

Datos Masivos I

2. Modelo de mapeo y reducción

Profa. Alondra Berzunza





Unidad 2 - Modelo de mapeo y reducción

2.1 Sistema de almacenamiento y procesamiento distribuido

2.2 Modelo de programación

2.3 Algoritmos con el modelo de mapeo y reducción

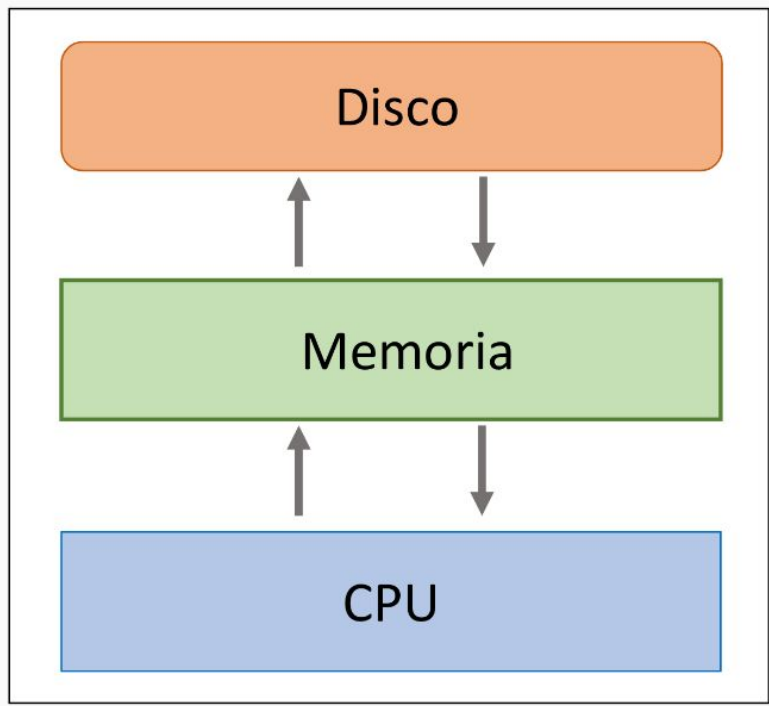
2.4 Extensiones

2.5 El modelo costo-comunicación

2.6 Teoría de la complejidad para el modelo de mapeo y reducción



2.1 Sistema de almacenamiento y procesamiento distribuido

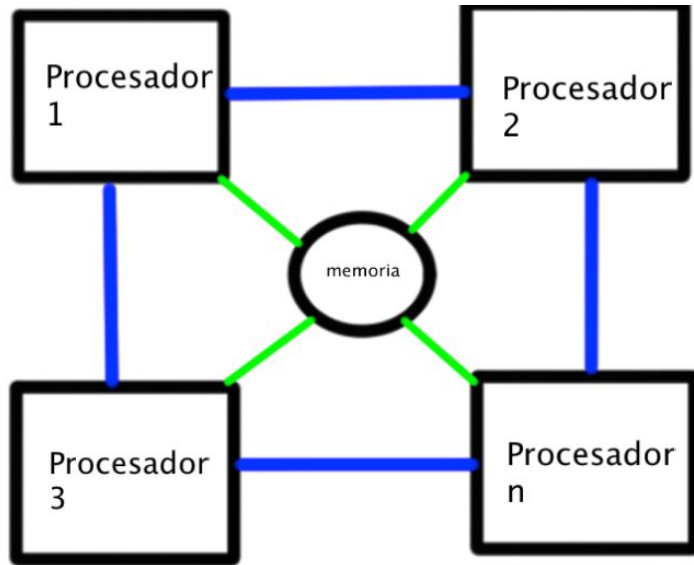


Cómputo con un solo procesador

Un solo procesador que atiende a un solo programa.

Esta fue la forma más común de procesamiento en los inicios de la computadora.

2.1 Sistema de almacenamiento y procesamiento distribuido



Multiprocesamiento

Una computadora con más de un procesador que tiene la capacidad de atender o llevar a cabo varias tareas en forma simultánea. El cómputo paralelo es un ejemplo de ésta clasificación.

2.1 Sistema de almacenamiento y procesamiento distribuido



Supercomputadora

Computadora especializada con múltiples procesadores.

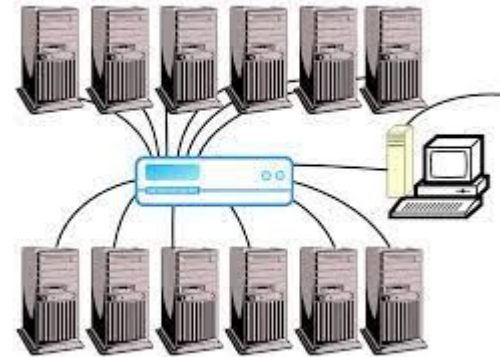
- Costosa
- Difícil de mantener

2.1 Sistema de almacenamiento y procesamiento distribuido

Los sistemas de programación están diseñados para obtener su paralelismo no de una "super-computadora", sino de "grupos de computación".

Cómputo en cluster

Lo podemos definir como un sistema de procesamiento paralelo o distribuido. Consta de un conjunto de computadoras independientes, interconectadas entre sí, de tal manera que funcionan como un solo recurso computacional.





2.1 Sistema de almacenamiento y procesamiento distribuido

Cómputo en clúster

Fallo en los nodos - Ventaja o desventaja?

Ventajas	Desventajas
Alto rendimiento	La Red puede hacer perder la fiabilidad.
Gran capacidad de expansión y de escalabilidad	La programación distribuída es complicada
	Seguridad



2.1 Sistema de almacenamiento y procesamiento distribuido

Fallo en los nodos

En clústeres grandes suelen ocurrir fallos en los nodos y en la red.

- Un nodo puede estar activo hasta por 1000 días (3 años)
- 1000 nodos, un fallo por día
- 1 millón de nodos, 1000 fallos al día

Retos:

- Almacenar los datos persistentemente y mantenerlos disponibles aún si los nodos fallan.
- Lidar con fallas de nodos de cómputo de larga duración.



2.1 Sistema de almacenamiento y procesamiento distribuido

Cuellos de botella en la red

Transferir demasiados datos a través de la red, puede ser muy lento.

- Con un ancho de banda de 1Gbps, tomaría cerca de un día transferir 10Tb de información de un nodo a otro.

Retos:

- Minimizar transferencia de datos a través de la red.



2.1 Sistema de almacenamiento y procesamiento distribuido

La programación distribuída es complicada.

Requiere considerar sincronización, carga de trabajo y comunicación, entre otras cosas.

Es necesario un modelo que oculte la complejidad posible de la programación distribuida.



MapReduce

MapReduce es un sistema de programación que permite atender los retos del cómputo en clúster. Proporciona un sistema de procesamiento de datos paralelo y distribuido.



2.1 Sistema de almacenamiento y procesamiento distribuido

Para explotar la computación en clúster, los archivos deben verse y comportarse de manera algo diferente de los sistemas de archivos convencionales que se encuentran en computadoras individuales.

Este nuevo sistema de archivos es a menudo llamado sistema de archivos distribuido o DFS

- El sistema de archivos de Google (GFS)
- Hadoop Distributed File System (HDFS)

Consideraciones

- Los archivos pueden ser enormes,
- Los archivos rara vez se actualizan.



