

Procesamiento Masivo de datos: Hadoop II

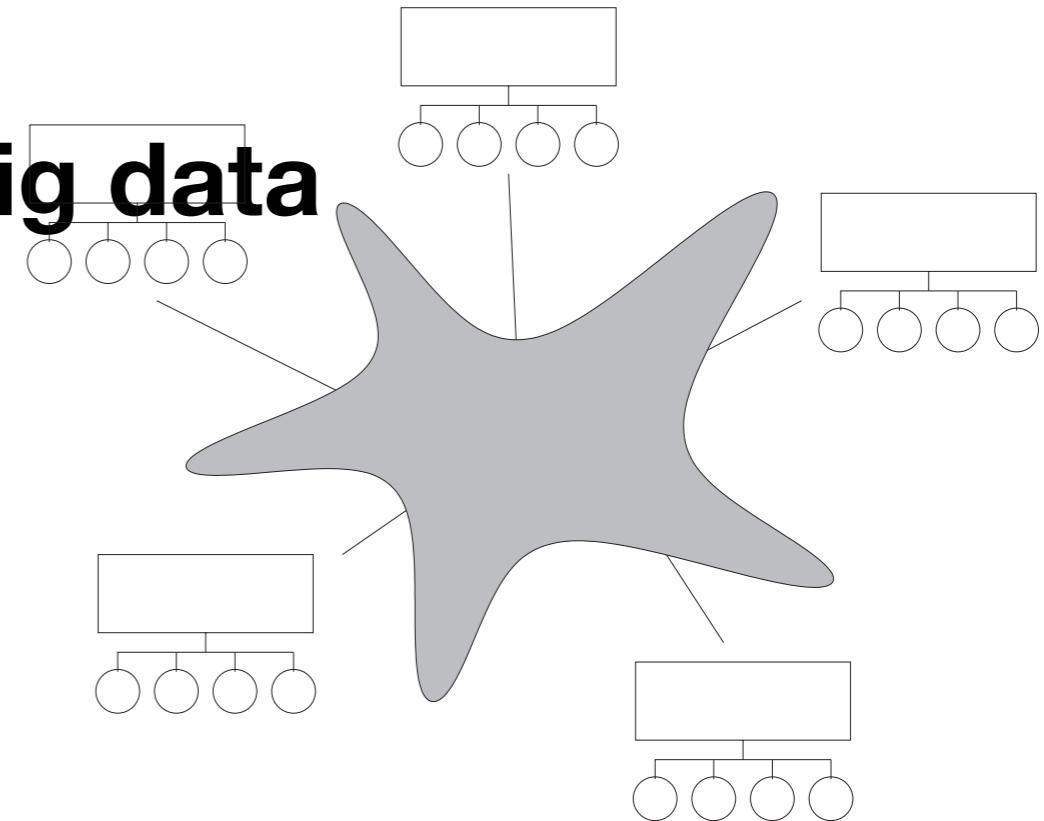
Alex Di Genova

06/10/2022

Hadoop

Un sistema operativo para big data

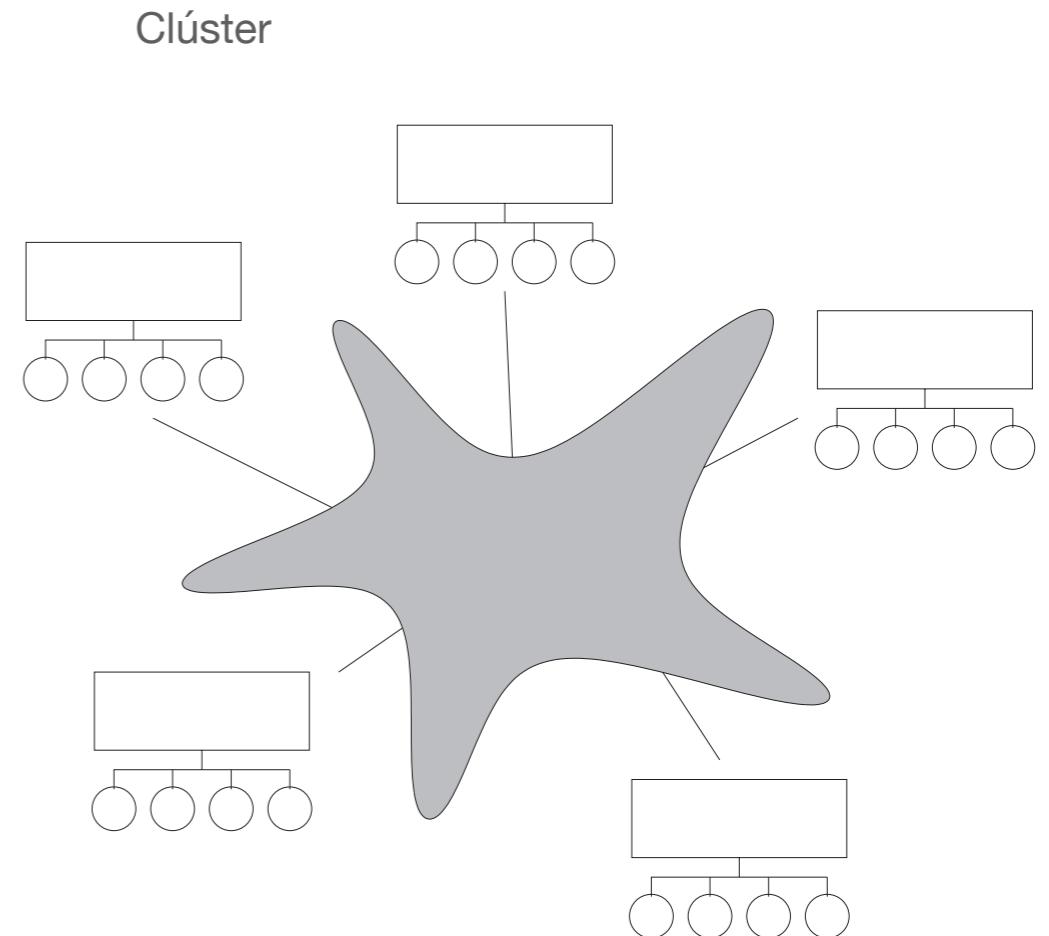
- Hadoop proporciona aplicaciones robustas para operar con datos.
- Hadoop implementa el paradigma computacional denominado Map/Reduce, donde la aplicación se divide en muchos pequeños fragmentos de trabajo, cada uno de los cuales se puede ejecutar o volver a ejecutar en cualquier nodo del clúster.
- Proporciona un sistema de archivos distribuido (HDFS) que almacena datos en los nodos de cómputo, proporcionando un acceso rápido y eficiente a los archivos.
- Tanto Map/Reduce como HDFS están diseñados para manejar errores en los nodos.
- Para realizar cálculos a escala, Hadoop distribuye un cálculo que involucra un conjunto masivo de datos a muchas máquinas que operan simultáneamente en su propia porción individual de datos.



