

Spark

ENTENDEMOS POR BIG DATA A DATOS QUE NO PUEDEN SER PROCESADOS POR MÉTODOS TRADICIONALES. POR EJEMPLO DATOS CUYO VOLUMEN REQUIERE QUE SEAN ALMACENADO EN UN CLUSTER O DATOS CUYA VELOCIDAD DE ARRIBO ES ∞ . AL HABLAR DE BIG DATA TENEMOS '5Vs':

- VOLUMEN
- VELOCIDAD
- VARIEDAD
- VERACIDAD
- VALENCIA

ALMACENAMIENTO DISTRIBUIDO

CONSISTE EN ALMACENAR INFORMACIÓN EN VARIOS EQUIPOS.

EN CADA EQUIPO DEL CLUSTER EXISTE UN FILE - SYSTEM LOCAL QUE SOLO ES ACCESIBLE DESDE EL EQUIPO Y UN FILE SYSTEM DISTRIBUIDO QUE ES GLOBAL A TODO EL CLUSTER, SIENDO POSIBLE MOVER ARCHIVOS DE UN EQUIPO A OTRO.

EL GESTOR INTERNO DEL FILE SYSTEM SE ENCARGA DE DIVIDIR CADA ARCHIVO EN 'BLOQUES' Y ALMACENA ESTOS BLOQUES EN DIFERENTES EQUIPOS DEL CLUSTER DE FORMA REPLICADA.

TODOS ESTOS DATOS SON ALMACENADOS Y MANEJADOS POR UN NODO LLAMADO 'MASTER NODE'. ESTE NODO 'SABE' DE QUE FORMA SE HAN REPARTIDO LOS ARCHIVOS EN EL CLUSTER. EL MASTER NODE TIENE UN BACKUP FUERA DE LÍNEA, DE FORMA TAL QUE SU CAIDA NO HAGA UN DESASTRE.

CUANDO SE QUIERE ACCEDER A UN REGISTRO, EL MASTED NODE SE ENCARGA DE DETERMINAR EN QUE EQUIPO ESTA DICHA INFORMACIÓN Y ACCEDER. SI EL EQUIPO NO RESPONDE EL MASTED SE ENCARGA DE ACCEDER A UNA COPIA Y DISPARAR LA CREACIÓN DE UNA NUEVA COPIA EN OTRO NODO DEL CLUSTER. LOS ARCHIVOS DISTRIBUIDOS SON INMUTABLES PUEDE LEERSE Y AGREGAR INFO AL FINAL PERO LA INFO EXISTENTE NO PUEDE MODIFICARSE

MAP REDUCE

MAP REDUCE ES UNA ABSTRACCIÓN INSPIRADA EN EL PARADIGMA DE PROGRAMACIÓN FUNCIONAL QUE PERMITE PROCESAR INFORMACIÓN QUE SE ENCUENTRA ALMACENADA DE FORMA DISTRIBUIDA EN UN CLUSTER. ESTE PROCESAMIENTO DE LA INFORMACIÓN DISTRIBUIDA SE PUEDE DIVIDIR EN 3 FASES:

FASE MAP

SE REALIZAN TODAS LAS OPERACIONES QUE PUEDAN HACERSE DE FORMA ATÓMICA SOBRE CADA REGISTRO. ESTO NOS PERMITE REALIZAR TRANSFORMACIONES DE DATOS CONVIERTIENDO CADA REGISTRO DEL ARCHIVO EN EL FORMATO QUE NECESITEMOS CREANDO UN NUEVO ARCHIVO DISTRIBUIDO, ADÉMÁS PODEMOS FILTRAR EUMINANDO LOS REGISTROS QUE NO NOS SIRVEN.

FASE REDUCE

SE ENCARGA DE LAS OPERACIONES QUE NECESITEN MAS DE UN REGISTRO AL MISMO TIEMPO PARA FUNCIONAR.