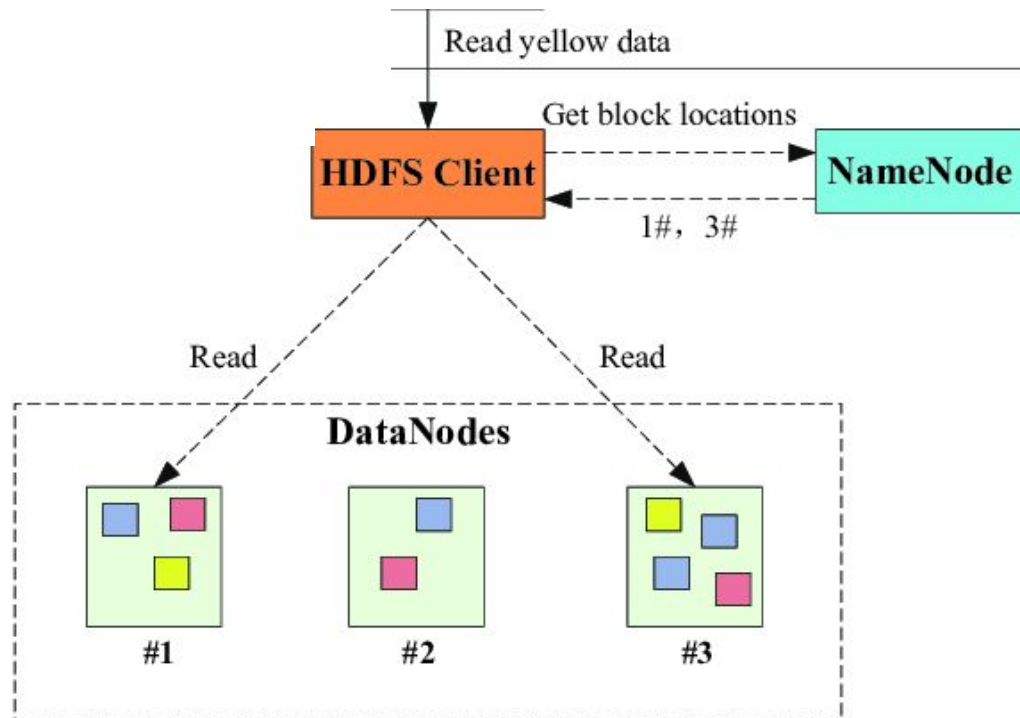
2.1 Sistema de almacenamiento y procesamiento distribuido

Los archivos se dividen en fragmentos o *chunks*.

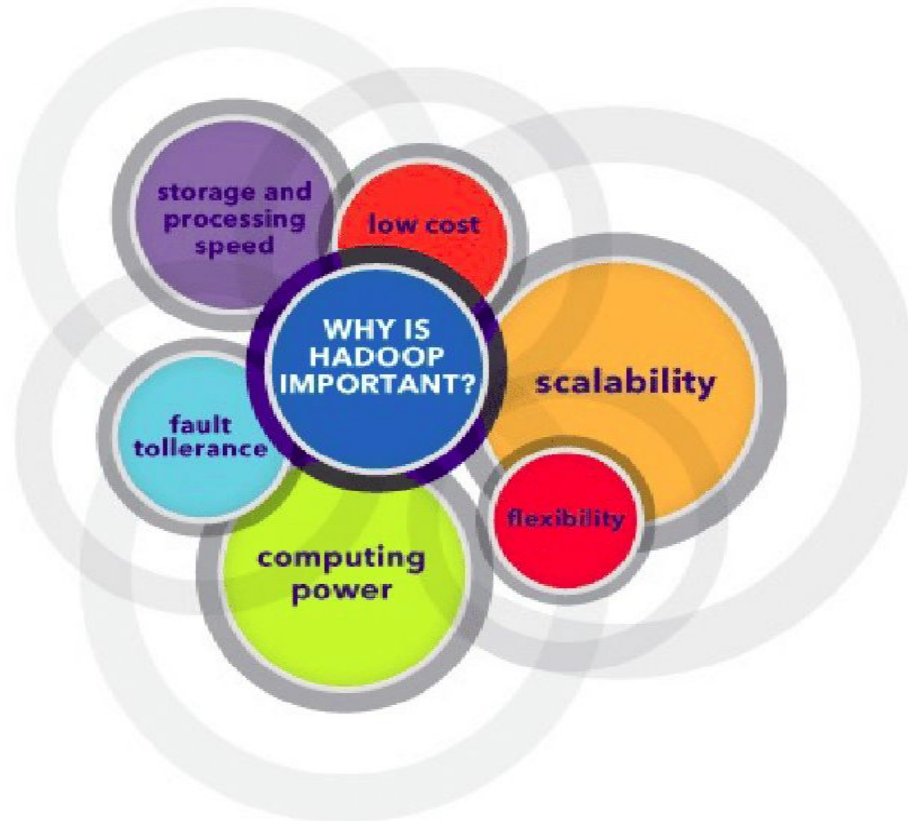
Cada fragmento se replica en diferentes nodos.

Cada uno de los “servidores de fragmento” actúan como servidores de cómputo

Existe un nodo maestro..



2.1 Sistema de almacenamiento y procesamiento distribuido





2.2 Modelo de programación

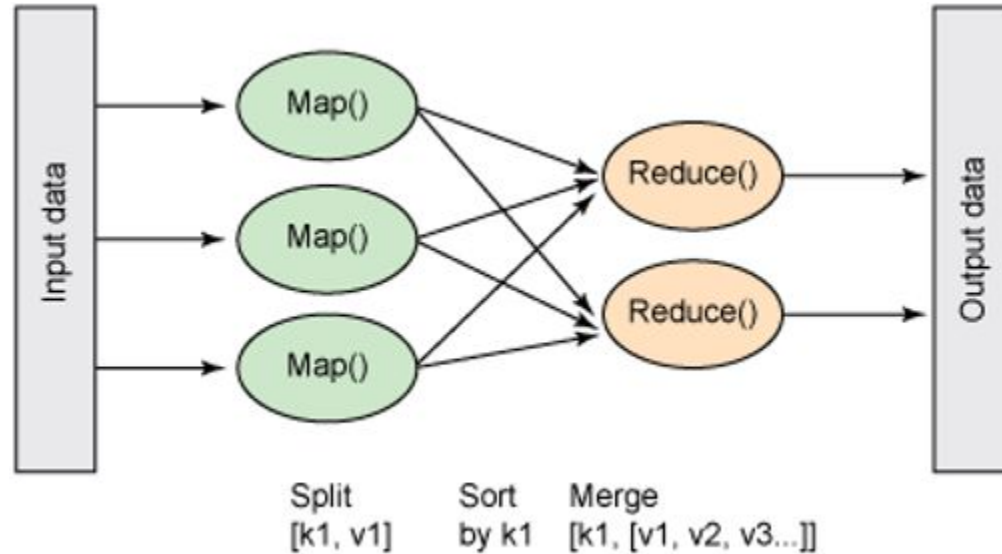
Es un modelo de programación que proporciona un sistema de procesamiento de datos paralelo y distribuido sobre grandes colecciones de datos por lo que usa HDFS.

- Fue inspirado en la programación funcional.
- Se caracteriza por ser simple y elegante.
- Permite la construcción de bloques.
- Está diseñado para ser ejecutado en clusters.
- Toma ventaja del paralelismo
- Tolerante a fallas.
- Es extensible a diferentes aplicaciones.

Permite...

- Realizar cálculos con grandes cantidades de datos a prueba de fallas
- Diseño de algoritmos capaces de procesar datos en tiempo real.