Hadoop

Un sistema operativo para big data

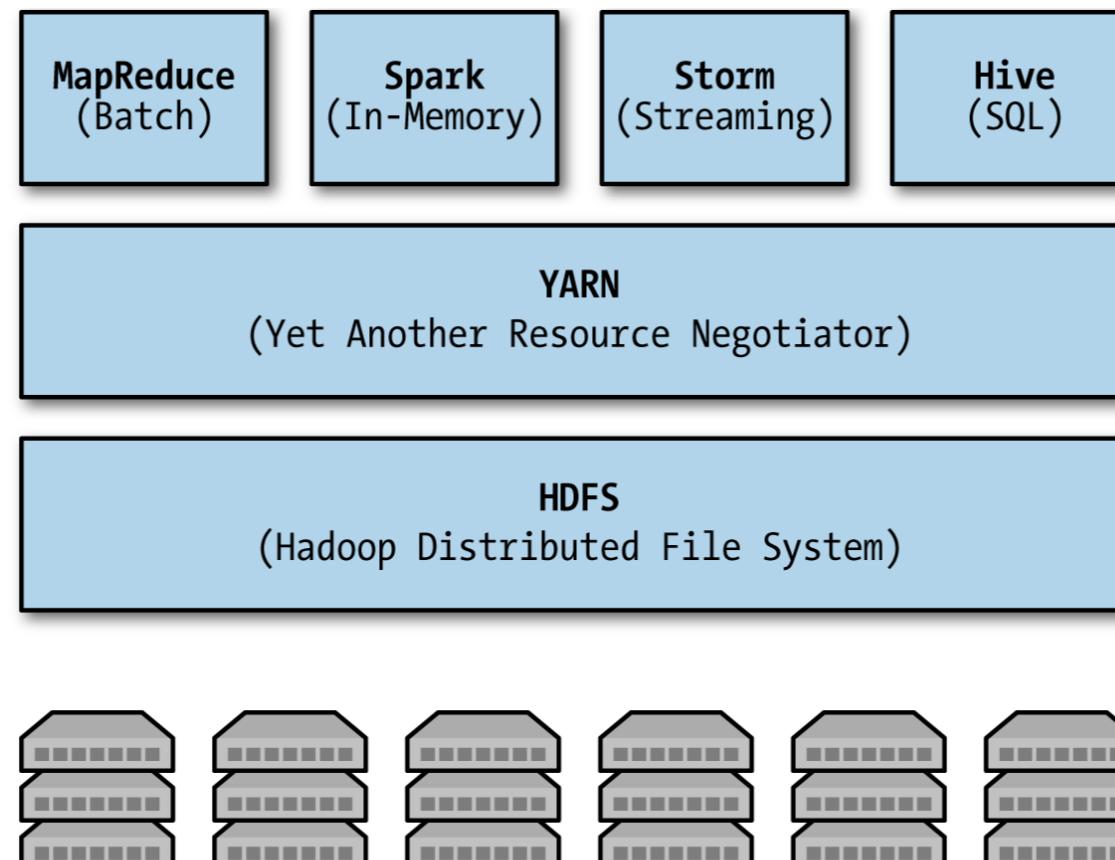
- Los datos se distribuyen inmediatamente cuando se agregan al clúster y se almacenan en varios nodos. Los nodos prefieren procesar los datos que se almacenan localmente para minimizar el tráfico en la red.
- un sistema distribuido debe cumplir con los siguientes requisitos:
 - **Tolerante a fallas**
 - Si un componente falla, no debería resultar en la falla de todo el sistema.
 - **Recuperable**
 - En caso de falla, no se deben perder datos.
 - **Consistente**
 - La falla de un trabajo o tarea no debe afectar el resultado final.
 - **Escalable**
 - Agregar carga (más datos, más cómputo) conduce a una disminución en el rendimiento, no a fallas; aumentar los recursos debería resultar en un aumento proporcional de la capacidad.



