

Práctica de ejecución de trabajos MapReduce python en Hadoop

Índice

Introdución.....	3
Interface para múltiples linguaxes.....	4
MapReduce en Python: exemplo de wordcount.....	5
Hadoop Streaming: Mapper.....	6
Hadoop Streaming: Reducer.....	7
Hadoop Streaming: envío do traballo a hadoop.....	10
Hadoop Streaming: saída resultante da execución dos traballos.....	14
Hadoop Streaming: outro exemplo sinxelo de wordcount distribuído usando MapReduce, con binarios.....	16
Hadoop Streaming: outro exemplo sinxelo de wordcount distribuído usando MapReduce, con scripts bash.....	17
Resolvendo outras consultas: froitas máis e menos repetidas.....	21
Ordea-los resultados mediante un traballo MapReduce.....	22
Configura-la orde de saída do mapper coa propiedade KeyFieldBasedComparator.....	23
Comandos para interactuar cos traballos.....	26
Exercicios MapReduce con Hadoop Streaming.....	27
Pautas e recomendacións para realiza-los exercicios.....	27
Exercicio 1. Cálculo das ventas totais de cada tenda.....	30
1ª parte.....	30
2ª parte.....	31
3ª parte.....	31
4ª parte.....	32
5ª parte.....	32
6ª parte.....	33
Exercicio 2. Cálculo do día máis chuvioso de cada ano.....	34
map: xerar pares ano-choiva, co ano e o valor de choiva.....	35
reducer: calcula o valor máximo das choivas rexistradas en cada ano.....	37
Execución en hadoop.....	39
Exercicio 3. Cálculo do día máis chuvioso de cada mes, para cada ano.....	40
map: xerar pares periodo-choiva, co ano e mes como chave e a cantidade de choiva rexistrada como valor.....	40
reducer: calcula o valor máximo das choivas rexistradas en cada ano.....	42
Execución en hadoop.....	44
Exercicio 4. Cálculo do día máis chuvioso de cada mes, para cada ano (nos últimos 20 anos)....	45
Exercicio 5. Cálculo do día máis caluroso de cada mes, para cada ano (nos últimos 20 anos)....	45
Exercicio 6.....	46
Exercicios propostos.....	47
Intro to Hadoop and MapReduce.....	48
Execución de traballos mapreduce.....	49
Opcións de execución.....	50
Número de reduces.....	50
Combiner.....	51

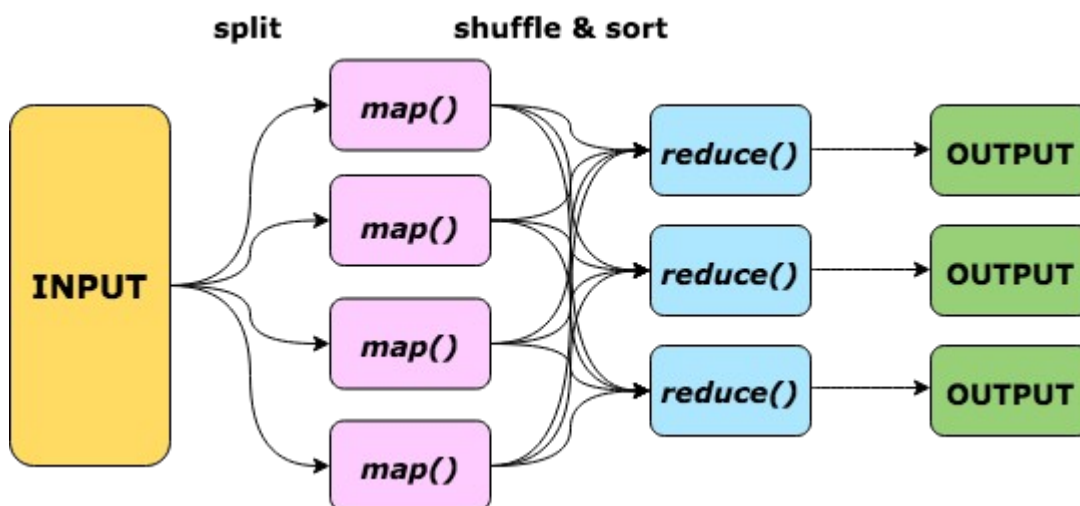
Manexo de datos.....	51
Orde secundario.....	52
Contadores.....	52
Funcionalidades comúns.....	53

Introdución

MapReduce é un paradigma informático deseñado para permiti-lo **procesamento paralelo distribuído de grandes cantidades de datos**.

Os datos repártense entre varios nodos, nos que son procesados por un ou máis **mapeadores**.

Os resultados emitidos polos **mapeadores** son primeiro ordeados e despois pásanse a un ou varios **redutores** que procesan e combinan os datos para devolve-lo resultado final.



Con **Hadoop Streaming** é posible utilizar calquera linguaxe de programación distinta de Java para definir un **mapeador** e/ou un **reductor** cos que executar aplicacións MapReduce.

O **mapeador** e/ou o **reductor** poden ser calquera tipo de executable que lea a entrada de **stdin** e emita a saída a **stdout**. Por defecto, a entrada é lida liña por liña e o contido dunha liña ata o primeiro carácter de tabulación é a **chave**; o resto da liña (excluíndo o carácter de tabulación) será o **valor**.

Se non hai un carácter de tabulación na liña, a liña enteira considérase chave e o valor é nulo. O formato de entrada predeterminado especificase na clase **TextInputFormat** (documentación da API: <https://hadoop.apache.org/docs/stable/api/org/apache/hadoop/mapred/TextInputFormat.html>), pero pódese personalizar, por exemplo, definindo outro separador de campos (documentación de Hadoop Streaming:

https://hadoop.apache.org/docs/stable/hadoop-streaming/HadoopStreaming.html#How_Streaming_Works).

- Documentación oficial de *Hadoop Streaming* para Hadoop:
<https://hadoop.apache.org/docs/stable/hadoop-streaming/HadoopStreaming.html>
- As opcións do comando Hadoop Streaming se documentan en:
https://hadoop.apache.org/docs/current/hadoop-streaming/HadoopStreaming.html#Streaming_Command_Options

Pódense ve-las opcións do comando de Streaming:

```
$ mapred streaming --help
```

Interface para múltiples linguaxes

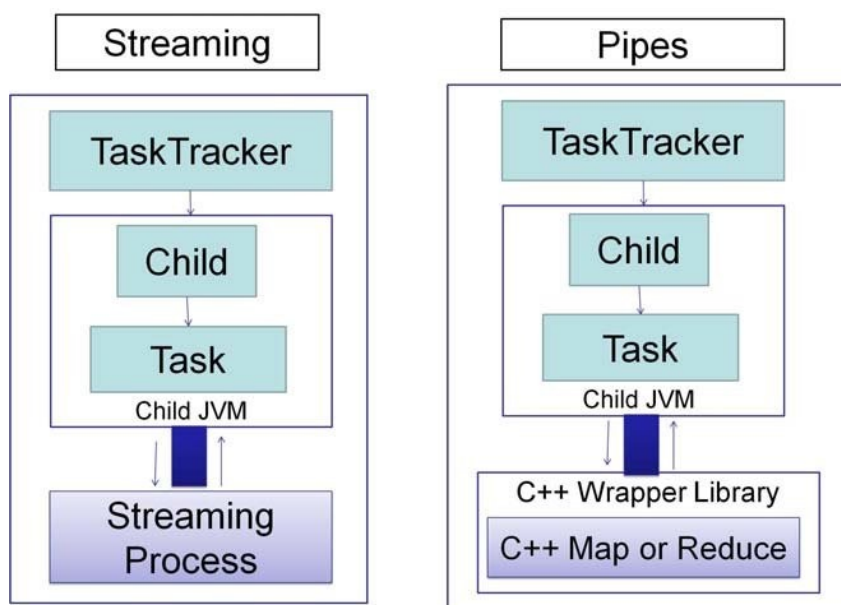
A API de MapReduce está escrita en Java, pero Hadoop provee APIs para escribi-las funcións *map* e *reduce* noutras linguaxes.

- **Hadoop Streaming**

- Permite programar rapidamente mediante scripts (python, perl, etc).
- Hadoop Streaming é unha API que permite escribir aplicacións con funcións map e reduce en linguaxes distintas de Java, usando *streams* estándar de Unix como interface entre Hadoop e os programas escritos en calquera linguaxe usuario/a.
- <https://hadoop.apache.org/docs/stable/hadoop-streaming/HadoopStreaming.html>

- **Hadoop Pipes**

- Permite programar eficientemente utilizando C++.
- Hadoop Pipes é unha interface C++ para Hadoop MapReduce que, a diferenza do *streaming*, usa *sockets* para a comunicación entre Java e C++.
 - Impleméntanse clases de C++ que herdan das clases *Mapper* e *Reducer* que provee Hadoop Pipes.
 - Os métodos map e reduce reciben un obxecto contexto.
 - Permite ler entrada e escribir saída.
- Xestión de procesos máis eficiente que Hadoop Streaming.
- <https://hadoop.apache.org/docs/r1.2.1/api/org/apache/hadoop/mapred/pipes/package-summary.html>



Fonte: https://www.researchgate.net/publication/224217002_Hybrid_Map_Task_Scheduling_for_GPU-Based_Heterogeneous_Clusters#pf3

MapReduce en Python: exemplo de wordcount

Mediante **Hadoop Streaming** podemos utilizar MapReduce con calquera linguaxe compatible co sistema de tubaxes Unix (`()`).

Para comprobalo faremos en Python o código correspondente ó programa *wordcount* de hadoop, o que implicará codificar en Python un programa mapeador e outro redutor.

Para empezar comprobamos que estea instalado python:

```
$ python3
```

```
hduser@hadoop-master:~/mr-wordcount-python$ python3
Python 3.8.10 (default, May 26 2023, 14:05:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
hduser@hadoop-master:~/mr-wordcount-python$
```

Utilizarase o ficheiro de texto *el_quijote.txt* para contabiliza-las súas palabras, polo que tamén deberá estar dispoñible.

```
hduser@hadoop-master:~/mr-wordcount-python$ ls -l
total 8
-rw-rw-r-- 1 hduser hduser 296 Nov  5 20:03 mapper.py
-rw-rw-r-- 1 hduser hduser 600 Nov  5 20:35 reducer.py
hduser@hadoop-master:~/mr-wordcount-python$ ls -l ../el_quijote.txt
-rw-rw-r-- 1 hduser hduser 1060259 Nov  5 13:13 ../el_quijote.txt
hduser@hadoop-master:~/mr-wordcount-python$
```

Excelente artigo base para esta práctica <https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/>

Hadoop Streaming: Mapper

Primeiro creamo-lo mapeador, que se encarga de *parsear* liña a liña o fragmento de documento que reciba, e xerar unha nova saída con tódalas palabras de maneira que cada nova liña se compoña dunha tupla formada pola palabra, un tabulador e o número 1 (que indica que hai unha ocorrencia da correspondente palabra).

- Recibe datos de texto liña a liña da entrada estándar
- Imprime tuplas de saída, separando chave e valor con TAB

```
$ nano mapper.py
```

```
#!/usr/bin/python3
# A primeira liña indica que é un script python

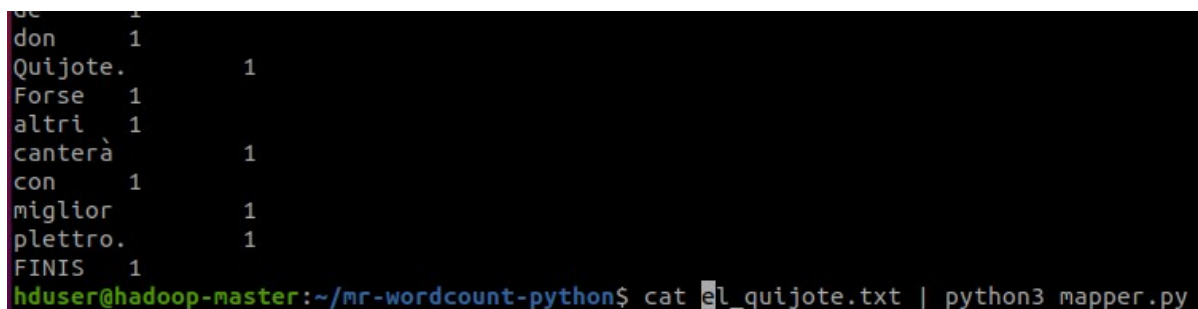
import sys
# Importamo-la librería sys para poder ler e escribir datos en STDIN e STDOUT

# Ler unha liña de STDIN (standard input):
# pasarémoslle un arquivo sobre o que conta-las palabras
for linha in sys.stdin:
    # Para cada liña eliminamo-los espazos ó principio e final
    linha = linha.strip()
    # Separa as palabras tomando como separador, por defecto, o espazo en branco
    # e gárdaas nunha lista
    palabras = linha.split()
    # Percorre cun bucle a lista de palabras imprimindo na saída estándar STDOUT
    # a palabra co valor de conteo 1
    for palabra in palabras:
        # Escribe os resultados no STDOUT (standard output);
        # o que se poña aquí será a entrada para o paso Reduce
        # Esta saída será o input para reducer.py
        print(palabra, "\t1")
    # creamos tuplas de (palabra, 1)
```

Pódese probar que o *mapper* funcione correctamente:

```
$ cat ../el_quijote.txt | python3 mapper.py
```

O resultado sería unha saída por pantalla dunha liña por cada palabra do texto seguida dun tabulador e o número 1.



```
don      1
Quijote.  1
Forse    1
altri    1
canterà  1
con      1
miglior  1
plettro. 1
FINIS    1
hduser@hadoop-master:~/mr-wordcount-python$ cat ../el_quijote.txt | python3 mapper.py
```

Hadoop Streaming: Reducer

De seguido, no *reducer* recibírase a saída do *mapper* e parseárase a cadea para separala palabra do contador.

