadoop y SparkSQL

Kevin Cheves Caicedo
Jose Garcia Lopez
David Zapata Espinoza
Universidad Técnica Estatal de Quevedo
Facultad de Ciencias de la Ingeniería
Quevedo, Ecuador
kevin.cheves2016@uteq.edu.ec
jose.garcial2016@uteq.edu.ec
victor.zapata2016@uteq.edu.ec

Resumen: El procesamiento de Big Data en tiempo real ha recibido una atención creciente, especialmente con la expansión de datos en volumen y complejidad. El big data se crea todos los días, desde el uso de Internet a través de redes sociales, dispositivos móviles, objetos conectados, videos, blogs y otros. Para garantizar un procesamiento de información confiable y rápido en tiempo real, las herramientas poderosas son esenciales para el análisis y procesamiento de Big Data. En este artículo, destacamos la implementación de las herramientas estandar tales como Hadoop MapReduce y también la implementación del framework Apache Spark.

I. Introducción

La cantidad de datos generados todos los días se está expandiendo de manera drástica. Big data es un término popular que se usa para describir los datos que están en zettabyte. Las empresas, organizaciones tratan de adquirir y almacenar datos sobre sus ciudadanos y clientes con el fin de conocerlos mejor y predecir el comportamiento del cliente. Los sitios web de redes sociales generan nuevos datos cada segundo y su manejo es uno de los principales desafíos a los que se enfrentan las empresas. Los datos que se almacenan en almacenes de datos están causando interrupciones porque están en un formato sin procesar, se debe realizar un análisis y procesamiento adecuados para producir información utilizable a partir de ellos.[1]

Se están utilizando nuevas herramientas para manejar una cantidad tan grande de datos en poco tiempo. ApacheHadoop es un marco de programación basado en Java que se utiliza para procesar grandes conjuntos de datos en un entorno informático distribuido. Hadoop se utiliza en un sistema en el que hay varios nodos que pueden procesar terabytes de datos. Hadoop utiliza el algoritmo MapReduce que divide el big data en fragmentos más pequeños y realiza las operaciones en él. Varias tecnologías vendrán de la mano para lograr esta tarea, como Spring Hadoop Data Framework para los fundamentos básicos y la ejecución de Map-Reduce , los sistemas más nuevos, incluidos Apache Flink y Apache Spark, estos se enfocan más en el necesidades de un procesamiento de datos distribuido eficiente: control del flujo de datos (incluido el soporte para el procesamiento iterativo), almacenamiento en caché de datos eficiente y operadores de procesamiento de datos declarativos.[2]

II. Hadoop

Es un marco de programación basado en Java que se utiliza para procesar grandes conjuntos de datos en un entorno informático distribuido. Hadoop se utiliza en un sistema en el que hay varios nodos que pueden procesar terabytes de datos. Hadoop utiliza su propio sistema de archivos HDFS que facilita la transferencia rápida de datos que puede sostener la falla del nodo y evitar la falla del sistema en su conjunto. Hadoop utiliza el algoritmo MapReduce que divide el big data en fragmentos más pequeños y realiza las operaciones en él. Varias tecnologías vendrán de la mano para lograr esta tarea, como Spring Hadoop Data Framework para los fundamentos básicos y la ejecución de Map-Reduce. Jobs, Apache Maven para la construcción distribuida del código, servicios web REST para la comunicación y, por último, Apache Hadoop para el procesamiento distribuido del enorme conjunto de datos.[1]

Hadoop fue desarrollado por MapReduce de Google, que es un marco de software donde una aplicación se divide en varias partes. El ecosistema actual de Appache Hadoop consiste en Hadoop Kernel, MapReduce, HDFS y varios componentes como Apache

Hive, Base y Zookeeper. HDFS y MapReduce como se muestra a continuación.[3]

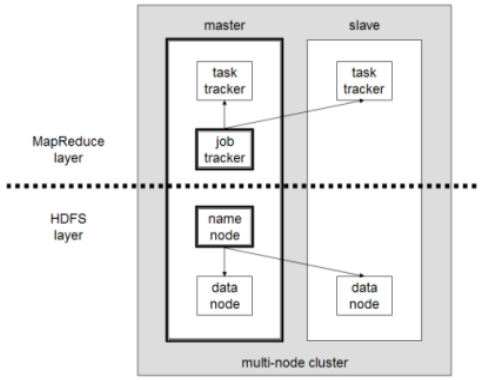

Ilustración 1 Ecosistema de Hadoop

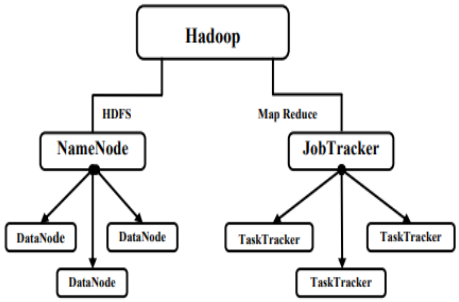

Ilustración 2 Hadoop Daemons

Se ejecuta en hardware básico y utiliza HDFS, que es una arquitectura de almacenamiento en clúster con un gran ancho de banda tolerante a fallas. Ejecuta MapReduce para el procesamiento de datos distribuidos y funciona con datos estructurados y no estructurados.[4]

Hadoop fue diseñado para analizar de forma rápida y fiable los datos estructurados y complejos. Como resultado, muchas empresas han optado por desplegar Hadoop junto a sus demás sistemas informáticos, lo que les permite combinar los datos antiguos y los nuevos de distintas formas novedosas.

Hadoop es muy útil cuando se van a realizar proyectos que necesiten de escalabilidad. Al disponer los datos de forma distribuida, la búsqueda se puede realizar muy rápidamente ya que Hadoop puede acceder a ella de forma paralela. Y aunque los datos estén distribuidos, no hay que preocuparse de fallos ya que dispone de un sistema de seguridad. Esta tecnología fue creada por Doug Cutting, el creador de Apache Lucene, la biblioteca de búsqueda de texto ampliamente utilizada. Hadoop tiene sus orígenes en Apache Nutch, un motor de búsqueda de código abierto, que es en sí una parte del proyecto Lucene.[5]

A. Formas de ejecución

- En modo Standalone: No se necesita configurar nada.
- En modo Servidor - nodo local: Un sistema basado en cliente servidor, pero que se ejecuta en modo local todo.
- En modo distribuido: Infraestructura completa con varios nodos de almacenamiento, ejecución, etc.

