Fall 2016

Shavadoop

Charlotte Eli & Mahzad Kalantari

Professor: Remy Sharrock

Contents

1 Context and Objectives

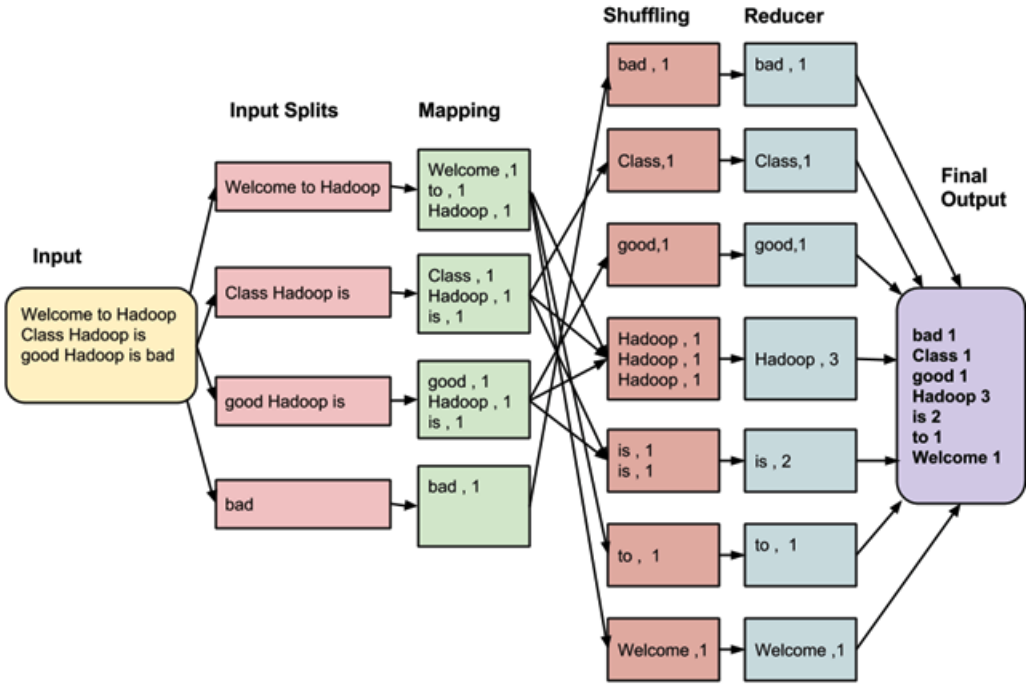
1.1 Wordcount

WordCount is a simple application that counts the number of occurrences of each word in a given input set.

- **Input** : text file
- **Output** : dictionnay <key, value> where keys are words and values correspond to their number of occurence

1.2 MapReduce

A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks.
The figure below represents a "word count" using MapReduce methodology:



1.3 Shavadoop

The goal of Shavadoop is to implement a "word count" in Java using MapReduce methodology. We use a simple network architecture : a Master and a certain number of slaves (workers), the number of slaves being an input of our program. The master's role is

to manage, oversee and launch programm remotly. Workers are in charge of running tasks according to master's instructions. Communication between the master and the workers is done via SSH protocol.

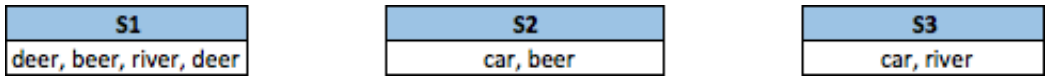
2 Program Architecture

2.1 From input.txt to wordcount: temporary files

Our program use the *input.txt* to create:

- Sx: the input file is split into n Sx where n corresponds to the number of workers

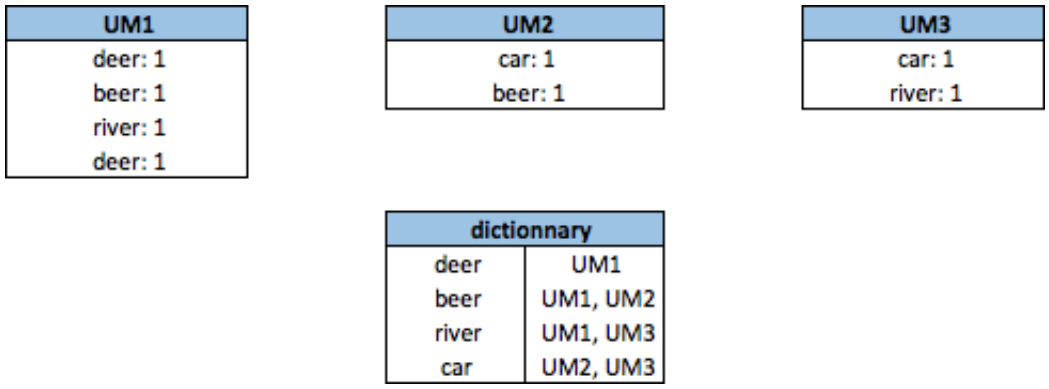
Figure 1: Step 1: input input.txt, output : Sx



- UMx: correponds to the list of words contained in Sx where each word appears as many times as it appears in Sx

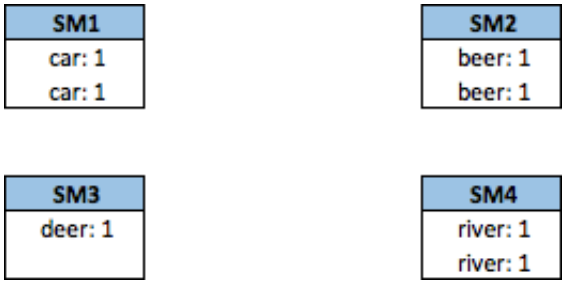
  Dictionnary <word,list of UMx in which word appears>

