



Hazelcast - MapReduce



MapReduce


MapReduce

MapReduce es un **modelo de programación** para el **procesamiento de grandes volúmenes de datos** de **manera paralela y distribuida** a partir de primitivas simples.

Este modelo, por sus características, está altamente ligado y fue piedra fundacional de la movida llamada Big Data

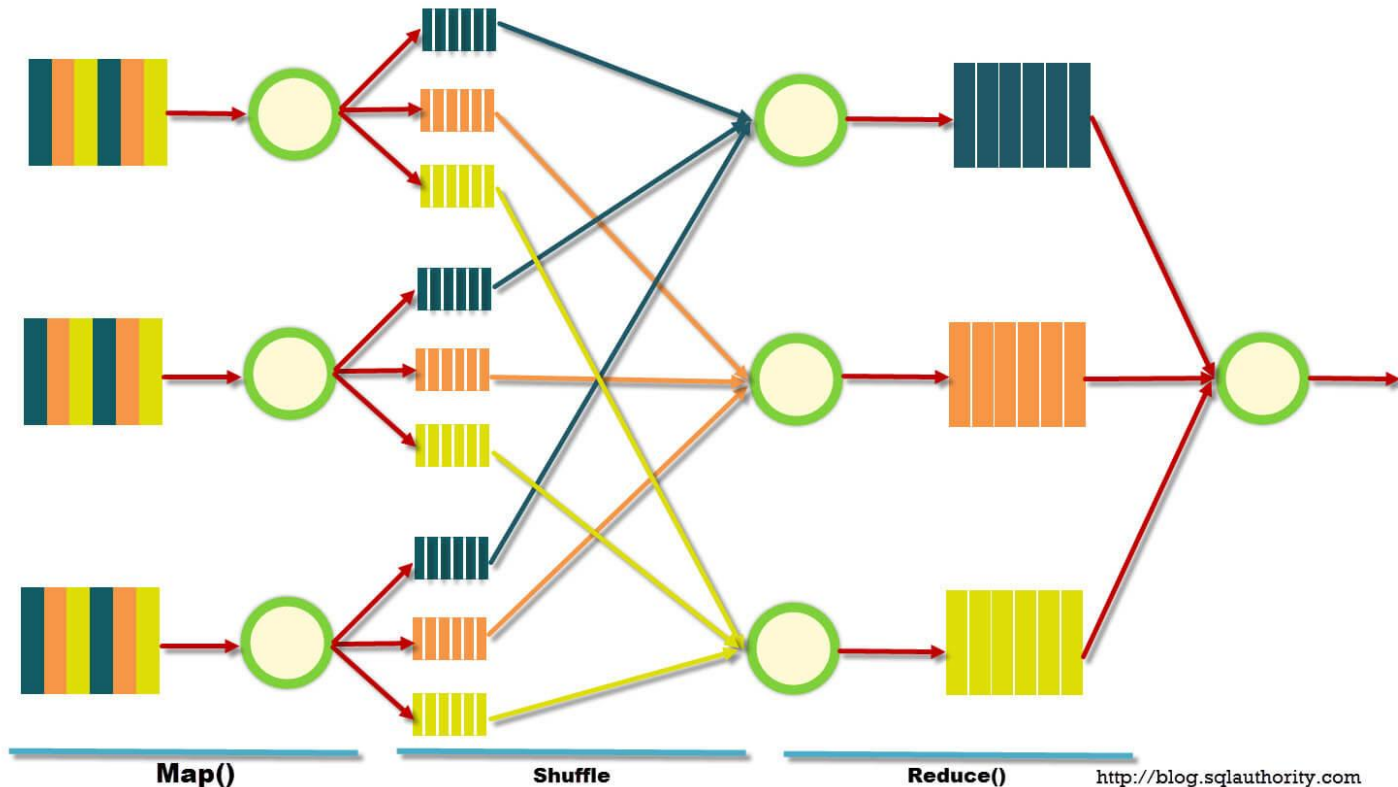


MapReduce

- Basado en un [paper realizado por Dean & Ghemawat](#) en Google.
 - El paper fue a partir de las conclusiones del trabajo del equipo de Google en la mejora del algoritmo [Page Rank](#).
 - La necesidad era **poder procesar el gran volumen de información** en mejores tiempos.
 - El modelo se basa en
 - Las **primitivas de programación funcional** que le dan nombre al modelo (**map y reduce**)
 - En poder subdividir la data para su **procesamiento distribuido** (**divide & conquer**).
 - Trabajar con la información local al nodo lo más posible, evitando trasladarla por la red (**Data Locality**).
- 

MapReduce - Modelo

How MapReduce Works?



Las dos operaciones se encargan de filtrar y transformar la data (**map**) para luego agregar esa data transformada para obtener el valor final (**reduce**)



MapReduce - Datos

Para el modelo **cada unidad de información** o dato que se utiliza como entrada y salida de las etapas cuenta de dos partes:

- » **una clave**: utilizada para clasificar o identificar la información
- » **un valor**: con el contenido del dato

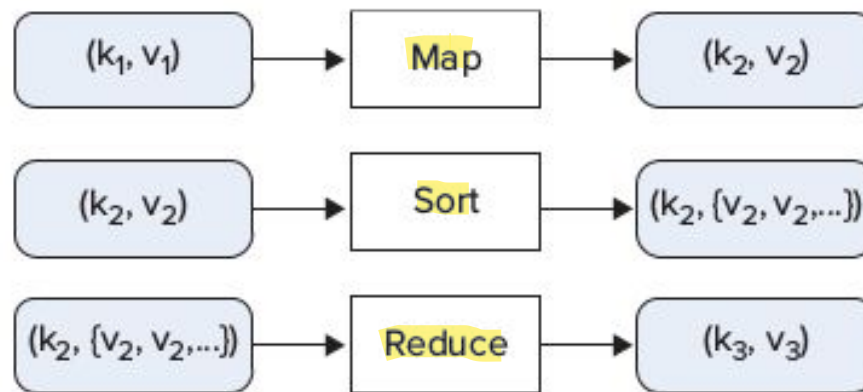
Cada uno puede ser tan complejo como cualquier objeto o registro de una base de datos, o tan simple como una primitiva.

Entonces cada etapa será una función que, aplicada un par clave valor, retorna otro par clave valor:

$$[k2, v2] = F(k1, v1)$$


MapReduce - Etapas

Entonces las etapas son:



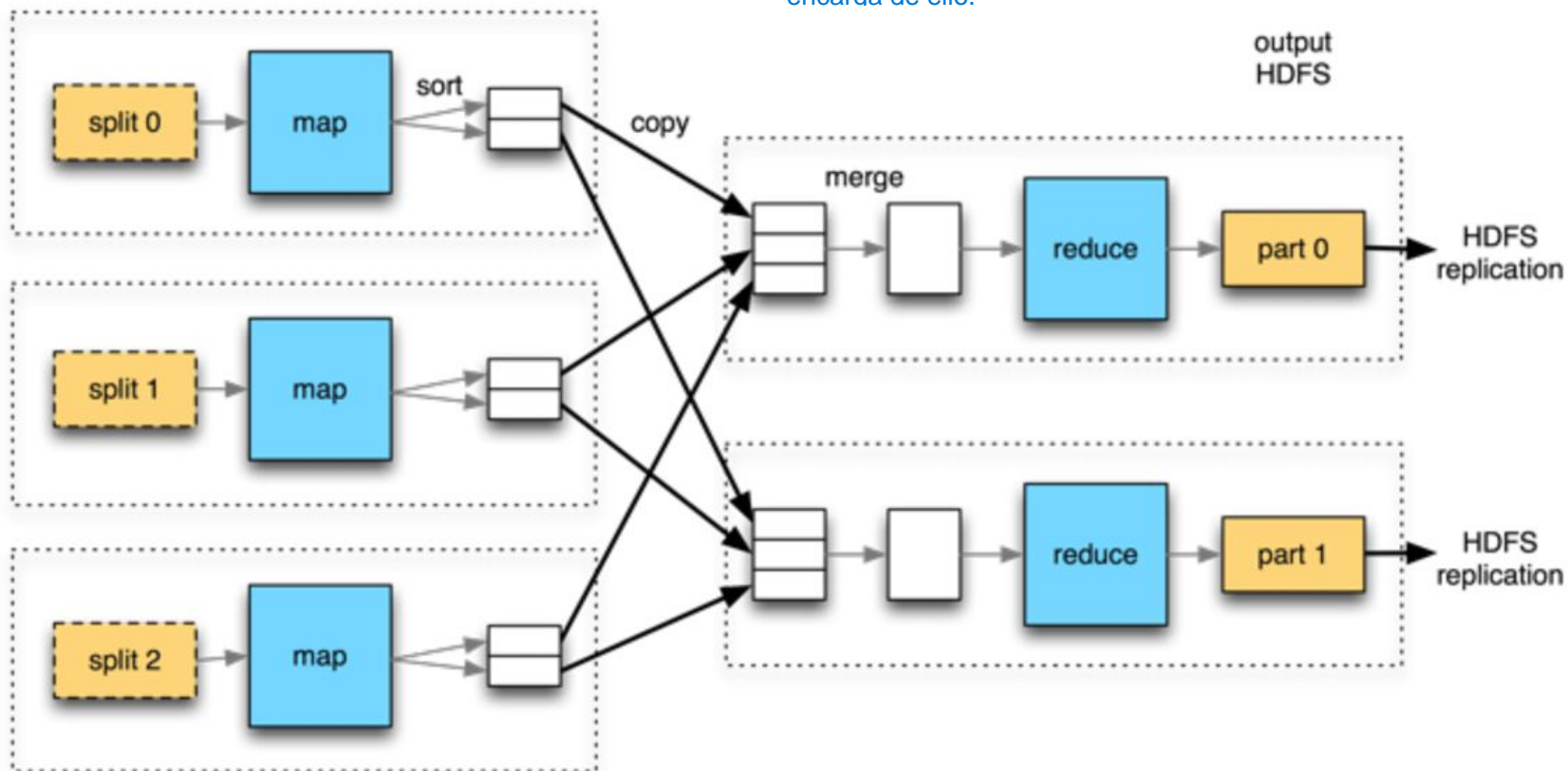
- » Map y Reduce se programan
- » Sort/Shuffle es provisto por el framework