B. Versiones de Hadoop

Existen diversas versiones que se pueden agrupar en Hadoop 1 y Hadoop 2 según los Daemons que las componen:[6]

Hadoop 1:

- **JobTracker:** corre en el nodo maestro siendo responsable de hablar con el Namenode para determinar la localización de los datos y enviará trabajos a los nodos esclavos que contienen los datos o están próximos a éstos.
- **TaskTracker:** se ejecuta en los nodos esclavos para lanzar tareas MapReduce sobre los datos que contienen (para la fase Map) o que se le mueven (para la fase Reduce)

Hadoop 2: Definido como MRv2 o YARN (*Yet Another Resource NegoCaCor*)

- **ResourceManager**: hacer las veces de JobTracker gesConando y planificando los recursos del clúster.
- **NodeManager**: sería equivalente al TaskTracker con la responsabilidad de los contenedores, monitorizar el uso de recursos y reportarlos al ResourceManager.

C. Yarm

Apache YARN (*Yet Another Resource Negotiator*) es el sistema de gestión de recursos del clúster de Hadoop. YARN se introdujo en Hadoop 2 para mejorar la implementación de MapReduce, pero es lo suficientemente general como para admitir otros paradigmas de computación distribuida.

YARN proporciona API para solicitar y trabajar con recursos del clúster, pero estas API no suelen ser utilizadas directamente por el código de usuario. En cambio, los usuarios escriben en API de nivel superior proporcionadas por marcos de computación distribuida, que a su vez se basan en YARN.y ocultar al usuario los detalles de la gestión de recursos.

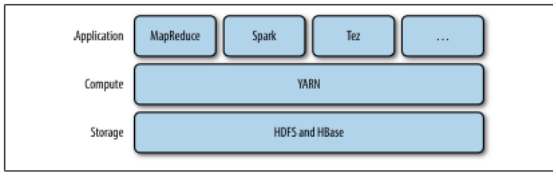

Ilustración 3 YARN applications

D. Hadoop I/O

Hadoop viene con un conjunto de primitivas para la E/S de datos. Algunas de estas son técnicas que son más generales que Hadoop, como la integridad y la compresión de datos, pero merecen una consideración especial cuando se trata de conjuntos de datos de varios terabytes. Otras son herramientas de Hadoop o API que forman los componentes básicos para el desarrollo de sistemas distribuidos, como marcos de serialización y estructuras de datos en disco.

E. Hadoop (HDFS)

El sistema de archivos distribuido de Hadoop (HDFS) es un sistema de archivos distribuido diseñado para ejecutarse en hardware básico. Tiene muchas similitudes con los sistemas de archivos distribuidos existentes. Sin embargo, las diferencias con otros sistemas de archivos distribuidos son significativas. HDFS es altamente tolerante a fallas y está diseñado para implementarse en hardware de bajo costo. HDFS proporciona acceso de alto rendimiento a los datos de la aplicación y es adecuado para aplicaciones que tienen grandes conjuntos de datos. HDFS relaja algunos requisitos POSIX para permitir el acceso de transmisión a los datos del sistema de archivos. HDFS se construyó originalmente como infraestructura para el proyecto del motor de búsqueda web Apache Nutch. es parte del proyecto Apache Hadoop Core.[7]

HDFS está diseñado para almacenar de manera confiable archivos muy grandes en máquinas en un clúster grande. Almacena cada archivo como una secuencia de bloques; todos los bloques de un archivo, excepto el último bloque, tienen el mismo tamaño. Los bloques de un archivo se replican para tolerancia a errores. El tamaño del bloque y el factor de replicación se pueden configurar por archivo. Una aplicación puede especificar el número de réplicas de un archivo. El factor de replicación se puede especificar en el momento de la creación del archivo y se puede cambiar más tarde. Los archivos en HDFS son de una sola escritura y tienen estrictamente un solo escritor en cualquier momento. NameNode toma todas las decisiones con respecto a la replicación de bloques. Recibe periódicamente un Heartbeat y un Blockreport de cada uno de los DataNodes del clúster. La recepción de un latido implica que el DataNode está funcionando correctamente. Un informe de bloques contiene una lista de todos los bloques en un DataNod.[4]

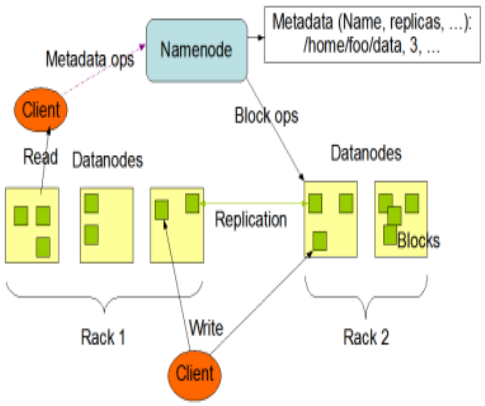

Ilustración 4 Arquitectura de HDFS

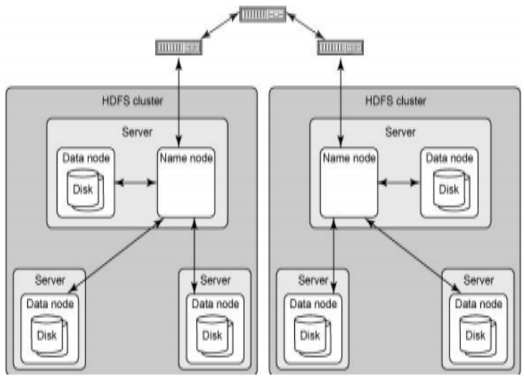

Ilustración 2 HDFS Clusters

Un clúster HDFS está formado por dos tipos de nodos: [8]

- **Datanode (slave):** contienen los bloques de información. Tienen capacidad para la realización de varias tareas simultáneas. Son

los encargados de los procesos de lectura/escritura. En algunos casos estos DataNodes son réplicas de otros. En un sistema de archivos vienen representados por dos ficheros, uno que contiene los datos y otro que contiene los metadatos y el checksum. Todos los DataNodes siempre están identificados por un ID de almacenamiento que les caracteriza frente a un NameNode para permitir su registro y acceso. El sistema soporta entre 10 y 4000 DataNodes.

- **Namenode (master):** es el nodo encargado del cierre, apertura y renombrado de directorios y ficheros. Además, contiene los espacios de nombres del fichero y por lo tanto un árbol con la topología que siguen el resto de nodos. Es también el encargado de la gestión de los metadatos y de la gestión y coordinación de los bloques de datos. Posee el control y la información sobre la asignación de los bloques en los DataNodes, así como de la creación, eliminación, movimiento, renombrado de directorios y supervisión del número de réplicas existentes y de su estado.

F. Integridad de datos en HDFS

HDFS suma de forma transparente todos los datos escritos en él y, de forma predeterminada, verifica las sumas de verificación al leer datos. Se crea una suma de comprobación separada para cada dfs.bytes-perchecksum bytes de datos. El valor predeterminado es 512 bytes y, debido a que una suma de comprobación CRC-32C tiene una longitud de 4 bytes, la sobrecarga de almacenamiento es inferior al 1%.

