



Big Data

Introducción - Map Reduce

Texto base: Dr. Juan Esquivel Rodríguez
Profesor: Dr. Luis Alexander Calvo Valverde



Objetivo General del Curso

Entender y aplicar técnicas de análisis de grandes cantidades de datos para la resolución de problemas concretos a través de tecnologías de manipulación, extracción y sintetización estadística.



Objetivos específicos

- Aplicar bibliotecas para la transformación de datos a gran escala para poder sintetizar el conocimiento para futuro análisis.
- Aplicar técnicas de análisis de datos para extraer patrones que mejoren el entendimiento de un problema concreto.
- Aplicar técnicas para aprendizaje automatizado de patrones, basado en datos existentes, para mejorar la certeza de la solución aplicada a problemas concretos.



Contenidos

- Introducción a procesamiento a gran escala
- Fuentes y repositorios de datos
- Procesamiento de fuentes (data frames)
- Procesamiento de atributos
- Organización de datos procesados
- Análisis de datos a gran escala
- Uso de modelos de aprendizaje automático a gran escala




Evaluación

- Proyecto Final 30%
 - Uso de conjunto de datos abiertos o empresa
 - Realizar extracción, transformación y carga
 - Adecuado para predicción (clasificación preferiblemente)
- Tareas 60%
 - Orientación primariamente programada (Apache Spark y Python)
- Participación en clase 10%
 - Basado en lecturas asignadas cada semana



Procesamiento de datos a gran escala

- Actualmente es sencillo acumular datos organizacionales
 - Nuevas características de un sistema pueden llevar a explosión exponencial
- Costos de almacenamiento en la nube son bajos
- Rendimiento se puede afectar orgánicamente
 - E.g. Dos años de transacciones de clientes pueden degradar significativamente
- No es factible almacenar todos los datos en una máquina física
 - Particionamiento es una necesidad
 - Punto donde empezamos a considerar el problema de "Big Data"



Las cinco “V”s del **Big Data**:



Lectura y transformación a gran escala

- Fuentes grandes de datos requieren "elementos de software" que permitan:
 - Leer en forma distribuida
 - Procesar en segmentos (e.g. batches y micro batches)
 - Almacenar de manera distribuida
- ETL
 - Extract
 - Transform
 - Load



ETL

- Extract
 - Se enfoca en la lectura
 - Fuentes típicas incluyen bases de datos relacionales, NoSQL, archivos planos (e.g. CSV, columnares)
- Transform
 - Crear datos más complejos
 - Ajustar de acuerdo a formato de salida
- Load
 - Cargar en la fuente donde serán consumidos
 - Fuentes pueden ser análogas a las de extracción



Frameworks ETL

- Clave para cualquier científico de datos
 - No se puede crear modelos sin datos de entrenamiento!
- Se basan en simplificar algoritmia de procesamiento a operaciones de un solo registro (e.g. fila)
 - Operación sobre un registro es independiente a cualquier otro registro
 - El diseño debe usar fuertemente transformaciones (map) y agregaciones (reduce) para resolver los problemas
- Map Reduce
 - Idea seminal que potenció el desarrollo tecnológico de Google Search
 - Base conceptual de frameworks actuales, como Apache Spark



MapReduce

- Tecnología desarrollada en Google para procesar distribuidamente
 - [Paper publicado en 2004](#)
- Ejemplo clásico: frecuencia de palabras
 - Corpus de millones de libros de texto
 - Queremos contar la frecuencia de cada palabra a través de todos los libros
 - Algoritmo fuerza bruta:

```
func count(words []string) {  
    var counts map[string]int  
  
    for _, word:=range words {  
        counts[word] += 1  
    }  
    printSorted(counts)  
}
```



Based on Functional Programming

Map = apply operation
to all elements

$$f(x) = y$$

Reduce = summarize
operation on elements

by University of California San Diego



Problemática de implementación

- Por qué no una implementación "tradicional" en una sola máquina?
 - Si el orden magnitud de archivos es en millones, la memoria se convierte en el cuello de botella
 - Tal vez no es necesario tener todos los datos en memoria pero hace más lento el procesamiento
- Múltiples núcleos de procesamiento
 - Por ejemplo, dividir entre 10 núcleos
 - Cada núcleo toma 1/10 de los archivos
 - Genera conteos parciales y devuelve a un controlador
 - El controlador integra los resultados




Tolerancia a fallos

- Al repetir este proceso con 1000 máquinas, un solo controlador colapsaría.
- Podemos agregar una jerarquía de controladores, por ejemplo:
 - Cada 10 máquinas envían a un controlador intermedio
 - Cada controlador intermedio envía a un controlador mayor
- Para poder escalar este tipo de arquitectura debe utilizarse un número alto de unidades de procesamiento
 - El fallo de una sola máquina, por sí sola, no es alta por una gran cantidad de meses, en la práctica
 - Cuando existen miles de máquinas la probabilidad es bastante alta



Tolerancia a fallos

- Error que una sola máquina falle: $\epsilon=0.001$
- Probabilidad que 10 máquinas ejecuten el proceso sin tener errores
 - $(1-\epsilon)^{10}=0.999^{10}=0.9900448802$
 - En 99% de las ejecuciones no debería existir errores en las máquinas
- Para 1000 máquinas
 - $(1-\epsilon)^{1000}=0.999^{1000}=0.3676954248$
 - Todos los núcleos son exitosos únicamente 37% de las veces
- El modelado debe incorporar tolerancia a fallos integralmente
 - Replicar entradas (e.g. 3 copias)
 - Distribuir trabajo entre máquinas no relacionadas entre sí
 - Utilizar checksums
 - Etc.