MapReduce - Etapas

input
HDFS

El map se ejecuta distribuido en cada nodo sobre los elementos que hay en ese nodo. Luego se van a sortear, y se pasan a reduce.

Lo importante es trabajar sobre la data que sabemos que tenemos, no sirve pensar que la data esta distribuida porque Hazelcast se encarga de ello.



MapReduce - Map

La operación Map tiene por objetivo transformar la data inicial en información útil para la operación final..

Para lo cual:

Todas estas operaciones necesitan ver las filas de a una, entonces las puedo ejecutar de manera distribuida

- **Filtra**: registros que no son necesarios no son emitidos
- **Proyecta**: no emite valores que no son necesarios para la operación final.
- **Expande**: agrega información que puede obtener de fuentes externas.
- **Multiplica**: para ciertas operaciones tal vez se necesite emitir más de un valor.

proyectar = elegir ciertos campos

Ejemplos

(de un registro genero muchos)

- dado un pais-> emitir un valor por provincia
- dado una frase -> emitir la palabra.

Entonces la operación map:

- recibe un par (key, value)
- emite 0, 1 ó más pares (key, value)



MapReduce - Sort

El framework se encarga de esta etapa por lo cual no requiere programación.

Va a tomar cada par emitido por el mapper y juntar los valores correspondientes a la misma clave.

Esto puede ser un punto complicado de implementación del framework por que requiere:

- conservar los pares hasta que el reducer pueda consumir los valores
- transmitir los valores por red para el nodo que está consumiendo la clave en cuestión.



El sort se encarga de agrupar por key todos los valores emitidos por los mappers para enviarlos a cada reducer.



MapReduce - Reduce

Se crea (o reutiliza) **un reducer por cada clave emitida**.

El mismo **recibe todos los valores** emitidos **por todos los mappers** para la **misma clave**.

A partir de esos valores el reducer los “procesa” para generar un “par final” de clave y valor (o valores).

Entonces la salida de cada reducer es un valor o valores para la clave que le fue asignada.



El reduce transforma todos los valores emitidos para una clave y los transforma en un valor final.

MapReduce - Implementación

Vista las etapas en el "Hello World" de big data: El Word Count

Le paso un texto de entrada y me devuelve la frecuencia de palabras

`IMap<String, String>`

Mapping

Grouping / Shuffling

Reducing

Final Result

Cuando yo pienso con mappers, no tengo que pensar que hay un texto. Tengo que pensar que hay una sola línea.

En realidad hay un solo par clave-valor donde el valor es la línea y es todo lo que me importa.