Los nodos de datos son responsables de verificar los datos que reciben antes de almacenar los datos y su suma de verificación. Esto se aplica a los datos que reciben de los clientes y de otros nodos de datos durante la replicación. Un cliente que escribe datos los envía a una canalización de nodos de datos, y el último nodo de datos de la canalización verifica la suma de comprobación.

Si el nodo de datos detecta un error, el cliente recibe una subclase de IOException, que debe manejar de una manera específica de la aplicación (por ejemplo, reintentando la operación).

G. MapReduce

El pilar de procesamiento en el ecosistema de Hadoop es el marco MapReduce. El marco permite aplicar la especificación de una operación a un gran conjunto de datos, dividir el problema y los datos y ejecutarlos en paralelo. Desde el punto de vista de un analista, esto puede ocurrir en múltiples dimensiones.

Por ejemplo, un conjunto de datos muy grande se puede reducir a un subconjunto más pequeño donde se pueden aplicar análisis. En un escenario de almacenamiento de datos tradicional.[3]

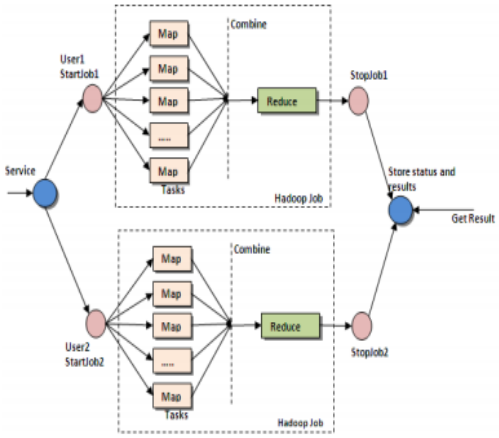

Ilustración 5 Arquitectura de MaoReduce

MapReduce es un modelo de programación para procesar conjuntos de datos a gran escala en grupos de computadoras. El modelo de programación MapReduce consta de dos funciones, map () y reduce (). Los usuarios pueden implementar su propia lógica de procesamiento especificando una función map () y reduce () personalizada. La función map () toma un par clave / valor de entrada y produce una lista de pares clave / valores intermedios. El sistema de tiempo de ejecución MapReduce agrupa todos los pares intermedios en función de las claves intermedias y las pasa a la función reduce () para producir los resultados finales. [4]

a. Map

(in_key,in_value)>list(out_key,intermediate_value)

b. Reduce

(out_key,list(intermediate_value))>list(out_value)

El framework Hadoop clasifica la salida de la fase Map que luego se da como entrada a la

fase Reduce para iniciar las tareas de reducción paralelas. Estos archivos de entrada y salida se almacenan en el sistema de archivos. De forma predeterminada, el marco MapReduce obtiene conjuntos de datos de entrada de HDFS. No es necesario que las tareas de Map y Reduce procedan de manera secuencial, es decir, las tareas de reduce pueden comenzar tan pronto como cualquiera de las tareas de Map complete su trabajo asignado. Tampoco es necesario que todas las tareas de Map se completen antes de que cualquier tarea de Reduce comience a funcionar. MapReduce funciona con pares clave-valor. Conceptualmente, una tarea de MapReduce toma el conjunto de datos de entrada como un par clave-valor y da salida en forma de par clave-valor solo procesando conjuntos de datos de entrada a través de las fases de MapReduce. La salida generada por la fase Map se denomina resultados intermedios que se dan como entrada para reducir la fase.[9]

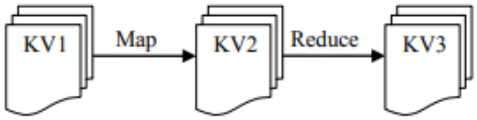

Ilustración 6 MapReduce key-value pairs

Similar a HDFS, MapReduce también explota la arquitectura maestra / esclavo en la que el daemon JobTracker se ejecuta en el nodo maestro y el daemon TaskTracker se ejecuta en cada nodo slave. La capa de procesamiento de MapReduce se compone de dos daemons: [9]

a) **JobTracker:** el servicio JobTracker se ejecuta en el nodo maestro y monitorea las tareas de MapReduce ejecutadas por TaskTracker en los nodos salve. El usuario en interacción con el nodo maestro envía el trabajo al JobTracker. JobTracker luego le pide a NameNode la ubicación real de los datos en HDFS para ser procesados. JobTracker ubica TaskTracker en los nodos esclavos y envía los trabajos a TaskTracker en los nodos esclavos. TaskTracker envía un mensaje de latido a JobTracker periódicamente para asegurarse de que TaskTracker en un nodo esclavo en particular esté activo y ejecutando su trabajo asignado. Si el mensaje de latido no se recibe dentro de un tiempo estipulado, entonces TaskTracker en un nodo de bálsamo en particular se considera no funcional y el trabajo asignado se programa en otro TaskTracker. JobTracker y askTracker se conocen como el motor MapReduce. JobTracker es un punto único de falla para el servicio Hadoop MapReduce, si falla, se detendrán todos los trabajos en ejecución.

b) **TaskTracker:** un demonio TaskTracker se ejecuta en los nodos esclavos de un clúster. Acepta trabajos de JobTracker y ejecuta operaciones MapReduce. Cada TaskTracker tiene un número finito de "espacios de tareas" según la capacidad de un nodo. El protocolo de latido permite a JobTracker saber cuántos "espacios de tareas" están disponibles en TaskTracker en un nodo esclavo. Es tarea de JobTracker asignar los trabajos apropiados al TaskTracker en particular dependiendo de cuántos espacios libres para tareas estén disponibles.

H. Funcionamiento

Desde que el cliente realiza la petición hasta que esta es completada, los sucesos que se producen son los siguientes:[8]

- El cliente realiza una petición, que es recibida por el nodo JobTracker.
- El nodo JobTracker una vez ha recibido la petición, dependiendo del tamaño de esta calcula si se la debe de asignar a uno o a varios nodos TaskTrackers. Para ello divide la petición en diferentes partes de entre 16 y 128 megabytes (este valor es ajustable por el usuario dependiendo de su preferencia). Tras esto busca tantos TaskTrackers en reposo como necesite, tanto para ejecutar las operaciones map como para las operaciones reduce. Finalmente envía las porciones de trabajo a los elegidos como nodos para la función map.
- Estos TaskTrackers realizan su tarea en paralelo. Van almacenando sus resultados en una memoria cache.
- Tras esto comienza la etapa “shuffle”. Todos los valores intermedios obtenidos de los procesos map se organizan teniendo en cuenta la clave que se le asigno en la etapa map y se les envía a los nodos encargados del proceso reduce.
- Estas respuestas se combinan mediante el método reduce
- El resultado final se le devuelve al JobTracker que inicio la comunicación una vez todos los procesos map y reduce han terminado.
-

Problemas

Los principales problemas que pueden existir en la funcionalidad MapReduce son la falta de memoria, la caída de un nodo o un hardware con características insuficientes. Ante los problemas de insuficiencia de memoria o hardware con capacidades de procesamiento insuficientes, MapReduce tiene la capacidad de ser escalable, lo que convierte un problema en una gran cualidad de MapReduce.