MapReduce - En detalle

- Base de MapReduce
 - Modelo de programación para procesar y generar grandes conjuntos de datos.
 - Todo el procesamiento se basa en expresar los datos en pares llave/valor.
- Se debe definir dos tipos de funciones
 - Función de mapeo: de los pares originales a una representación intermedia
 - Función de reducción: une todas las llaves en una sola entrada
- Existe una fase oculta a los usuarios que ordena los datos (shuffle)
 - Si se tienen R máquinas se asignan datos intermedios a nodos
 - Selección basada en índice de nodo = $\text{hash}(\text{llave}) \% R$
- En la práctica una gran cantidad de problemas SI se pueden modelar
- MapReduce incluía un ambiente de ejecución encargado de muchas de estas tareas

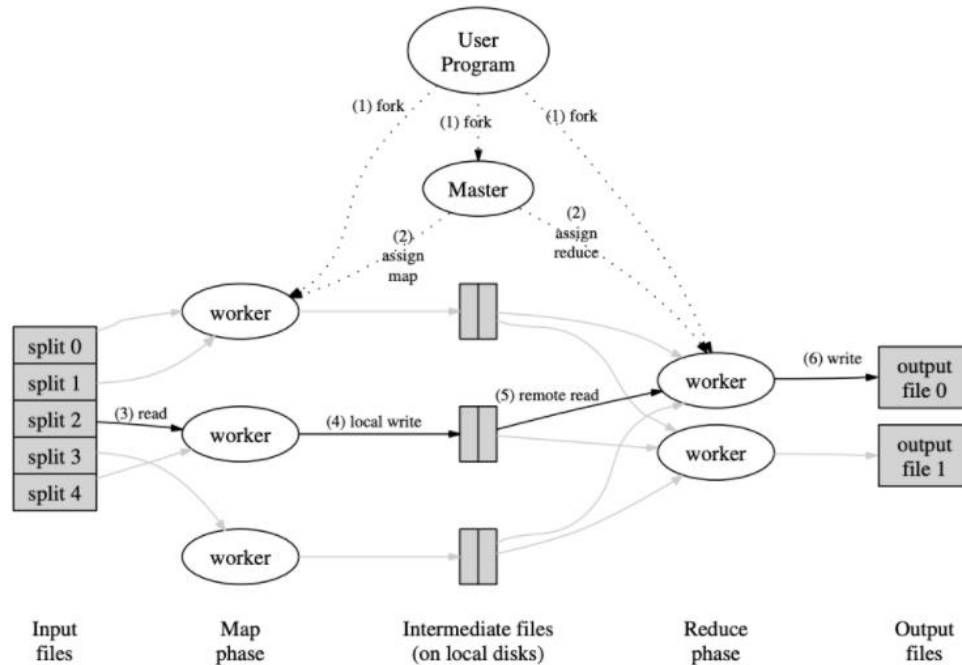


MapReduce - Conteo de palabras

```
map(String key, String[] value):  
    // key: document name  
    // value: document contents  
    for each word in value:  
        EmitIntermediate(w, "1")
```

```
reduce(String key, Iterator values):  
    // key: a word  
    // values: a list of counts  
    int result = 0;  
    for each v in values:  
        result += ParseInt(v);  
    Emit(AsString(result));
```

Ambiente ejecución MapReduce





Limitaciones MapReduce

- Los problemas deben expresarse como una colección de elementos llave/valor
 - Si no se puede expresar de esta forma, no se puede resolver
- Problemas con algoritmos iterativos
 - La premisa de independencia entre registros lo impide
- Desarrollar en MapReduce es complejo
 - Tema para futura clase!



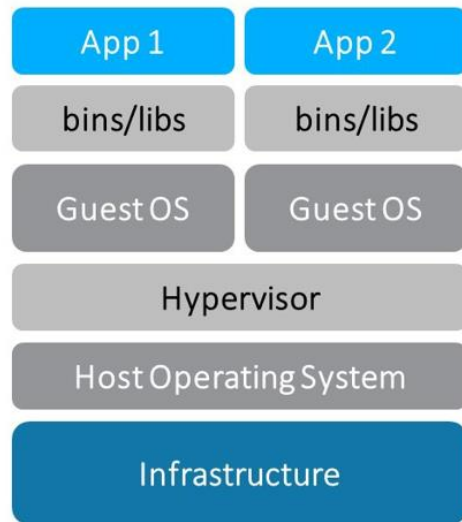
Apache Spark

- Proyecto desarrollado en UC Berkeley y ahora mantenido por Apache Foundation
- Posee integraciones e implementaciones en las plataformas primarias de computación en la nube
 - AWS
 - GCP
 - Azure
- Basado en una abstracción llamada Resilient Distributed Dataset (RDD)
 - Colección de elementos que pueden ser almacenados a lo largo de múltiples nodos
- Spark permite el uso de múltiples fuentes de datos
- La tarea primaria de un programador es diseñar las operaciones sucesivas para transformar el RDD de su forma original a la salida deseada