Para leva-la conta das palabras, métese dentro dun dicionario no que se incrementan as ocorrencias atopadas.

- Recibe unha liña por cada par chave-valor emitido
- Ó gardarse nun dicionario, unha chave non pode estar en varias liñas

```
$ nano reducer.py
```

```
#!/usr/bin/python3
# A primeira liña indica que é un script python

import sys
# Importamo-la librería sys para poder ler e escribir datos en STDIN e STDOUT

# Inicializamo-lo dicionario
dicPalabras = {}

# ler unha liña de STDIN (standard input):
# pasarémoslle un arquivo cunha palabra co valor 1 en cada liña
for linha in sys.stdin:
    # Para cada liña eliminamo-los espazos ó principio e final
    linha = linha.strip()
    # Parseamo-la entrada de mapper.py;
    # separa as palabras tomando o tabulador como separador:
    # o formato proporcionado por mapper.py era unha palabra e un número (un),
    # así que só se fai 1 troceado, para obte-los dous valores
    palabra, conta = linha.split('\t', 1)
    # Converter conta (que realmente é string) a int
    try:
        conta = int(conta)
    # Cando conta non é un número, sixilosamente,
    # se ignora/descarta esta liña
    except ValueError:
        continue
    # Se incrementa o valor gardado no dicionario para palabra
    # grazas a que esta é a chave do mesmo
    try:
        dicPalabras[palabra] += conta
    except:
        # Se palabra aínda non estaba no dicionario se mete co valor conta (1)
        dicPalabras[palabra] = conta

# Percórrese todo o dicionario
for palabra in dicPalabras.keys():
    # Escríbese cada palabra e a súa conta na STDOUT
    print(palabra, "\t", dicPalabras[palabra])
```

Pódese comprobar que o código funciona correctamente, probando o proceso MapReduce completo, executando o seguinte comando:

```
$ cat ../el_quijote.txt | python3 mapper.py | python3 reducer.py
> ../conta_quijote.tsv
```

Se abrimo-lo ficheiro, podemos ve-lo resultado:

```
hduser@hadoop-master:~/nr-wordcount-python$ cat ../el_quijote.txt | python3 mapper.py | python3 reducer.py > ../conta_quijote.tsv
hduser@hadoop-master:~/nr-wordcount-python$ head ../conta_quijote.tsv
DON      6
QUIJOTE  6
DE       17
LA       13
MANCHA   2
Miguel   3
de       8947
Cervantes 1
Saavedra 1
PRIMERA  1
hduser@hadoop-master:~/nr-wordcount-python$ tail ../conta_quijote.tsv
declárase. 1
Tiénesa 1
vigilias 1
sacallos 1
Forse 1
altrí 1
canterá 1
miglior 1
pletto. 1
FINIS 1
```

Podemos darle **permisos de ejecución** a ambos scripts para que *Hadoop Streaming* poida executalos sen necesidade de invocalos precedidos do intérprete python:

```
$ chmod u+x mapper.py
$ chmod u+x reducer.py
```

```
$ cat ../el_quijote.txt | ./mapper.py | ./reducer.py > ../conta_quijote.tsv
```


Observade que os resultados son as palabras contabilizadas na orde en que primeiro aparecían no texto, non hai ningún proceso de ordenación, salvo que no propio código se inclúa: ollo que nisto hai unha **importante diferencia a cando se execute con hadoop streaming!**

Con **hadoop streaming** os datos que saen do proceso *map* son ordeados pola chave automaticamente antes de pasarse ó proceso *reduce*, sempre e cando haxa algún proceso *reduce*: **o resultado dos *mapper* é ordeado se o número de *reducers* é >0.**

Na execución local de proba, poderíamos aproveitar para ordear-los resultados alfabeticamente (interesante para atopar cadeas que non se deberían contar, ou palabras que se contabilizaron erroneamente por diferenciarse en maiúsculas/minúsculas, ou estar unidas a signos de puntuación,...)

```
$ cat ../el_quijote.txt | python3 mapper.py | python3 reducer.py | sort -t$'\t' -k1,1 > ../conta_quijote_ordeada.tsv
```

```
hduser@hadoop-master:~/mr-wordcount-python$ head ../conta_quijote_ordeada.tsv
,      1
-      4
.      2
i      1
-      1
1:     1
10:    1
11:    1
12:    1
13:    1
```

Ou poderíamos obte-los resultados ordeados polo número de repeticións de cada palabra e así ver cales son as máis/menos repetidas no texto da obra:

```
$ cat ../el_quijote.txt | python3 mapper.py | python3 reducer.py | sort -t$'\t' -nk2 > ../conta_quijote_ordeada_conta.tsv
```

```
hduser@hadoop-master:~/mr-wordcount-python$ tail ../conta_quijote_ordeada_conta.tsv
los      2122
se       2382
no       2786
el       3726
en       3883
a        4725
la       4941
y        8042
de       8947
que      10351
```

Nota: ollo coa memoria RAM!

Nun caso real, traballando con Big Data, ó almacenar tódolos datos que recibimos en memoria, poderían non coller na RAM de cada datanode. Por elo, se recomenda o uso da librería *itertools*, por exemplo, utilizando a función *groupby()*: <https://docs.python.org/es/3/library/itertools.html>

Hadoop Streaming: envío do traballo a hadoop

Una vez comprobado que os algoritmos de mapeo e redución funcionan, **imos procesalos dentro de Hadoop para aproveita-la computación distribuída** do noso clúster.

Para elo, usaremos **Hadoop Streaming**, que permite executar jobs MapReduce con calquera script (codificado en calquera linguaxe de programación) que poida ler da entrada estándar (*stdin*) e escribir na saída estándar (*stdout*). Deste xeito, Hadoop Streaming envía os datos en crú ó **mapper** vía *stdin* e tras procesarse, os pasa ó **reducer** vía *stdout*.

Hadoop-streaming ten 2 parámetros obrigatorios (**-input** e **-output**) e outros opcionais, aínda que para o noso caso os parámetros **-mapper** e **-reducer** tamén teremos que usalos:

Parámetro		Exemplo	Observacións
-input	Ruta HDFS e nome do directorio/arquivo de entrada (obrigatorio)	-input /el_quijote.txt	
-output	Ruta HDFS e nome do directorio de saída dos resultados (obrigatorio)	-output /contando_quijote	
-mapper	Comando que se executará como mapeador: script ou executable ou JavaClassName (obrigatorio)	-mapper 'python3 mapper.py'	Neste exemplo ademais do nome do programa .py indícase o intérprete de Python.
-reducer	Comando que se executará como redutor. (obrigatorio)	-reducer reducer.py	Neste exemplo só se indica o nome do programa .py, sen o intérprete de Python, porque o script ten permiso de execución.
-files	Ruta local e nomes dos ficheiros mapper e reducer (se se usa, debe porse antes que os parámetros obrigatorios)	-files ~/Descargas/mapper.py,/home/hduser/Descargas/reducer.py	Olo con non deixar espazo tra-la coma e con que non acepta rutas relativas despois da coma.

Referencia: <https://hadoop.apache.org/docs/stable/hadoop-streaming/HadoopStreaming.html>

Atención!: a orde dos argumentos é importante, xa que, por razóns de compatibilidade con versións anteriores, **-files** e **-archives** deben ir antes do argumento **-input**; deben estar ó principio como cos argumentos **-D**.

Nota: os arquivos de script a usar deben ter permisos de execución se non se invocan precedidos do intérprete de Python (é dicir, para executar “**mapper.py**” en vez de “**python3 mapper.py**” hai que asignarlle permiso de execución):

```
$ chmod +x *.py
```

A sintaxe para executa-los *jobs* é:

```
mapred streaming \  
-files mapper.py, reducer.py \  
-input meuCartafoIEntradaHDFS \  
-output meuCartafoISaídaHDFS \  
-mapper scriptMapper \  
-reducer scriptReducer
```

Nota: **Recomendable mapred en vez de hadoop jar**. En versións máis antigas de Hadoop, en vez de utiliza-lo comando **mapred**, utilízase o comando **hadoop jar**:

```
hadoop jar rutaDeHadoopStreaming.jar \  
-files mapper.py, reducer.py
```

```
-input meuCartafo1EntradaHDFS \  
-output meuCartafo1SaídaHDFS \  
-mapper scriptMapper \  
-reducer scriptReducer
```

Normalmente a ruta do jar é: `$HADOOP_HOME/share/hadoop/tools/lib`

Antes de nada, **para poder aproveita-la capacidade de computación distribuída, hai que subir ó sistema hdfs o ficheiro a tratar, *el_quijote.txt***

```
$ hdfs dfs -put ../el_quijote.txt
```

Os **scripts python podemos collelos do sistema local**, ou poderíamos subilos tamén ó sistema hdfs, pero como non aporta nada subi-lo código a executar, usámoslos localmente, e iso implica que indiquemos os parámetros **-file** para referirnos ós arquivos locais correspondentes.

```
$ mapred streaming -input el_quijote.txt -output saida_quijote -mapper mapper.py  
-file mapper.py -reducer reducer.py -file reducer.py
```

Como ó executarse, en pantalla se indica que o parámetro **-file** está desaconsellado, pódese utilizar no seu lugar un único parámetro **-files** para indica-la ruta e nome de cada ficheiro local *mapper* e *reducer*:

```
-files ruta/reducer.py,ruta/mapper.py
```

ollo!! -files ten que ir antes que -input

A mesma execución anterior co parámetro **-files**:

```
$ mapred streaming -files ./mapper.py,./reducer.py -input el_quijote.txt -output  
saida_quijote -mapper mapper.py -reducer reducer.py
```

Unha vez finalizado o *job*, pódense ve-las primeiras liñas dos resultados xerados en HDFS mediante:

```
$ hdfs dfs -head /user/hduser/saida_quijote/part-00000
```

```
hduser@hadoop-master:~/mr-wordcount-python$ hdfs dfs -head /user/hduser/saida_quijote/part-00000  
"Apenas" 1  
"Caballero" 4  
"Conde" 1  
"Donde" 1  
"Más" 1  
"Miau", 1  
"No" 1  
"Rastrea" 1  
"Ricamonte" 1
```

Podemos darlle **permisos de execución** ós **scripts** para que *Hadoop Streaming* poida executalos sen necesidade de invocalos precedidos do intérprete python:

```
$ chmod u+x mapper.py
$ chmod u+x reducer.py
```

```
$ mapred streaming \
-files ./mapper.py,./reducer.py \
-mapper mapper.py \
-reducer reducer.py \
-input ./el_quijote.txt \
-output ./contando_quijote_sen_x8
```

```
hduser@hadoop-master:~/mr-wordcount-python$ mapred streaming \
> -files ./mapper.py,./reducer.py \
> -mapper mapper.py \
> -reducer reducer.py \
> -input ./el_quijote.txt \
> -output ./contando_quijote_sen_x8
packageJobJar: [] [/usr/share/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.1.jar] /tmp/streamjob4014838181292134924.jar tmpDir=null
2023-11-07 22:35:11,844 INFO client.DefaultNoHARMFaloverProxyProvider: Connecting to ResourceManager at hadoop-master/10.0.2.12:8032
2023-11-07 22:35:12,024 INFO client.DefaultNoHARMFaloverProxyProvider: Connecting to ResourceManager at hadoop-master/10.0.2.12:8032
2023-11-07 22:35:12,334 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hduser/.staging/job_1699388564941_0008
2023-11-07 22:35:13,137 INFO mapred.FileInputFormat: Total input files to process : 1
2023-11-07 22:35:13,383 INFO mapreduce.JobSubmitter: number of splits:2
2023-11-07 22:35:13,664 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1699388564941_0008
```

Advertencia: como se pode apreciar ó executa-lo programa, *mapreduce* produce unha saída con moita información (*verbose output*), pero se todo vai ben, deberíamos ver algunha liña do tipo: **INFO mapreduce.Job: Job ... completed successfully**

Erro habitual: cando executamos repetidas veces un traballo que fallou nalgún punto, ou se tras facer algún cambio nun traballo volvemos a lanzalo, hai que ter en conta que os resultados non poden gardarse nun cartafol que xa exista previamente, xa que non se permite sobrescribir un directorio.

```
ERROR streaming.StreamJob: Error Launching job : Output directory ... already exists
Streaming Command Failed!
```