Hadoop

Arquitectura

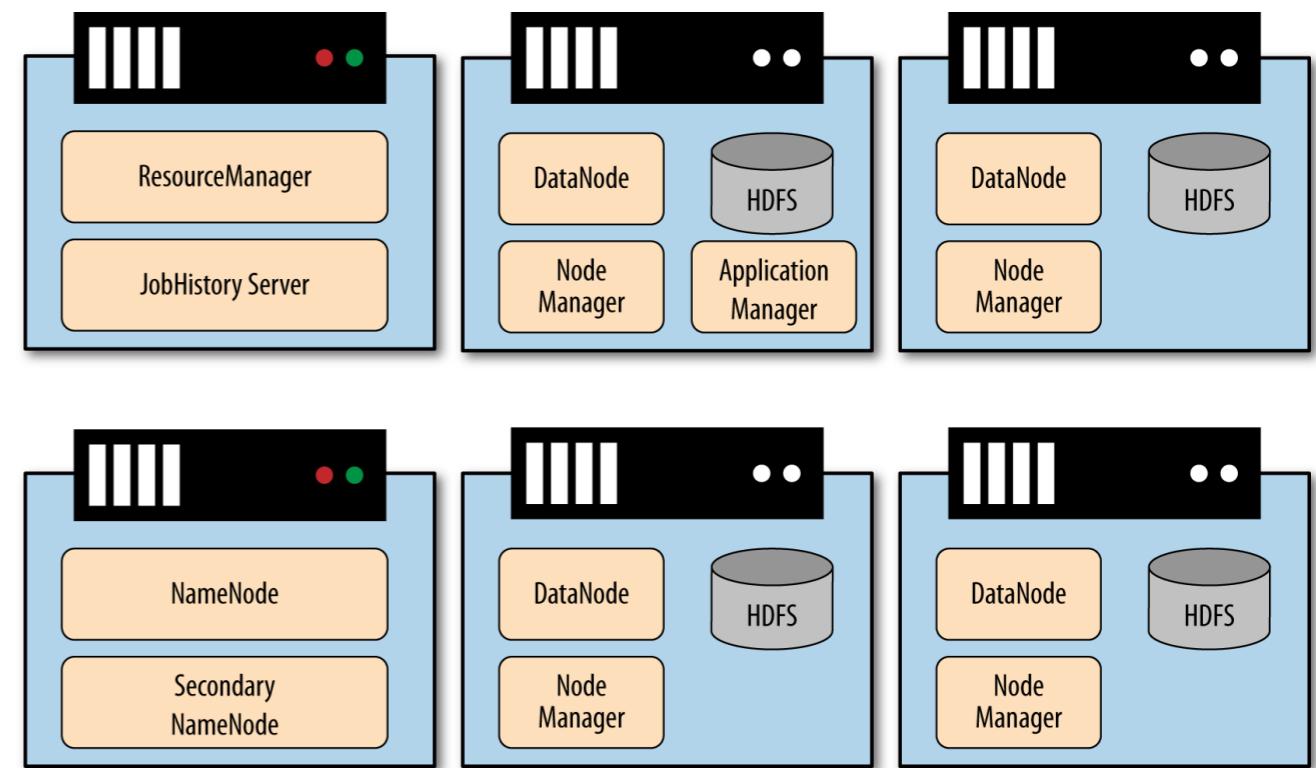
- Hadoop se compone de dos componentes principales que implementan los conceptos básicos de computación y almacenamiento distribuido.
- **HDFS** (a veces abreviado como DFS) es el sistema de archivos distribuidos de Hadoop, responsable de administrar los datos almacenados en discos en todo el clúster.
- **YARN** actúa como un administrador de recursos del clúster, asignando recursos computacionales (disponibilidad de procesamiento y memoria en los nodos trabajadores) a las aplicaciones que desean realizar una computación distribuida.
- HDFS y YARN funcionan en conjunto para minimizar la cantidad de tráfico de red en el clúster, principalmente al garantizar que los datos sean locales para el cálculo requerido. La duplicación de datos y tareas garantiza la tolerancia a fallas, la capacidad de recuperación y la consistencia.
- HDFS y YARN son una plataforma sobre la cual se construyen aplicaciones de big data (proporcionan un sistema operativo para big data)