Frente a problemas de caídas de nodos TaskTracker, tiene la capacidad de activar otro nodo, que se encuentre en ese momento en reposo, para continuar la operación que el anterior no ha podido ejecutar. Es decir, envía a un nuevo nodo el trabajo que el anterior no ha podido terminar antes de que se cayera, reinicia el nodo caído y le devuelve al estado de reposo, de este modo queda ocioso para nuevas peticiones. Esto es posible debido a que el nodo JobTracker está siempre en continuo contacto con el resto de nodos para así detectar las caídas o fallos que se produzcan. [8]

III. Spark

Apache Spark es un motor de computación en clúster de propósito general con API en Scala, Java y Python y bibliotecas para transmisión, procesamiento de gráficos y aprendizaje automático [6]. Lanzado en 2010, es uno de los sistemas más utilizados con una API "integrada en el lenguaje" similar a DryadLINQ, y el proyecto de código abierto más activo para el procesamiento de big data. Spark tuvo más de 400 colaboradores en 2014 y está empaquetado por varios proveedores.[10]

Apache Spark llegó para acelerar el tiempo de procesamiento de Hadoop y para utilizar de manera eficiente más tipos de operaciones. Estas operaciones pueden incluir consultas interactivas y procesamiento de secuencias. En estas operaciones, la principal preocupación es mantener una alta velocidad en el procesamiento de grandes conjuntos de datos para reducir el tiempo de espera entre consultas y el tiempo de espera para ejecutar un programa. Como Spark tiene su propio administrador de distribución de computación, se puede usar sin Hadoop. Dicho esto, Spark todavía se puede implementar usando Hadoop. Spark puede usar Hadoop para almacenamiento o procesamiento. Como Spark tiene sus propios componentes SQL internos, no necesita ningún almacén para instalar.[11]

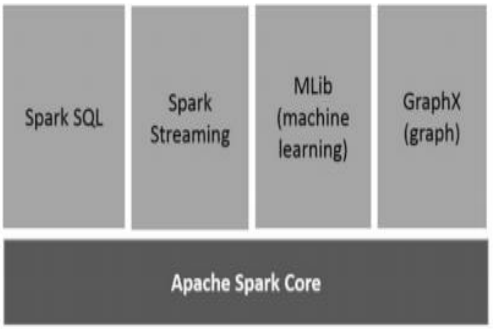

Ilustración 7 Spark Components

Spark está escrito en Scala, el cual es más conciso, permitiendo más expresividad en menos líneas de código. No necesita Hadoop para ser ejecutado, sin embargo, es compatible tanto con Hadoop como con su estructura de datos, permitiendo la capacidad de procesar datos de cualquier fuente. Existen varios componentes que definen el ecosistema de Apache Spark, entre los cuales se encuentran: [12]

- **Spark Core Engine**, es el motor responsable para la planificación, distribución y monitorización de aplicaciones ejecutadas en un clúster; Spark SQL, el cual da soporte para las consultas interactivas.
- **Spark Streaming,** es un componente que permite procesar y analizar datos caso en tiempo real; MLlib, es la librería de aprendizaje automático de Spark y proporciona una gama amplia de algoritmos de alta calidad.
- **GrapX,** es un motor para el análisis de grafos, permitiendo al usuario crear, transformar y obtener conclusiones sobre datos estructurados; SparkR, es un paquete que integra Spark con el lenguaje estadístico R.

Spark es un marco de computación en clúster de código abierto desarrollado originalmente en AMPLab en UC Berkeley. Al permitir que los programas de usuario carguen datos en la memoria de un clúster y los consulten repetidamente, Spark se está convirtiendo en la tecnología central de big data y computación en la nube. El proyecto Spark integra SparkSQL, la tecnología Spark Streaming, resolviendo el big data en lotes,

transmisión, consultas ad hoc y otros tres problemas centrales.[13]

Apache Spark convierte los datos de entrada en conjuntos de datos distribuidos resilientes (RDD) que “permiten a los programadores realizar cálculos en memoria en grandes clústeres de manera tolerante a fallas”. Las transformaciones perezosas se aplican a los RDD, creando nuevos RDD, donde la ejecución de dichas transformaciones no ocurre hasta que sea necesario para el consumo de una "acción", por ejemplo, para la recopilación en el controlador o para el almacenamiento en HDFS. La API principal de Spark presenta varias transformaciones y acciones genéricas, y se realiza a través de uno de los tres administradores de recursos: Spark Standalone; Hadoop YARN; o Apache Mesos. Se han creado API adicionales sobre la API principal para proporcionar un soporte de mayor nivel para varios contextos. Estas API se proporcionan para diferentes lenguajes de programación. La API principal actualmente es compatible con Scala, Java, Python y R.[2]

La iteración se puede realizar de forma similar a Apache Hadoop MapReduce; usando un bucle en el controlador. Sin embargo, en lugar de configurar, iniciar y bloquear nuevos trabajos que escriben desde y hacia HDFS, Spark simplemente aplicaría transformaciones perezosas adicionales y seguiría buscando la siguiente acción. Spark principalmente realiza cálculos en memoria en un intento de minimizar la comunicación del disco. [2]

A. Implementación de Apache Spark

Spark está diseñado para funcionar en un entorno Hadoop y para cumplir con las limitaciones de MapReduce, es la alternativa de map/reduce por lotes. Spark se puede utilizar para realizar procesos de datos en tiempo real y consultas rápidas que finalizan en unos segundos.[14]

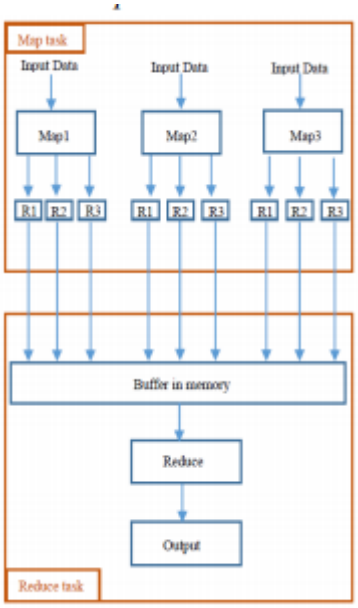

Ilustración 8 Map Reduce in Spark

- Escribir la salida de la fase Map en la memoria caché del búfer del sistema operativo
- Decidir si los datos permanecerán en el búfer en la memoria o se derramarán en el disco
- Fusionando La salida de Map proveniente de los mismos núcleos en un solo archivo
- Generando varios archivos de derrames como el número de Reductores
- Enviar los datos como archivos intermedios a los Reductores
- Escribiendo los datos en la memoria de Reducir.
- Procesando los datos mediante la función Reducir.