De seguido podemos proba-la execución usando o *hadoop-streaming* de **hadoop jar**:

```
$ hdfs dfs -put ../el_quijote.txt / # non fai falta porque xa o subimos antes
```

O programa **hadoop-streaming** que ven coa distribución de Hadoop, */usr/share/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.1.jar* permite crear e executar traballos Map/Reduce con calquera arquivo executable ou script como o *mapper* e/ou o *reducer*.

```
$ /usr/share/hadoop/bin/hadoop jar
/usr/share/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.1.jar -file
./mapper.py -mapper 'python3 mapper.py' -file ./reducer.py -reducer 'python3
reducer.py' -input ./el_quijote.txt -output ./contando_quijote
```

```
hduser@hadoop-master:~/mr-wordcount-python$ /usr/share/hadoop/bin/hadoop jar /usr/share/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.1.jar
-file ./mapper.py -mapper 'python3 mapper.py' -file ./reducer.py -reducer 'python3 reducer.py' -input ./el_quijote.txt -output ./contando_quijote
_sen_x3
2023-11-07 22:08:01,271 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [./mapper.py, ./reducer.py, /tmp/hadoop-unjar4149062273524316776/] [] /tmp/streamjob1495805255852789092.jar tmpDir=null
2023-11-07 22:08:02,893 INFO client.DefaultNoHARMFaloverProxyProvider: Connecting to ResourceManager at hadoop-master/10.0.2.12:8032
2023-11-07 22:08:03,254 INFO client.DefaultNoHARMFaloverProxyProvider: Connecting to ResourceManager at hadoop-master/10.0.2.12:8032
2023-11-07 22:08:03,701 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hduser/.staging/job_1699388564941_0003
2023-11-07 22:08:07,215 INFO mapred.FileInputFormat: Total input files to process : 1
2023-11-07 22:08:09,751 INFO mapreduce.JobSubmitter: number of splits:2
2023-11-07 22:08:11,710 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1699388564941_0003
```

A mesma execución anterior co parámetro *-files*:

```
$ /usr/share/hadoop/bin/hadoop jar \
/usr/share/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.1.jar -files \
./mapper.py,./reducer.py -mapper 'python3 mapper.py' -reducer 'python3 \
reducer.py' -input ./el_quijote.txt -output ./contando_quijote_sen_x4
```

Podemos darle **permisos de ejecución** ós **scripts** para que *Hadoop Streaming* pueda ejecutarlos sin necesidad de invocarlos precedidos do intérprete python:

```
$ chmod u+x mapper.py
$ chmod u+x reducer.py
```

```
$ /usr/share/hadoop/bin/hadoop jar \
/usr/share/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.1.jar \
-files ./mapper.py,./reducer.py \
-mapper mapper.py \
-reducer reducer.py \
-input ./el_quijote.txt \
-output ./contando_quijote_sen_x5
```

```
hduser@hadoop-master:~/mr-wordcount-python$ /usr/share/hadoop/bin/hadoop jar \
> /usr/share/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.1.jar \
> -files ./mapper.py,./reducer.py \
> -mapper mapper.py \
> -reducer reducer.py \
> -input ./el_quijote.txt \
> -output ./contando_quijote_sen_x7
packageJobJar: [/tmp/hadoop-unjar2465208559345722960/] [] /tmp/streamjob3723604674518913980.jar tmpDir=null
2023-11-07 22:29:45,836 INFO client.DefaultNoHARMFalloverProxyProvider: Connecting to ResourceManager at hadoop-master/10.0.2.12:8032
2023-11-07 22:29:46,032 INFO client.DefaultNoHARMFalloverProxyProvider: Connecting to ResourceManager at hadoop-master/10.0.2.12:8032
2023-11-07 22:29:46,363 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hduser/.staging/job_1699388564941_0007
2023-11-07 22:29:47,039 INFO mapred.FileInputFormat: Total input files to process : 1
2023-11-07 22:29:47,287 INFO mapreduce.JobSubmitter: number of splits:2
2023-11-07 22:29:47,700 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1699388564941_0007
```

Hadoop Streaming: saída resultante da execución dos traballos

Os resultados da execución do traballo gárdanse no cartafol indicado no parámetro **-output**.

Ese cartafol de destino non pode existir previamente como medida de seguridade para evitar que os resultados dun traballo sobrescriban os doutra execución previa (pensade que realizar un traballo pode durar moitas horas, como para que se corra o risco de que se perdan os seus resultados).

A saída dun traballo que remata correctamente é unha serie de ficheiros gardados no cartafol:

- Ficheiro **_SUCCESS**
 - Está baleiro pero indica que a tarefa completouse correctamente
- Ficheiros **part-X-nnnnn**
 - Son os resultados do traballo
 - Se $X = m$ → resultados do *map* (cando non hai *reduce*)
 - Se $X = r$ → resultados do *reduce* (o máis habitual)
 - No caso de hadoop-streaming non hai X , así que o nome dos ficheiros resultado serían da forma: *part-00000*
 - **nnnnn** = número da tarefa
- Cada tarefa xera un ficheiro diferente

Así, por exemplo, se un job utilizase 30 *reducers* habería 30 ficheiros chamados dende *part-r-00000* a *part-r-00029*, un por cada tarefa *reducer*.

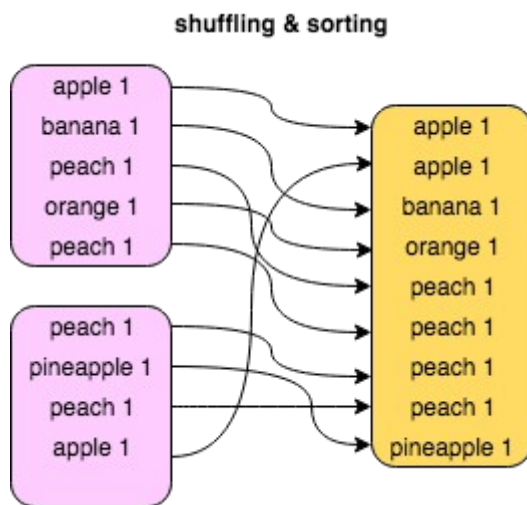
Traballando en Big Data o máis recomendable é botar unha ollada ós resultados usando comandos como **head** ou **tail**, en vez de **cat**, debido ó seu eventual gran tamaño da saída.

```
$ hdfs dfs -head /user/hduser/contando_quijote_sen_x5/*
```

```
hduser@hadoop-master:~$ hdfs dfs -ls /user/hduser/contando_quijote_sen_x5/
Found 2 items
-rw-r--r--  2 hduser supergroup      0 2023-11-07 22:24 /user/hduser/contando_quijote_sen_x5/_SUCCESS
-rw-r--r--  2 hduser supergroup 326047 2023-11-07 22:24 /user/hduser/contando_quijote_sen_x5/part-00000
hduser@hadoop-master:~$ hdfs dfs -cat /user/hduser/contando_quijote_sen_x5/_SUCCESS
hduser@hadoop-master:~$ hdfs dfs -head /user/hduser/contando_quijote_sen_x5/part-00000
"Apenas"      1
"Caballero"   4
"Conde"       1
"Donde"       1
"Más"         1
"Miau"        1
```

Observando os datos obtidos como resultado da saída do proceso *map-reduce*, chama a atención que se atopan ordeados pola chave (primeira columna), a diferenza de cando executáramos eses mesmos procesos *map* e *reduce* directamente usando localmente *python*.

Con **hadoop streaming** os datos que saen do proceso *map* son ordeados pola chave automaticamente antes de pasarse ó proceso *reduce*, sempre e cando haxa algún proceso *reduce*: o resultado dos *mapper* é ordeado se o número de *reducers* é >0 .



Hadoop Streaming: outro exemplo sinxelo de wordcount distribuído usando MapReduce, con binarios.

Do mesmo xeito que executamos o código fonte codificado en Python, poderíamos incluso lanzar binarios executables.

Por exemplo, os comandos do sistema linux, **cat** que permite amosa-lo contido de arquivos, e **wc** que permite contar -entre outras cousas- o número de palabras dun ficheiro, poderíanse executar de forma distribuída nun clúster usando *MapReduce*.

```
$ mapred streaming \  
-mapper /bin/cat \  
-reducer /usr/bin/wc \  
-input ./el_quijote.txt \  
-output ./contando_quijote_bin
```

O resultado son 3 datos do arquivo analizado:

- O número de liñas
- O número de palabras
- O número de bytes

Podemos compara-lo resultado do traballo map-reduce coa execución local dos comandos no propio sistema para corroborar que se obteñen os mesmos cálculos.

```
$ hdfs dfs -cat ./contando_quijote_bin/*

$ wc ../el_quijote.txt
      10 ERROR=0
      WRONG_LENGTH=0
      WRONG_MAP=0
      WRONG_REDUCE=0
File Input Format Counters
      Bytes Read=1064355
File Output Format Counters
      Bytes Written=25
2023-11-10 20:11:10,198 INFO streaming.StreamJob: Output directory: ./contando_quijote_bin
hduser@hadoop-master:~/mr-wordcount-python$ hdfs dfs -cat ./contando_quijote_bin/*
2186 187018 1062445
hduser@hadoop-master:~/mr-wordcount-python$ wc ../el_quijote.txt
2186 187018 1060259 ../el_quijote.txt
hduser@hadoop-master:~/mr-wordcount-python$ hdfs dfs -ls ./contando_quijote_bin/*
-rw-r--r--  2 hduser supergroup      0 2023-11-10 20:11 contando_quijote_bin/_SUCCESS
-rw-r--r--  2 hduser supergroup    25 2023-11-10 20:11 contando_quijote_bin/part-00000
hduser@hadoop-master:~/mr-wordcount-python$
```

A diferencia no número de bytes é seguramente debida a que a estrutura física de almacenamento ou os metadatos do arquivo *el quijote.txt* é distinta ó gardarse no sistema hdfs ou no sistema ext4???

```
hduser@hadoop-master:~/mr-wordcount-python$ hdfs dfs -ls ./el_quijote.txt
-rw-r--r-- 2 hduser supergroup 1060259 2023-11-05 13:25 el_quijote.txt
hduser@hadoop-master:~/mr-wordcount-python$ ls -ls ./el_quijote.txt
1036 -rw-rw-r-- 1 hduser hduser 1060259 Nov 5 13:13 ./el_quijote.txt
hduser@hadoop-master:~/mr-wordcount-python$
```


Hadoop Streaming: outro exemplo sinxelo de wordcount distribuído usando MapReduce, con scripts bash.

- Exemplo obtido de:
https://github.com/groda/big_data/blob/master/mapreduce_with_bash.ipynb

O script *mapper* será o ficheiro **map.sh**, que troceará cada liña dun ficheiro de texto de entrada, en palabras e para cada palabra devolverá unha liña que conteña a palabra e o valor 1, separado por un tabulador:

map.sh

```
#!/bin/bash
while read line
do
  for word in $line
  do
    #sáltase as liñas baleiras
    if [[ "$line" =~ [^[:space:]] ]]
    then
      if [ -n "$word" ]
      then
        echo -e ${word} "\t1"
      fi
    fi
  done
done
```

Para proba-lo script podemos pasarlle uns datos de entrada e observa-lo resultado.