2.2 Modelo de programación



2.2 Modelo de programación

El programador indica
cuántos nodos necesita
para cada tarea
(mapeo-reducción)
5 y 3

Hoy empecé la dieta verde:
verde lejos la pizza,
verde lejos los tamales,
verde lejos las tortas,
verde lejos el pan.

Mapeo

Lee una entrada
y produce un
conjunto de pares
llave - valor

(hoy, 1)
(empece, 1)
(la, 1)
(dieta, 1)
(verde, 1)
(verde, 1)
(lejos, 1)
(la, 1)
(pizza, 1)
(verde, 1)
(lejos, 1)
(los, 1)
(tamales, 1)
(verde, 1)
(lejos, 1)
(las, 1)
(tortas, 1)
(verde, 1)
(lejos, 1)
(el, 1)
(pan, 1)

(llave, valor)

Agrupar por llaves:

colecciona todos
los pares con la
misma llave

(hoy, 1)
(empece, 1)
(la, 1)
(la, 1)
(dieta, 1)
(verde, 1)
(verde, 1)
(verde, 1)
(verde, 1)
(verde, 1)
(lejos, 1)
(lejos, 1)
(lejos, 1)
(lejos, 1)
(pizza, 1)
(los, 1)
(tamales, 1)
(las, 1)
(tortas, 1)
(el, 1)
(pan, 1)

(llave, valor)

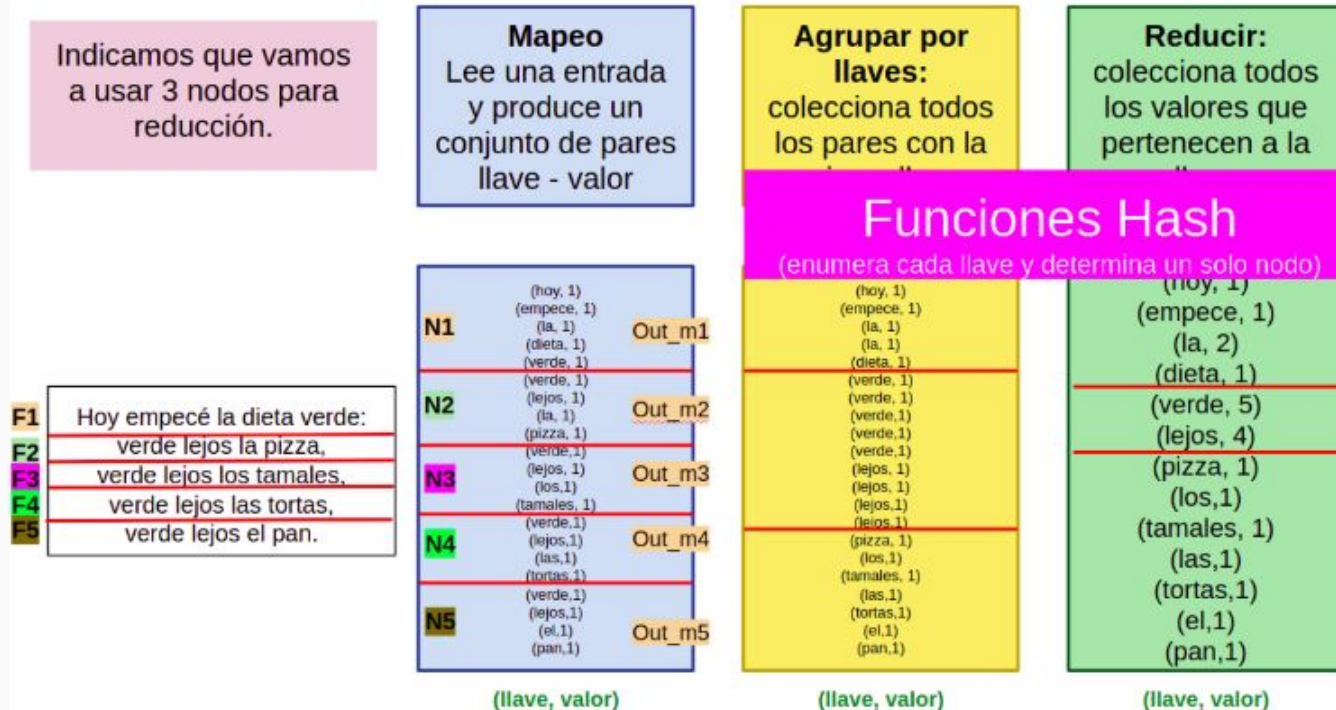
Reducir:

colecciona todos
los valores que
pertenecen a la
llave

(hoy, 1)
(empece, 1)
(la, 2)
(dieta, 1)
(verde, 5)
(lejos, 4)
(pizza, 1)
(los, 1)
(tamales, 1)
(las, 1)
(tortas, 1)
(el, 1)
(pan, 1)

(llave, valor)

2.2 Modelo de programación



2.2 Modelo de programación

El nodo maestro sabe que para acceder a los resultados debe consultar tres nodos

La función de mapeo se va a ejecutar **paralelamente** en los 5 nodos

Mapeo
Lee una entrada y produce un conjunto de pares llave - valor

Agrupar por llaves:
colecciona todos los pares con la misma llave

Reducir:
colecciona todos los valores que pertenecen a la llave

F1 Hoy empecé la dieta verde:
F2 verde lejos la pizza,
F3 verde lejos los tamales,
F4 verde lejos las tortas,
F5 verde lejos el pan.

N1	(hoy, 1)
	(empece, 1)
	(la, 1)
	(dieta, 1)
	(verde, 1)
N2	(verde, 1)
	(lejos, 1)
	(la, 1)
	(pizza, 1)
	(verde, 1)
N3	(lejos, 1)
	(los, 1)
	(tamales, 1)
	(verde, 1)
N4	(lejos, 1)
	(las, 1)
	(tortas, 1)
	(verde, 1)
N5	(lejos, 1)
	(el, 1)
	(pan, 1)

(llave, valor)

(hoy, 1)
(empece, 1)
(la, 1)
(dieta, 1)
(verde, 1)
(verde, 1)
(verde, 1)
(verde, 1)
(verde, 1)
(lejos, 1)
(lejos, 1)
(lejos, 1)
(lejos, 1)
(lejos, 1)
(pizza, 1)
(los, 1)
(tamales, 1)
(las, 1)
(tortas, 1)
(el, 1)
(pan, 1)

(llave, valor)

(hoy, 1)
(empece, 1)
(la, 2)
(dieta, 1)
(verde, 5)
(lejos, 4)
(pizza, 1)
(los, 1)
(tamales, 1)
(las, 1)
(tortas, 1)
(el, 1)
(pan, 1)

(llave, valor)

2.2 Modelo de programación

Las lecturas
secuenciales son
mucho más eficientes
que los accesos
aleatorios

Mapeo
Lee una entrada
y produce un
conjunto de pares
llave - valor

**Agrupar por
llaves:**
colecciona todos
los pares con la
misma llave

Reducir:
colecciona todos
los valores que
pertenecen a la
llave

F1 Hoy empecé la dieta verde:
F2 verde lejos la pizza,
F3 verde lejos los tamales,
F4 verde lejos las tortas,
F5 verde lejos el pan.