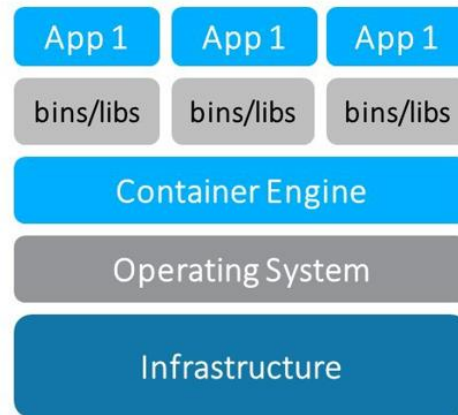


Spark en Docker

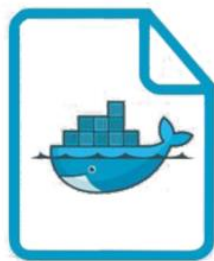
- Configuración de un ambiente Spark puede ser complejo
- Docker es un framework para manejo de contenedores
 - En el repositorio del curso hay 2 Dockerfile(s) que se usarán
- El objetivo del curso NO es aprender Docker
 - Es posible usar un ambiente local, sin embargo, nuestra recomendación es usar Docker
 - Máquinas virtuales han dado resultados menos satisfactorios en complejidad de configuración



Virtual Machines

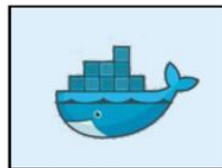


Containers



Dockerfile

Build →




Docker
Image


Run →




Docker
Container




 [Product](#) [Team](#) [Enterprise](#) [Explore](#) [Marketplace](#) [Pricing](#)

Search


 [juanmlesquivel / bigdataclass](#) Public








 N

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#)

 main  1 branch  0 tags

[Go to file](#) [Code](#)

 [juanmlesquivel Update primary Dockerfile to the light version](#) 82ae991 on 21 Jul 2021 19 commits

 basic	Add simple CSV read and print example (#2)	2 years ago
 db	Add configuration to load a psql instance. Add sample code to read us...	2 years ago
 kafka	Add Kafka sample (#11)	2 years ago
 sparkml	Add Spark ML Jupyter Example (#10)	2 years ago
 sparkpsql	Add configuration to load a psql instance. Add sample code to read us...	2 years ago
 testsample	Add pytest spark sample (#6)	2 years ago
 transactions	Add answer to practice question, class 1 (#5)	2 years ago



Docker - Comandos iniciales

- `docker run -dp 80:80 docker/getting-started`
- `docker image ls`
- `docker container ls`
- `docker ps -a`
- `docker container stop <container>`
- `docker container rm <container>`
- `docker image prune`



Ejemplo básico de lectura en Spark

- Herramientas requeridas:
 - Docker Desktop
 - Git: Código en Github
 - Pueden descargar Github Desktop o bien herramienta de línea de comandos
- <https://github.com/juanmlesquivel/bigdataclass/tree/main/basic>
- `docker build --tag bigdata .`
- `docker run -i -t bigdata /bin/bash`



Ejemplo transacciones

- Muestra versiones sencillas de diferentes operaciones
 - Lectura
 - Transformación
 - Agregación
 - Unión
- <https://github.com/juanmlesquivel/bigdataclass/tree/main/transactions>



Ejercicio práctico

- Tomar la salida del ejemplo de transacciones y ajustar por el tipo de cambio
 - Archivo de tipo de cambios en repositorio
- Se puede asumir que los valores en la tabla de transacciones están en dólares y deben ser ajustados al tipo de cambio de los dos clientes (uno en Colones otro en Euros)
- La salida del programa modificado debe contener una columna con los valores monetarios ajustados



pytest

- Framework para pruebas unitarias en Python
- Objetivo ejemplificado
 - Tenemos una función `unir(una_lista, otra_lista)`
 - Queremos revisar que `unir([1, 2], [3, 4])` devuelva una lista del 1 al 4
 - En código esto sería revisar que `[1, 2, 3, 4] == unir([1, 2], [3, 4])`
- Las pruebas unitarias estándar tienen 3 fases
 - Preparación
 - Ejecución de código bajo prueba
 - Revisión de aserción
- Deben ser
 - Confiables
 - Mantenibles
 - Fácil de leer



Referencias

- Schutt, R; O'Neill C. Doing Data Science - Straight Talk from the Frontline. O'Reilly Media. 2013 (Capítulo 14)
- Dean, J; Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters.
<https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf>