

Procesamiento Masivo de datos: Hadoop

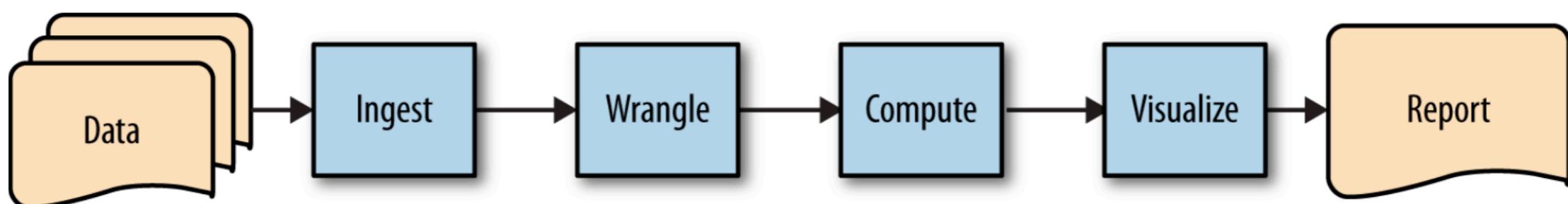
Alex Di Genova

03/10/2022

Hadoop

Introducción

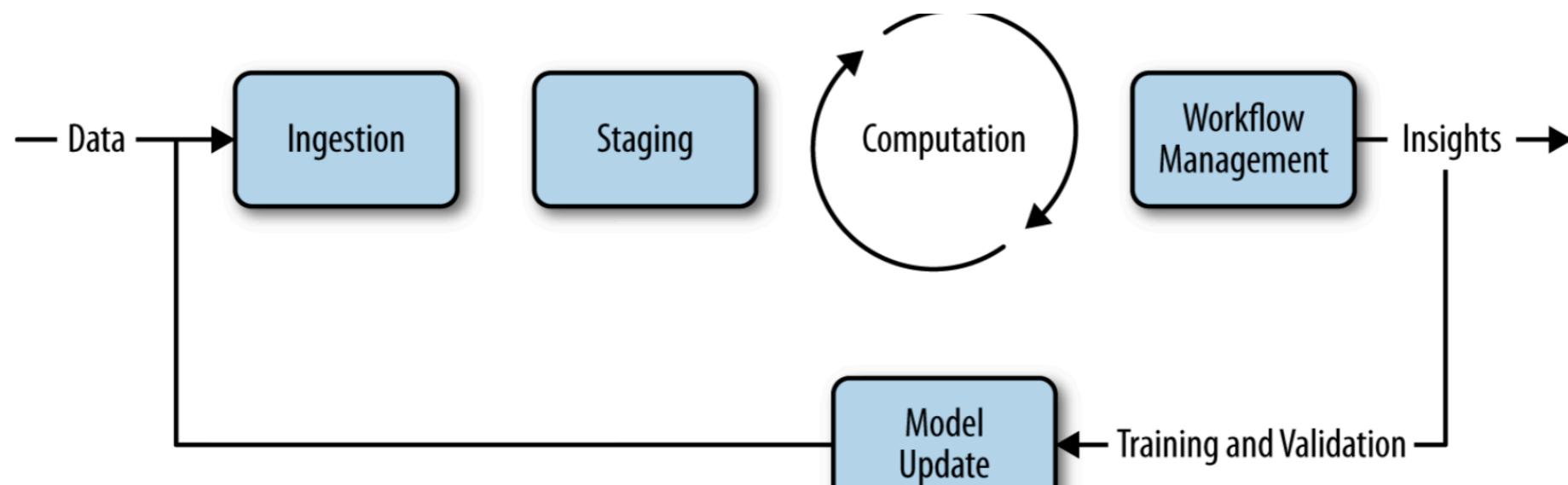
- Data science workflow.
 - 44 zettabytes (2022)
- Conjuntos de datos que son tan grandes o complejos que el software de procesamiento de datos tradicional es inadecuado para manejarlos.
- El análisis de datos requiere un software masivamente paralelo que se ejecuta en varios servidores.