N1	(hoy, 1) (empece, 1) (la, 1) (dieta, 1) (verde, 1)	Out_m1
N2	(verde, 1) (lejos, 1) (la, 1) (pizza, 1)	Out_m2
N3	(verde, 1) (lejos, 1) (los, 1) (tamales, 1)	Out_m3
N4	(verde, 1) (lejos, 1) (las, 1) (tortas, 1)	Out_m4
N5	(verde, 1) (lejos, 1) (el, 1) (pan, 1)	Out_m5

(llave, valor)

(hoy, 1) (empece, 1) (la, 1) (dieta, 1) (verde, 1) (verde, 1) (verde, 1) (verde, 1) (verde, 1) (lejos, 1) (lejos, 1) (lejos, 1) (lejos, 1) (pizza, 1) (los, 1) (tamales, 1) (las, 1) (tortas, 1) (el, 1) (pan, 1)
--

(llave, valor)

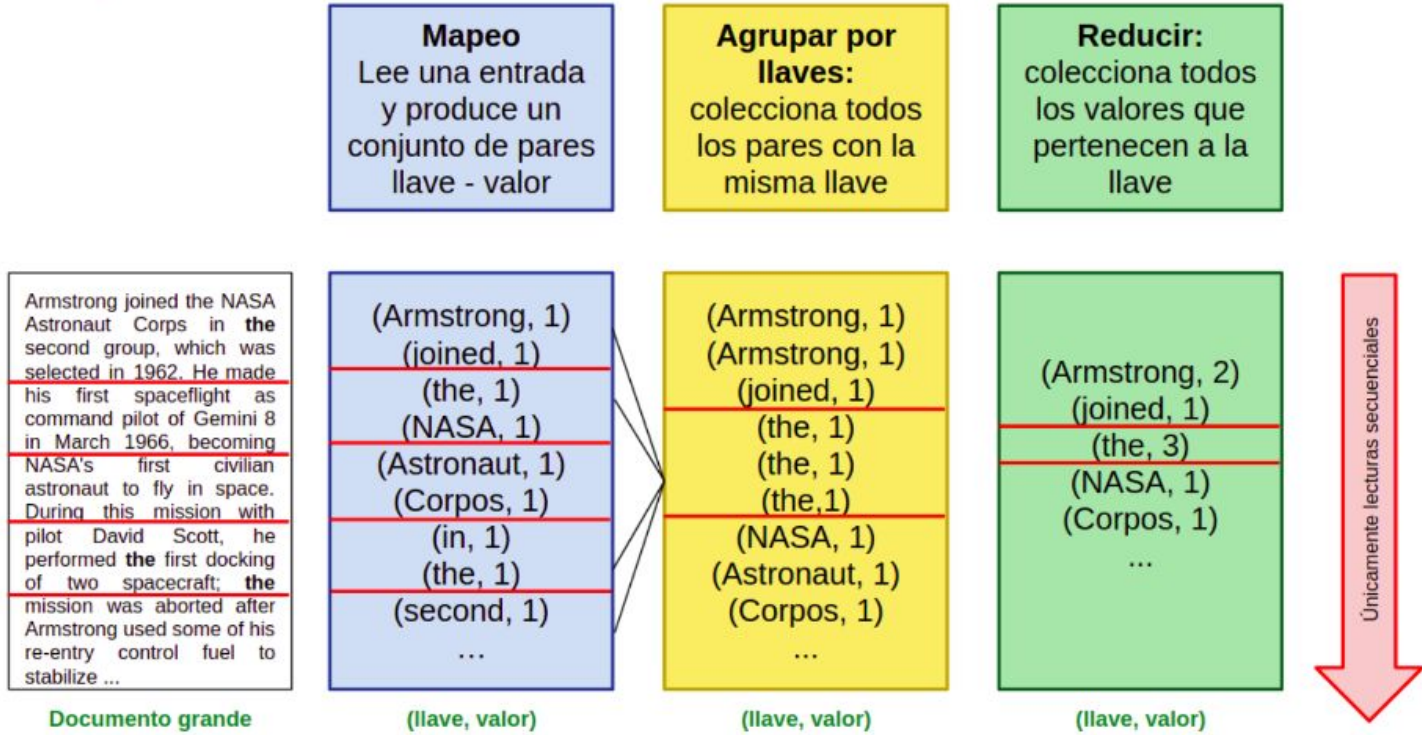
(hoy, 1) (empece, 1) NR1 (la, 2) (dieta, 1) NR2 (verde, 5) (lejos, 4) (pizza, 1) (los, 1) (tamales, 1) NR3 (las, 1) (tortas, 1) (el, 1) (pan, 1)
--

(llave, valor)

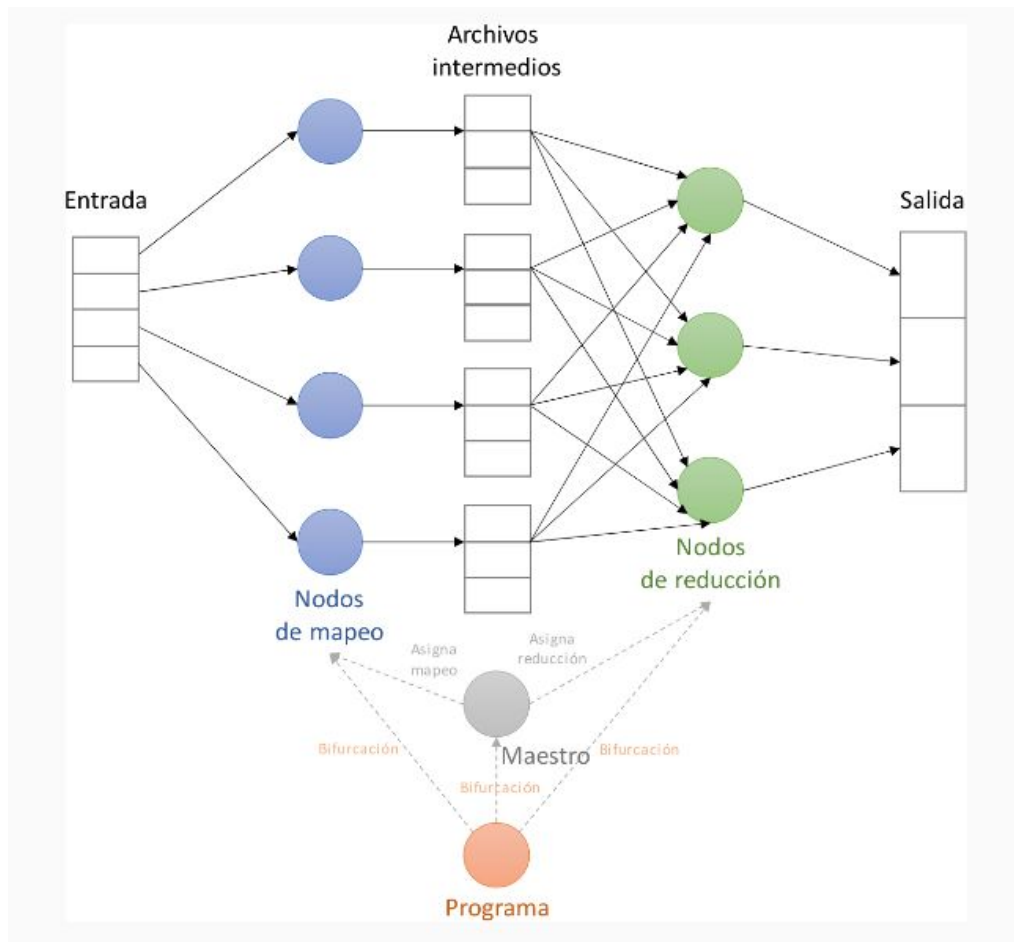
Únicamente lecturas secuenciales

MapReduce está construido sobre lecturas de archivos secuenciales y
nunca sobre accesos aleatorios