Os datos de entrada obterémolos dun xerador de froitas ó chou no que podemos programar cantas froitas incluír nun ficheiro de texto de saída:

froitaria.py

```
#!/usr/bin/env python3

# froitaria.py
# froitaria é unha modificación de charlatan.py crea un ficheiro de texto con
# froitas pseudoaleatorias que se toman
# dunha lista que pode definir o/a programador/a
#
# Pode definirse o número de palabras para crear un ficheiro máis grande

import random

ficheiro = open("froitas.txt","a")
pseudoPalabrasAleatorias = ["Mazá ", "Laranxa ", "Pera ", "Pexego ", "Figo ",
"Peladillo ", "Breva ", "Plátano ", "Níspero ", "Fatón ", "Ameixa ", "Cirola "]
cantidadePalabrasAleatorias = len(pseudoPalabrasAleatorias) - 1

#Aumenta NUMERO_PALABRAS_RESULTANTES para crear un ficheiro maior
#150000000 -> 112MB
#300000000 -> 231MB
```

```
#50 -> 1KB
NUMERO_PALABRAS_RESULTANTES = 50

indice = 0
for x in range(NUMERO_PALABRAS_RESULTANTES):
    indice = random.randint(0, cantidadPalabrasAleatorias)
    ficheiro.write(pseudoPalabrasAleatorias[indice])
    if x % 20 == 0:
        ficheiro.write('\n')
```

Para xera-lo ficheiro de texto **froitas.txt** coas froitas de exemplo basta con executar:

```
$ /usr/bin/python3 froitaria.py
```

Pódese ve-lo contido do ficheiro resultante:

```
$ cat froitas.txt
```

E agora xa se pode testea-lo **mapper**:

```
$ cat froitas.txt | bash map.sh
```

Se lle damos permiso de execución ó ficheiro **map.sh** é posible invoca-lo **map.sh** directamente:

```
$ chmod u+x map.sh
$ cat froitas.txt | ./map.sh

hduser@hadoop-master:~/mr-wordcount-bash$ ls -l map.sh
-rw----- 1 hduser hduser 208 Nov 11 16:00 map.sh
hduser@hadoop-master:~/mr-wordcount-bash$ cat froitas.txt | ./map.sh
bash: ./map.sh: Permiso denegado
hduser@hadoop-master:~/mr-wordcount-bash$ chmod u+x map.sh
hduser@hadoop-master:~/mr-wordcount-bash$ ls -l map.sh
-rwx----- 1 hduser hduser 208 Nov 11 16:00 map.sh
hduser@hadoop-master:~/mr-wordcount-bash$ cat froitas.txt | ./map.sh
Cirola 1
Ameixa 1
Figo 1
Figo 1
```

Podemos modifica-lo script **froitaria.py** para xerar un arquivo máis grande, de 200MB aprox., e subilo ó clúster HDFS.

```
$ hdfs dfs -mkdir -p mr-wordcount-bash/input
$ hdfs dfs -put froitas.txt mr-wordcount-bash/input
$ hdfs dfs -ls -l -R mr-wordcount-bash/input

hduser@hadoop-master:~/mr-wordcount-bash$ hdfs dfs -rm -R mr-wordcount-bash
Deleted mr-wordcount-bash
hduser@hadoop-master:~/mr-wordcount-bash$ hdfs dfs -mkdir -p mr-wordcount-bash/input
hduser@hadoop-master:~/mr-wordcount-bash$ hdfs dfs -put froitas.txt mr-wordcount-bash/input
hduser@hadoop-master:~/mr-wordcount-bash$ hdfs dfs -ls -h -R mr-wordcount-bash/input
-rw-r--r-- 2 hduser supergroup 206.5 M 2023-11-11 17:08 mr-wordcount-bash/input/froitas.txt
hduser@hadoop-master:~/mr-wordcount-bash$
```

Para proba-lo programa **map.sh** no clúster, podemos usar un **reducer** ficticio como **/bin/cat**, que simplemente devolverá os mesmos datos que reciba.

```
$ mapred streaming \
  -files map.sh \
  -input mr-wordcount-bash/input \
```

```
-output mr-wordcount-bash/output \
-mapper map.sh \
-reducer /bin/cat
```

Se o traballo se completou correctamente o resultado estará dispoñible no cartafol de saída creado.

```
$ hdfs dfs -ls -h -R mr-wordcount-bash/output
$ hdfs dfs -head mr-wordcount-bash/output/part-00000
$ hdfs dfs -tail mr-wordcount-bash/output/part-00000
```

```
hduser@hadoop-master:~/mr-wordcount-bash$ hdfs dfs -head mr-wordcount-bash/output/part-00000
Ameixa 1
Ameixa 1
Ameixa 1
Ameixa 1
Ameixa 1
Ameixa 1
Ameixa 1
Ameixa 1
Ameixa 1
Ameixa 1
```

```
hduser@hadoop-master:~/mr-wordcount-bash$ hdfs dfs -tail mr-wordcount-bash/output/part-00000
1
Plátano 1
Plátano 1
Plátano 1
Plátano 1
Plátano 1
Plátano 1
Plátano 1
```

Obsérvase que, aínda que non pedimos ordear os datos resultantes, con **hadoop streaming** os datos que saen do proceso *map* son ordeados pola chave automaticamente antes de pasarse ó proceso *reduce*, sempre e cando haxa algún proceso *reduce*: o resultado dos *mapper* é ordeado se o número de *reducers* é >0.

O script *reducer* será o ficheiro ***reduce.sh***, que toma os 2 valores que separa o tabulador en cada liña: o primeiro será a chave (o nome da froita) e o segundo o número de repeticións (o valor 1 que ven do proceso mapper) desa chave. Como o proceso *reducer* devolveu ordeados os resultados, de seguido se comproba se a chave lida en cada iteración é igual á anteriormente lida: se é a mesma, súmase o valor lido ó total acumulado, e se non o é, devólvese a chave e ó total acumulado, antes de volver a reinicia-lo total acumulado para a seguinte chave:

reduce.sh

```
#!/bin/bash
currkey=""
currcount=0
while IFS=$'\t' read -r key val
do
    if [[ $key == $currkey ]]
    then
        currcount=$(( currcount + val ))
    else
        if [ -n "$currkey" ]
        then
            echo -e ${currkey} "\t" ${currcount}
        fi
        currkey=$key
        currcount=1
    fi
done
# last one
echo -e ${currkey} "\t" ${currcount}
```

Podemos testea-lo *reducer* antes de envia-lo traballo ó clúster:

```
$ chmod u+x reduce.sh
$ cat froitas.txt | ./map.sh |sort| ./reduce.sh
hduuser@hadoop-master:~/mr-wordcount-bash$ cat froitas.txt | ./map.sh |sort| bash ./reduce.sh
Ameixa      75
Breva       83
Cirola      93
Fatón       80
Figo        82
Laranxa          75
Mazá        83
Nispero          82
Peladillo     75
Pera         77
Pexego       87
Plátano      79
```

E unha vez todo preparado, borramo-lo directorio de saída se se creou anteriormente -lembremos que do contrario abortarase a execución porque non permite sobreescribi-lo directorio de resultados- e lanzamo-lo traballo.

```
$ hdfs dfs -ls -R -h mr-wordcount-bash/output
$ hdfs dfs -rm -R mr-wordcount-bash/output
$ mapred streaming \
  -files map.sh,reduce.sh \
  -input mr-wordcount-bash/input \
```

```
-output mr-wordcount-bash/output \
-mapper map.sh \
-reducer reduce.sh
```

Se o traballo se completou correctamente o resultado estará dispoñible no cartafol de saída creado.

```
$ hdfs dfs -ls -h -R mr-wordcount-bash/output
$ hdfs dfs -head mr-wordcount-bash/output/part-00000
$ hdfs dfs -tail mr-wordcount-bash/output/part-00000
```

```
File Output Format Counters
  Bytes Written=206
2023-11-11 18:56:41,346 INFO streaming.StreamJob: Output directory: mr-wordcount-bash/output
hduser@hadoop-master:~/mr-wordcount-bash$ hdfs dfs -ls -h -R mr-wordcount-bash/output
-rw-r--r--  2 hduser supergroup          0 2023-11-11 18:56 mr-wordcount-bash/output/_SUCCESS
-rw-r--r--  2 hduser supergroup    206 2023-11-11 18:56 mr-wordcount-bash/output/part-00000
hduser@hadoop-master:~/mr-wordcount-bash$ hdfs dfs -head mr-wordcount-bash/output/part-00000
Ameixa    2497248
Breva     2499759
Cirola    2501110
Fatón     2500401
Figo      2501498
Laranxa   2500812
Mazá      2499635
Nispero   2497593
Peladillo 2498881
Pera      2500684
Pexego    2501307
Plátano   2501122
hduser@hadoop-master:~/mr-wordcount-bash$
```

Resolvendo outras consultas: froitas máis e menos repetidas

O resultado anteriormente obtido era unha lista de froitas que aparecen no ficheiro seguidas das súas frecuencias.

A saída do redutor está ordeada por chave (a froita) porque esa é a orde na que se lle chegan os datos ó redutor a partir do mapeador. Se queremos ordear os datos por frecuencia, podemos utilizar o comando **sort** de Unix (coas opcións **k2**, **n** e **r** respectivamente para ordear polo campo 2, de forma numérica, e en orde inversa).

```
$ hdfs dfs -cat mr-wordcount-bash/output/part-00000 | sort -k2nr
hduser@hadoop-master:~/mr-wordcount-bash$ hdfs dfs -cat mr-wordcount-bash/output/part-00000 | sort -k2nr
Figo      2501498
Pexego    2501307
Plátano   2501122
Cirola    2501110
Laranxa   2500812
Pera      2500684
Fatón     2500401
Breva     2499759
Mazá      2499635
Peladillo 2498881
Nispero   2497593
Ameixa    2497248
hduser@hadoop-master:~/mr-wordcount-bash$
```

Pero, por suposto, tamén é posible ordear-la saída resultante mediante outro traballo MapReduce.

Ordea-los resultados mediante un traballo MapReduce

Preparamos un novo script que simplemente intercambia a orde dos 2 elementos de cada liña: en primeiro lugar estará o número de repeticións, e logo do tabulador a froita correspondente.

swap_keyval.sh

```
#!/bin/bash
# This script will read one line at a time and swap key/value
# For instance, the line "word 100" will become "100 word"

while read key val
do
    printf "%s\t%s\n" "$val" "$key"
done
```

Asígnaselle o permiso de execución ó script:

```
$ chmod u+x swap_keyval.sh
```

Neste caso, para executa-lo script mapeador que intercambia a froita e as repeticións obtidos no anterior traballo MapReduce, tomaremos como entrada a saída anterior, **output**, e gardaremo-la saída do traballo actual nun novo cartafol **output_sorted**.

O bo de executar un traballo usando a saída dun traballo anterior é que non se necesita cargar ficheiros a HDFS porque os datos xa están en HDFS.

Nota: *Apache Spark* soluciona unha das principais **deficiencias de MapReduce**, xa que en vez de **escribir datos no disco en cada paso dun pipeline de transformación de datos (o cal leva tempo e iso pode ser moi custoso para canalizacións de datos máis longas)** mantén en memoria eses datos.

```
$ hdfs dfs -rm -R mr-wordcount-bash/outputsorted
$ mapred streaming \
  -files swap_keyval.sh \
  -input mr-wordcount-bash/output \
  -output mr-wordcount-bash/outputsorted \
  -mapper swap_keyval.sh
```

Se o traballo se completou correctamente o resultado estará dispoñible no cartafol de saída creado.

```
$ hdfs dfs -ls -l -h -R mr-wordcount-bash/outputsorted
$ hdfs dfs -head mr-wordcount-bash/outputsorted/part-00000
$ hdfs dfs -tail mr-wordcount-bash/outputsorted/part-00000
```

```
Bytes Written=182
2023-11-11 19:10:30,981 INFO streaming.StreamJob: Output directory: mr-wordcount-bash/output2
hduser@hadoop-master:~/mr-wordcount-bash$ hdfs dfs -ls -l -h -R mr-wordcount-bash/output2
-rw-r--r--  2 hduser supergroup      0 2023-11-11 19:10 mr-wordcount-bash/output2/_SUCCESS
-rw-r--r--  2 hduser supergroup    182 2023-11-11 19:10 mr-wordcount-bash/output2/part-00000
hduser@hadoop-master:~/mr-wordcount-bash$ hdfs dfs -head mr-wordcount-bash/output2/part-00000
2497248 Anelxa
2497593 Nispero
2498081 Peladillo
2499635 Maza
2499759 Breva
2500401 Fatón
2500684 Pera
2500812 Laranxa
2501110 Cirola
2501122 Plátano
2501307 Pexego
2501498 Figo
hduser@hadoop-master:~/mr-wordcount-bash$
```

Configura-la orde de saída do mapper coa propiedade **KeyFieldBasedComparator**

Pódese configurar como os mapeadores van face-la ordeación da súa saída mediante a propiedade **KeyFieldBasedComparator**.

```
-D mapreduce.job.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator
```

Esta clase ten algunhas opcións similares ó **sort** de Unix (**-n** para ordenar numericamente, **-r** para a ordenación inversa, **-k pos1[,pos2]** para especificar campos para ordenar).

Por defecto o separador é o tabulador, pero pódese configurar coa propiedade **mapreduce.map.output.key.field.separator**

Para ordear-las froitas das máis repetidas ás menos fariamo-la seguinte execución:

```
$ comparator_class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator
$ mapred streaming \
-D mapreduce.job.output.key.comparator.class=$comparator_class \
-D mapreduce.partition.keycomparator.options=-nr \
-files swap_keyval.sh \
-input mr-wordcount-bash/output \
-output mr-wordcount-bash/output3 \
-mapper swap_keyval.sh
```

```
File Output Format Counters
      Bytes Written=182
2023-11-11 19:52:40,188 INFO streaming.StreamJob: Output directory: mr-wordcount-bash/output3
hduser@hadoop-master:~/mr-wordcount-bash$ hdfs dfs -head mr-wordcount-bash/output3/part-00000
2501498 Figo
2501307 Pexego
2501122 Plátano
2501110 Cirola
2500812 Laranxa
2500684 Pera
2500401 Fatón
2499759 Breva
2499635 Mazá
2498881 Peladillo
2497593 Nispero
2497248 Ameixa
hduser@hadoop-master:~/mr-wordcount-bash$
```

Para usa-la segunda columna como campo de ordeación:

```
$ mapred streaming \
-D mapreduce.job.output.key.comparator.class=$comparator_class \
-D stream.num.map.output.key.fields=2 \
-D mapreduce.partition.keycomparator.options='-k2r' \
-files swap_keyval.sh \
-input mr-wordcount-bash/output \
-output mr-wordcount-bash/output4 \
-mapper swap_keyval.sh
```



```

2023-11-11 21:02:09,677 INFO streaming.StreamJob: Output directory: mr-wordcount-bash/output4
hduser@hadoop-master:~/mr-wordcount-bash$ hdfs dfs -head mr-wordcount-bash/output4/part-00000
2501122 Plátano
2501307 Pexego
2500684 Pera
2498881 Peladillo
2497593 Nispero
2499635 Mazá
2500812 Laranxa
2501498 Figo
2500401 Fatón
2501110 Cirola
2499759 Breva
2497248 Ameixa
hduser@hadoop-master:~/mr-wordcount-bash$ hdfs dfs -rm -R mr-wordcount-bash/output4

```

A opción **-D** permite especificar variables de configuración para cambia-los valores por defecto establecidos no ficheiro **mapred-default.xml**.

