UNIVERSIDAD COMPLUTENSE MADRID

MapReduce

Enrique Martín (emartinm@ucm.es)
Sistemas de Gestión de Datos y de la Información
Master Ing. Informática
Fac. Informática

Contenidos

Bibliografía

Bibliografía

- *Hadoop: The Definitive Guide, 4th edition*. Tom White. O'Reilly (2015)
- *Mining of Massive Datasets, 3rd edition*. Jure Leskovec, Anand Rajaraman, Jeff Ullman. Cambridge University Press, 2014. Capítulo 2. Disponible en `http://www.mmds.org`

Introducción

Introducción

- **MapReduce** es un **esquema de cómputo** diseñado para procesar **grandes cantidades de datos** en clústeres de manera **distribuida**
- Utiliza un **sistema de ficheros distribuido**
- Diseñado originalmente por Google para poder calcular la relevancia de las páginas usando el algoritmo PageRank
- Aparece por primera vez en el artículo «MapReduce: Simplified Data Processing on Large Clusters» de Jeffrey Dean y Sanjay Ghemawat, OSDI 2004[1]
  - Explica cómo Google divide, procesa y agrega de nuevo conjuntos de datos de tamaño muy grande en sus clústeres

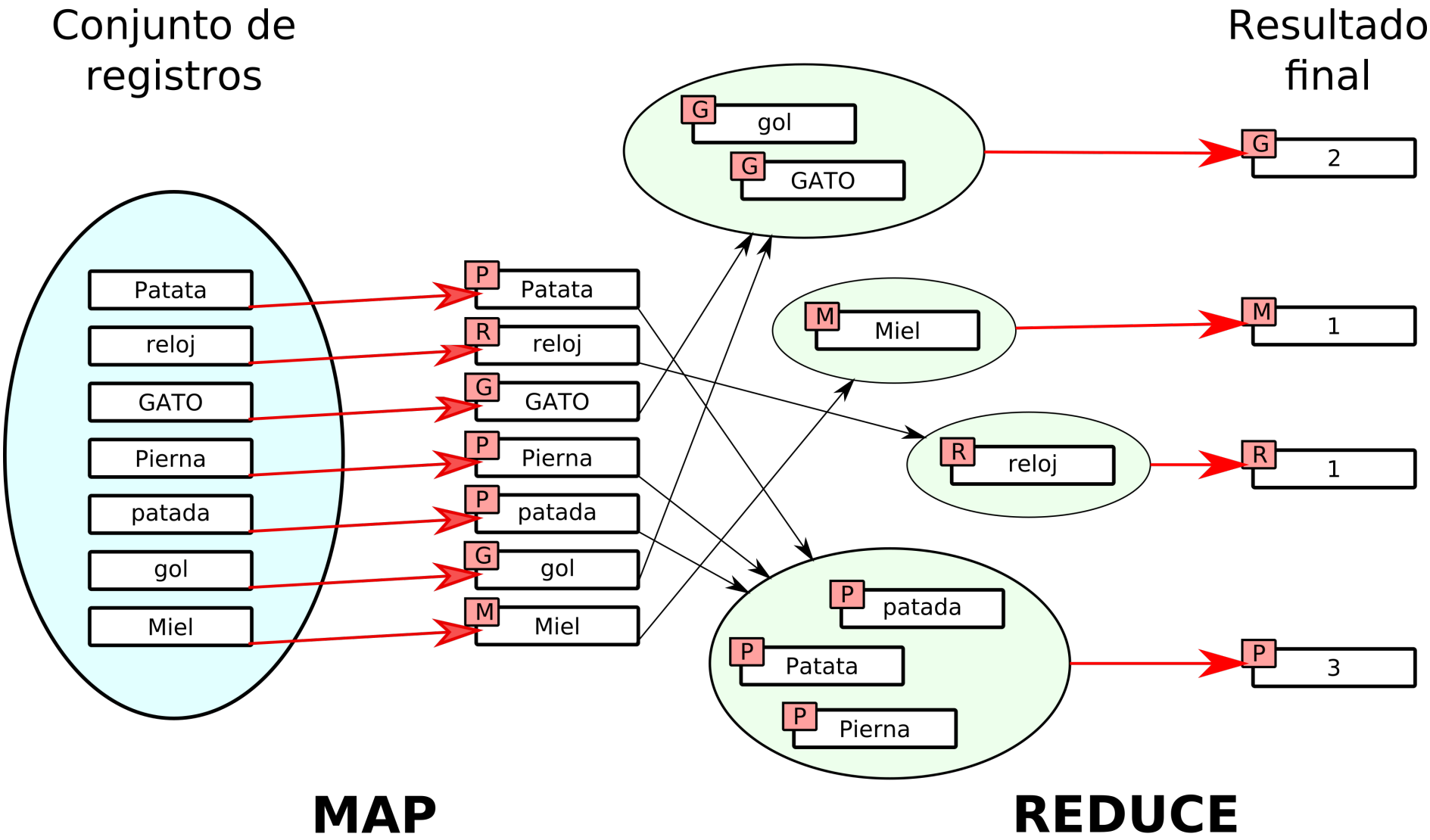
[1] Disponible en `https://www.usenix.org/conference/osdi-04/mapreduce-simplified-data-processing-large-clusters`

MapReduce

- Basado en la estrategia «Divide y Vencerás» para explotar el potencial de procesamiento del clúster
- Divide el análisis completo en análisis sobre fragmentos más pequeños de los datos. Cada uno se realizará en una máquina del clúster
- Al final, se agregan los resultados obtenidos en cada máquina

Intuición de MapReduce

Contar el número de palabras que comienza con cada letra



Fases de MapReduce

- Las dos fases principales de MapReduce son:
  1. **Map**: aplica una función a los datos. P.ej: filtrar ciertos valores
  2. **Reduce**: Combina los valores obtenidos por Map
- Primero se aplica Map, y luego se invoca a Reduce con los resultados obtenidos
- Ambas trabajan con parejas (clave,valor), y son proporcionadas por el usuario

Origen de MapReduce

Los términos Map y Reduce provienen del mundo funcional (Haskell, ML), ya que realizan una acción similar que estos:

- `map` aplica la función a cada uno de los elementos de una lista

```
map :: (a -> b) -> [a] -> [b]
map (+2) [1,2,5] = [3,4,7]
```

- `reduce` (también llamado **fold**) agrega los elementos utilizando una función

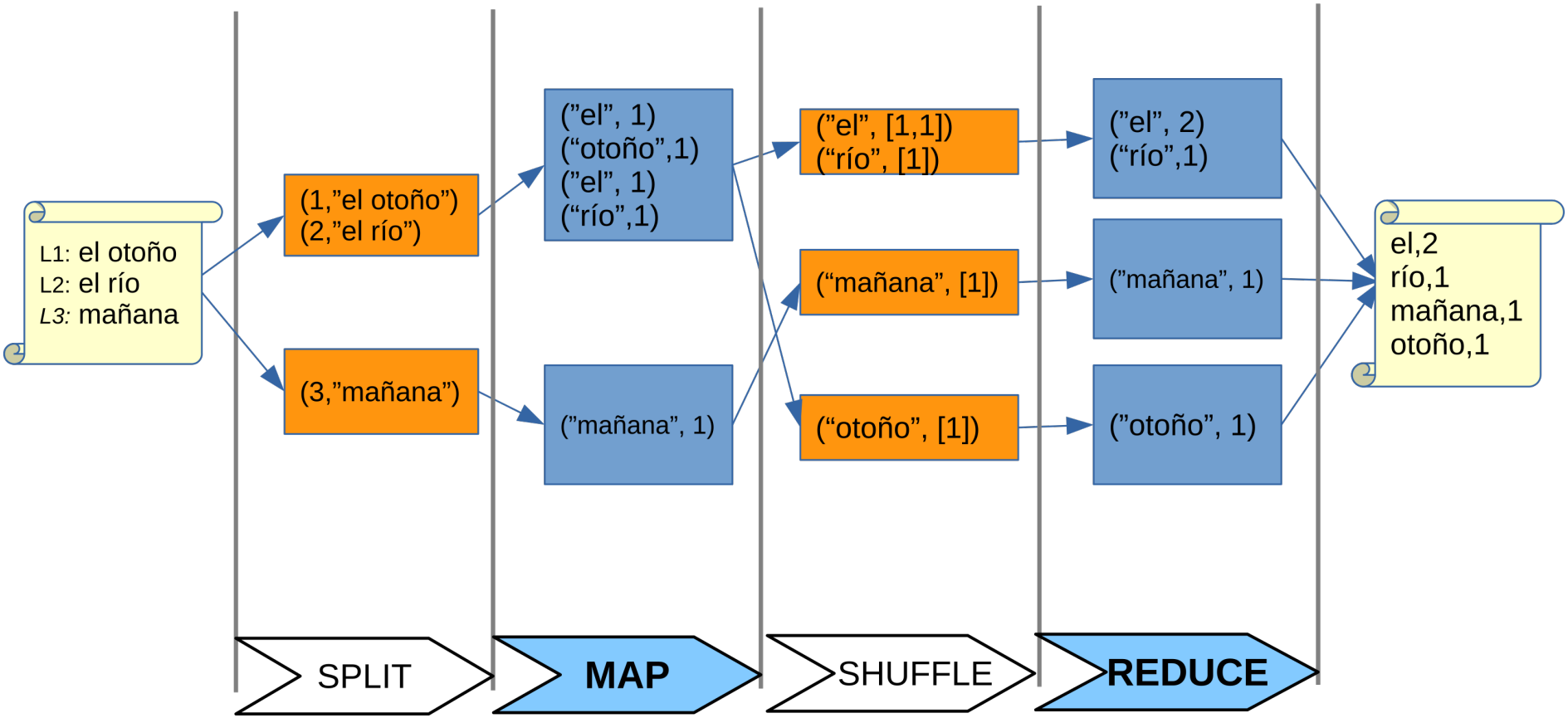
```
reduce :: (a -> a -> a) -> [a] -> a
reduce (+) [1,2,3,4] = 1+2+3+4 = 10
```

Detalle de las fases

- La función `map` toma una pareja de tipo (k_1, v_1) y emite parejas de tipo (k_2, v_2)
- La función `reduce` toma una pareja de tipo $(k_2, [v_2])$ y emite parejas de tipo (k_3, v_3) :
- Las parejas que acepta la función `reduce` **siempre están determinadas** por las parejas emitidas por la función `map`:
  - Si `map` emite parejas `(str, int)`, `reduce` recibirá parejas `(str, [int])`
  - Si `map` emite parejas `(str, str)`, `reduce` recibirá parejas `(str, [str])`
  - Si `map` emite parejas `(int, float)`, `reduce` recibirá parejas `(int, [float])`
  - ...
- Tanto `map` como `reduce` pueden emitir 0, 1 o varias parejas, es decir, pueden no emitir nada o emitir muchas parejas

Fases de una tarea MapReduce

Esquema general de MapReduce



Fase Split

- La fase **Split** divide el fichero (o ficheros) en varios *chunks*. Es una fase predeterminada
- Los *chunks* son grandes (64–128MB). Idealmente son bloques del sistema de ficheros distribuido
- Cada *chunk* es asignado a un *mapper*, que se ejecutará en un equipo del clúster
- Se trata de maximizar la localidad de datos, es decir, que el *mapper* se ejecute en el nodo que ya almacena el *chunk* de datos

Fase Map

- El *chunk* es dividido en varias parejas (clave,valor). Normalmente:
  - **Clave**: *offset* en el archivo o número de línea. No se suele usar e incluso a veces es el valor constante *null*
  - **Valor**: cadena de texto con el contenido de la línea
- Para cada pareja (clave,valor), el *mapper* aplica la función `map` proporcionada por el usuario. Esta función genera 0, 1 o varias parejas
- Las parejas generadas por las invocaciones a la función `map` se almacenan de manera local en el equipo del clúster
  - Al terminar el *mapper*, el nodo tendrá almacenadas una serie de parejas intermedias
  - Estas parejas intermedias deben ser repartidas y enviadas a los equipos del cluster adecuados que ejecutarán la fase Reduce

Fase Map en Python

Ejemplo de función `map` para contar las apariciones de cada palabra (en el *framework* MRJob para Python la función `map` se debe llamar `mapper`)

```
# key : None
# line: str
def mapper(self, key, line):
  for word in line.split():
    yield (word, 1) # Emite la pareja
```

Fase Shuffle and Sort

- Es una fase predeterminada de MapReduce
- Esta fase se encarga de repartir las parejas intermedias generadas en la fase Map entre los distintos *reducers*
- Este reparto se puede hacer de varias formas: rangos, hashing, letra inicial...pero debe cumplir que:

Dos parejas con la misma clave irán a parar siempre al mismo *reducer*

Fase Shuffle and Sort

- Una vez repartidas, las claves intermedias se ordenan por clave y se agrupan aquellas que tienen la misma clave
- La agrupación fusiona las parejas de igual clave generando una cuyo **valor es una secuencia** de los valores asociados a dicha clave. Por ejemplo:
  - Parejas ordenadas: `(1,pepe), (2,eva), (2, juan), (5,ana)`
  - Parejas agrupadas: `(1,[pepe]), (2,[eva,juan]), (5,[ana])`
- Estas parejas fusionadas serán la entrada que se pasará a la función `reduce`
- Por tanto, si la función `map` produce parejas de tipo (k_2, v_2), debido a la fusión la función `reduce` deberá aceptar parejas $(k_2, [v_2])$.

Fase Reduce

- Se aplica la función `reduce` proporcionada por el usuario a cada pareja de tipo $(k_2, [v_2])$ producto de la fusión
- La función `reduce` puede generar 0, 1 o varias parejas resultado en cada invocación
- Estas parejas resultado forman parte de la salida de la tarea MapReduce global, y se almacenan en el sistema de ficheros distribuido
  - Usualmente cada *reducer* volcará su salida en un fichero distinto del sistema de ficheros distribuido
  - Estos ficheros se pueden combinar después o utilizar como entrada de nuevas tareas MapReduce

Fase Reduce

Ejemplo de función `reduce` para contar las apariciones de cada palabra (en el *framework* MRJob para Python la función `reduce` se debe llamar `reducer`)

```
# key    : str
# values: generador de int
def reducer(self, key, values):
  yield (key, sum(values)) # Emite pareja
```

Importante

En el *framework* MRJob el parámetro `values` de un `reducer` es un **generador** (no una lista) así que **únicamente se puede recorrer una vez**. Tratar de recorrerlo más de una vez lanzará una excepción.

Ejemplos de tareas MapReduce

Contar apariciones de cada palabra

```
class MRWordCount(MRJob):

  def mapper(self, key, line):
    for word in line.split():
      yield (word, 1)

  def reducer(self, key, values):
    yield (key, sum(values))
```

Contar palabras, líneas y caracteres

Contar palabras, líneas y caracteres **que formen parte de palabras**

```
class MRCounts(MRJob):

  def mapper(self, key, line):
    nchar = 0
    for word in line.split():
    # Manera muy básica de detectar palabras
      nchar += len(word)
      yield ("#words", 1)
    yield ("#lines", 1)
    yield ("#chars", nchar)

  def reducer(self, key, values):
    yield (key, sum(values))
```

Ejercicios MapReduce

- Filtrar un log para obtener el número de páginas web servidas a navegadores Chrome cada hora de cada día.
- Formato de log (CSV):

```
fecha,hora,recurso,navegador
2012/12/03,10:30,/index.html,Chrome
2012/12/03,17:31,/perro.png,Chrome
2012/12/03,18:59,/index.html,Safari
...
```

- Resultado:

```
2012/12/03-17    1254
2012/12/03-18    58476
2012/12/03-19    258
...
```

Ejercicios MapReduce

- Longitud mínima y máxima de los mensajes de Twitter por horas (de cualquier día)
- Formato del fichero: JSON con un mensaje **en cada línea**:

```
{ "user": {"name": "pepe", "location": "Magaluf, ES"},
  "date": "2018/08/29",
  "time": "5:25:34",
  "text": "viva el vino @juanpedro!"
}
```

- Resultado:

```
4   (min:50,max:140)
5   (min:5,max:120)
6   (min:17,max:110)
...
```

Ejercicios MapReduce

- Índice invertido de menciones en tuits, es decir, por cada mención asociar una lista de usuarios que las han realizado
- Formato del fichero: JSON **en cada línea** (como antes):

```
{
  "user": {"name": "pepe", "location": "Magaluf, ES"},
  "date": "2018/08/29",
  "time": "5:25:34",
  "text": "viva el vino @juanpedro!"
}
```

- Resultado:

```
@juanpedro  [pepe, ana]
@ana        [pepe,eva,ana,evaristo]
@luis       [eva,evaristo,julian]
...
```

Ejercicios MapReduce

- Listado de nombres de usuarios de Twitter únicos que han escrito algún mensaje
- Formato del fichero: JSON **en una línea** (como antes):

```
{ "user": {"name": "pepe", "location": "Magaluf, ES"},
  "date": "2018/08/29",
  "time": "5:25:34",
  "text": "viva el vino @juanpedro!"
}
```

- Resultado:

```
ana
eva
evaristo
pepe
...
```

Optimizaciones de MapReduce

Optimizaciones

- Transferir una gran cantidad de parejas entre nodos congestiona la red (que es un recurso muy preciado) y afecta al tiempo total de la tarea
- Existen varias optimizaciones para minimizar el tráfico generado entre la fase Map y Reduce:
  - Uso de la fase **Combiner**
  - Uso de estructuras estáticas en los nodos *mapper* o *reducer*
- Nos centraremos en la fase Combiner

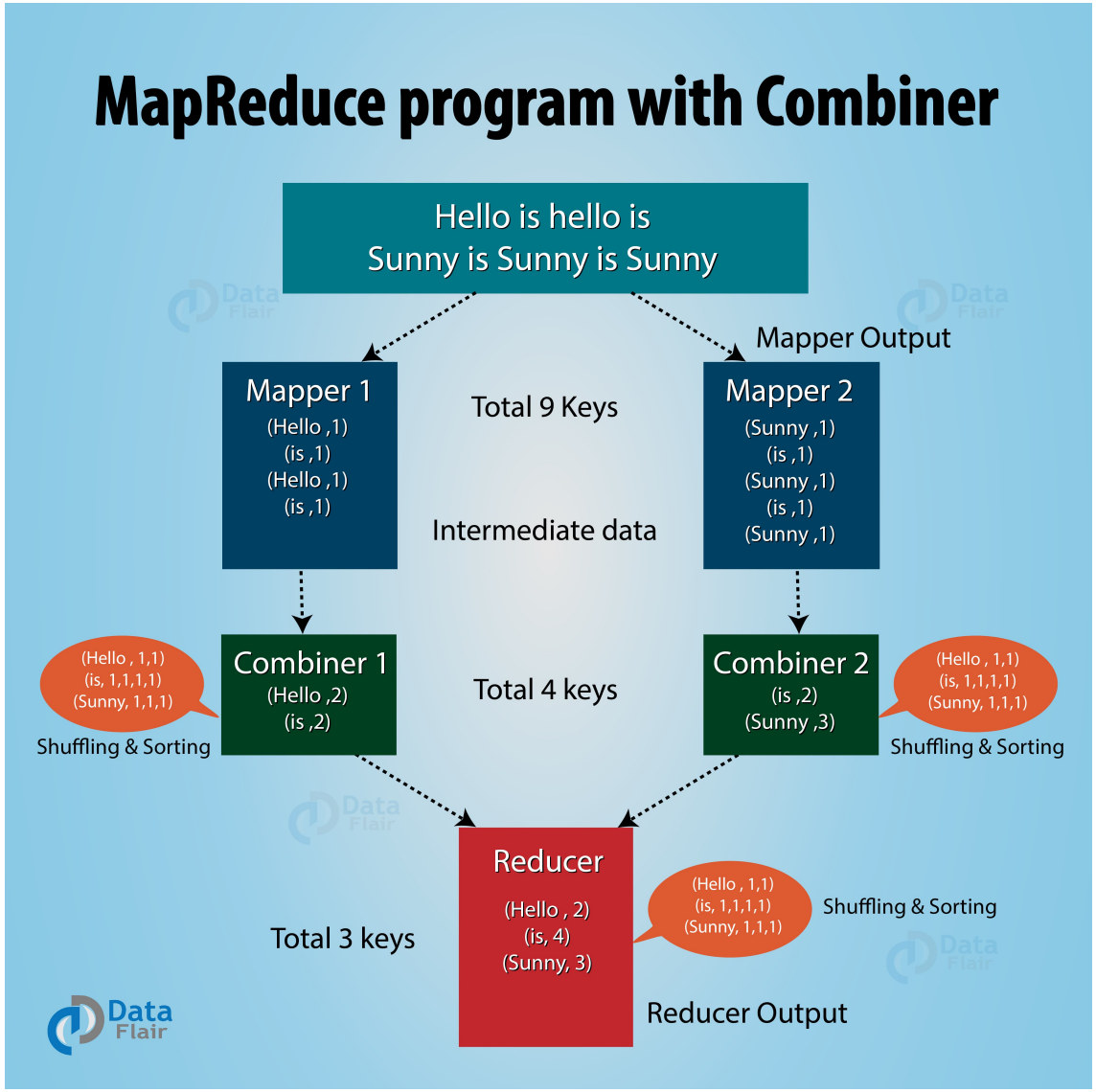
Fase Combiner

- El Combiner es una fase adicional que se puede incluir en una tarea MapReduce
- Está situada entre la fase Map y la fase Shuffle. Se puede considerar una fase Reduce **local a cada *mapper***
- Agrega las parejas generadas **por el *mapper*** para reducir la cantidad de datos a transmitir por la red de nodos *mapper* a nodos *reducer*

Fase Combiner

- Las parejas de entrada que toma la función `combine` son similares a las de entrada de `reduce`, de tipo $(k_2, [v_2])$
- Por tanto, antes de ejecutar el Combiner, las parejas generadas por cada *mapper* deben combinarse (de manera local al nodo) tal y como ocurre en la fase Shuffle
- Las parejas generadas por `combine` deben ser el mismo tipo generado por `map` (pues luego irán a la fase Shuffle), es decir, de tipo (k_2, v_2)
- Un `combiner` es una función de tipo $(k_2, [v_2]) \rightarrow (k_2, v_2)$

Diagrama de una fase Combiner para contar palabras



Fuente: `https://data-flair.training/blogs/hadoop-combiner-tutorial/`

Fase Combiner en MRJob

Ejemplo de función `combine` para contar las apariciones de cada palabra (en el framework MRJob para Python la función `combine` se llama `combiner`)

```
# key    : str
# values: generador de int
def combiner(self, key, values):
  yield (key, sum(values))
```

En este caso el código es el mismo que el de la función `reducer`, pero no siempre será así

Ejecución de la fase Combiner

- La fase Combiner es una optimización, y el sistema **no tiene la obligación de ejecutarla**. Podemos pensar que es una ayuda que el sistema MapReduce puede aplicar *si le viene bien*
- De hecho, un nodo *mapper* podría aplicar esta fase a:
  - Ninguna pareja
  - Todas las parejas
  - Un subconjunto de las parejas
  - Distintos subconjuntos de parejas y luego agregar de nuevo estos resultados obtenidos

Ejecución de la fase Combiner

- Por tanto la tarea MapReduce debe producir la misma salida tanto si se ejecuta la fase Combiner completa en todos los nodos *mapper*, como si se ejecuta parcialmente en alguno de ellos o no se ejecuta en absoluto.
- Para cumplir esto, la función `combine` debe ser **conmutativa** (no importa el orden) y **asociativa** (no importa en cuántas etapas se ejecute):
  - Conmutativa $\rightarrow x + y = y + x$
  - Asociativa $\rightarrow (x + y) + z = x + (y + z)$

Operaciones en la fase Combiner

- La operación `max` es segura para la fase Combiner. Imaginemos la secuencia de valores $1, 2, 3, 4$:
  - `max(max(1,2), max(3,4)) = max(2, 4) = 4`
  - `max(max(1,2), 3, 4) = max(2, 3, 4) = 4`
  - `max(max(3,4), max(1,2)) = max(4, 2) = 4`
- El **promedio no es seguro** para la fase Combiner. Consideremos la secuencia de valores $1, 2, 3$:
  - `avg(1,2, 3) = 2`
  - `avg(avg(1,2), 3) = avg(1.5, 3) = 2.25`