2.2 Modelo de programación



2.2 Modelo de programación





2.2 Modelo de programación

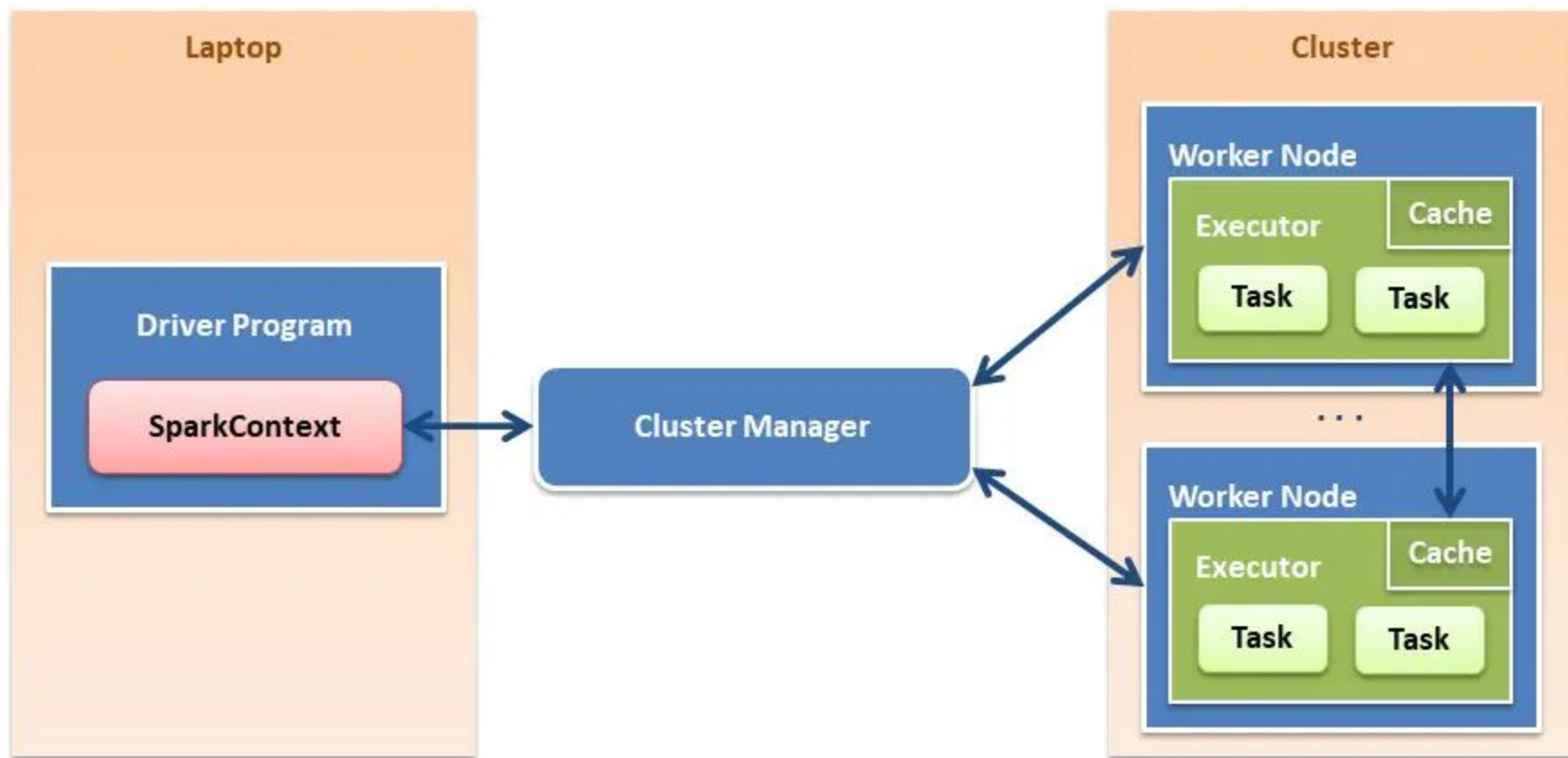
- Una tarea de mapeo puede producir muchos pares con la misma llave, lo cual aumenta el tamaño del archivo que se transfiere a los nodos de reducción.
- Los combinadores realizan una combinación preliminar de los valores de la tarea de mapeo.
- Usualmente se usa la misma función que la de reducción, solo se puede utilizar si la función es asociativa y conmutativa.
- Para decidir a qué nodo de reducción se va una llave, se usa una función de partición por defecto: $\text{has}(\text{llave}) \bmod r$
- Es posible definir una función de partición distinta $\text{has}(\text{autor}(\text{documento})) \bmod r$



MapReduce & Spark

MapReduce, uno de los primeros sistemas de computación paralela, donde la tarea se separa en minitareas para hacerlas en unidades de procesamiento diferentes. Escribir tareas con MapReduce es complejo. Hive, desarrollado por Facebook, corre encima de Hadoop.

Spark es más moderno y no reemplaza a Hadoop, pero reemplaza las tareas de MapReduce. Trabaja con RDDs y tiene interfaces diferentes, entre ellas pyspark.





Ejercicio

Se encuentran trabajando en una empresa que realiza soluciones de inteligencia artificial. El cliente ya había entregado un set de datos pequeño que tenía disponible al momento de la contratación del servicio y se comprometió a entregar el resto en cuanto los tuviera listos. El cliente entrega el resto de los datos a unos pocos días antes de la entrega del primer avance y debe realizarse la extracción de los mismos.

Los datos que deben extraerse son muchos y con el hardware disponible para ese proyecto no es posible almacenar o cargar los datos para analizarlos o hacerles el tratamiento respectivo. Los recursos de la empresa en ese instante no alcanzan para contratar los recursos faltantes en la nube.

¿Qué solución darías para poder realizar la extracción, análisis y carga de los datos para cumplir con los objetivos de la primer entrega?

En equipos, discutan para llegar a una solución y posteriormente compartirla con el grupo.



2.3 Algoritmos con el modelo de mapeo y reducción

MapReduce no es una solución para todos los problemas.

Usos más comunes:

- Operaciones de álgebra relacional
- Ordenamiento
- Búsqueda
- Multiplicación de una matriz y un vector
- Multiplicación de dos matrices



2.3 Algoritmos con el modelo de mapeo y reducción: Operaciones de álgebra relacional

Las operaciones que se pueden utilizar con MapReduce, son:

- Selección
- Proyección
- Unión, Intersección, and Diferencia
- Join
- Agrupaciones



2.3 Algoritmos con el modelo de mapeo y reducción: Ordenamiento

Problema: Ordenar un conjunto de archivos con un valor por línea.

Función de Mapeo: Recibe pares con nombre de archivo y número de línea como llave y el contenido de la línea como valor. ((nombre, línea), valor). Regresa el valor como llave.

Función de Reducción: Sort.

Aprovecha el ordenamiento de llaves y se define una función tal que,

$$k_1 < k_2 \implies \text{hash}(k_1) < \text{hash}(k_2)$$



2.3 Algoritmos con el modelo de mapeo y reducción: Búsqueda

Problema: Encontrar documentos que contienen un patrón dado..

Función de Mapeo:

- Recibe pares con nombre de archivo y número de línea como llave y el patrón como valor. ((nombre, línea), patrón)
- Regresa el nombre del archivo si es que encontró el patrón en el contenido.

Función de Reducción: Identidad.