Spark copia los datos en la RAM (procesamiento en memoria). Este tipo de procesamiento reduce el tiempo necesario para interactuar con los servidores físicos y esto hace que Spark sea más rápido que Hadoop MapReduce. Para la recuperación de datos en caso de falla, Spark usa RRD.

El motor computacional en memoria, Spark, alivia las costosas E/S de disco para almacenar los resultados intermedios, mejora significativamente el rendimiento, especialmente para los cálculos interactivos e iterativos. Spark proporciona numerosas API, incluido MapReduce, para una programación eficiente. Hoy en día, en la llamada era 'post-Hadoop', el sistema de archivos distribuido de Hadoop (HDFS) sigue siendo muy poderoso para admitir el procesamiento de big data de una manera rentable, administrando datos

masivos en el lago de datos de Hadoop, y es la solución de almacenamiento más popular para Apache Big Data Stack (ABDS). Además, el paradigma MapReduce también se desarrolla y aplica a HPC y problemas interactivos y en tiempo real. Ha sido ampliamente adoptado en investigaciones científicas, como minería de datos, procesamiento gráfico y análisis genético.[15]

B. Spark SQL

Se lanzó en mayo de 2014 y ahora es uno de los componentes desarrollados más activamente en Spark. Spark SQL ya se ha implementado en entornos de gran escala.

Por ejemplo, una gran empresa de Internet utiliza Spark SQL para crear canalizaciones de datos y ejecutar consultas en un clúster de 8000 nodos con más de 100 PB de datos. Cada consulta individual opera regularmente en decenas de terabytes. Además, muchos usuarios adoptan Spark SQL no solo para consultas SQL, sino en programas que lo combinan con el procesamiento de procedimientos. Por ejemplo, 2/3 de los clientes de Databricks Cloud, un servicio alojado ejecutando Spark, use Spark SQL dentro de otros lenguajes de programación. En cuanto al rendimiento, encontramos que Spark SQL es competitivo con los sistemas solo SQL en Hadoop para consultas relacionales. También es hasta 10 veces más rápido y más eficiente en memoria que el código Spark ingenuo en cálculos expresables en SQL. [10]

C. DataFrames

Si bien los RDD ofrecen un control granular, pueden ser más lentos que sus contrapartes de Scala y Java cuando se implementan en Python. La solución a esto fue la creación de una nueva estructura de datos: Spark DataFrames. Al igual que los RDD, los DataFrames son colecciones distribuidas inmutables de objetos; sin embargo, a diferencia de los RDD, los DataFrames se organizan en columnas con nombre (y tipo). De esta manera, son conceptualmente similares a una base de datos relacional (o un DataFrame de pandas). La diferencia más importante entre una base de datos relacional y Spark DataFrames está en la ejecución de transformaciones y acciones. Cuando se trabaja con DataFrames, Catalyst Optimizer de Spark crea y optimiza un plan de ejecución lógico antes de enviar instrucciones a los controladores. Una vez que se ha formado el plan lógico, se crea y ejecuta un plan físico óptimo. Esto proporciona un aumento significativo del rendimiento, especialmente cuando se trabaja con grandes cantidades de datos. Dado que Catalyst Optimizer funciona de la misma manera en todas las API de lenguaje, DataFrames brinda paridad de rendimiento a todas las API de Spark.[16]

D. Spark SQL and DataFrames

Crear un DataFrame a partir de un archivo de texto, csv o JSON existente es generalmente más fácil que crear un RDD. La API de DataFrame también tiene argumentos para lidiar con los encabezados de los archivos o para inferir automáticamente el esquema.

Los DataFrames se pueden actualizar, consultar y analizar fácilmente mediante operaciones SQL. Spark le permite ejecutar consultas directamente en DataFrames de manera similar a cómo realiza transformaciones en RDD.

Además, el módulo pyspark.sql.functions contiene muchas funciones adicionales para un análisis más detallado. [16]

IV. Conclusiones

El paradigma de programación Hadoop MapReduce y HDFS se utilizan cada vez más para procesar conjuntos de datos grandes y no estructurados. Hadoop permite interactuar con el modelo de programación MapReduce mientras oculta la complejidad de implementar, configurar y ejecutar los componentes de software en la nube pública o privada. Hadoop permite a los usuarios crear un clúster de servidores básicos. MapReduce se ha modelado como una plataforma independiente, como una capa de servicio adecuada para diferentes requisitos de los proveedores de la nube. También permite a los usuarios comprender el procesamiento y análisis de datos.

Apache Spark parece sin duda el candidato perfecto si queremos emplear un framework para programar Big Data, sin embargo, no cuenta con su propio sistema de ficheros distribuido y es donde se apoya en el sistema de Hadoop, HDFS. También permite trabajar con su gestor de recursos Hadoop Yarn y con muchos del resto de módulos con los que cuenta Hadoop.

Es por esto que como conclusión podemos decir que en la actualidad emplear la simbiosis entre Apache Spark + Apache Hadoop es la mejor opción para trabajar con Big Data, haciendo uso del músculo de Hadoop (Hadoop YARN, HDFS…) y el cerebro de Spark.