HAY DOS FORMAS DE REDUCE:

REDUCE

ESTA FASE SE EN CARGA DE HACER OPERACIONES QUE DEBEN SER ASOCIATIVAS Y COMUTATIVAS ENTRE REGISTROS DEL ARCHIVO DONDE EL RESULTADO PASA A SER PARTE DEL INPUT DE LA SIGUIENTE OPERACIÓN.

REDUCE BY KEY

EN ESTA FASE SUPONEMOS QUE CADA REGISTRO ESTA FORMADO POR UNA TUPLA (CLAVE, VALOR).

EL SISTEMA VA A AGRUPAR TODOS LOS REGISTROS PARA LOS CUALES LA CLAVE ES LA MISMA Y APLICAR UN REDUCE.

FASE SHUFFLE AND SORT

ESTA FASE ES LA ENCAJADA DE HACER QUE EL SISTEMA PUEDA EJECUTAR LA FASE DE REDUCE. ES LA PARTE MÁS COSTOSA DE TODO EL PROCESO.

AHORA NOS INTERESA SPARK; UNA HERRAMIENTA QUE NOS PERMITE EL PROCESAMIENTO DE DATOS DISTRIBUIDOS EN MODO BATCH REDO APROVECHANDO LOS DATOS EN MEMORIA.

- CREAMOS SPARK CONTEXT

SPARK = SPARK SESSION. BUILDER. GETORCREATE()

- PARALELIZAMOS UNA COLECCIÓN DE PYTHON

RDD = SC. PARALLELIZE (LISTA, N) → CANTIDAD DE PARTICIONES

- LEEMOS ARCHIVOS CON TEXTFILE

RDD = SPARKCONTEXT. TEXTFILE (ARCHIVO.TXT)

ACCIONES



CUIDADO!

COUNT

CUENTA LA CANTIDAD

- COUNT()

SI SE QUIERE ACONTAR ALGO ES PECIFICO
DE NTRO DE UN REGISTRO

- COUNT('?')

TAKE

OBTIENE LOS PRIMEROS N REGISTROS DEL RDD.

- TAKE(N)

COLLECT

OBTIENE TODOS LOS REGISTROS DEL RDD

* DEBEMOS SABER QUE EL RDD ES
ACOTADO.

- COLLECT()

FIRST

OBTIENE EL PRIMER REGISTRO DEL RDD

- FIRST()

TAKE ORDERED

OBTIENE LOS PRIMEROS N REGISTROS EN BASE A UN ORDEN INDICADO.

* UTILIZABLE DE MANERA ACOTADA

- TAKE ORDERED (N, KEY = LAMBDA X :
 ASCEND.
 DESCEND.)
↓
CANTIDAD

TAKE SAMPLE

OBTIENE UNA MUESTRA DE N REGISTROS CON O SIN REEMPLAZO

- TAKESAMPLE (BOOL, N)
↓
TRUE FALSE
↓
CANT.

REPETIDAS
O NO

REDUCE

OBTIENE UN SOLO REGISTRO, COMBINANDO EL RESULTADO EN BASE A UNA FUNCIÓN DADA.

- REDUCE (LAMBDA A, B : ...)

TENER EN CUENTA QUE ACA A, B TOMAN TODO EL REGISTRO (CLAVE, VALOR).

SI SE DESEA COMPARAR POR EL VALOR HAY QUE ACCEDER A [1] B[1]

COUNTBYKEY

CUENTA OCURRENCIAS DE REGISTROS PARA CADA CLAVE

RECORDAR QUE UN REGISTRO TIENE UNA TUPLA UNICA (C,V) PERO DENTRO PUEDE HABER ((), ())

- COUNTBYKEY(, ,)

TRANSFORMACIONES

MAP

TRANSFORMA CADA REGISTRO EN BASE A UNA FUNCIÓN DADA

- MAP (LAMBDA X : ---)

PODÉ LO GENERAL LA UTILIZAMOS PARA GENERAR NUEVOS BDD APLICANDOLE CAMBIOS A SUS COLUMNAS.

FILTER

FILTRA REGISTROS EN BASE A UNA FUNCIÓN DADA

- FILTER (LAMBDA X : ---)

FLAT MAP

SIMILAR A MAP PERO CADA REGISTRO PUEDE GENERAR 0, 1 O MAS REGISTROS

- FLATMAP (LAMBDA X : [(X), (X-1)])

↓ A MODO DE EJ

REDUCE BY KEY

COMBINA LOS REGISTROS PARA UNA MISMA CLAVE EN BASE A UNA FUNCIÓN DE REDUCE.

LA FUNCIÓN DE REDUCE DEBE SER COMUTATIVA Y ASOCIATIVA.

- REDUCEBYKEY(LAMBDA A,B : ...)

TENED EN CUENTA QUE A Y B SON LOS VALORES.

GROUP BY KEY

AGRUPA LOS REGISTROS PARA CADA CLAVE. SE OBTIENE TODOS LOS REGISTROS PARA CADA CLAVE.



SOLO SE DEBE UTILIZAR SI ES NECESARIO LA INFO DE CADA REGISTRO Y ES ACOTADO

- GROUPBYKEY()

DISTINCT

ELIMINA REGISTROS DUPLICADOS (TODO EL REGISTRO DEBE COINCIDIR)

- DISTINCT()

PUEDE UTILIZARSE PARA OBTENER PALABRAS ÚNICAS

SORT BY KEY

ORDENA LOS REGISTROS POR CLAVE.

- SORTBYKEY()

TRANSFORMACIONES ENTRE DOS RDD

UNIÓN

REALIZA LA UNION ENTRE DOS RDD
(NO IMPORTA SI LOS REGISTROS SON IGUALES O NO)

$$\text{UNION} = \text{RDD1}. \text{UNION}(\text{RDD2})$$

INTERSECCIÓN

REALIZA LA INTERSECCIÓN ENTRE DOS RDD DE REGISTROS.

$$\text{INTERSECT} = \text{RDD1}. \text{INTERSECTION}(\text{RDD2})$$

SUBSTRACT

ELIMINA DEL PRIMER RDD LOS REGISTROS QUE APAREZCAN EN EL SEGUNDO RDD.

$$\text{SUBTRACT} = \text{RDD1}. \text{SUBTRACT}(\text{RDD2})$$

JOINS

CON LOS JOINS SE COMBINAN DOS RDD EN BASE A LAS CLAVES DE LOS REGISTROS. JUNTA CADA REGISTRO DEL RDD1 CON CADA REGISTRO RDD2 QUE TENGAN MISMA CLAVE.

JOIN

SI TENGO (k, v) (k, w) DEVUELVE $(k, (v, w))$
CON TODOS LOS PARES DE ELEMENTOS PARA CADA CLAVE (LAS QUE ESTAN EN COMÚN EN AMBOS RDD)

$$\text{JOIN} = \text{RDD1}. \text{JOIN}(\text{RDD2})$$

LEFT OUTER JOIN

SI TENGO (k, v) (k, w) DEVUELVE $(k, (v, w))$
ASEGURANDO QUE TODOS LOS DATOS DE LA IZQ ESTABAN

$$\text{LEFTJOIN} = \text{RDD1}. \text{LEFTOUTERJOIN}(\text{RDD2})$$

RIGHT OUTER JOIN

ES ANALOGO AL LEFT JOIN, PERO SE ASEGURAN LOS DE LA DERECHA

RIGHTJOIN = RDD1. OUTER RIGHT JOIN (RDD2)

PRIMERO
RIGHT

FULL JOIN

SI TENGO $(k, v)(k, w)$ DEVUELVE $(k, (v, w))$ ASEGURANDOSE QUE TODOS LOS DATOS ESTEN, AUNQUE LAS CLAVES NO CONCUERDEN (MATCH)

FULL = RDD1. FULLOUTERJOIN (RDD2)