2.3 Algoritmos con el modelo de mapeo y reducción: Multiplicación de una matriz y un vector

Problema: Dada una matriz **M** de $n \times n$ y un vector **v** de tamaño n , calcular $x = M \cdot v$

$$x_i = \sum_j M_{i,j} \cdot v_j$$

Función de Mapeo:

- Se lee a **v** si no se ha hecho
- Se aplica a cada elemento de **M**
- Regresa pares $(i, M_{i,j} \cdot v_j)$

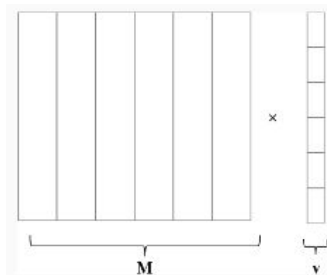
Función de Reducción:

- Suma todos los valores asociados a la llave i .
- Regresa pares (i, x_i)



2.3 Algoritmos con el modelo de mapeo y reducción: Multiplicación de una matriz y un vector

Si \mathbf{v} no cabe en memoria, se divide \mathbf{M} en k segmentos verticales y \mathbf{v} en k segmentos horizontales.



El mapeo se aplica a cada elemento de los segmentos correspondientes de \mathbf{M} y \mathbf{v} .

Las tareas de Mapeo y Reducción se realizan de igual forma que cuando \mathbf{v} cabe en memoria.



2.3 Algoritmos con el modelo de mapeo y reducción: Multiplicación de dos matrices

Problema: Dadas dos matrices de $n \times n$, calcular $C = A \cdot B$

$$C_{i,k} = \sum_j A_{i,j} \cdot B_{j,k}$$

Función de Mapeo:

- Para cada elemento A_{ij} regresa un par llave-valor $(j, (es(A), i, A_{i,j}))$
-
- Para cada elemento B_{jk} regresa un par llave-valor $(j, (es(B), k, B_{j,k}))$



2.3 Algoritmos con el modelo de mapeo y reducción: Multiplicación de dos matrices

Función Reducción:

- Para cada llave j examina su lista de valores asociados.
- Para cada valor (i, A_{ij}) que viene de A , y cada valor (k, B_{jk}) que viene de B , produce una llave-valor con la llave (i,k) y valor $A_{i,j} \cdot B_{j,k}$
- Para cada llave (i,k) regresa la suma de valores asociados.
- Cada resultado $((i,k),v)$ corresponde a un elemento de la matriz C .



2.4 Extensiones: Entorno de Trabajo

Los entornos de trabajo de Mapeo Reducción se encargan de:

- Dividir los datos de entrada.
- Planificar la ejecución de los programas en todo el clúster.
- Agrupar por llaves (en la práctica esto es un cuello de botella)
- Manejo de fallas
- Administrar la comunicación entre máquinas.



2.4 Extensiones: Nodo Maestro

El nodo maestro se encarga de la coordinación de las actividades en el clúster de cómputo.

- Estatus de las tareas/ nodos: “esperando”, “en proceso”, “completada”.
- Planifica las tareas de mapeo y reducción.
- Cuando una tarea de mapeo está completa, el nodo maestro envía la localización y tamaño de los archivos intermedios (R) a los nodos de reducción.
- Para detectar fallas, el nodo maestro periódicamente se comunica con los demás nodos.



2.4 Extensiones: Flujo de Trabajo

- Las entradas y salidas son almacenadas en el sistema de archivos distribuidos (DFS)
- Las tareas de mapeo son planificadas en nodos cercanos a la localización física de los datos.
- Los resultados intermedios son almacenados en sistemas locales de archivos (nodos de mapeo reducción): evitar tráfico en la red y sobrecarga de datos.
- Las salidas son frecuentemente la entrada hacia otra tarea de mapeo.



2.4 Extensiones: Manejo de Fallas

Pueden fallar los nodos de:

- Mapeo
- Reducción
- Maestro



2.4 Extensiones:

¿Cuántas tareas de MapReduce debemos planificar?

Objetivo: Identificar M (número de tareas de mapeo) y R (número de tareas de reducción) que deben ser planificadas.

- El número de tareas M debe ser más grande que el número de nodos en el clúster.
 - Mejora el balanceo de cargas y acelera la recuperación en caso de fallas.
- Usualmente R es más pequeño de M
- El archivo de salida se reparte en los nodos R .



2.4 Extensiones: Problemas con el modelo de Mapeo Reducción

- Dificultad para programar directamente: muchos algoritmos no se describen fácilmente con funciones mapeo reducción.
- Cuellos de botella: Para preservar la persistencia de los datos, el tiempo que se toma en dividir y almacenar los datos es considerable.
 - Incurre en gastos considerables debido a la replicación de datos E/S de disco y serialización.



2.4 Extensiones

MapReduce ha demostrado ser tan influyente que ha generado una serie de extensiones y modificaciones.

Estos sistemas comparten las siguientes características:

1. Están contruidos sobre un sistema de archivos distribuidos.
2. Manejan un gran número de tareas que son instancias de un pequeño número de funciones escritas por el usuario.
3. Incorporar un método para tratar la mayoría de fallas que ocurren durante la ejecución de trabajo grande, sin tener que reiniciar el trabajo desde el principio.



2.4 Extensiones: Sistemas de flujo de trabajo

El cómputo se expresa como un grafo acíclico que representa el flujo de trabajo de un conjunto de funciones.

- Que un vértice ***a*** se conecte a otro vértice ***b*** representa que la salida de la función ***a*** es la entrada de la función ***b***.
- El modelo de mapeo y reducción es un sistema de flujo de trabajo con dos pasos.

Los datos fluyen de una función a otra y cada función se puede realizar en múltiples tareas con distintas partes de los datos de entrada.

Un nodo maestro se encarga de dividir las tareas en distintos nodos y resolver posibles fallas.



2.4 Extensiones: Modelo de Spark

Es una extensión del modelo de mapeo reducción.

- Se basa en un sistema de operaciones (transformaciones/acciones) realizadas sobre colecciones de datos distribuidos (RDD).
- Actualmente, es el sistema más popular de flujo de datos.
-

Más rápido

- Evita guardar resultados intermedios a disco.
- Activa la caché de datos para consultas repetitivas.
- Presenta funciones extras (más allá de mapeo y reducción)
- Compatible con Apache Hadoop.

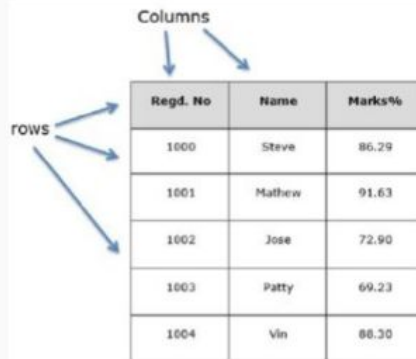


2.4 Extensiones: Apache Spark

- Es código abierto (Apache Foundation)
- Soporta Java, Scala y Python
- Principal contribución: Conjunto de datos distribuidos y resilientes (RDD)
- Integra APIs de alto nivel
 - En las versiones más recientes de Spark incluye Dataframes y Datasets.
 - Ofrece APIs para agregar datos, lo cual permite soportar SQL.

2.4 Extensiones: ¿Qué es un dataset?

- Es un conjunto de datos.
- Puede contener cualquier tipo de información.



The diagram shows a table with three columns and five rows. Above the table, the word 'Columns' has two arrows pointing to the 'Regd. No' and 'Name' headers. To the left of the table, the word 'ROWS' has three arrows pointing to the first three rows of the table body.