V. Anexos

SparkSQL Code:

```
#Importar SparkContext y
SparkConf, de Apache Spark al
proyecto en Python mediante la
libreria pyspark
from pyspark import
SparkContext, SparkConf;
#Importar SQLContext de
pyspark.sql para poder realizar
sentencias tipo sql
from pyspark.sql import
SQLContext
#Iniciar la ejecución
if __name__ == "__main__":

#Añadir la configuración a
SparkContext, y dar ese
contexto a SQLContext para
iniciar con el manejo del
DataFrame
  config =
SparkConf().setAppName("Data
Frame Join");
  sc =
SparkContext(conf=config);
  sqlContext = SQLContext(sc);

#Crear el DataFrame del archivo
people.json
  df =
sqlContext.read.json("people.js
on");

#Manejo del DataFrame

#Mostrar el DataFrame(mostrará
solo 20 filas por defecto)
  df.show();

#Mostrar el DataFrame completo
  df.show(df.count());

#Mostrar esquema del DataFrame
  df.printSchema();
  df.describe().show()

#Describir solo una columna
  df.describe('born').show()

#Limpieza de DataFrame,
eliminar datos duplicados
  dataframe_dropdup =
df.dropDuplicates()
  dataframe_dropdup.show()
  dataframe_dropdup.count()

#Sentencias select
  df.select("name").show();
  df.select(df['name'],
df['mother']).show();
  df.select("name", "born",
"sex").show()

#Mostrar Datos mediante filtros
  df.filter(df['died'] >
1800).show();
  df.filter((df["died"] > 1800)
& (df["died"] < 1900)).show()
  df.filter((df.died > 1800) &
(df.died < 1900)).show()

#Mostrar campos especificos de
datos filtrados
  df.select("name", "born",
"sex").filter((df["died"] >
1800) & (df["died"] <
1900)).show()

#Mostrar cierta cantidad de
carcateres de un campo en
especificos
  df.select(df.name.substr(1,
3).alias("title")).show()
  df.select(df.name.substr(1,
3).alias("title"),
df.died).show()

#Condultas más elaboradas,
mostrar un campo haciendoce
valer de operaciones entre
otros campos
  df.select((df.died -
df.born).alias("Age of
death")).show()

#Agrupar el DataFrame

df.groupBy("died").count().show
()

#Agrupar el DataFrame de forma
ascendente y descendente

df.groupBy("died").count().sort
("count",
ascending=False).show(df.count(
))

df.groupBy("died").count().sort
("count",
ascending=True).show(df.count()
)
```

```
#Parar el contexto de spark
(SparkContext)
sc.stop();
```

EJEMPLO HADOOP

Este ejemplo es realizado con una máquina virtual proporcionada por cloudera.com. Esta viene desplegada en un sistema operativo CENTOS el cual funciona como servidor. En este vienen programados los diferentes servicios que necesita hadoop para funcionar y además viene con una JDK Java y el editor Eclipse IDE para desarrollar las aplicaciones .jar que requiera el desarrollador para ejecutar junto con hadoop y realizar los procesos que este programe para el trabajo con big data.

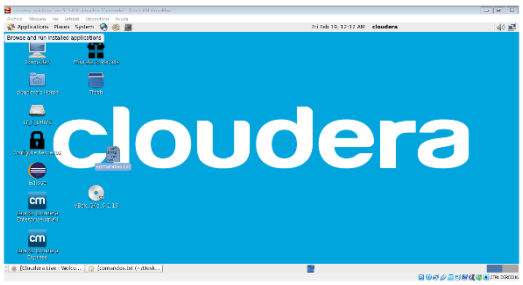

Ilustración 9: Vista de servidor CENTOS

Una vez desplegada la máquina virtual, la cual ejecuta por defecto todos los servicios de hadoop al iniciar, este necesitara de aplicaciones .jar que contengan la lógica sobre la cual se va a basar para procesar los datos, para ello accedemos a la aplicación Eclipse IDE para realizar este proceso.

Dentro del IDE se procede a crear un proyecto asignándole las librerías de hadoop que se encuentra en la ruta /usr/lib/haddop/ y de hadoop client que se encuentra en la ruta /usr/lib/haddop/client/.

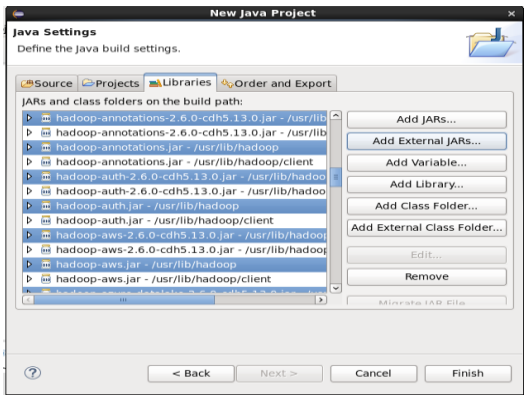

Ilustración 10: Creación de proyecto hadoop

Después, procedemos a crear una un nuevo objeto .java el cual denominaremos WordCount.java sobre el cual se realizará la programación del código MapReduce para hadoop, el código va a ser el siguiente:

1. Las importaciones necesarias para ejecutar el código implementando MapReduce

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import
org.apache.hadoop.mapreduce.lib.input.FileInpu
tFormat;
import
org.apache.hadoop.mapreduce.lib.output.FileOut
putFormat;
```

2. Nombre de la clase `WordCount` que va a contener la funcionalidad MapReduce.

```
public class WordCount {
```

3. Clase estática interna `TokenizerMapper` a la clase `WordCount` la cual va a contener la funcionalidad del map la cual está en la función map la cual se va a encargar de dividir la información a procesar en listas por cada palabra diferente que encuentra.

```
public static class TokenizerMapper
       extends Mapper<Object, Text, Text,
IntWritable>{

    private final static IntWritable one = new
IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value,
Context context
                    ) throws IOException,
InterruptedException {
      StringTokenizer itr = new
StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```

4. Clase estática interna `IntSumReducer` que se encarga de sumar las listas y enviar ese número junto con la key (palabra) que está recibiendo en ese momento, esta es ejecutada en diferentes nodos hijos hadoop.

```
public static class IntSumReducer
       extends
Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new
IntWritable();

    public void reduce(Text key,
Iterable<IntWritable> values,
                       Context context
                       ) throws IOException,
InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }
```

5. El método main que ejecutara la clase principal al iniciar la aplicación. Este obtiene un objeto `Configuration` el cual carga la configuración de hadoop y es pasada a un Job. Después se le pasa al Job la clase que contendrá la clase con hadoop implementada. Después se le pasa la clase estática que contendrá la funcionalidad map y seguido a esto se le pasa la clase que contendrá la funcionalidad reduce.
   Después se le pasa una clase que servirá de formato para la salida de los datos `Text.class` para el key y `IntWritable.class` para el value.
   Una vez configurado el job se le añade el archivo de entrada de los datos a procesar con `FileInputFormat.addInputPath(job, new Path(args[0]));` y el archivo de salida con `FileOutputFormat.setOutputPath(job, new Path(args[1]));`.
   Con esto ya se podrá poner en ejecución el job con `System.exit( job.waitForCompletion( true) ? 0 : 1);` esto lo que hace es enviar la tarea a ejecutar y finaliza la aplicación una vez se complete la tarea. La última línea es la de cierre de la clase.

```
public static void main(String[] args) throws
Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word
count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new
Path(args[0]));
    FileOutputFormat.setOutputPath(job, new
Path(args[1]));
    System.exit(job.waitForCompletion(true) ?
0 : 1);
  }
}
```

Una vez programada la aplicación se tiene que crear un .jar, para ello se da clic derecho sobre el proyecto y del menú desplegable se escoge la opción exportar.