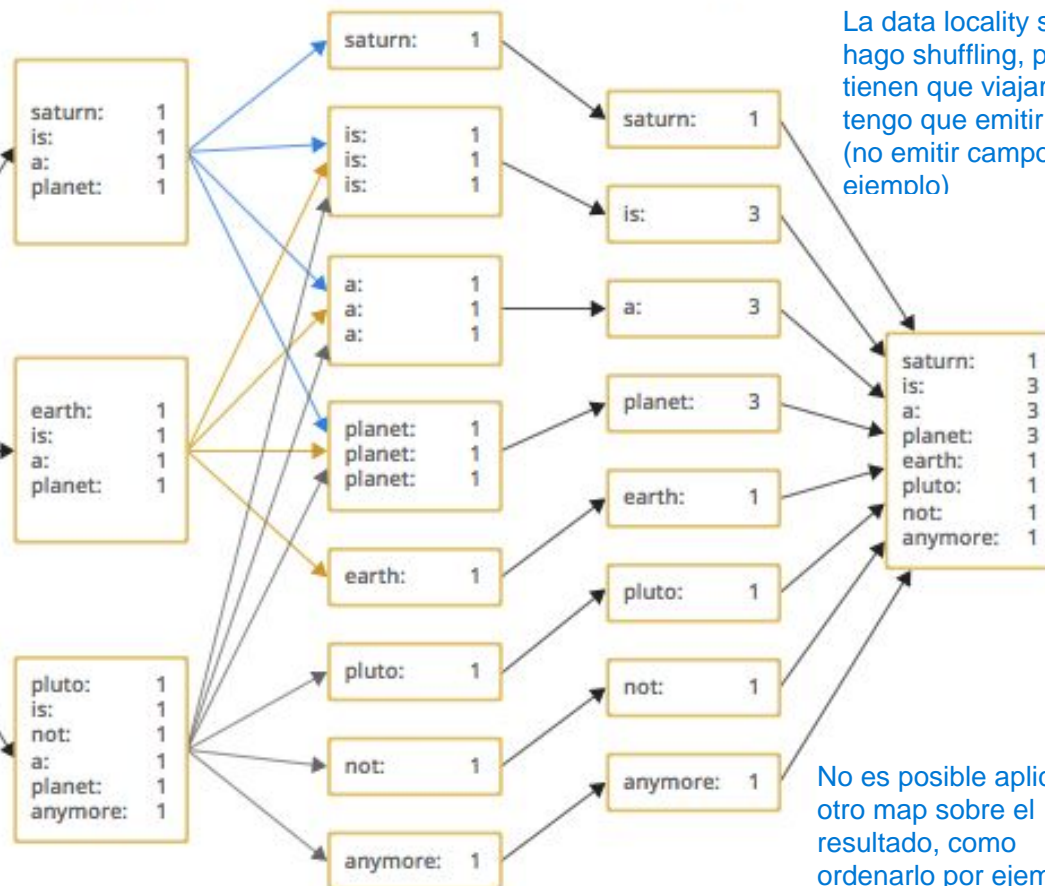
Cada vez que encuentro una palabra nueva en el mapper, emito el par:

Clave = palabra

Valor = 1

Si ya existe la clave

entonces le sumo 1 al valor



La data locality se muere cuando hago shuffling, porque mis datos tienen que viajar por la red. Por eso tengo que emitir lo menos posible (no emitir campos innecesarios por ejemplo)

No es posible aplicar otro map sobre el resultado, como ordenarlo por ejemplo



MapReduce - **Etapas** (Oculta)


El mismo paper dice que a veces hace falta mas que solo Map y Reduce

El modelo al pasar de lo teórico a lo práctico tiene varias etapas más que las mencionadas:

- **Una etapa preliminar** donde se **carga la información** y se dispone la información para el framework. Esto casi ni se toca (se hace automatico). Me deja elegir como particionar mi info.
- **Una etapa intermedia** post-mapper llamada **Combiner** donde se **sumarizan los datos**
- **Una etapa final de post procesado** que permite dar una **última transformación al total** de la información (tal vez ordenar, o guardar en algún medio).

Las dos últimas son opcionales.

Todas las implementaciones tienen Combiner. Se pone en la salida del mapper y hace una pre-reduccion, para reducir al maximo la informacion que pasa por la red. Debe ser implementada al igual que el Reduce, y a veces la implementacion es la misma.



Ejercicio 1 MapReduce

Suponiendo un dataset de los contenidos de diversos libros. Donde cada elemento tiene como **clave** “nombre del libro - número de línea” y como **valor** el **contenido de la línea** para dicho libro.

Para todos los casos, indicar el procesamiento en el Mapper, cuál es el par clave, valor de salida del Mapper y cuál es el procesamiento en el Reducer para llegar al resultado final.

Se quiere tener:

1. El word count de las palabras.
2. El word count de las palabras que empiezan con t.
3. Para las palabras que comienzan con la letra t (case insensitive) del libro "Drácula", indicar el porcentaje de las mismas que consta de 4 letras.
4. Un índice que muestre por cada palabra en cuáles libros está.
5. La frecuencia de las frecuencias de palabras. O sea a partir de la salida de un wordcount (palabra -> cantidad), obtener la frecuencia de las cantidades (cantidad -> número de palabras con esa cantidad).

1)
MAPPER: tokeniza y sanitiza la oracion en palabras. Por cada palabra emite [palabra, 1]
REDUCER: recibe todos los [palabra, 1] y emite los [palabra, total]

2)
Es como el 1 pero se aplica el filtro de que la palabra empiece con t en el mapper

3)
El truco es que tiene que devolver un solo valor (raro en map-reduce).
MAPPER: sanitizo, tokenizo la oracion en palabras, paso a minuscula, etc. Por cada palabra que empieza con t, si es de Dracula, emito [D, 1|0], con 1 si es de 4 y 0 sino.
REDUCER: sumo los valores 1 para obtener el total de las palabras de 4 letras y sumo todas las claves para tener el total de palabras. Emito [D, cant4/cantT]



Hazelcast - MapReduce

Hazelcast - MapReduce Job Tracker

Hazelcast provee una **implementación de MapReduce** además de otras herramientas de procesamiento y query distribuido.

La API de Hazelcast para MapReduce está armada a partir de un DSL, que permite describir la tarea mediante el patrón Builder.

El **punto de entrada es la clase JobTracker**, la misma se obtiene a partir de la instancia de Hazelcast (sea un client node o un member node).

```
// Client Config
ClientConfig clientConfig = new ClientConfig();

// Group Config
GroupConfig groupConfig = new
GroupConfig().setName("group-name").setPassword("group-pass");
clientConfig.setGroupConfig(groupConfig);

HazelcastInstance hazelcastInstance =
HazelcastClient.newHazelcastClient(clientConfig);

JobTracker t = hz.getJobTracker("word-count")
```


Hazelcast - MapReduce Key Value Source

Un **job** obtiene la información a procesar mediante a la clase [KeyValueSource](#), que permite al job iterar sobre los valores de una colección.

La clase **KeyValueSource** provee métodos estáticos para construirla a partir de las colecciones distribuidas (IMap, IList, ISet, Multimap).

El mapa ya tiene clave-valor por default, pero las listas no por ejemplo. KeyValueSource nos deja wrappear nuestras clases para que tenga key-values

```
final IMap<String, String> map = client.getMap("libros");

final KeyValueSource<String, String> source = KeyValueSource.fromMap(map);

final IList<String> list = hazelcastInstance.getList("my-list");
final KeyValueSource<String, String> source = KeyValueSource.fromList(list);
```

En caso de colecciones sin clave aparente (o sea todo lo que no sea mapas), la clave es el nombre de la colección.

Hazelcast - MapReduce - Mapper


La interfaz Mapper provee un método map que por cada elemento del se de datos recibe como parámetros la clave, el valor del ítem, y una instancia de Context que permite emitir los valores de salida.

```
public class TokenizerMapper implements Mapper<String, String, String, Long> {  
    private static final Long ONE = 1L;  
  
    @Override  
    public void map(String key, String document, Context<String, Long> context) {  
        StringTokenizer tokenizer = new StringTokenizer(document.toLowerCase());  
        while (tokenizer.hasMoreTokens()) {  
            context.emit(tokenizer.nextToken(), ONE);  
        }  
    }  
}
```

En caso de precisar acceder dentro del mapa, alguna otra clase o elemento de Hazelcast se puede implementar [HazelcastInstanceAware](#).



Hazelcast - MapReduce Mapper

- A cada mapper se le asigna una partición del nodo en que se ejecuta (si hay), para aprovechar data locality.
 - Si algún nodo tiene varias particiones podrían tener más de una instancia de mapper.
 - Si el mapper termina con su partición se le puede asignar una nueva del nodo.
 - Por cómo está armada la operación map es naturalmente thread safe
- 

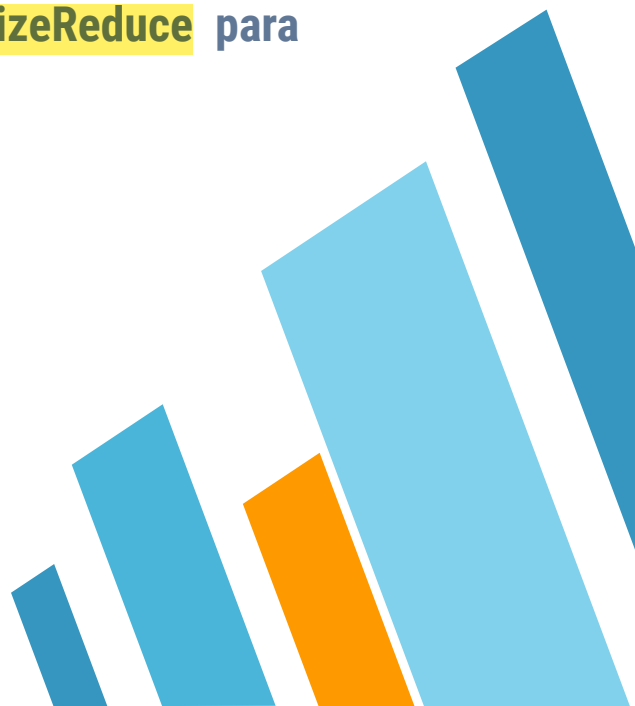
Hazelcast - MapReduce Reducer

```
public class WordCountReducerFactory implements ReducerFactory<String, Long, Long> {  
    @Override  
    public Reducer<Long, Long> newReducer(String key) {    implementar factory  
        return new WordCountReducer();  
    }  
  
    private class WordCountReducer extends Reducer<Long, Long> {  
        private long sum;  
        @Override  
        public void beginReduce () {    como arranco el reduce  
            sum = 0;  
        }  
  
        @Override  
        public void reduce( Long value ) {  
            sum += value.longValue();  
        }  
  
        @Override  
        public Long finalizeReduce() {  
            return sum;  
        }  
    }  
}
```

Para implementar un reducer se deben implementar dos clases asociadas:
ReducerFactory y Reducer.




Hazelcast - MapReduce **Reducer**

- Se requiere un factory porque se ejecuta **un reducer por cada clave única** emitida por los mapper (La clave es el parámetro del método newReducer).
 - El ciclo de vida de un reducer es:
 - Al **llegar una clave** que aún no fue asignada a un reducer se utiliza **el factory para instanciar al nuevo reducer**.
 - Se llama al método **beginReduce** una vez.
 - **Por cada valor** asociado a la clave **se invoca al método reduce**.
 - **Un vez que se terminaron los valores** se invoca a **finalizeReduce** para obtener el valor final para dicha clave.
- 



Hazelcast - MapReduce Reducer

- Como se envía de a chunks, los reducers pueden llegar a quedar suspendidos hasta que aparezcan nuevos valores para la clave asociada.
 - En el caso de una suspensión probablemente no se reinicie en el mismo Thread. Antes de la versión 3.3 no había garantías de visibilidad de variables entre los threads por lo que había que definir todas las variables de instancia como volatile.
- 

Hazelcast - MapReduce Creación del Job

Una vez obtenida la `KeyValueSource` y definidos los mappers y reducers se utiliza el `jobTracker` para instanciar un `Job`, que funciona como Builder para setear los diferentes elementos del job y poder submitear el mismo

```
...  
Job<String, String> job = jobTracker.newJob( source );  
CompletableFuture<Map<String, Long>> future = job  
    .mapper( new TokenizerMapper() )  
    .reducer( new WordCountReducerFactory() )  
    .submit();
```

```
// Attach a callback listener  
future.andThen( buildCallback() );
```

Resultado obtenido por vía asincrónica

```
// Wait and retrieve the result  
Map<String, Long> result = future.get();
```

Resultado obtenido por vía sincrónica



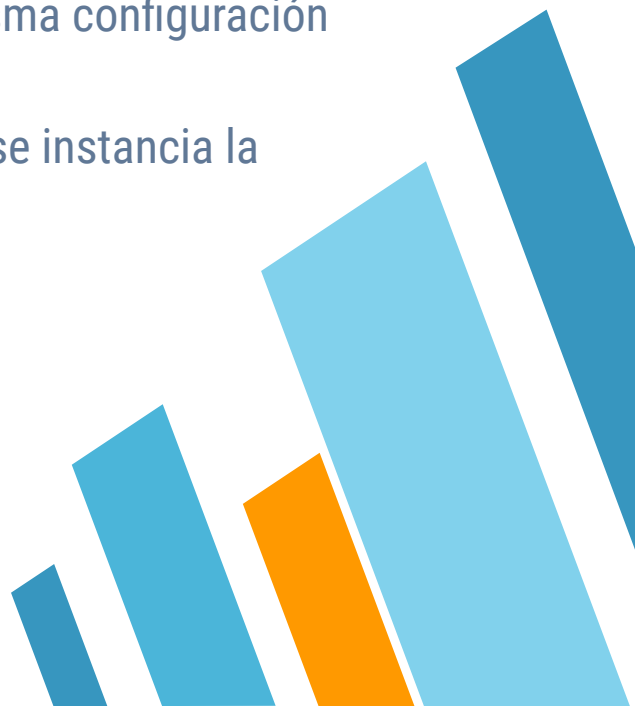
Hazelcast - MapReduce Ejecución

Para ejecutar un job en un cluster se requiere que **cada nodo tenga las clases necesarias para ejecutarlas en el CLASSPATH** (no hay carga dinámica de clases):

- » Mapper
- » Reducer
- » Objetos que se utilicen como keys y/o values.
- » Otros objetos de apoyo para el job.

Todos los nodos deben tener el mismo hazelcast.xml o la misma configuración programática para que formen el cluster.

El **cliente puede ser parte del cluster o no**, depende de cómo se instancia la **HazelcastInstance**.

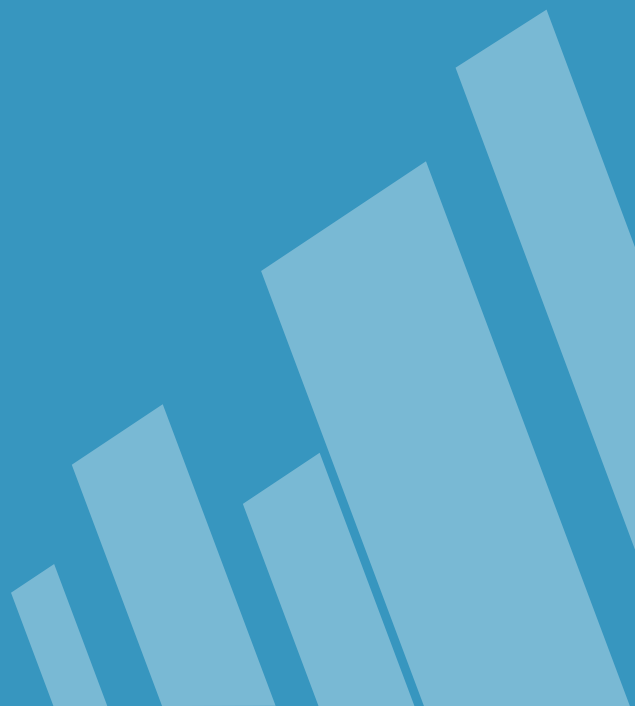




Ejercicio 2

Implementar los punto 1, 2 y 5 que se plantearon en el ejercicio 1.

En campus pueden encontrar el archivo `books.tar.gz` con algunos libros conocidos para usar como fuente de datos.





Hazelcast MapReduce Optimizaciones





Hazelcast - MapReduce **Combiner**

Pensemos el caso donde el texto analizar tiene exactamente 1 millón de veces la palabra "Hola" y 1 millón de veces la palabra "Chau". Dado el wordcount como está planteado corriendo en un cluster con 4 nodos que cada uno corre 1 mapper y 1 reducer.

¿Cuántos valores emitirían los mappers?

- 1 millón de [Hola, 1]
- 1 millón de [Chau, 1]

Si a esto le pongo un combiner, estaría emitiendo dos valores por la red en vez de 2 millones

¿Es esto eficiente?

No, estamos mandando muchos 1 por la red para las mismas palabras. Esto es todo tráfico por la red, congestionándola.