Regd. No	Name	Marks%
1000	Steve	86.29
1001	Mathew	91.63
1002	Jose	72.90
1003	Patty	69.23
1004	Vin	88.30

Imagen tomada de IMF BUSINESS SCHOOL

2.4 Extensiones: RDD

- Archivo de objetos de un tipo
- Estructura principal de Spark
- Está particionado sobre los nodos del clúster
- Son inmutables: cuando transformamos un RDD realmente estamos creando uno nuevo.
- Distribuye los datos en modo lectura.
- Tolerante a fallos
- Pueden ser creados desde Hadoop.

Existen dos formas comunes para crear RDDs:

- A través del objeto *sparkContext*
- A partir de conjuntos de datos externos





2.4 Extensiones: Creando RDDs

El método `sparkContext.parallelize` nos permite crear un RDD a partir de una lista o una tupla.

```
lista = ['en', 'un', 'lugar', 'de', 'un', 'gran', 'país']
```

```
lista_rdd = sc.parallelize(lista, 4)
```

`sc` es una instancia de la clase `sparkContext`

La lista se pasa como argumento a `sc` y se paraleliza automáticamente por Spark.

El programador puede decidir en cuántas partes debe paralelizarse un RDD.



2.4 Extensiones: Creando RDDs

A partir de una fuente de almacenamiento, se puede utilizar la función *textFile* del *sparkContext*.

```
text = sc.sparkContext.textFile('file.txt')
```

Como argumento se pasa un archivo (texto, binario) almacenado en disco.

El método *textFile* carga el archivo como un RDD



2.4 Extensiones: RDD

Los RDDs no son valiosos solamente por los datos que contienen, sino por las operaciones que podemos realizar sobre ellos.

Spark, proporciona un conjunto de acciones para procesar y extraer información:

`collect(), reduce(), count(), foreach()`



Acciones RDD

- `collect()`, retorna todos los elementos de un RDD como una lista de python.
- `count()`, retorna el número de elementos del RDD.
- `countByValue()`, retorna un diccionario con el número de apariciones de cada elemento en un RDD.
- `reduce()`, agrega los elementos de un RDD según la función que se le pase como parámetro. La función debe cumplir con las siguientes propiedades, para que pueda ser calculada en paralelo.
 - Conmutativa, asegurando que el resultado será independiente del orden de los elementos en el RDD.
 - Asegurando que cualquiera de los elementos asociados en la agregación, a la vez no afecta el resultado final.
- `foreach()`, ejecuta la función que se le pasa por parámetro sobre cada elemento del RDD.
- `collectAsMap()`, retorna los elementos de un RDD clave/valor como un diccionario de python.
- `saveAsTextFile(directorio)` guarda el RDD como un conjunto de archivos de texto dentro de directorio.



Acciones RDD

Función	Valor que retorna
<i>first()</i>	Devuelve el primer valor del RDD
<i>mean()</i>	Devuelve el valor medio
<i>variance()</i>	Devuelve la varianza
<i>stdev()</i>	Devuelve la desviación estándar
<i>take(n)</i>	Devuelve una lista con los n elementos del RDD

No siempre se podrá ejecutar las acciones directamente sobre un RDD debido a las siguientes razones.

- El RDD podría no estar en un formato adecuado
- El RDD podrá tener más datos de los necesarios a analizar (calcular la media de las edades)
- El RDD podrá no contener todos los datos necesarios.



Transformaciones

Antes de realizar acciones sobre los RDD para garantizar que contenga los datos unificados, filtrados y formateados para evitar errores.

Al aplicar una transformación sobre un RDD original, regresará un RDD.

Las transformaciones no modifican el RDD original, Spark evalúa las transformaciones de manera 'perezosa' (lazy evaluation).

Las transformaciones construcción RDDs a través de operaciones como `map()`, `filter()`, `sample()`, y `union()`.



Transformaciones

- `map()`, retorna un nuevo RDD, resultado de pasar cada uno de los elementos de un RDD original como parámetro de la función.
- `filter()`, retorna un nuevo RDD que contiene los elementos que cumplen la función.
- `distinct()`, retorna un nuevo RDD que contiene una sola copia de los diferentes elementos del RDD
- `union()`, retorna un nuevo RDD que contiene la unión de los elementos de un RDD y del que se le pasa como argumento.

Función	Valor que retorna
<i>intersection()</i>	Devuelve la intersección de 2 RDDs
<i>keys()</i>	Devuelve únicamente las llaves del RDD
<i>sortBy(func)</i>	Ordena un RDD según un criterio



Ejercicio

Se realizó la extracción y carga de datos a una base de datos SQL y archivos CSV, por parte de un equipo de arquitectura que contaba con los recursos necesarios para realizar dichas tareas. Los recursos son exclusivos de esa área.

Se requiere hacer un análisis de dichos datos, el problema es que para su carga en cualquier IDE para poder visualizar los datos, se es posible visualizar el total de columnas que se tiene en cada tabla o archivo, además de que también contienen demasiados registros.

¿Qué solución propondrías para poder visualizar los datos completos?

En equipos, discutan para llegar a una solución y posteriormente compartirla con el grupo.



Ejercicio

Una vez resuelto el problema de la visualización de las columnas de cada tabla/archivo ¿qué solución propondrías para realizar la transformación de los datos?



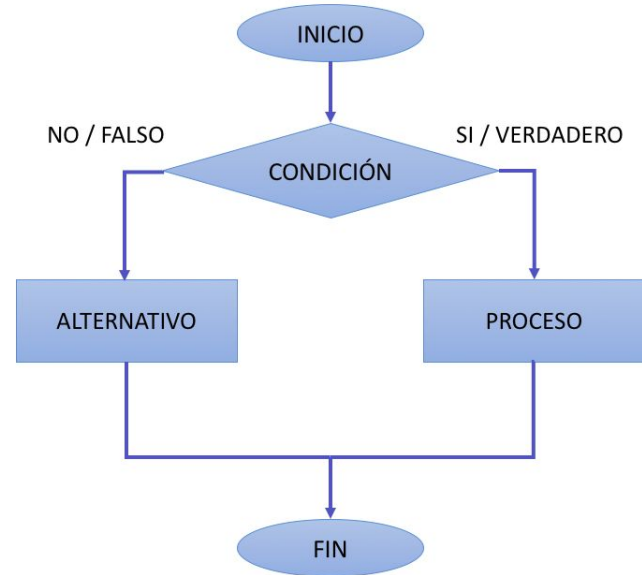
2.5 El modelo costo-comunicación

De forma general, el rendimiento de un algoritmo se puede medir a través de los siguientes costos:

- Tiempo
- Espacio
- Energía

El objetivo es encontrar:

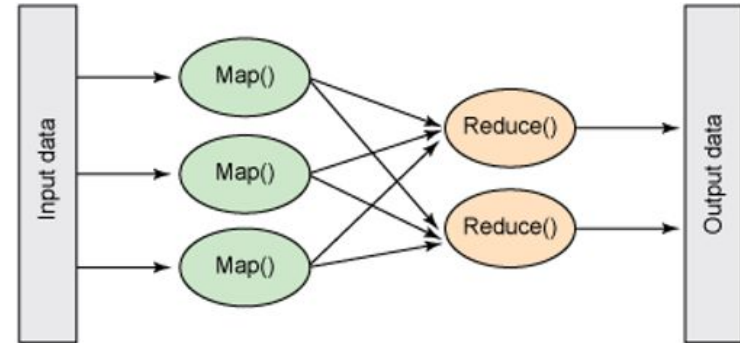
- $T(n)$ -> costo en el tiempo
- $S(n)$ -> costo del espacio
- $E(n)$ -> costo de energía



2.5 El modelo costo-comunicación

Costo en el Modelo de Mapeo-Reducción

1. Costo en cómputo
 - a. Trabajos de Mapeo
 - b. Trabajos de Reducción
 - c. Sistema del modelo



2. Costo de comunicación
 - a. Transmisión de los pares de los nodos de mapeo a los de reducción
 - b. Costos de lectura en disco (nodos de mapeo)



2.5 El modelo costo-comunicación

Costo en el Modelo de Mapeo-Reducción

En un escenario común:

- El costo de comunicación domina al costo de cómputo.
- El costo de los trabajos de mapeo es una pequeña fracción del costo de comunicación.
- El costo de ordenamiento es proporcional al costo de comunicación.