Hadoop

A Hadoop cluster

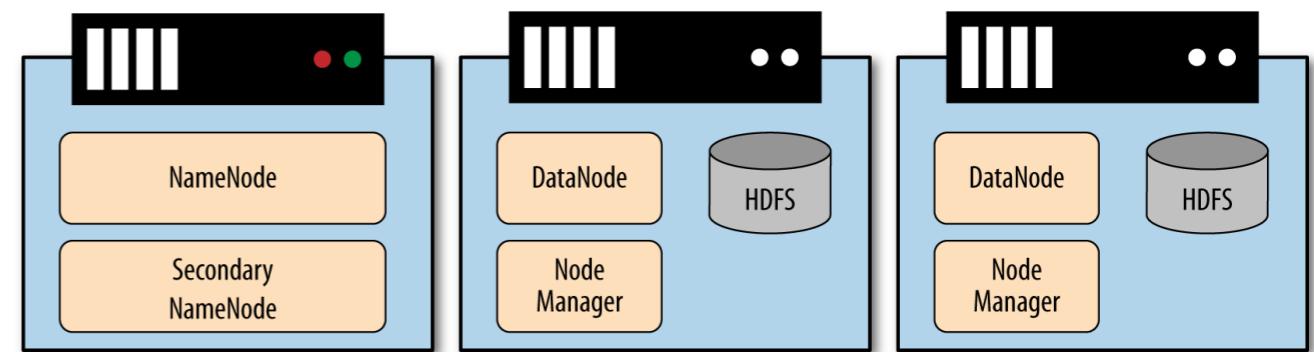
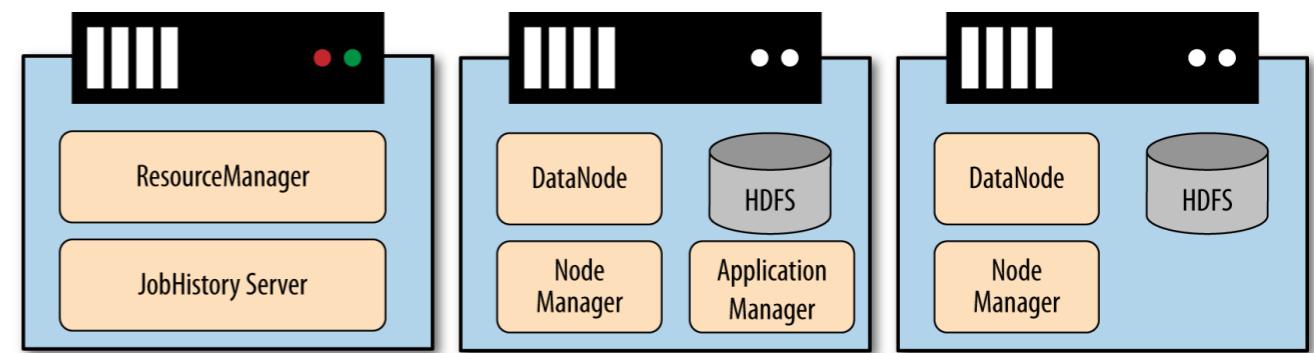
- Para HDFS, los servicios maestro y trabajador son los siguientes:
 - NameNode (Master)
 - Almacena el árbol de directorios del sistema de archivos, los metadatos del archivo y las ubicaciones de cada archivo en el clúster.
 - Secondary NameNode (Master)
 - Realiza tareas de limpieza y puntos de control para el NameNode
 - DataNode (Worker)
 - Almacena y administra bloques HDFS en el disco local



Hadoop

A Hadoop cluster

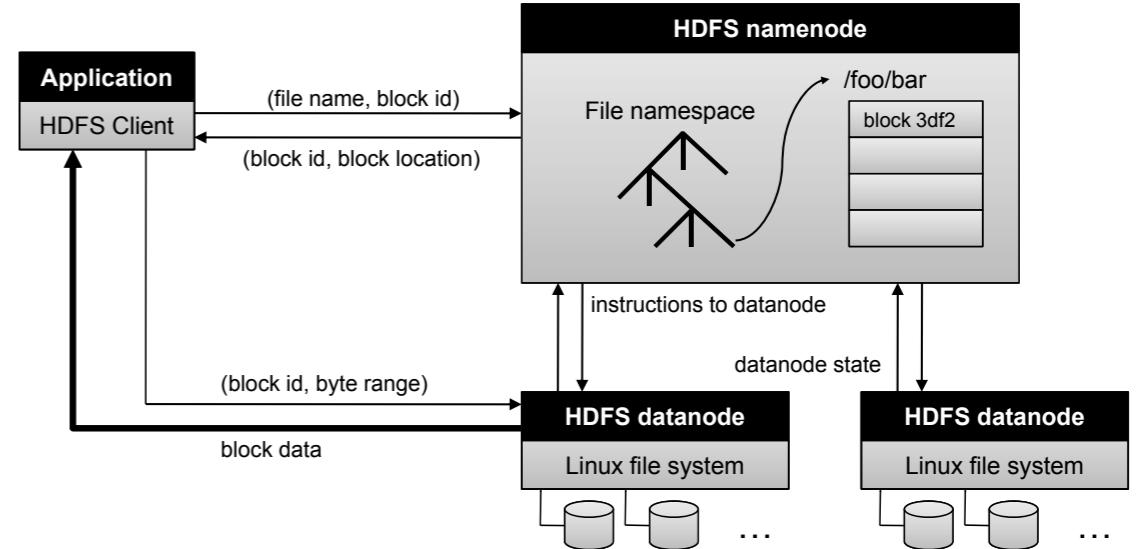
- YARN tiene múltiples servicios maestros y un servicio de trabajador:
 - ResourceManager (Master)
 - Asigna y supervisa los recursos de clúster disponibles a las aplicaciones, además de manejar la programación de trabajos en el clúster.
 - ApplicationMaster (Master)
 - Coordina una aplicación particular que se ejecuta en el clúster según lo programado por ResourceManager.
 - NodeManager (Worker)
 - Ejecuta y administra tareas de procesamiento en un nodo individual y también informa sobre el estado y el estado de las tareas a medida que se ejecutan.



Hadoop

HDFS

- HDFS proporciona almacenamiento redundante para big data al almacenar los datos en un grupo de computadoras.
- HDFS es una capa de software sobre un sistema de archivos nativo como ext4 o xfs.
- Los archivos HDFS se dividen en bloques, generalmente de 64 MB o 128 MB.
- El tamaño del bloque es la cantidad mínima de datos que se pueden leer o escribir en HDFS.
- Los bloques permiten dividir archivos muy grandes y distribuirlos a muchas máquinas en tiempo de ejecución. Se almacenan diferentes bloques del mismo archivo en diferentes máquinas para proporcionar un procesamiento distribuido más eficiente.
- Los bloques se replican en los DataNodes (3x). Por lo tanto, cada bloque existe en tres máquinas diferentes y tres discos diferentes, y si fallan incluso dos nodos, los datos no se perderán.
 - Almacenamiento en clúster / 3
- El **NameNode** maestro almacena qué bloques componen un archivo y dónde se encuentran esos bloques. El **NameNode** se comunica con los **DataNodes**, los nodos que realmente contienen los bloques en el clúster. Los metadatos asociados con cada archivo se almacenan en la memoria del **NameNode** para realizar búsquedas rápidas.