Hadoop

Big Data workflows

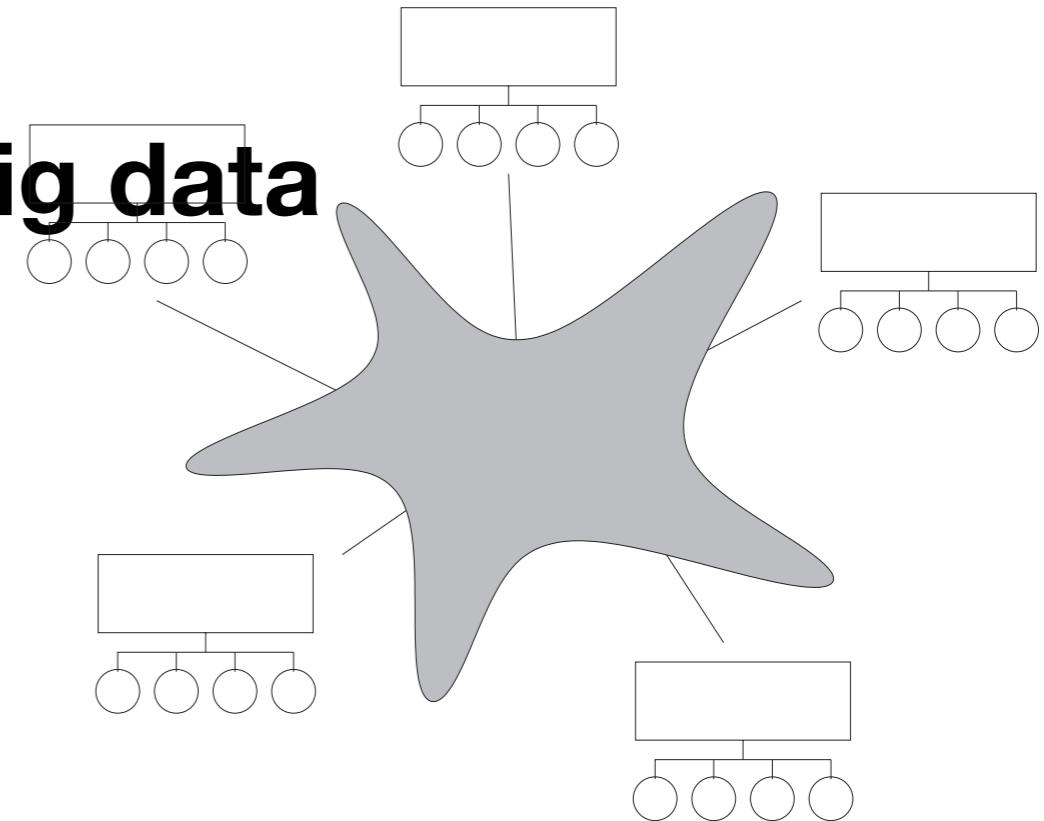
- Escalable y automatico.
- De datos a información.
- Estapas:
 - Fase de consumo
 - Se especifican las ubicaciones de los datos o se anotan datos.
 - Fase de puesta a punto (staging)
 - Se realizan transformacion a los datos para almacenarlos y dejarlos disponibles para procesamiento (normalización y estandarización).
 - Fase de computo
 - Minar los datos para obtener información.
 - Desarrollando reportes
 - Construyendo modelos para recomendacion, clustering o clasificacion.
 - Fase de control de workflow
 - Realiza la abstraccion, orquesta y automatiza tareas para permitir que los pasos del worflow sean operacionales (script, applicacion, job).