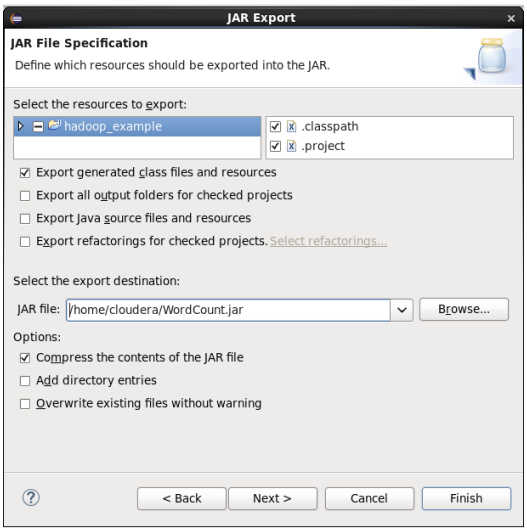

Ilustración 11: Ventana de exportación de aplicación

Ahora se procede a ejecutar la aplicación con hadoop, para ello vamos a crear un archivo de texto con datos de ejemplo con el siguiente código.

cat > /home/cloudera/processFile.txt

El ultimo parámetro es la ruta donde se va a crear el archivo y el nombre con el que se lo va a crear.

```
[cloudera@quickstart ~]$ cat > /home/cloudera/processFile.txt
prueba
preba hadoop
prueba hadoop cloudera
cloudera
sistemas distribuidos
sistemas informaticos
^Z
[1]+  Stopped                 cat > /home/cloudera/processFile.txt
[cloudera@quickstart ~]$
```

Ilustración 12: Creación de archivo de texto de prueba

Ahora ejecutamos el siguiente comando para verificar los archivos y carpetas que tiene el sistema de archivo de hadoop actualmente.

hdfs dfs -ls /

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 10 items
drwxrwxrwx   - hdfs  supergroup          0 2017-10-23 12:15 /benchmarks
drwxr-xr-x   - hbase supergroup          0 2021-02-19 15:31 /hbase
drwxr-xr-x   - hdfs  supergroup          0 2021-02-09 06:26 /inputfolder1
drwxr-xr-x   - hdfs  supergroup          0 2021-02-09 08:31 /inputfolder2
drwxr-xr-x   - hdfs  supergroup          0 2021-02-09 06:34 /ouput1
drwxr-xr-x   - hdfs  supergroup          0 2021-02-09 08:44 /ouput2
drwxr-xr-x   - solr  solr                0 2017-10-23 12:18 /solr
drwxrwxrwt   - hdfs  supergroup          0 2021-02-09 02:59 /tmp
drwxr-xr-x   - hdfs  supergroup          0 2021-02-09 05:55 /user
drwxr-xr-x   - hdfs  supergroup          0 2017-10-23 12:17 /var
[cloudera@quickstart ~]$
```

Ilustración 13: Visualización de información de sistema de archivos hadoop

Ahora procederemos a almacenar el archivo de texto en haddop, para ello se procede a crear una carpeta en el sistema de archivos hadoop con el siguiente comando.

hdfs dfs -mkdir /inputfolder3

```
[cloudera@quickstart ~]$ hdfs dfs -mkdir /inputfolder3
[cloudera@quickstart ~]$
```

Ilustración 14: Crear carpeta en hadoop

Con esto podemos verificar si se creó listando los archivos con uno de los comandos anteriores y después procedemos a enviar el archivo de texto creado a la carpeta destinada en el sistema de archivos hadoop.

Para ello utilizamos el siguiente comando el cual obtiene la ruta de archivo de entrada y la ruta de destino:

hdfs dfs -put /home/cloudera/processFile.txt /inputfolder3/

```
[cloudera@quickstart ~]$ hdfs dfs -put /home/cloudera/processFile.txt /inputfolder3/
[cloudera@quickstart ~]$
```

Ilustración 15: Enviar archivo a sistema de archivo hadoop

Después, procedemos a verificar si se creó correctamente en el Sistema de archivos Hadoop y si los datos están correctos con el siguiente comando

hdfs dfs -cat /inputfolder3/processFile.txt

```
[cloudera@quickstart ~]$ hdfs dfs -cat /inputfolder3/processFile.txt
prueba
preba hadoop
prueba hadoop cloudera
cloudera
sistemas distribuidos
sistemas informaticos
[cloudera@quickstart ~]$
```

Ilustración 16: Visualizar información de archivo en sistema de archivos hadoop

Con los datos listos para procesar se procede a ejecutar la aplicación hadoop que va a procesarlos (WordCount.jar creada anteriormente) con el siguiente comando.

hadoop jar /home/cloudera/WordCount.jar WordCount /inputfolder3/processFile.txt /ouput3

Este comando le dice a Hadoop que va a ejecutar un jar con la ruta /home/cloudera/WordCount.jar, que el nombre de la clase principal que tiene la lógica MapReduce es WordCount, que el archivo a procesar está en la ruta hadoop /inputfolder3/processFile.txt y que los muestre en una carpeta de salida denominada /ouput3 (si no encuentra la carpeta la crea).

```
[cloudera@quickstart ~]$ hadoop jar /home/cloudera/WordCount.jar WordCount /inputfolder3/processFile.txt /ouput3
21/02/19 16:51:02 INFO client.RMProxy: Connecting to ResourceManager at quickstart.cloudera/10.0.2.15:8032
21/02/19 16:51:08 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with t
he Runner to remedy this.
21/02/19 16:51:13 INFO input.FileInputFormat: Total input paths to process : 1
21/02/19 16:51:14 INFO mapreduce.JobSubmitter: number of splits:1
21/02/19 16:51:17 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1613771433589_0001
21/02/19 16:51:21 INFO impl.YarnClientImpl: Submitted application application_1613771433589_0001
21/02/19 16:51:22 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1613771433589_0001/
21/02/19 16:51:22 INFO mapreduce.Job: Running job: job_1613771433589_0001
21/02/19 16:53:12 INFO mapreduce.Job: Job job_1613771433589_0001 running in uber mode : false
21/02/19 16:53:12 INFO mapreduce.Job:  map 0% reduce 0%
21/02/19 16:54:33 INFO mapreduce.Job:  map 100% reduce 0%
21/02/19 16:55:50 INFO mapreduce.Job:  map 100% reduce 100%
21/02/19 16:55:54 INFO mapreduce.Job: Job job_1613771433589_0001 completed successfully
21/02/19 16:55:58 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=112
                FILE: Number of bytes written=294541
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=221
                HDFS: Number of bytes written=78
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=36140032
                Total time spent by all reduces in occupied slots (ms)=38312960
                Total time spent by all map tasks (ms)=70586
                Total time spent by all reduce tasks (ms)=74830
                Total vcore-milliseconds taken by all map tasks=70586
                Total vcore-milliseconds taken by all reduce tasks=74830
                Total megabyte-milliseconds taken by all map tasks=36140032
                Total megabyte-milliseconds taken by all reduce tasks=38312960
        Map-Reduce Framework
                Map input records=6
                Map output records=11
                Map output bytes=140
                Map output materialized bytes=108
                Input split bytes=125
                Combine input records=11
                Combine output records=7
                Reduce input groups=7
                Reduce shuffle bytes=108
                Reduce input records=7
                Reduce output records=7
                Spilled Records=14
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=1376
                CPU time spent (ms)=8080
                Physical memory (bytes) snapshot=249143296
                Virtual memory (bytes) snapshot=1457635328
                Total committed heap usage (bytes)=96468992
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=96
        File Output Format Counters
                Bytes Written=78
```

Ilustración 17:Ejecución de aplicación en hadoop

Con esto ya tenemos los datos procesados con los resultados en la carpeta especificada anteriormente. Con el siguiente comando vemos los archivos re resultado obtenidos.

hdfs dfs -ls /ouput3