Por exemplo podería resultar útil cando houberse problemas de falta de memoria na fase de ordeación, cambia-lo tamaño en MB da memoria reservada para ordear, e iso faríase indicando un novo tamaño de memoria á propiedade **mapreduce.task.io.sort.mb**:

```
-D mapreduce.task.io.sort.mb=512
```

```

$ mapred streaming \
-D mapreduce.job.output.key.comparator.class=$comparator_class \
-D mapreduce.partition.keycomparator.options=-nr \
-D mapreduce.task.io.sort.mb=50 \
-files swap_keyval.sh \
-input mr-wordcount-bash/output \
-output mr-wordcount-bash/output5 \
-mapper swap_keyval.sh

```


-----> COUSAS QUE TEÑO QUE REVISAR

Probar se hai diferenza entre a execución cando os SCRIPTS están en local ou cando están no hdfs: prodúcese o mesmo proceso de computación distribuída igualmente en ambos casos?

-file ou **-files** cando se usan????

Entendo que co parámetro **-files** se indica que os arquivos dos scripts son locais e que se suban ós nodos do clúster para executarse neles...?

Poderíamos subí-los scripts do sistema local ó hdfs e indicar que xa están no hdfs? Tería algunha diferenza de funcionamento?

Para subir arquivos conservando os seus permisos (incluído o de execución) úsase o parámetro **-p**

```
$ hdfs dfs -put -p mapper.py  
$ hdfs dfs -put -p reducer.py
```

COUSAS QUE TEÑO QUE REVISAR <-----

Comandos para interactuar cos traballos

O comando `hadoop job` permite interactuar cos traballos, pero actualmente se desaconsella o seu uso a prol de `mapred job`:

- Para ve-la lista de traballos en execución:

```
$ $HADOOP_HOME/bin/mapred job -list all
```

- Para ve-lo estado dun traballo:

```
$ $HADOOP_HOME/bin/mapred job -status <JOB-ID>
```

Por exemplo:

```
$ $HADOOP_HOME/bin/mapred job -status job_201310191043_0004
```

- Para ve-lo rexistro de saída do directorio do traballo:

```
$ $HADOOP_HOME/bin/mapred job -history <DIR-NAME>
```

Por exemplo:

```
$ $HADOOP_HOME/bin/mapred job -history /user/hduser/saida
```

- Para matar un traballo:

```
$ $HADOOP_HOME/bin/mapred job -kill <JOB-ID>
```

Por exemplo:

```
$ $HADOOP_HOME/bin/mapred job -kill job_201310191043_0004
```

WARNING: “hadoop job” is deprecated

USAR: "mapred job" instead

Exercicios MapReduce con Hadoop Streaming

Pautas e recomendacións para realiza-los exercicios

1) Antes de mandar un traballo a execución distribuída sobre Hadoop podes **verifica-lo teu código na túa máquina local** (ou no bash da máquina hadoop do CESGA), para o que vos recomendo as seguintes pautas:

Crear un extracto do ficheiro de datos para facer probas:

```
$ head -n100 purchases.txt > mini_purchases.txt
```

```
ricardo@ubustu:~/Descargas$ head mini_purchases.txt
2012-01-01    09:00    San Jose      Men's Clothing    214.05    Amex
2012-01-01    09:00    Fort Worth    Women's Clothing    153.57    Visa
2012-01-01    09:00    San Diego     Music      66.08     Cash
2012-01-01    09:00    Pittsburgh    Pet Supplies    493.51    Discover
2012-01-01    09:00    Omaha        Children's Clothing    235.63    MasterCard
2012-01-01    09:00    Stockton      Men's Clothing    247.18    MasterCard
2012-01-01    09:00    Austin        Cameras    379.6     Visa
2012-01-01    09:00    New York      Consumer Electronics    296.8     Cash
2012-01-01    09:00    Corpus Christi    Toys      25.38     Discover
2012-01-01    09:00    Fort Worth    Toys      213.88    Visa
```

Comproba o contido do novo ficheiro creado:

```
$ cat mini_purchases.txt
```

Manda o ficheiro ó *mapper* cunha redirección (***pipe***):

```
$ cat mini_purchases.txt | python mapper.py
```

Ordena o resultado con ***sort***:

```
$ cat mini_purchases.txt | python mapper.py | sort
```

Manda o resultado, ordeado, ó *reducer*:

```
$ cat mini_purchases.txt | python mapper.py | sort | python reducer.py
```

(opcional) Para evitar chamadas a ***python*** podes face-los ficheiros executables:

```
$ chmod +x mapper.py reducer.py
```

Deste xeito a execución anterior sería máis simple:

```
$ cat mini_purchases.txt | ./mapper.py | sort | ./reducer.py
```

Olo! Estes comandos funcionan se temos tódolos ficheiros no directorio actual; en caso contrario hai que actualiza-las rutas.

2) Depois de comprobar que o teu código funciona en local podes **executalo no Clúster Hadoop**.

A sintaxe sería a seguinte:

```
$ mapred streaming \  
-files mapper.py, reducer.py \  
-input meuCartafoIEntradaHDFS \  
-output meuCartafoISaídaHDFS \  
-mapper scriptMapper \  
-reducer scriptReducer
```

Documentación de Hadoop Streaming:

<https://hadoop.apache.org/docs/stable/hadoop-streaming/HadoopStreaming.html>

Partindo dun ficheiro comprimido **purchases.txt.gz** ímolo descomprimir, colocar nun cartafol vendas e subir ambos, cartafol e ficheiro, ó sistema hdfs:

```
$ ls -lh purchases.txt.gz
$ mkdir vendas
$ ls -lh vendas
$ gzip -l purchases.txt.gz
      compressed      uncompressed  ratio uncompressed_name
      38454568         211312924  81.8% purchases.txt
$ gunzip -k -c purchases.txt.gz > ./vendas/purchases.txt
```

A opción **-k** evita que **gunzip** elimine o arquivo comprimido unha vez descomprimido, e a opción **-c** co redireccionado permite escribi-la saída no terminal e saca-lo descomprimido cara a outra ubicación.

```
hduser@hadoop-master:~/Descargas$ ls -lh vendas/
total 0
hduser@hadoop-master:~/Descargas$ ls -lh purchases.txt.gz
-rw-rw-r-- 1 hduser hduser 37M Nov 10 17:20 purchases.txt.gz
hduser@hadoop-master:~/Descargas$ gzip -l purchases.txt.gz
      compressed      uncompressed  ratio uncompressed_name
      38454568         211312924  81.8% purchases.txt
hduser@hadoop-master:~/Descargas$ gunzip -k -c purchases.txt.gz > ./vendas/purchases.txt
hduser@hadoop-master:~/Descargas$ ls -lh vendas/
total 202M
-rw-rw-r-- 1 hduser hduser 202M Nov 10 17:41 purchases.txt
hduser@hadoop-master:~/Descargas$
```

Por exemplo, para subir un cartafol chamado *vendas*, co ficheiro *purchases.txt* a procesar, ó directorio por defecto do noso sistema hdfs:

```
$ hdfs dfs -put vendas
```

```
hduser@hadoop-master:~/Descargas$ hdfs dfs -put vendas
hduser@hadoop-master:~/Descargas$ hdfs dfs -ls vendas
Found 1 items
-rw-r--r--  2 hduser supergroup  211312924  2023-11-10 17:54 vendas/purchases.txt
hduser@hadoop-master:~/Descargas$
```

De seguido, para lanza-la execución do noso programa *MapReduce* composto por un ficheiro *mapper.py* e outro *reducer.py* situados no noso directorio actual:

```
$ mapred streaming -files mapper.py,reducer.py -input vendas/purchases.txt -
output resultado_consulta -mapper "python mapper.py" -reducer "python
reducer.py"
```

Exercicio 1. Cálculo das vendas totais de cada tenda

Utilizarase como ficheiro de datos *purchases.txt*.

E os scripts de partida son os ficheiros *mapper.py* e *reducer.py*.

1ª parte

Como se pode ver no código, non se trata dun algoritmo complexo e ademais pode executarse desde a consola lanzando os scripts de xeito manual.

A vantaxe que ofrece Hadoop é que poden utilizarse os mesmos algoritmos sobre ficheiros moito máis grandes, repartidos nun sistema de ficheiros distribuído e utilizar múltiples computadores para procesa-los datos.

mapper.py

```
#!/usr/bin/python
# Format of each line is:
# date\ttime\tstore name\titem description\tcost\tmethod of payment
#
# We want elements 2 (store name) and 4 (cost)
# We need to write them out to standard output, separated by a tab
import sys
for line in sys.stdin:
    data = line.strip().split("\t")
    date, time, store, item, cost, payment = data
    print(store+"\t"+cost)
```

reducer.py

```
#!/usr/bin/python
import sys
salesTotal = 0
oldKey = None
# Loop around the data
# It will be in the format key\tval
# Where key is the store name, val is the sale amount
#
# All the sales for a particular store will be presented,
# then the key will change and we'll be dealing with the next store
for line in sys.stdin:
    data_mapped = line.strip().split("\t")
    if len(data_mapped) != 2:
        # Something has gone wrong. Skip this line.
        continue

    thisKey, thisSale = data_mapped
    # Escribe un par key:value ante un cambio na key
    # Reinicia o total
    if oldKey and oldKey != thisKey:
        print(oldKey+"\t"+str(salesTotal))
        oldKey = thisKey;
        salesTotal = 0
    oldKey = thisKey
    salesTotal += float(thisSale)
# Escribe o ultimo par, unha vez rematado o bucle
if oldKey != None:
    print(oldKey+"\t"+str(salesTotal))
```

A partir do código inicial do *mapper* e *reducer* vistos, realiza os seguintes exercicios modificando o código que sexa necesario.

Crea unha nova carpeta para cada exercicio: *parte2*, *parte3*, ... dentro dunha carpeta xeral "*mapreduce-vendas*", para non sobreescribi-lo código de cada exercicio.

2ª parte

A función do *mapper* executarase sobre todas e cada unha das liñas do ficheiro de datos. O código do *mapper* anterior non funcionará correctamente se se atopa con algunha liña que non encaixa co número exacto de valores separados por tabulador.

Mellora o código de xeito que se atopa algunha liña cun formato non axeitado a descarte e siga traballando coa liña seguinte.

Solúciónase engadindo unha liña no *mapper* que comprobe que en cada liña que procesa hai exactamente 6 elementos. En caso contrario non trata esa liña e pasa á seguinte.

```
for line in sys.stdin:
    data = line.strip().split("\t")
    if len(data) == 6:
        date, time, store, item, cost, payment = data
        print(f'{store}\t{cost}')
```

3ª parte

A categoría do produto vendido identifícase na 4ª columna do ficheiro csv.

Redefine o *mapper* e o *reducer* de xeito que devolvan un ficheiro co total de vendas por categoría.

Fai as probas cun extracto do ficheiro de vendas.

Executa o traballo desde Hadoop sobre o ficheiro *purchases.txt* almacenado en HDFS.

Solucionamos cambiando o *mapper*, de maneira que o par que envía ao *reducer* é o par: *item:cost*, en lugar de *store:cost*. Desta maneira a suma será das vendas por *item*, é dicir, por categoría.

```
for line in sys.stdin:
    data = line.strip().split("\t")
    if len(data) == 6:
        date, time, store, item, cost, payment = data
        print(f'{item}\t{cost}')
```

4ª parte

O tipo de pago en cada venda identifícase na 6ª columna do ficheiro csv.

Redefine o *mapper* e o *reducer* de xeito que se obteña a venda máis alta para cada tipo de pago das rexistradas en todo o ficheiro.

Fai as probas cun extracto do ficheiro de vendas.

Executa o traballo desde Hadoop sobre o ficheiro *purchases.txt* almacenado en HDFS.

Consulta o resultado: a que pode ser debido?

O *mapper* devolve o par: *payment:cost*

```
for line in sys.stdin:
    data = line.strip().split("\t")
    if len(data) == 6:
        date, time, store, item, cost, payment = data
        print(f'{payment}\t{cost}')
```

O *reducer* non utiliza a operación suma para ir acumulando valores, senón que compara, pois o que busca son os máximos por tipo de *payment*.

```
oldKey = thisKey
if thisSale >= salesMax:
    salesMax = float(thisSale)
```

5ª parte

Nesta ocasión só nos importa o importe de cada venda.

Modifica o exercicio anterior para que o resultado da execución sexa o máximo absoluto de todas as vendas rexistradas.

Fai as probas cun extracto do ficheiro de vendas.

Executa o traballo desde Hadoop sobre o ficheiro *purchases.txt* almacenado en HDFS.

Unha posible solución sería modificar unicamente o *mapper* e crear unha única categoría "*all*" en lugar de utiliza-los diferentes valores para *payment*. Desta maneira o *reducer* pensará que todas as vendas son co mesmo tipo de pago, ou dito doutro xeito, non se preocupará de que tipo de pago se utilizou.

```
for line in sys.stdin:
    data = line.strip().split("\t")
    if len(data) == 6:
        date, time, store, item, cost, payment = data
        print(f'all\t{cost}')
```


6ª parte

De novo unicamente nos interesan os importes de cada venda.

Redefine o *mapper* e o *reducer* de xeito que se obteña o total de vendas.

Fai as probas cun extracto do ficheiro de vendas.

Executa o traballo desde Hadoop sobre o ficheiro *purchases.txt* almacenado en HDFS.

Igual que no caso anterior, podemos utilizar no *mapper* unha única categoría "*all*".

```
for line in sys.stdin:
    data = line.strip().split("\t")
    if len(data) == 6:
        date, time, store, item, cost, payment = data
        print(f'all\t{cost}')
```

Exercicio 2. Cálculo do día máis chuvioso de cada ano

Fonte de datos: histórico meteorolóxico da estación meteorolóxica do campus universitario de As Lagoas-Marcosende, en Vigo (situada a 460 m de altura), correspondente ós **valores diarios da choiva** do período de consulta do 13-11-2018 ó 13-11-2023, obtidos dos datos abertos de Meteogalicia: <https://www.meteogalicia.gal/observacion/estacioneshistorico/consultar.action>

- Ficheiro de datos: **choiva-diaria-CUVI-5-anos-13-11-2018-13-11-2023.csv**

Histórico da estación Vigo-Campus. Vigo. Pontevedra

REALIZAR NOVA CONSULTA

Variables diarias. Período de consulta: 13-11-2018 a 13-11-2023

* Datos en horario UTC
Códigos de validación

Resultados en formato CSV

Resultados en formato CSV en columnas

Resultados en formato JSON

Resultados en formato PDF

Código validación	Data*	Código parámetro	Parámetro	Valor	Unidades
1	13-11-2018	PP_SUM_1.5m	Chuvia	0.0	L/m2
1	14-11-2018	PP_SUM_1.5m	Chuvia	0.0	L/m2
1	15-11-2018	PP_SUM_1.5m	Chuvia	0.0	L/m2
1	16-11-2018	PP_SUM_1.5m	Chuvia	0.0	L/m2
1	17-11-2018	PP_SUM_1.5m	Chuvia	0.0	L/m2
1	18-11-2018	PP_SUM_1.5m	Chuvia	3.6	L/m2

- Obxectivo: calcula-la cantidade de choiva máxima caída nun día para cada ano, a partir do rexistro histórico

Os datos están almacenados nun arquivo csv coa coma como separador, pechados *entre comiñas* dobres e cunha *ringreira de encabezado*, da seguinte forma:

```
"Código validación","Data","Código parámetro","Parámetro","Valor","Unidades"
"1","2018-11-13 00:00:00.0","PP_SUM_1.5m","Chuvia","0.0","L/m2"
"1","2018-11-14 00:00:00.0","PP_SUM_1.5m","Chuvia","0.0","L/m2"
"1","2018-11-15 00:00:00.0","PP_SUM_1.5m","Chuvia","0.0","L/m2"
"1","2018-11-16 00:00:00.0","PP_SUM_1.5m","Chuvia","0.0","L/m2"
"1","2018-11-17 00:00:00.0","PP_SUM_1.5m","Chuvia","0.0","L/m2"
...
```

A primeira columna corresponde ó código de validación que ten os seguintes posibles valores e significados:

Códigos de validación

- 0 - Dato sen validar
- 1 - Dato válido orixinal
- 2 - Dato sospeitoso
- 3 - Dato erróneo
- 5 - Dato válido interpolado
- 9 - Dato non rexistrado

Polo tanto só teremos en conta rexistros co código de validación 1 e 5

Das restantes columnas só nos interesan o trozo de ano da data e a do valor da choiva caída.

map: xerar pares ano-choiva, co ano e o valor de choiva

O código, para cada liña de datos de entrada:

- 1º Elimina espazos en branco (por diante e por detrás) co método **strip()**.
- 2º Extrae o código de validación, o instante de lectura (no que se atopa o ano) e a cantidade de choiva rexistrada utilizando a coma (,) como separador de cada campo.
- 3º Só tomará rexistros correspondentes ós dous códigos de validación válidos, o **1** e o **5**.
- 4º Por último devolve co **print()** os valores chave-valor separados por un espazo en branco unha vez eliminadas as comiñas e truncada a data para que só sexa os 4 díxitos do ano.

mapperMaxChoiva.py

```
#!/usr/bin/python3
# -*- coding: iso-8859-15 -*-

'''
O formato de cada liña do ficheiro de entrada é:
"Código validación","Data","Código parámetro","Parámetro","Valor","Unidades"

Obxectivo: Obter, a partir do rexistro histórico, a cantidade de choiva diaria
para cada lectura e devolvelo precedido de só o número de ano
Para cada lectura obtemo-los pares <ano, choiva_recollida>

p.e.: 2023 4.2
'''

import sys

# Iterar sobre as liñas de entrada dende sys.stdin
for linha in sys.stdin:

    # Eliminar espazos en branco ó principio e ó final da liña
    linha = linha.strip()

    # Descompo-la liña en campos separados por comas
    codigo, instante_lectura, lixo1, lixo2, choiva, lixo3 = linha.split(",")

    # Verificar se o string de código (eliminando as comiñas) está na lista ["1",
    "5"]
    if codigo.strip("'") in ["1", "5"]:

        # Imprimi-lo ano e a cantidade de choiva
        print(instante_lectura[1:5],choiva.strip("'"))
```

Convertemos en executable o mapper:

```
$ chmod u+x mapperMaxChoiva.py
```

Podemos proba-lo código:

```
$ cat choiva-diaria-CUVI-5-anos-13-11-2018-13-11-2023.csv | ./mapperMaxChoiva.py
```

O resultado serán 2 columnas de datos separados por un espazo en branco: a primeira é o ano en formato de 4 díxitos e a segunda un número real cun díxito decimal. Ademais, os datos están ordeados ascendentemente polo ano.

```
2022 0.0
2022 0.0
2022 48.3
2022 30.4
2022 8.7
2022 34.1
2023 54.7
2023 4.7
2023 0.0
2023 0.0
2023 0.0
2023 0.0
2023 38.4
2023 31.5
```

reducer: calcula o valor máximo das choivas rexistradas en cada ano

O código, para cada liña de datos de entrada:

- 1º Trátase a primeira liña recibida fóra do bucle para evitar ter que comprobar para cada unha das seguintes liñas se xa se inicializaron as variables; deste xeito repítese código pero o código é máis eficiente.
- 2º Para cada rexistro recibido na forma dun par chave-valor se separa o ano e a cantidade de choiva.
- 3º No primeiro dato recibido considérase máxima a primeira cantidade de choiva e cando se colle o seguinte dato, compárase a súa choiva coa choiva máxima almacenada; se é superior a choiva actual, actualízase a choiva almacenada.
- 4º Isto faise para tódalas temperaturas ata que cambia a data do ano, na que cambiamos de ano actual, emitindo a solución d anterior ano (ano e choiva máxima).
- 5º O último **print()** emite a solución do último ano.

reducerMaxChoiva.py

```
#!/usr/bin/python3
# -*- coding: iso-8859-15 -*-

'''
O formato de cada liña do ficheiro de entrada é:
- un número enteiro de 4 díxitos correspondente ó ano,
- un espazo en branco e
- un número real correspondente ás choivas en litros/metro cadrado
p.e: 2023 24.6

Obxectivo: calcula-la cantidade de choiva máxima diaria de cada ano a partir do
rexistro histórico
p.e.: 2018 39.0
'''

import sys

# Inicializar variables
ano_actual = None
choiva_maxima_actual = None

# Ler a primeira liña fóra do bucle para evitar comprobar sempre se é
# a primeira liña para cada novo rexistro lido
primeira_linha = sys.stdin.readline()

# Descompo-la primeira liña
ano_actual, choiva_maxima_actual = primeira_linha.strip().split(" ", 1)
choiva_maxima_actual = float(choiva_maxima_actual)

# Iterar sobre as demais liñas de entrada
for linha in sys.stdin:
    ano, choiva_str = linha.strip().split(" ", 1)

    # Converte-la cantidade de choiva a float
```

```

choiva = float(choiva_str)

# Se é o mesmo ano, comprobar se a cantidade de choiva é a máxima
if ano_actual == ano:
    choiva_maxima_actual = max(choiva_maxima_actual, choiva)
else:
    # Se cambia o ano, emitir resultado e actualizar variables
    print("%s\t%s" % (ano_actual, choiva_maxima_actual))
    ano_actual = ano
    choiva_maxima_actual = choiva

# Emiti-lo resultado do último ano
if ano_actual is not None:
    print("%s\t%s" % (ano_actual, choiva_maxima_actual))

```

Convertemos en executable o mapper:

```
$ chmod u+x reducerMaxChoiva.py
```

Podemos proba-lo código:

```
$ cat choiva-diaria-CUVI-5-anos-13-11-2018-13-11-2023.csv | ./mapperMaxChoiva.py
| ./reducerMaxChoiva.py
```

```

r-clima-cuvi-python$ cat choiva-diaria-CUVI-5-anos-13-11-2018-13-11-2023.csv | ./mapperMaxChoiva.py | ./reducerMaxChoiva.py
2018    39.0
2019    89.2
2020    70.8
2021    59.5
2022    71.9
2023    63.5

```

Hai que sinalar que o código do *reducer* funciona porque neste caso o *mapper* está devolvendo os valores xa ordeados pola chave, grazas a que xa estaban ordeados cronoloxicamente no seu ficheiro de entrada, pero de non ser así, para poder proba-lo *map-reduce* sen hadoop, teriamos que ordear-la saída do *map* antes de pasarllo ó *reduce*:

```
$ cat choiva-diaria-CUVI-5-anos-13-11-2018-13-11-2023.csv | ./mapperMaxChoiva.py
| sort -k1,1 | ./reducerMaxChoiva.py
```

Execución en hadoop

Subir ó clúster HDFS o arquivo a tratar.

```
$ hdfs dfs -mkdir -p mr-clima-cuvi-python/input
$ hdfs dfs -put choiva-diaria-CUVI-5-anos-13-11-2018-13-11-2023.csv mr-clima-cuvi-python/input
```

Lanza-lo traballo no clúster:

```
$ mapred streaming \
  -files mapperMaxChoiva.py, reducerMaxChoiva.py \
  -input mr-clima-cuvi-python/input \
  -output mr-clima-cuvi-python/output \
  -mapper mapperMaxChoiva.py \
  -reducer reducerMaxChoiva.py
```

Comprobar que o traballo xerou o ficheiro de resultados esperado no cartafol de saída.

```
$ hdfs dfs -cat mr-clima-cuvi-python/output/*
```

Exercicio 3. Cálculo do día máis chuvioso de cada mes, para cada ano

Obxectivo: calcula-la cantidade de choiva máxima caída nun día, para cada mes de cada ano, a partir do rexistro histórico

Usaremo-lo mesmo ficheiro de entrada:

- Ficheiro de datos: *choiva-diaria-CUVI-5-anos-13-11-2018-13-11-2023.csv*

Os datos están almacenados nun arquivo csv coa *coma* como separador, pechados *entre comiñas* *dobres* e cunha *ringleira de encabezado*, da seguinte forma:

```
"Código validación","Data","Código parámetro","Parámetro","Valor","Unidades"
"1","2018-11-13 00:00:00.0","PP_SUM_1.5m","Chuvia","0.0","L/m2"
"1","2018-11-14 00:00:00.0","PP_SUM_1.5m","Chuvia","0.0","L/m2"
"1","2018-11-15 00:00:00.0","PP_SUM_1.5m","Chuvia","0.0","L/m2"
"1","2018-11-16 00:00:00.0","PP_SUM_1.5m","Chuvia","0.0","L/m2"
"1","2018-11-17 00:00:00.0","PP_SUM_1.5m","Chuvia","0.0","L/m2"
...
```

map: xerar pares periodo-choiva, co ano e mes como chave e a cantidade de choiva rexistrada como valor

Respecto do anterior problema, agora só hai que cambiar no *mapper*, que se inclúa o mes xunto ó ano.