- hadoop fs -help
- hadoop fs –copyFromLocal kmers.txt kmers.txt
- hadoop fs -mkdir text_db
- hadoop fs –ls .
- hadoop fs –cat kmers.txt | less
- hadoop fs –tail shakespeare.txt | less
- hadoop fs –get kmers.txt ./kmers.from-remote.txt
- hadoop fs –chmod 664 kmers.txt
- hadoop fs -test <ruta>
- hadoop fs -count <ruta>
- hadoop fs -du -h <ruta>

Hadoop

MapReduce: Un modelo de programación funcional

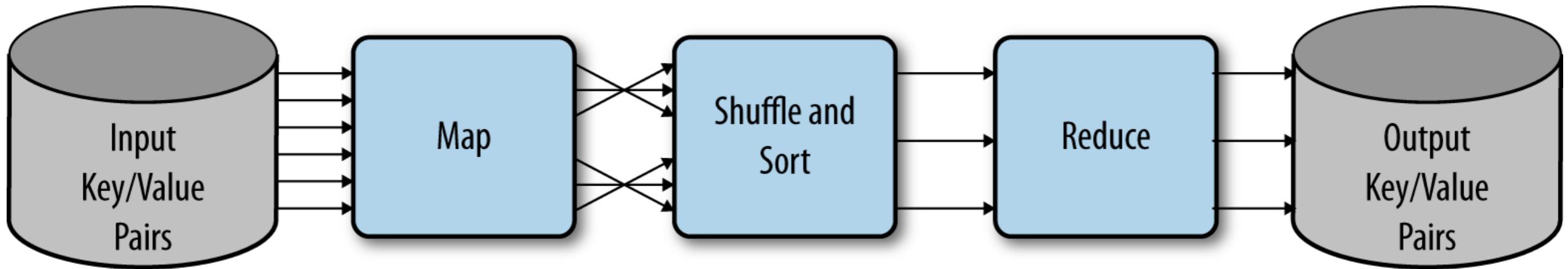
- MapReduce es un marco computacional simple pero muy potente diseñado específicamente **para habilitar el cómputo distribuido (tolerante a fallas) en un grupo de máquinas administradas centralmente.**
 - Idea: múltiples tareas independientes ejecutan una función en fragmentos locales de datos y agregan los resultados después del procesamiento.
 - La programación funcional es un estilo de programación que garantiza las funciones dependen solo de sus entradas, y están cerradas y no comparten estado. Los datos se transfieren entre funciones enviando la salida de una función como entrada a otra función totalmente independiente.
 - Los datos que se operan como entrada y salida en estas funciones no son utilizan pares de "clave/valor" para coordinar el cálculo.

```
def map(key, value):  
    # Perform processing  
    return (intermed_key, intermed_value)
```

```
def reduce(intermed_key, values):  
    # Perform processing  
    return (key, output)
```

Hadoop

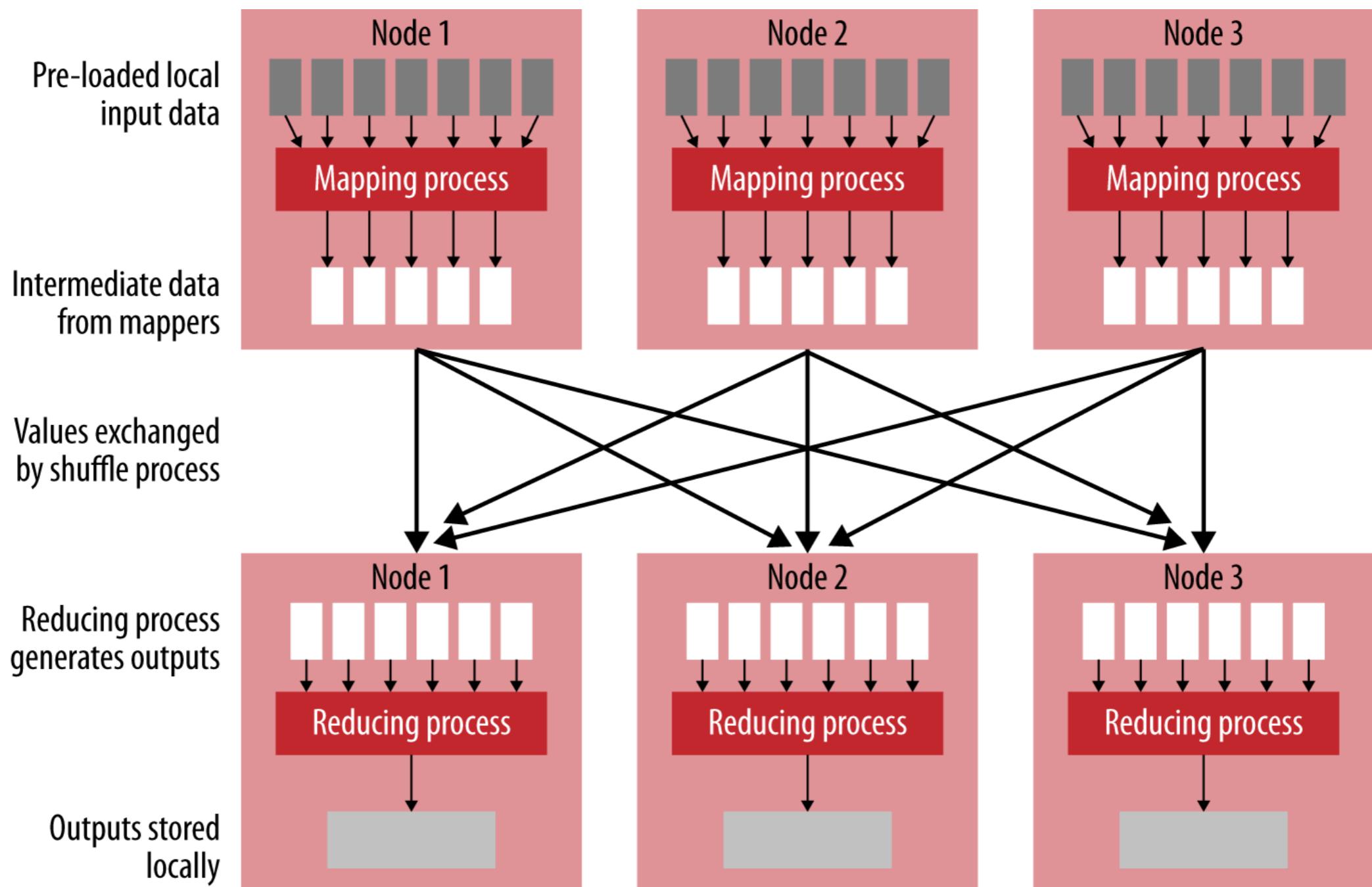
MapReduce: Un modelo de programación funcional