Hadoop

Un sistema operativo para big data

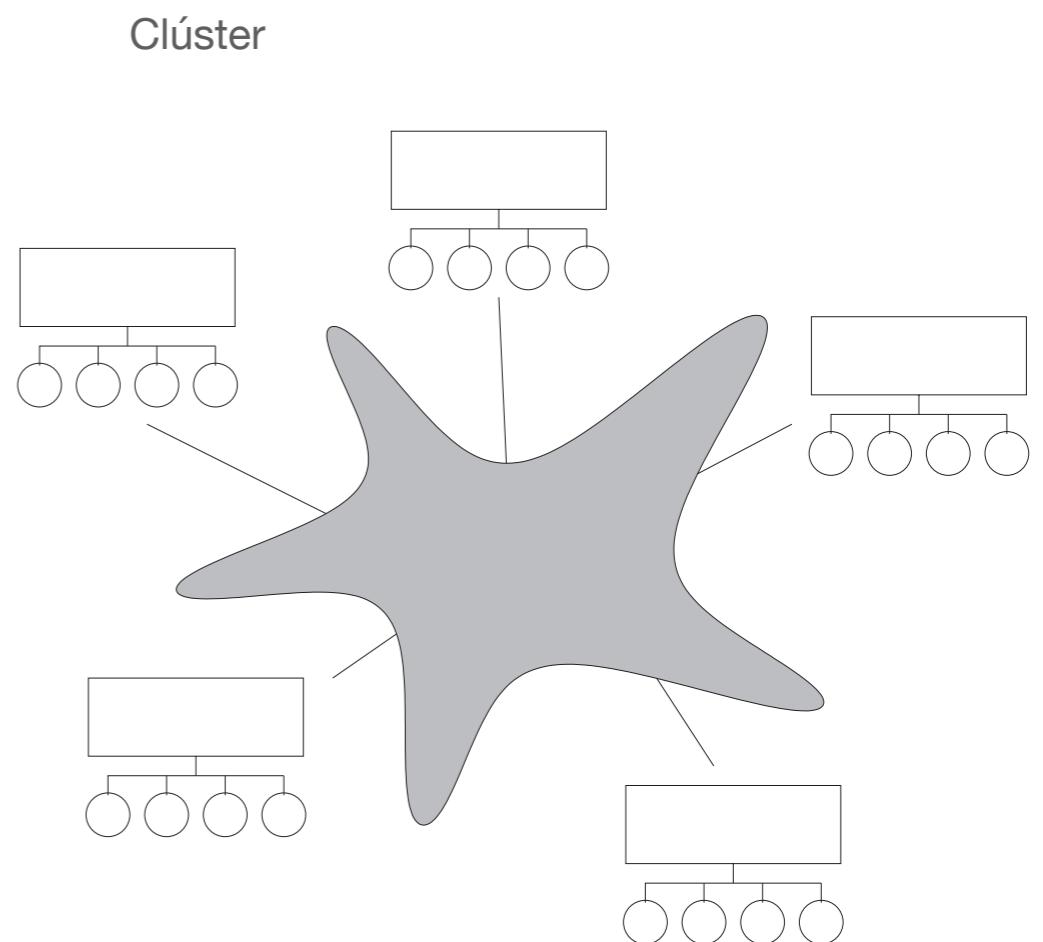
- Hadoop proporciona aplicaciones robustas para operar con datos.
- Hadoop implementa el paradigma computacional denominado Map/Reduce, donde la aplicación se divide en muchos pequeños fragmentos de trabajo, cada uno de los cuales se puede ejecutar o volver a ejecutar en cualquier nodo del clúster.
- 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.

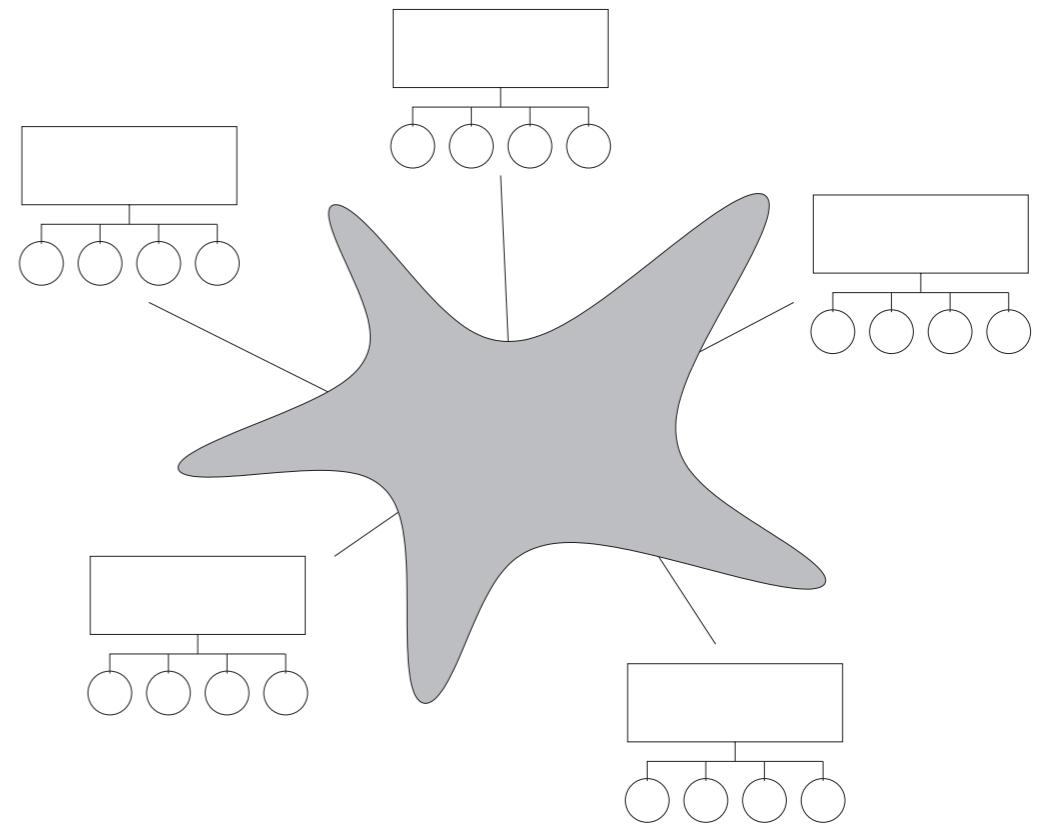


Hadoop

Un sistema operativo para big data

- La computación distribuida no es nueva, pero es un desafío técnico que requiere el desarrollo de algoritmos distribuidos, la administración de máquinas en el clúster y la resolución de detalles de redes y arquitectura.
- Los datos se almacenan en bloques de un tamaño fijo (normalmente 128 MB) y cada bloque se duplica varias veces en el sistema para proporcionar redundancia y seguridad de datos.
- Un cálculo se suele denominar trabajo; los trabajos se dividen en tareas donde cada nodo individual realiza la tarea en un solo bloque de datos.
- Los trabajos se escriben a un alto nivel sin preocuparse por la programación de la red, el tiempo o la infraestructura de bajo nivel, lo que permite a los desarrolladores centrarse en los datos y la computación en lugar de los detalles de la programación distribuida.
- El sistema debe minimizar de forma transparente la cantidad de tráfico de red entre los nodos. Cada tarea debe ser independiente y los nodos no deben tener que comunicarse entre sí durante el procesamiento para garantizar que no haya dependencias entre procesos que puedan provocar “deadlock”.
- Los trabajos son tolerantes a fallas generalmente a través de la redundancia de tareas, de modo que si falla un solo nodo o tarea, el cálculo final no es incorrecto ni está incompleto.
- Los programas maestros asignan trabajo a los nodos trabajadores de modo que muchos nodos trabajadores puedan operar en paralelo, cada uno en su propia porción del conjunto de datos.

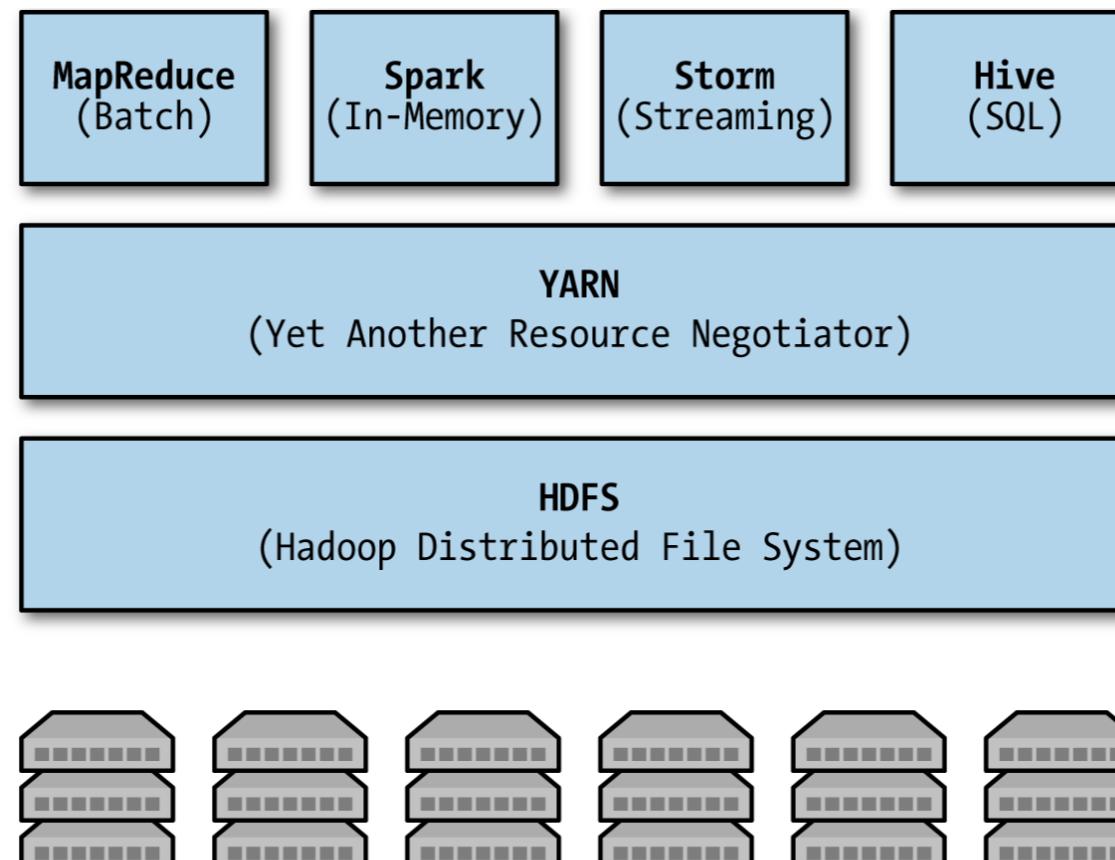
Clúster



Hadoop

Arquitectura

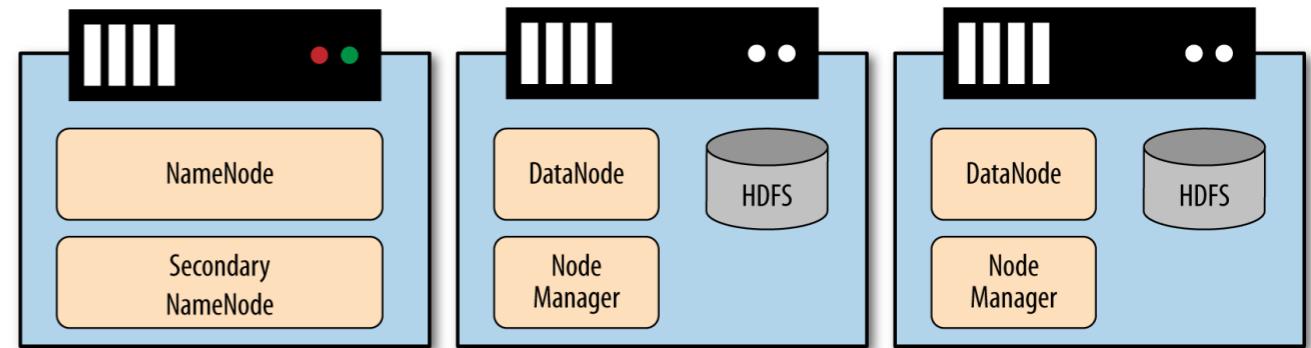
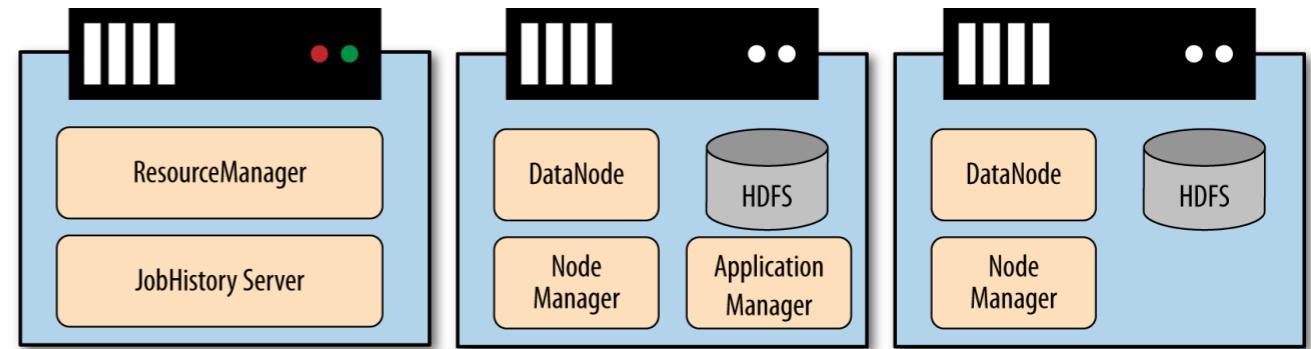
- Hadoop se compone de dos componentes principales que implementan los conceptos básicos de computación y almacenamiento distribuido.
- **HDFS** (a veces abreviado como DFS) es el sistema de archivos distribuidos de Hadoop, responsable de administrar los datos almacenados en discos en todo el clúster.
- **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