O resultado serán 2 columnas de datos separados por un espazo en branco: a primeira é o ano en formato de 4 díxitos, un guión e 2 díxitos para o mes, e a segunda columna será un número real cun dígito decimal. Ademais, os datos están ordeados ascendentemente polo ano-mes.

mapperMaxChoivaMensual.py

Realmente bastaría cunha pequena modificación na última liña do código para que tome os 7 primeiros caracteres en vez dos 4 primeiros da columna de Data:

```
# Imprimi-lo ano-mes e a cantidade de choiva
print(instante_lectura[1:8],choiva.strip(''))
```

O código completo corrixido sería o seguinte:

```
#!/usr/bin/python3
# -*- coding: iso-8859-15 -*-

'''
O formato de cada liña do ficheiro de entrada é:
"Código validación","Data","Código parámetro","Parámetro","Valor","Unidades"

Obxectivo: Obter, a partir do rexistro histórico, a cantidade de choiva máxima
caída nun día dun mes e devolvelo precedido de só o número de ano - número de
mes
Para cada lectura obtemo-los pares <ano, choiva_recollida>
```



```

p.e: 2023-10 49.7

'''
import sys

# Iterar sobre as liñas de entrada dende sys.stdin
for linha in sys.stdin:

    # Eliminar espazos en branco ó principio e ó final da liña
    linha = linha.strip()

    # Descompo-la liña en campos separados por comas
    codigo, instante_lectura, lixo1, lixo2, choiva, lixo3 = linha.split(",")

    # Verificar se o string de código (eliminando as comiñas) está na lista ["1",
"5"]
    if codigo.strip('') in ["1", "5"]:

        # Imprimi-lo ano-mes e a cantidade de choiva
        print(instante_lectura[1:8],choiva.strip(''))

```

Convertemos en executable o mapper:

```
$ chmod u+x mapperMaxChoivaMensual.py
```

Podemos proba-lo código:

```
$ cat choiva-diaria-CUVI-5-anos-13-11-2018-13-11-2023.csv |
./mapperMaxChoivaMensual.py
```

```

2022-12 0.0
2022-12 0.0
2022-12 48.3
2022-12 30.4
2022-12 8.7
2022-12 34.1
2023-01 54.7
2023-01 4.7
2023-01 0.0
2023-01 0.0
2023-01 0.0
2023-01 0.0
2023-01 38.4
2023-01 31.5

```

reducer: calcula o valor máximo das choivas rexistradas en cada ano

Non é necesario cambiar absolutamente nada, xa que vai recibir unha cadea como chave, que en vez dos 4 díxitos do ano, serán os 4 díxitos do ano, un guión e 2 díxitos do mes, pero o código que o trata é idéntico, posto que simplemente vai comparar os valores coa mesma chave para devolver aquel que teña o valor de choiva máis alto.

De seguido reproduzo o código, aínda que o único cambio foi dos nomes dos identificadores para que o seu significado sexa coherente co novo obxectivo do proceso.

reducerMaxChoivaMensual.py

```
#!/usr/bin/python3
# -*- coding: iso-8859-15 -*-

'''
O formato de cada liña do ficheiro de entrada é:
- unha cadea composta dun número enteiro de 4 díxitos correspondente ó ano_mes,
- un guión,
- un número enteiro de 2 díxitos correspondente ó mes,
- un espazo en branco e
- un número real correspondente ás choivas en litros/metro cadrado
p.e: 2023-11 50.4

Obxectivo: calcula-la cantidade de choiva total mensual de cada ano_mes-mes a
partir do rexistro histórico
'''

import sys

# Inicializar variables
ano_mes_actual = None
choiva_acumulada_actual = None

# Ler a primeira liña fóra do bucle para evitar comprobar sempre se é
# a primeira liña para cada novo rexistro lido
primeira_linha = sys.stdin.readline()

# Descompo-la primeira liña
ano_mes_actual, choiva_acumulada_actual = primeira_linha.strip().split(" ", 1)
choiva_acumulada_actual = float(choiva_acumulada_actual)

# Iterar sobre as demais liñas de entrada
for linha in sys.stdin:
    ano_mes, choiva_str = linha.strip().split(" ", 1)

    # Converte-la cantidade de choiva a float
    choiva = float(choiva_str)

    # Se é o mesmo ano_mes, comprobar se a cantidade de choiva é a máxima
    if ano_mes_actual == ano_mes:
        choiva_acumulada_actual = max(choiva_acumulada_actual, choiva)
    else:
        # Se cambia o ano_mes, emitir resultado e actualizar variables
        print("%s\t%s" % (ano_mes_actual, choiva_acumulada_actual))
```

```

        ano_mes_actual = ano_mes
        choiva_acumulada_actual = choiva

# Emiti-lo resultado do último ano_mes
if ano_mes_actual is not None:
    print("%s\t%s" % (ano_mes_actual, choiva_acumulada_actual))

```

Convertemos en executable o reducer:

```
$ chmod u+x reducerMaxChoivaMensual.py
```

Podemos probalo código:

```
$ cat choiva-diaria-CUVI-5-anos-13-11-2018-13-11-2023.csv |
./mapperMaxChoivaMensual.py | ./reducerMaxChoivaMensual.py
```

```

r-clima-cuvi-python$ cat choiva-diaria-CUVI-5-anos-13-11-2018-13-11-2023.csv | ./mapperMaxChoivaMensual.py | ./reducerMaxChoivaMensual.py
2018-11 33.5
2018-12 39.0
2019-01 89.2
2019-02 19.5
2019-03 49.3
2019-04 21.0

```

Execución en hadoop

Subir ó clúster HDFS o arquivo a tratar, se non o fixemos antes.

```
$ hdfs dfs -mkdir -p mr-clima-cuvi-python/input
$ hdfs dfs -put choiva-diaria-CUVI-5-anos-13-11-2018-13-11-2023.csv mr-clima-cuvi-python/input
```

Lanza-lo traballo no clúster:

```
$ mapred streaming \
  -files mapperMaxChoivaMensual.py, reducerMaxChoivaMensual.py \
  -input mr-clima-cuvi-python/input \
  -output mr-clima-cuvi-python/output2 \
  -mapper mapperMaxChoivaMensual.py \
  -reducer reducerMaxChoivaMensual.py
```

Aínda que estamos usando unha chave considerando que o guión (-) forma parte dela, en realidade estamos usando 2 campos chave.

En casos como este, nos que a chave está conformada por varios valores, poderíamos indicarlle ó comando Hadoop que o *mapper* vai cunha chave composta (neste caso 2 valores), e que ademais cada parte vai separada por un determinado carácter (un guión neste caso).

Podemos probalo, aínda que agora hai que especificar unha saída diferente porque Hadoop non sobrescribe a anterior e daría un erro:

```
$ mapred streaming \
  -Dstream.num.map.key.fields=2 \
  -Dmap.output.key.field.separator="-" \
  -files mapperMaxChoivaMensual.py, reducerMaxChoivaMensual.py \
  -input mr-clima-cuvi-python/input \
  -output mr-clima-cuvi-python/output3 \
  -mapper mapperMaxChoivaMensual.py \
  -reducer reducerMaxChoivaMensual.py
```

Comprobar que o traballo xerou o ficheiro de resultados esperado no cartafol de saída.

```
$ hdfs dfs -cat mr-clima-cuvi-python/output3/*
```

Exercicio 4. Cálculo do día máis chuvioso de cada mes, para cada ano (nos últimos 20 anos)

Corroborar os datos obtidos dos últimos 5 anos, no anterior exercicio, cos datos rexistrados no ficheiro *choiva-temperatura-mensual-CUVI-20-anos-11-2006-11-2023.csv*, que contén os valores mensuais de choiva e temperatura do período de consulta do 31-10-2006 ó 13-11-2023.

Os datos están almacenados nun arquivo csv coa coma como separador, pechados entre comiñas dobres e cunha ringleira de encabezado, da seguinte forma:

```
"Instante lectura","Chuvia","Chuvia diaria máxima","Temperatura máxima a 1.5m","Temperatura media a 1.5m","Temperatura mínima a 1.5m"
"2006-11-01 00:00:00.0","-9999.0","-9999.0","22.2","13.5","6.9"
"2006-12-01 00:00:00.0","289.2","-9999.0","14.4","9.3","3.2"
"2007-01-01 00:00:00.0","64.1","9.8","16.2","9.1","1.2"
"2007-02-01 00:00:00.0","216.0","37.2","14.1","9.5","4.0"
```

Notas:

- O valor -9999.0 indica dato non rexistrado.
- Os valores de choiva son en L/m²

Os resultados dos exercicios 3 e 4 son os mesmos nos anos coincidentes?

Exercicio 5. Cálculo do día máis caluroso de cada mes, para cada ano (nos últimos 20 anos)

Usando de novo os datos rexistrados no ficheiro *choiva-temperatura-mensual-CUVI-20-anos-11-2006-11-2023.csv*, que contén os valores mensuais de choiva e temperatura do período de consulta do 31-10-2006 ó 13-11-2023, calcula a temperatura máxima mensual para cada ano a partir do rexistro histórico.

É complicado modifica-lo código das anteriores consultas para adaptalo ó novo obxectivo?

Exercicio 6

A función do mapper executarase sobre todas e cada unhas liñas do ficheiro de datos. O código do mapper anterior non funcionará correctamente se se encontra algunha liña que non encaixa co número exacto de valores separados por tabulador.

Mellora o código de xeito que se encontra algunha liña cun formato non axeitado a descarte e siga traballando coa liña seguinte.

Solucionamos engadindo unha liña no mapper que comprobe que en cada liña que procesa hai exactamente 6 elemento. Caso contrario non trata esa liña e pasa á seguinte.

```
for line in sys.stdin:
```

```
    data = line.strip().split("\t")
```

```
    if len(data) == 6:
```

```
        date, time, store, item, cost, payment = data
```

```
        print(f'{store}\t{cost}')
```

Exercicios propostos

Exercicio 1: <https://grouplens.org/datasets/movielens/>

Modifica-lo programa WordCount para que conte o número de palabras que teñen os nomes de películas do ficheiro de datos chamado *u.item*

<https://files.grouplens.org/datasets/movielens/ml-100k.zip>

<https://files.grouplens.org/datasets/movielens/ml-100k/>

Exercicio 2:

Deseñar un algoritmo utilizando MapReduce que, partindo dos datos de *Movielens*, xere unha lista ordeada dos anos das películas e o promedio de valoración para cada ano.

Implementa-lo algoritmo deseñado utilizando Hadoop

Exemplos código map e código reduce: Solucións ó curso Udacity Intro to Hadoop and MapReduce

- Exemplo mapper.py e reducer.py para hadoop

Código python para calcula-las vendas

<https://github.com/juandecarrion/udacity-hadoop-course/tree/dba5d5ec87e1e807403f1f1f8cb3f5cf57097575>

- Máis exemplos e inclúe un ficheiro purchases grande:

<https://github.com/CodeMangler/udacity-hadoop-course>

- purchases.txt

<https://github.com/juandecarrion/udacity-hadoop-course/blob/master/testdata/purchases.txt>

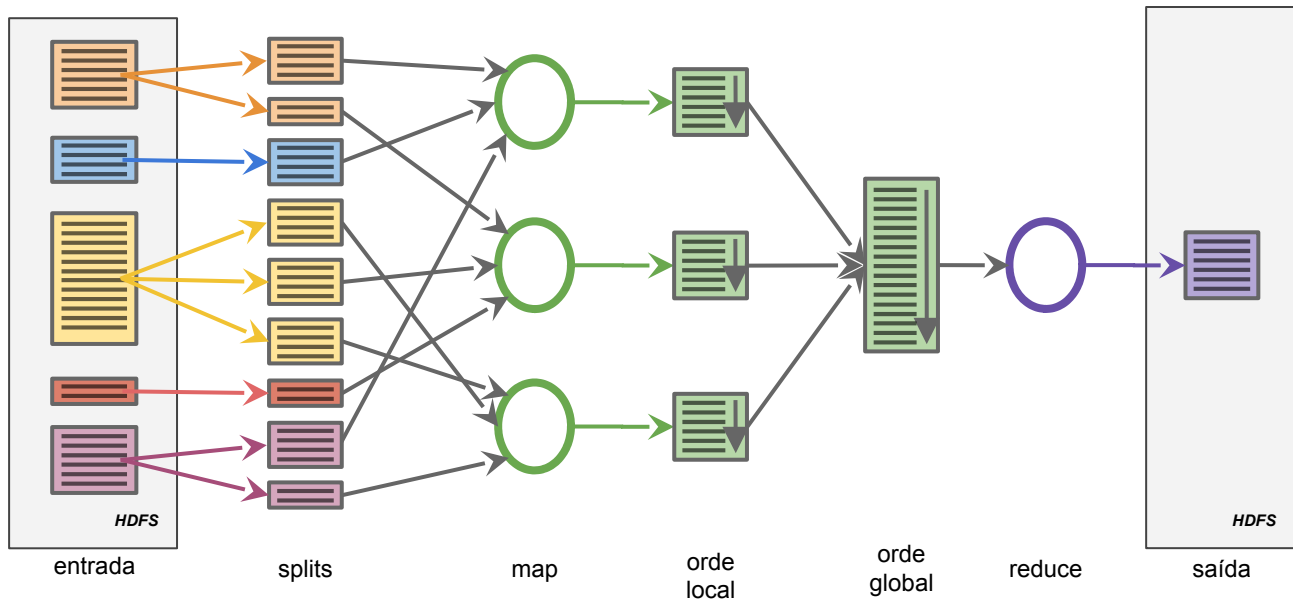
Intro to Hadoop and MapReduce

<https://classroom.udacity.com/courses/ud617>

<https://classroom.udacity.com/courses/ud617/lessons/308873795/concepts/3092715800923>