Hazelcast - MapReduce Combiner

Un Combiner funciona como un "reducer" dentro del mapper acumulando los valores que salen del mismo antes de emitirlos por la red.

Al igual que los reducer se instancia uno por clave (y por mapper) así que requiere un factory.

```
public class WordCountCombinerFactory
    implements CombinerFactory<String, Long, Long> {

    @Override
    public Combiner<Long, Long> newCombiner(String key) {
        return new WordCountCombiner();
    }
}
```

Hazelcast - MapReduce Combiner


```
class WordCountCombiner extends Combiner<Long, Long> {  
    private long sum = 0;  
  
    @Override  
    public void combine(Long value) {  
        sum++;  
    }  
  
    @Override  
    public Long finalizeChunk() {  
        return sum;  
    }  
  
    @Override  
    public void reset() {  
        sum = 0;  
    }  
}
```



Hazelcast - MapReduce **Combiner**

A diferencia de los Reducers no se espera a recibir todos los valores, sino que se **acumula hasta que se acaban una cierta cantidad de valores (chunk) y se emite ese resultado parcial.**

El ciclo de vida de un combiner es el siguiente:

- Al **detectar una nueva clave** se utiliza el **factory** para instanciar el combiner.
 - Se **inicializa** con el método **beginCombine**.
 - Se **acumula** mediante el método **combine**.
 - Al **acabarse** el chunk se llama al método **finalizeChunk** para obtener el resultado parcial.
 - En caso de llegar un **nuevo valor** se llama al método **reset** para re-inicializar el conteo.
- 

Hazelcast - MapReduce Collator

El Collator es una clase que se puede agregar opcionalmente al final del proceso para realizar un post-procesado del resultado final del job. Esto sirve para ordenar al final por ejemplo

Recibe un iterable con todas las salidas del proceso, de manera tal que se puede operar tanto con las claves como con los valores de salida.

```
public class WordCountCollator implements Collator<Map.Entry<String, Long>, Long> {  
  
    @Override  
    public Long collate( Iterable<Map.Entry<String, Long>> values ) {  
        long sum = 0;  
        for ( Map.Entry<String, Long> entry : values ) {  
            sum += entry.getValue().longValue();  
        }  
        return sum;  
    }  
}
```

Hazelcast - MapReduce Key Predicate

Pre-filtrar los valores antes de que entren al mapa

KeyPredicate se puede agregar para **pre-seleccionar los valores** que serán procesados. Cuenta con un **método evaluate** que recibe la clave y **en caso de devolver falso el par no se procesa**.

En el caso de que el KeyValueSource implemente el método getAllKeys, el hazelcast puede filtrar las claves antes de iniciar la operación.

```
public class WordCountKeyPredicate implements KeyPredicate<String> {  
  
    @Override  
    public boolean evaluate( String s ) {  
        return s != null && s.toLowerCase().contains( "hazelcast" );  
    }  
  
}
```


Hazelcast - MapReduce Optimización

- El **Combiner** y el **KeyPredicate** se agregan al **Job** de la misma manera que los mappers y reducers
- El Collator se agrega como parámetro al submit

```
final Job<String, String> job = jobTracker.newJob( source );
final ICompletableFuture<Long> future = job
    .keyPredicate(new WordCountKeyPredicate())
    .mapper(new TokenizerMapper())
    .combiner(new WordCountCombinerFactory())
    .reducer(new WordCountReducerFactory())
    .submit(new WordCountCollator());

// Wait and retrieve the result
final Map<String, Long> result = future.get();
```

El submit hay que ponerlo siempre, turrin no sabe por que el collator va en submit y no en un .collator()

Ejercicio 3

- » Analizar para cada uno de los jobs realizados en los ejercicios anteriores si es útil implementar Collator, Combiner o Key Predicate. En caso afirmativo agregarlos.



Referencias y para profundizar

- » [Paper MapReduce](#)
 - » [Documentación Hazelcast Map Reduce](#)
- 



CREDITS

Content of the slides:

- » POD - ITBA

Images:

- » POD - ITBA
- » Or obtained from: commons.wikimedia.org

Slides theme credit:

Special thanks to all the people who made and released these awesome resources for free:

- » Presentation template by [SlidesCarnival](https://slidescarnival.com)
 - » Photographs by [Unsplash](https://unsplash.com)
- 