- 1.Los datos locales se cargan en un proceso de mapeo como pares clave/valor de HDFS.
- 2.Los mapeadores generan cero o más pares clave/valor, asignando valores calculados a una clave en particular.
- 3.Luego, estos pares se ordenan/barajan en función de la clave y luego se pasan a un reductor de modo que todos los valores de una clave estén disponibles.
- 4.Los reductores deben generar cero o más pares clave/valor finales, que son la salida.

Hadoop

Map/Reduce



Hadoop

Map/Reduce

```
def map(dockey, line):
    for word in value.split():
        emit(word, 1)

def reduce(word, values):
    count = sum(value for value in
values)
    emit(word, count)
```

```
# Input to WordCount mappers
(27183, "The fast cat wears no hat.")
(31416, "The cat in the hat ran fast.")

# Mapper 1 output
("The", 1), ("fast", 1), ("cat", 1), ("wears", 1),
("no", 1), ("hat", 1), (".", 1)

# Mapper 2 output
("The", 1), ("cat", 1), ("in", 1), ("the", 1),
("hat", 1), ("ran", 1), ("fast", 1), (".", 1)

# Input to WordCount reducers
# This data was computed by shuffle and sort

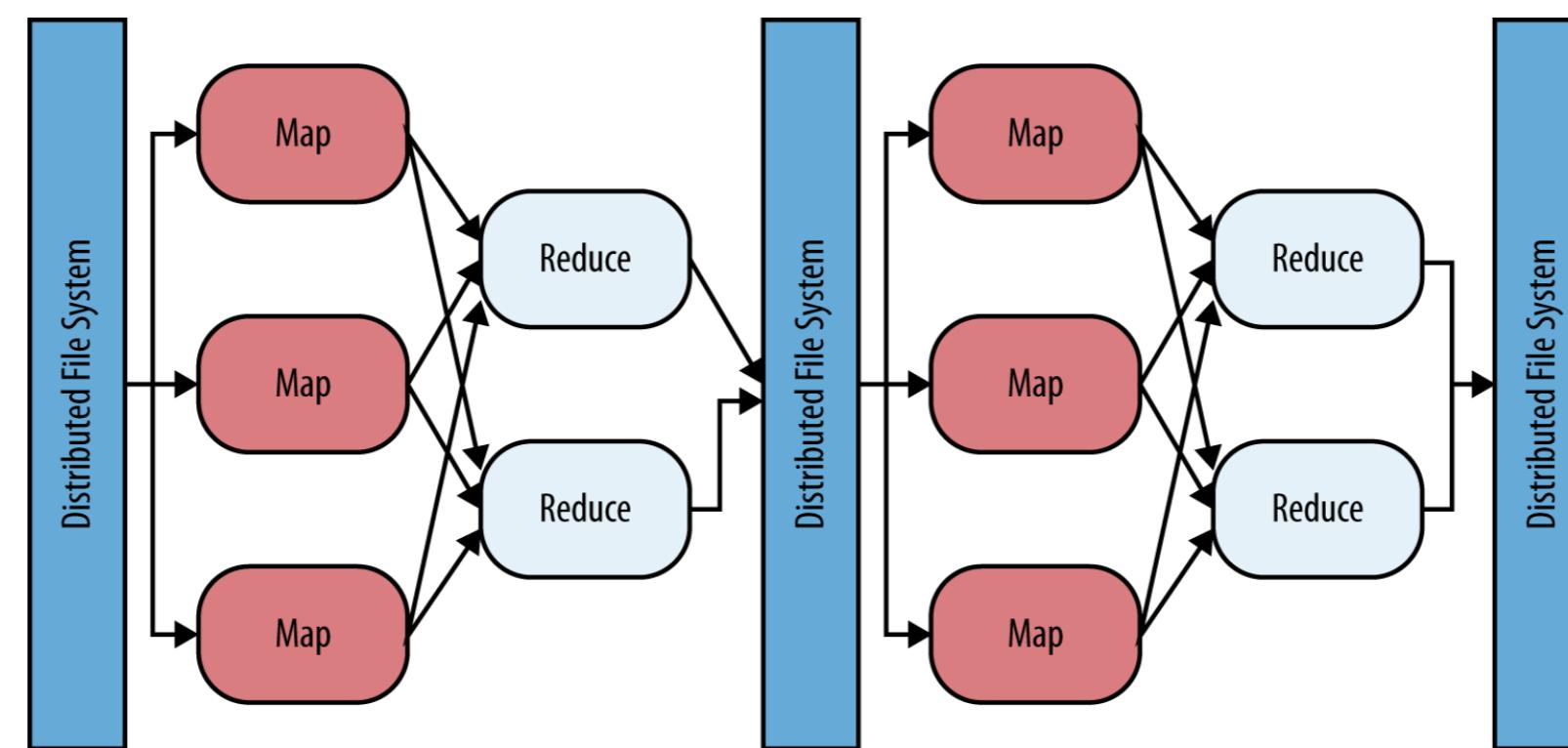
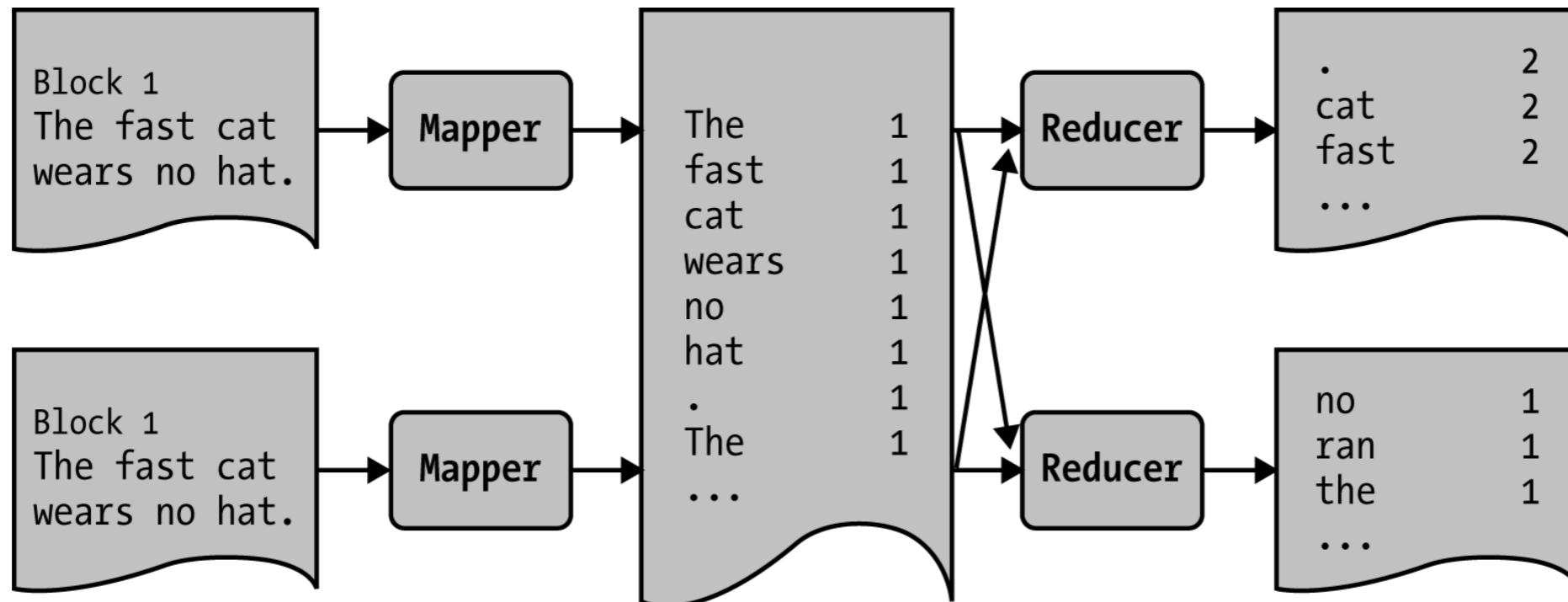
(".", [1, 1])
("cat", [1, 1])
("fast", [1, 1])
("hat", [1, 1])
("in", [1])
("no", [1])
("ran", [1])
("the", [1])
("wears", [1])
("The", [1, 1])

# Output by all WordCount reducers

(".", 2)
("cat", 2)
("fast", 2)
("hat", 2)
("in", 1)
("no", 1)
```