```
[cloudera@quickstart ~]$ hdfs dfs -ls /ouput3
Found 2 items
-rw-r--r--   1 hdfs supergroup          0 2021-02-19 16:55 /ouput3/_SUCCESS
-rw-r--r--   1 hdfs supergroup         78 2021-02-19 16:55 /ouput3/part-r-00000
[cloudera@quickstart ~]$
```

Ilustración 18: Archivos resultado de hadoop

Hadoop crear dos archivos de resultado, el archivo _SUCCESS contiene la metadata del proceso y el archivo part-r-00000 contiene los datos de resultado. Con el siguiente comando vemos el resultado obtenido en el archivo part-r-00000.

hdfs dfs -cat /ouput1/part-r-00000

```
[cloudera@quickstart ~]$ hdfs dfs -cat /ouput3/part-r-00000
cloudera        2
distribuidos    1
hadoop  2
informaticos    1
preba   1
prueba  2
sistemas        2
[cloudera@quickstart ~]$
```

Ilustración 19: Datos resultantes de hadoop

Este archive nos muestra cada palabra encontrada y al lado el número de veces que se repite.

Con ello finalizamos el ejemplo de hadoop.

VI. Referencias

[1] J. Nandimath, E. Banerjee, A. Patil, P. Kakade, and S. Vaidya, "Big data analysis using Apache Hadoop," in *Proceedings of the 2013 IEEE 14th International Conference on Information Reuse and Integration, IEEE IRI 2013*, Aug. 2013, pp. 700–703, doi: 10.1109/IRI.2013.6642536.

[2] B. Akil, Y. Zhou, and U. Rohm, "On the usability of Hadoop MapReduce, Apache Spark & Apache flink for data science," in *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017*, Jul. 2017, vol. 2018-Janua, pp. 303–310, doi: 10.1109/BigData.2017.8257938.

[3] G. Tafese and D. Desta, "The Roles of Civics and Ethical Education in Shaping Attitude of the Students in Higher Education: The Case of Mekelle University," *Int. J. Sci. Res. Publ.*, vol. 4, no. 10, pp. 680–683, 2014.

[4] S. G. Manikandan and S. Ravi, "Big data analysis using apache hadoop," in *2014 International Conference on IT Convergence and Security, ICITCS 2014*, Jan. 2014, pp. 1–4, doi: 10.1109/ICITCS.2014.7021746.

[5] F. M. Rodriguez Sánchez, "Herramientas para Big Data: Entorno Hadoop.," *Univ. Politécnica Cart.*, pp. 9–13, 2014.

[6] F. J. L. Sevilla, "Introducción a Hadoop . Instalación en AWS."

[7] Apache Software Foundation, "Apache Hadoop 3.2.0 – HDFS Architecture," *Apache Softw. Found.*, pp. 1–14, 2019.

[8] A. Álvaro Palacios Tolón Tutor Rafael Herradón Díez and U. DE Politécnica Madrid, "BIG DATA: ANÁLISIS Y ESTUDIO DE LA PLATAFORMA HADOOP," 2013.

[9] M. R. Ghazi and D. Gangodkar, "Hadoop, mapreduce and HDFS: A developers perspective," *Procedia Comput. Sci.*, vol. 48, no. C, pp. 45–50, 2015, doi: 10.1016/j.procs.2015.04.108.

[10] V. L. Qin, "Spark SQL: Relational Data Processing in Spark," *Acad. Psychiatry*, vol. 41, no. 6, p. 763, 2017, doi: 10.1007/s40596-017-0796-z.

[11] M. Alsmirat, Y. Jararweh, and M. Al-Ayyoub, "Speeding DBLP querying using hadoop and spark," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 459, no. 1, 2018, doi: 10.1088/1757-899X/459/1/012003.

[12] E. N. Sistemas, Y. Computación, and L. F. De, "PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR FACULTAD DE INGENIERÍA INGENIERÍA DE SISTEMAS Y COMPUTACIÓN DISERTACIÓN PREVIA A LA OBTENCIÓN DEL TÍTULO DE INGENIERA "ANÁLISIS COMPARATIVO DE LAS HERRAMIENTAS DE BIG DATA EN," 2017.

[13] S. Chen, C. Wu, and Y. Yu, "Analysis of Plant Breeding on Hadoop and Spark," *Adv. Agric.*, vol. 2016, pp. 1–6, 2016, doi: 10.1155/2016/7081491.

[14] K. Aziz, D. Zaidouni, and M. Bellafkih, "Real-time data analysis using Spark and Hadoop," in *Proceedings of the 2018 International Conference on Optimization and Applications, ICOA 2018*, May 2018, pp. 1–6, doi: 10.1109/ICOA.2018.8370593.

[15] X. Yang, S. Liu, K. Feng, S. Zhou, and X. H. Sun, "Visualization and adaptive subsetting of earth science data in HDFS: A novel data analysis strategy with Hadoop and Spark," in *Proceedings - 2016 IEEE International Conferences on Big Data and Cloud Computing, BDCloud 2016, Social Computing and Networking, SocialCom 2016 and Sustainable Computing and Communications, SustainCom 2016*, Oct. 2016, pp. 89–96, doi: 10.1109/BDCloud-SocialCom-SustainCom.2016.24.

[16] С. Е. Попов and Р. Ю. Замараев, "Apache Spark," *Программирование*, no. 1, pp. 39–53, 2020, doi: 10.31857/s0132347420010057.