En los servicios de la nube se paga tanto por el cómputo como por la comunicación, por lo tanto es importante tener un balance de ambos en el diseño de los algoritmos.



2.5 El modelo costo-comunicación

- El costo de comunicación de una tarea es el tamaño de su entrada
- El costo de comunicación de un algoritmo es la suma de los costos de todas las tareas que requiere.
- La eficiencia de un algoritmo se mide basado en su costo de comunicación.
 - El cómputo que realiza cada tarea es usualmente muy simple y con complejidad lineal en el tamaño de la entrada.
 - Transmitir una entrada sobre la red puede ser muy lento y puede requerir tiempo para poder recibirla de otros nodos debido al tráfico en la red.
 - Incluso cuando la tarea se ejecuta en el mismo nodo donde están las entradas, es necesario leerlos de disco y esto puede ser más costo que el cómputo que se realiza sobre las mismas.



Problema de ejemplo: Join

- Queremos resolver: $R(A,B) \bowtie S(B,C)$

Las tareas de mapeo tienen fragmentos de cualquiera de las de las dos relaciones.

- Función de mapeo:
 - Recibe una tupla t de R o S
 - Produce un par llave-valor
- Función reducción:
 - Recibe la llave con la lista de valores, la cual puede contener la tupla de R , la de S o ambas
 - Identifica tuplas que son de R de aquellas que son de S y las empareja.
 - Produce un par (t,t)



Problema de ejemplo: Join Mapeo

Supongamos que los tamaños de R y S son r y s respectivamente.

La entrada de las tareas de mapeo es un fragmento de los archivos que contienen a R o a S .

La suma del costo de comunicación de cada tarea de mapeo es $r + s$.

El costo de comunicación del mapeo es $O(r + s)$



Problema de ejemplo: Join Reducción

La reducción se aplica a uno o más atributos de B.

Cada reducer toma las entradas que recibe y los divide entre tuplas que vienen de R y los que vienen de S,

Cada tupla de R se empareja con cada tupla de S para producir una salida.

El tamaño de la salida puede ser mayor o menor a $r + s$, dependiendo de que tanto se unan las tuplas.



Tiempo Reloj

Al diseñar un algoritmo para un clúster es importante tomar en cuenta el **tiempo** que toma un algoritmo en ejecutarse en paralelo.

Una posible forma de reducir el costo de comunicación es simplemente asignar todo el trabajo a una sola tarea y el tiempo para llevarse a cabo sería muy alto.

Problema de ejemplo: Multiple Join

- **Input:** three tables

$R(X, Y)$, $S(Y, Z)$, $T(Z, X)$

$\text{size}(R) + \text{size}(S) + \text{size}(T) = M$

- **Output:** compute

$Q(x, y, z) = R(x, y) \bowtie S(y, z) \bowtie T(z, x)$

	T	
	Z	X
S	Fred	Alice
	Jim	Jim
R	Fred	Alice
	Jim	Jim
	Jim	Alice
	Jim	
	Jack	Jim
	Alice	
	Fred	Jim
	Carol	Alice
	...	

Imagen tomada de Dan Suciu, University of Washington



Problema de ejemplo: Multiple Join

Paso 1: Calcular la primera unión (almacenarlo en *temp*)

$$temp(X, Y, Z) = R(X, Y) \bowtie S(Y, Z)$$

Paso 2: Calcular una segunda unión

$$Q(X, Y, Z) = temp(X, Y, Z) \bowtie T(Z, X)$$

Cada unión fue implementada en el modelo de mapeo reducción *temp* puede generar largas relaciones intermedias.

El tiempo de comunicación y computación transcurrido pueden verse afectados (dependerá del tamaño de *temp*)



Costos de algoritmos

El costo de un algoritmo depende de varias características:

- La semántica del modelo de programación
- La topología de la red
- El tipo de datos
- Protocolos de comunicación



2.6 Teoría de la complejidad para el modelo de mapeo y reducción

Los modelos de mapeo reducción tienen dos parámetros que los caracterizan:

1. El tamaño del reductor (**q**)
 - Indica el límite superior del número de valores que pueden aparecer en la lista asociada a cada una de las llaves.
2. La tasa de replicación (**r**)
 - Número de pares llave-valor dividido por todas las entradas.



Unión por similitud

Supongamos que tenemos un gran conjunto de elementos X y una medida de similitud $s(x, y)$ que indica cuán similares son los elementos x & y del conjunto X .

Para cada par de elementos de X debemos determinar su similitud aplicando la función s , y supongamos que s es simétrica:

$$s(x, y) = s(y, x)$$

La salida del algoritmo son aquellos pares cuya similitud excede un umbral dado t .



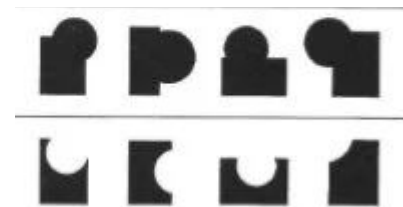
Unión por similitud

Ejemplo

Supongamos que tenemos una colección de un millón de imágenes, cada una de tamaño de 1 mb, por lo tanto, el conjunto de datos tiene un tamaño de 1 terabyte.

El objetivo sería descubrir pares de imágenes que muestran el mismo tipo de objeto o escena. Este problema es extremadamente difícil, pero la clasificación por distribución de colores generalmente es de ayuda para lograr este objetivo.

La entrada son pares llave-valor (**i**, **Pi**), donde **i** es in ID para la imagen y **Pi** es la imagen misma. Queremos comparar cada par de imágenes, así que usamos una llave para cada conjunto de dos ID **{i, j}**





Unión por similitud

Ejemplo

Queremos que cada llave $\{i, j\}$ esté asociada con los dos valores P_i & P_j , por lo que la entrada correspondiente será $(\{i, j\}, [P_i, P_j])$

Entonces la función de reducción puede simplemente aplicar la función de similitud s a las dos imágenes en su lista de valores $s(P_i, P_j)$, y decida si la similitud de las dos imágenes está por encima del umbral.

Función Map: Toma un elemento de entrada (i, P_i) y genera $g - 1$ pares de llaves-valor.

Función Reduce: Toma cada (i, P_i) y (j, P_j) , donde i, j pertenecen a distintos grupos y aplica la función de similitud $s(P_i, P_j)$.

NOTA: Es importante considerar que no queremos comparar varias veces los mismos pares.



Unión por similitud

Ejemplo

r es igual al número de grupos

$$q \text{ es } \frac{2 \times 10^6}{g}$$

No importa que valor tenga g , se aplica a cada par una vez, así que aunque el trabajo se divida en reductores todos los grupos hacen el mismo cálculo.