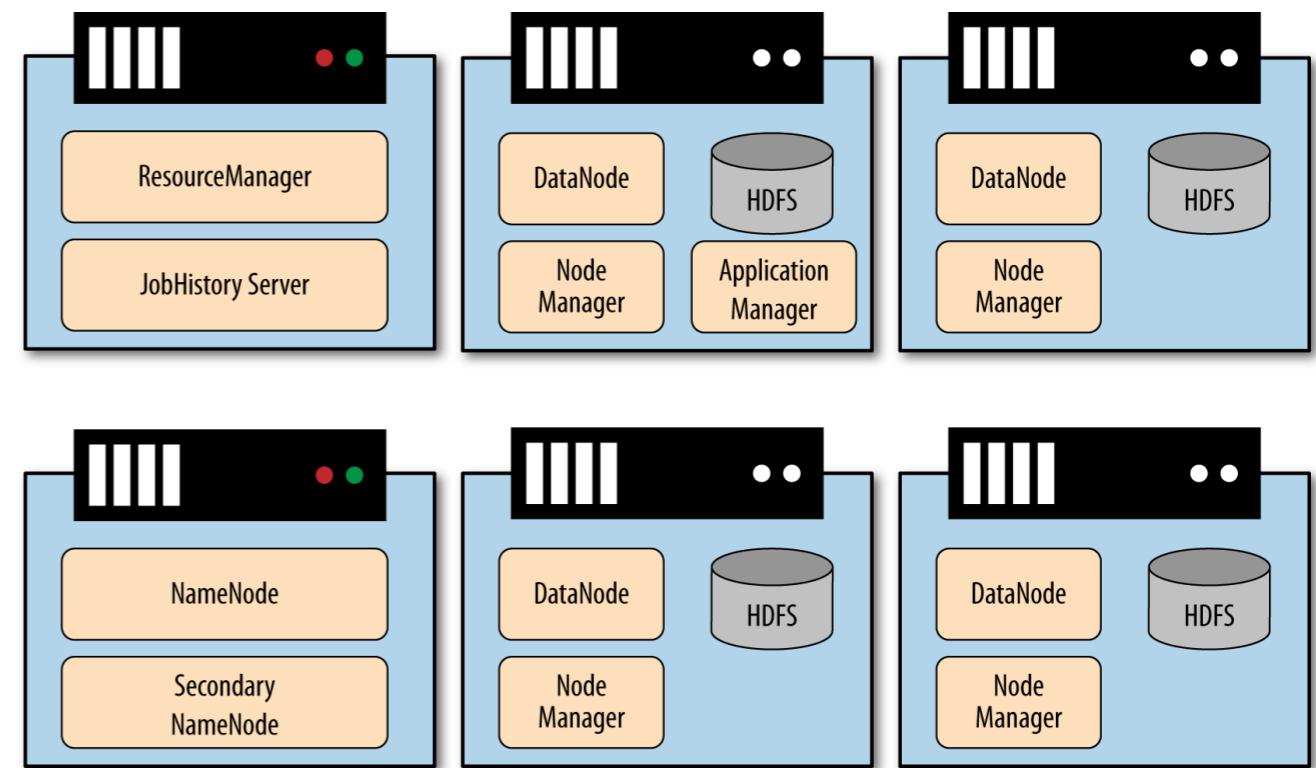
- Hadoop no es hardware que deba comprar o mantener. Hadoop es en realidad el nombre del software que se ejecuta en un clúster.
- YARN y HDFS se implementan mediante varios procesos (demonios), es decir, software que se ejecuta en segundo plano y no requiere intervención del usuario. Los procesos de Hadoop son servicios, lo que significa que se ejecutan todo el tiempo en un nodo de clúster y aceptan entradas y entregan salidas a través de la red (similar a un servidor HTTP)
- Cada uno de estos procesos se ejecuta dentro de su propia máquina virtual Java (JVM), por lo que cada demonio tiene su propia asignación de recursos del sistema y el sistema operativo lo administra de forma independiente.
- Nodos maestros:
 - Estos nodos ejecutan servicios de coordinación para los trabajadores de Hadoop y, por lo general, son los puntos de entrada para el acceso de los usuarios al clúster.
- Nodos trabajadores:
 - Estos nodos son la mayoría de las computadoras en el clúster. Los nodos trabajadores ejecutan servicios que aceptan tareas de los nodos maestros, ya sea para almacenar o recuperar datos o para ejecutar una aplicación en particular.