Execución de trabajos mapreduce

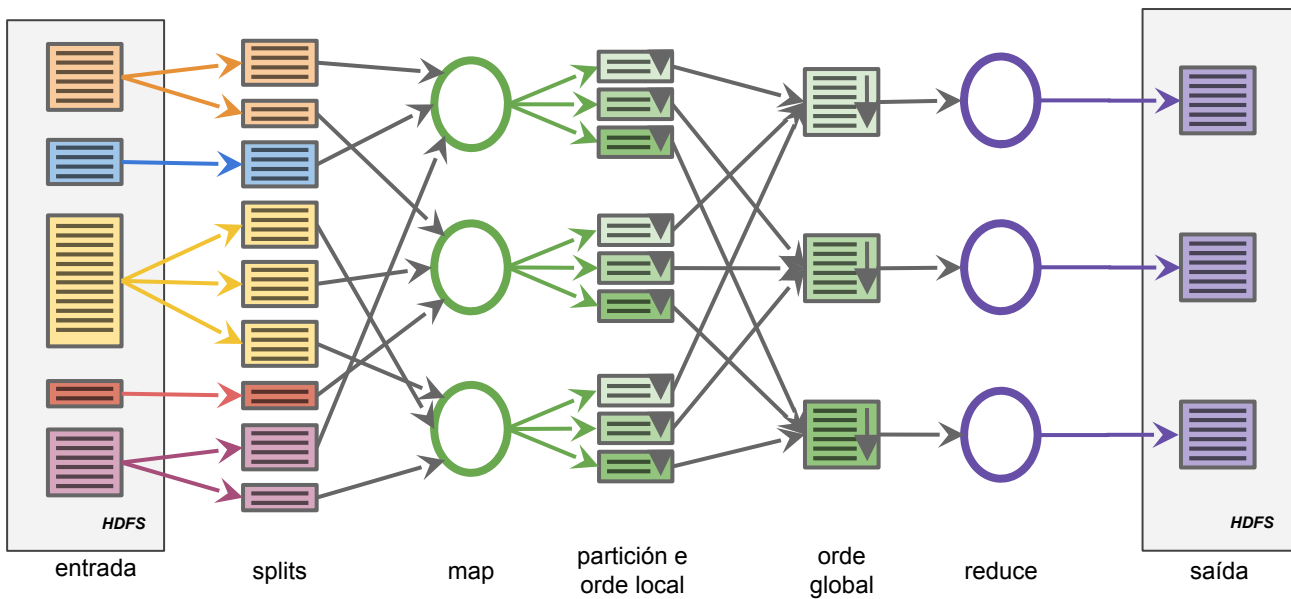
A execução da tarefa implica varios traballos:



Opcións de execución

Número de reduces

- É posible executar varias tarefas de redución en paralelo
 - Recomendado para tarefas de redución con procesamento intensivo
 - Obtéñense varios ficheiros de saída
- Utilízase unha función de **partición** para repartir las claves entre as diferentes tarefas de redución
 - Por defecto, aplícase unha función de hash ás claves



- Valor por defecto: 1
 - Función por defecto
- Valor recomendado: $x * c$
 - $0.95 < x < 1.75$
 - c = máximo total de contadores
- Valor posible: 0
 - Non se executa función de redución
 - Almacénanse os resultados do *map*
- Cambiar por código

```
job.setNumReduceTasks(4)
```

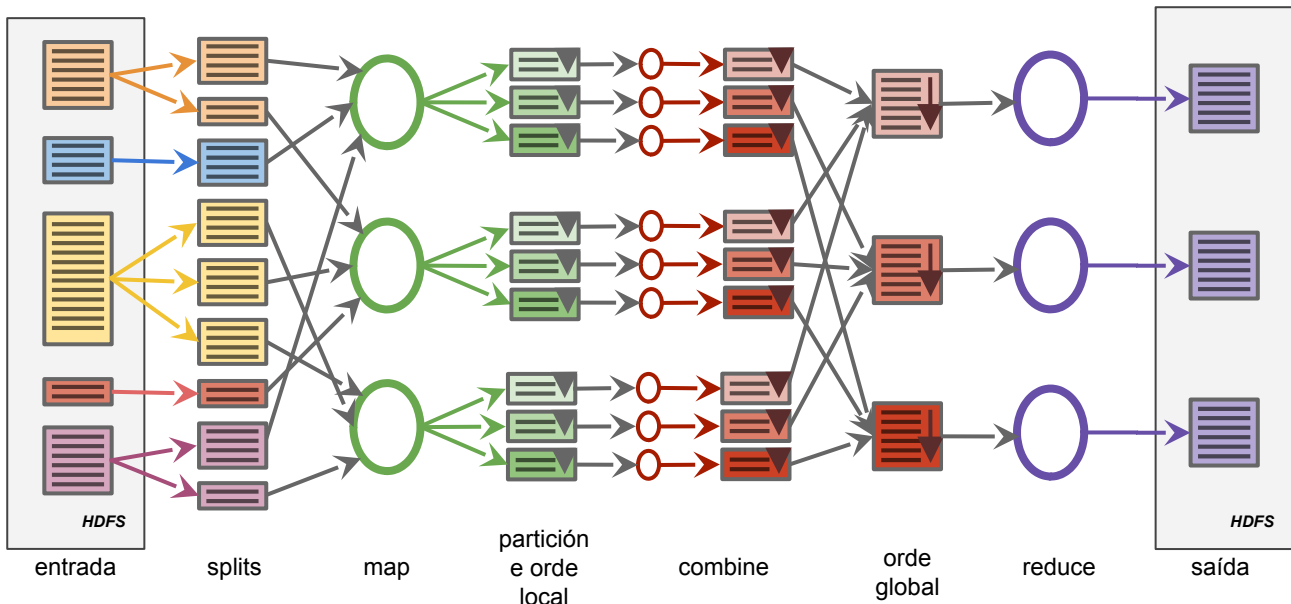
- Fichero de configuración **mapred-site.xml**

```
<configuration>
  <property>
    <name>mapreduce.job.reduces</name>
    <value>4</value>
  </property>
</configuration>
```

Combiner

- É posible realizar unha operación de redución parcial despois do *map* antes de envia-los resultados á etapa de redución
- Permite reducir información transmitida entre os nodos
- Os tipos de entrada e saída deben coincidir cos do *reducer*

```
job.setCombinerClass(WordCountReducer.class);
```



Exemplo de uso dun combiner:

```
$ mapred streaming \
-Dstream.num.map.key.fields=2 \
-Dmap.output.key.field.separator="-" \
-files mapperMaxChoivaMensual.py,reducerMaxChoivaMensual.py \
-input mr-clima-cuvi-python/input \
-output mr-clima-cuvi-python/output2 \
-mapper mapperMaxChoivaMensual.py \
-reducer reducerMaxChoivaMensual.py \
-combiner reducerMaxChoivaMensual.py
```

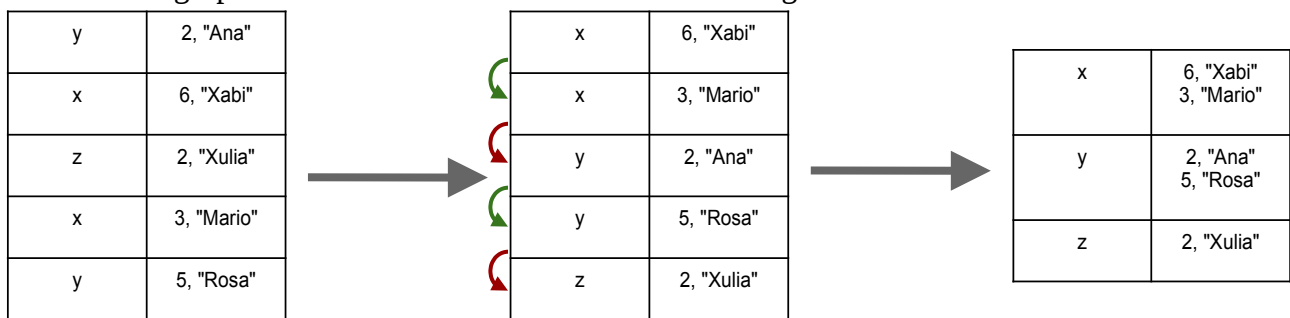
Manexo de datos

- Pode manexar diferentes formatos de entrada e saída
 - Ficheiros de texto
 - 1 ou máis liñas como valor, chave-valor separados por tab ou outro carácter, documentos XML
 - Ficheiros binarios
 - Múltiples ficheiros, con diferente formato
 - Permite traballar con varios ficheiros de entrada ou saída
 - Bases de datos relacionais
 - (!) As tarefas poderían sobrecargar ó servidor de orixe

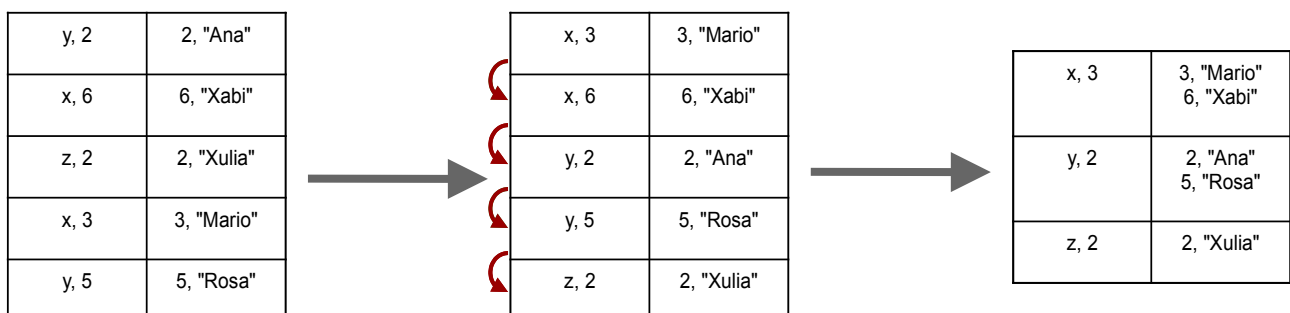
- Pode utilizar diferentes ferramentas para serialización dos datos en formato binario
 - Writables
 - Pode manexar estruturas complexas mediante código Java
 - Apache Avro
 - Permite interoperar con outras linguaxes
 - Permite definir estruturas complexas en formato JSON
 - Permite definir esquema ó escribir e ó le-los datos
- Pode utilizar compresión
 - Reduce o tamaño dos datos para almacenar ou transmitir
 - Aplicable a datos de entrada e de saída e á saída da función de mapeo
- Pode traballar con ficheiros agrupados
 - Permite almacenar varios ficheiros pequenos nun mesmo bloque

Orde secundario

- A agrupación de valores por chave realízase en dúas partes
 - Ordear tódolos pares chave-valor por chave
 - Agrupar tódolos valores de chaves consecutivas iguais



- É posible modifica-lo criterio para agrupar pares
 - Permite definir chaves con máis información para ordear-la saída do mapeo por máis dun criterio



Contadores

- Almacenan información para control e depuración dos procesos
 - Contadores de sistema

- Por tarefas, por traballo
- Exemplo: número de tarefas de *map* executadas
- Contadores de usuario/a
 - Definidos mediante código Java
 - Exemplo: número de rexistros con formato inválido

Funcionalidades comúns

Hadoop inclúe unha librería de clases con funcionalidades comúns

- Atópanse dentro do espazo do nome ***org.apache.hadoop.mapreduce.lib***
- Permiten minimizar ou elimina-la necesidade de programar tarefas frecuentes
- JOIN de datos antes da fase de mapeo
- Análise dunha mostra dos datos
 - Permite particiona-los datos para a fase de redución de forma que sexa posible obter datos ordeados concatenando os diferentes ficheiros de saída
- Concatenador de mappers
 - Executa varios mappers nunha soa tarefa de mapeo ou redución
 - Permite definir pasos reutilizables por ser máis pequenos, pero evita a sobrecarga do trasiego de datos
- Mappers ou reducers comúns
 - Inversión chave $\leftarrow \rightarrow$ valor
 - Tokenizador de palabras
 - Buscador de expresións regulares
 - Agregación de datos