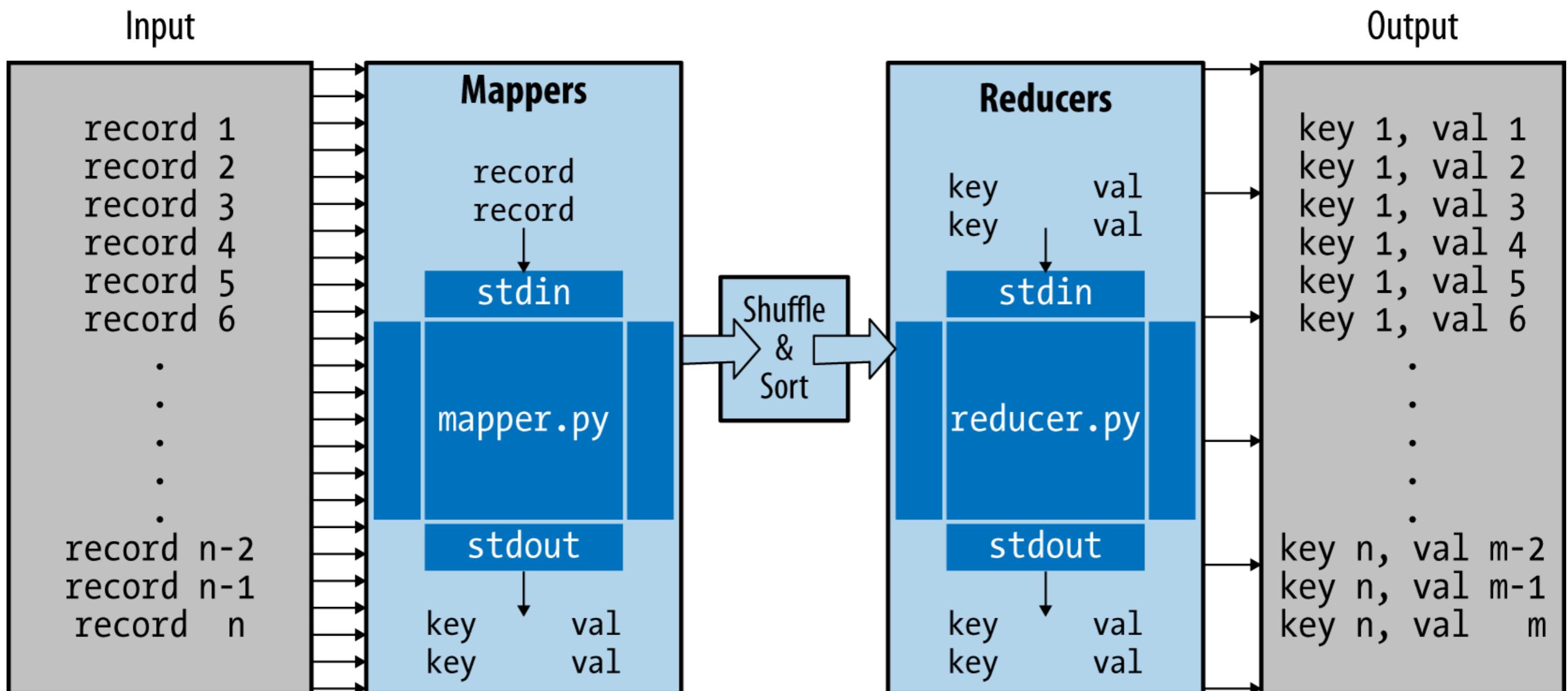
Hadoop

Map/Reduce



Hadoop streaming

Map/Reduce



Google Colab

The screenshot shows a Google Colab notebook titled "Rust-C-basics.ipynb". The notebook contains two sections: "Hola Mundo" and "Arreglos y funciones".

Hola Mundo:

```
[ ] 1 !apt install rustc cargo
[ ] 1 @@writefile helloworld.rs
2 // This is a comment, and is ignored by the compiler
3
4 // This is the main function
5 fn main() {
6     // Statements here are executed when the compiled binary is called
7     // Print text to the console
8     println!("Hola Mundo!");
9 }
10

Writing helloworld.rs

[ ] 1 !rustc /content/helloworld.rs
2 ./helloworld

[ ] Hola Mundo!
```

Arreglos y funciones:

```
[ ] 1 @@writefile arreglos.rs
2
3 use std::mem;
4
5 // This function borrows a slice
6 fn analyze_slice(slice: &[i32]) {
7     println!("primer elemento del arreglo: {}", slice[0]);
8     println!("el ultimo elemento es: {}", slice[slice.len()-1]);
9     println!("el arreglo tiene {} elementos", slice.len());
10 }

11
12 fn main() {
13     // arreglo de largo 5 fijo
14     let xs: [i32; 5] = [1, 2, 3, 4, 5];
15     // arreglo de largo 500 inicializado en 0
16     let ys: [i32; 500] = [0; 500];
17
18     // indices comienzan en 0
19     println!("primer elemento arreglo: {}", xs[0]);
20     println!("tercer elemento del arreglo: {}", xs[2]);
21
22     // `len` retorna el largo del arreglo
23     println!("numero de elementos arreglo xs: {}", xs.len());
24     println!("numero de elementos arreglo ys: {}", ys.len());
25
26     // Arrays are stack allocated
27     println!("tamaño en memoria xs {} bytes", mem::size_of_val(&xs));
28     println!("tamaño en memoria ys {} bytes", mem::size_of_val(&ys));
29     // Arrays can be automatically borrowed as slices
30     println!("pasamos el arreglo por puntero a función");
31     analyze_slice(&xs);
32     analyze_slice(&ys);
33
34     println!("pasamos una slice del array");
35     analyze_slice(&ys[1 .. 4]);
36
37     // Se puede acceder a las matrices de forma segura mediante `.get`,
38     // que devuelve un
39     // `Opción`. Esto se puede combinar como se muestra a continuación, o se puede usar con
40     // `.expect()` si desea que el programa finalice con un buen
41     // mensaje en lugar de continuar.
42     for i in 0..xs.len() + 1 { // iteramos un elemento mas del largo
43         match xs.get(i) {
44             Some(xval) => println!("{}: {}", i, xval),
45             None => println!("Fuera de indice! {} no existe!", i),
46         }
47     }
48 }
```

<https://github.com/adigenova/uohpmd/blob/main/code/HadoopII.ipynb>

Consultas?

Consultas o comentarios?

Muchas gracias