Hadoop

A Hadoop cluster

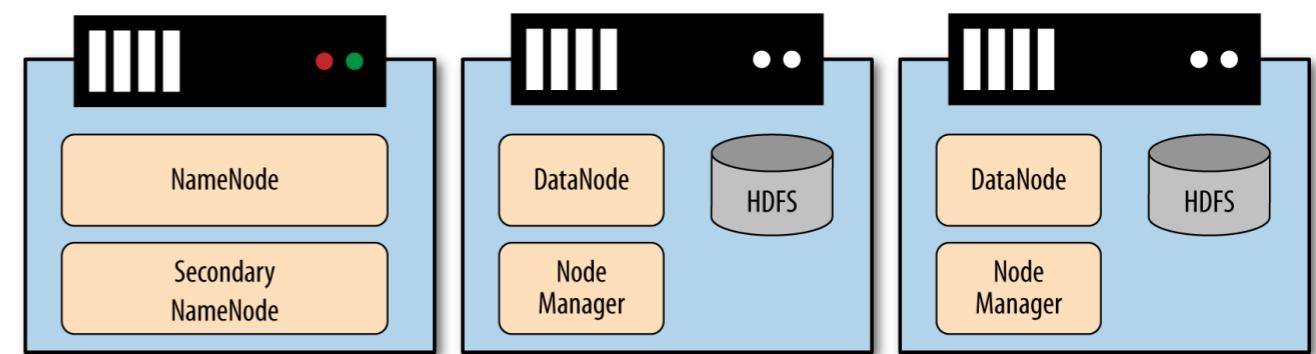
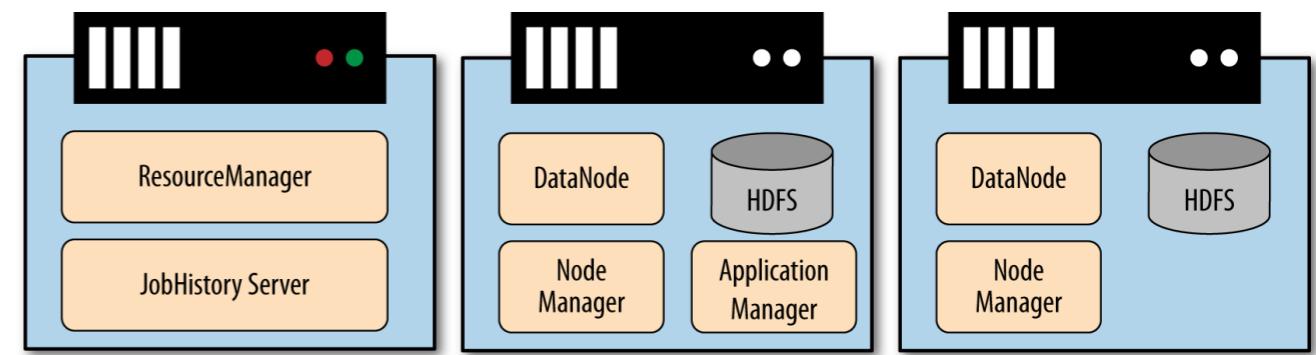
- Para HDFS, los servicios maestro y trabajador son los siguientes:
 - NameNode (Master)
 - Almacena el árbol de directorios del sistema de archivos, los metadatos del archivo y las ubicaciones de cada archivo en el clúster.
 - Secondary NameNode (Master)
 - Realiza tareas de limpieza y puntos de control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



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 elemenos 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 una 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 elemto mas del largo
43         match xs.get(i) {
44             Some(xval) => println!("{}: {}", i, xval),
45             None => println!("Fuerza de indice! {} no existe!", i),
46         }
47     }
```

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

Consultas?

Consultas o comentarios?

Muchas gracias