Figure 2: Step 2: input Sx output : Umx + dictionnary



- SMx: one SM per word, SMx contain every occurence of the corresponding word

Figure 3: Step 3: input Umx + dictionnary output SMx



- RMx: one RM per word, RMx sums the occurences of the SMx

Figure 4: Step 4: input SMx output RMx

- Final Wordcount

Figure 5: Final Step: input RMx, output final

Wordcount	
deer	1
beer	2
river	2
car	2

2.2 Code Architecture

Code is organized around four packages:

- Master : Contains all features link to the Master : split, dictionnaries, merge, slaves'launching...
- Slave : Contains all features link to workers (map,reduce, sort, shuffle...)

- TestConnectionSSH : Contains all functions link to networks and SSH protocol (network verification, identification of machines available for computation...)
- MasterConnectSlave : Enables master to manage slaves (workers, reducers..)

3 Program's Operations

3.1 Master

The Master class contains three dictionnaries neccessary for MapReduce implementation :

- $< UM_x,$ Machine >
- < Work, UM_x list >
- $< RMS_x,$ Machine >

We have added the following dictionnary:

< Machine, list of words to be reduced >

The Master class is composed of 8 methods:

- *InitialiseDossierShava* is the first method that should be called. It aims at cleaning the destination folder.
- *SplitFichier(nb workers)* takes as input the number of machines that we want/need for mapping's execution $SM_x \rightarrow UM_x$. It was our decision to make the number of data-processing machines flexible. This should be the second method to be called.
- *setMachine(machines list)* allows to identify n operative machines at Telecom Paristech where n is the number fixed in the previous class.
- *LaunchSlaveModeSxUMx(folder's name)* launches the mapping $S_x \rightarrow UM_x$ performed by the slaves.
- *SetDicoUMHost* constructs $< UM_x,$ Machine > dictionnary.
- *SetDicoRMSMachine* constructs $< RMS_x,$ Machine > dictionnary.
- *LaunchSlaveModeUMxSMx(folder's name)* performs $UM_x \rightarrow SM_x$ and RM_x mapping. Shuffling and Reducing are also performed in this class. The number of reducers is flexible, each reducing a certain number of words.
- *ReduceFinal* gathers all RM_x files from slaves and creates a file *Resultats* with the final results.

3.2 Slave

Slave class performs mapping operations $SM_x \rightarrow UM_x$ and $UM_x \rightarrow RM_x$. We specify the type of mapping that we want using the class *typeMapping*, it uses :

- Splitmapping Class for $SM_x \rightarrow UM_x$
- MappingSM Class for $UM_x \rightarrow RM_x$.

The reducing part is performed by *MappingSm* class.

3.3 MasterConnectSlave

This class launches .jar file on each machine.

4 Shavadoop in Practise

4.1 How to use the program?

The Master Class requires the following arguments :

- Input.txt: input to our wordcount program.
- Folder's name: folder where we want to store the temporary files necessary to preform the wordcount and the final result file.
- List of Telecom Paristech machines'IP.
- Number of workers that we want to use.
- Number of reducers that we want to use.
- Username : prior to launching the program, the user will have to ensure that he has a public and a private key to use SSH protocol.
- Folder containing .jar files

4.2 Results

Once we have performed these operations, we can launch the Master file using Eclipse or the Master.jar directly. The results are stored in the specified folder.

We have obtained the following execution time using 4 workers and 3 reducers (the user can use as many as wanted):

Figure 6: Code Forestier Mayotte

```
c45-18% java -jar master.jar forestier_mayotteClean.txt /cal/homes/mkalantari/Shavadoop_files
liste_machines.txt 4 3 mkalantari /cal/homes/mkalantari/SHAVA/bin

Début du split du fichier Input
Fin du split du fichier Input => Exécution en 17ms

Début du test sur la connection
Fin du test sur la connection des machines => Exécution en 14212ms

Début du mapping en UM
Fin du mapping en UM => Exécution en 4015ms

Début du mapping en RSM
[TestConnectionSSH c130-04.enst.fr] Exception in thread "main" java.lang.
ArrayIndexOutOfBoundsException: 3
[TestConnectionSSH c130-04.enst.fr]     at Slave.main(Slave.java:217)
Fin du mapping en RSM => Exécution en 20126ms
Tout est fini
```

Figure 7: Code Deontologie Police Nationale

```
c45-18% java -jar master.jar deontologie_police_nationaleClean.txt /cal/homes/mkalantari/
Shavadoop_files liste_machines.txt 4 3 mkalantari /cal/homes/mkalantari/SHAVA/bin

Début du split du fichier Input
Fin du split du fichier Input => Exécution en 34ms

Début du test sur la connection
Fin du test sur la connection des machines => Exécution en 14189ms

Début du mapping en UM
Fin du mapping en UM => Exécution en 1339ms

Début du mapping en RSM
[TestConnectionSSH c130-04.enst.fr] Exception in thread "main" java.lang.
ArrayIndexOutOfBoundsException: 3
[TestConnectionSSH c130-04.enst.fr]     at Slave.main(Slave.java:217)
Fin du mapping en RSM => Exécution en 141937ms
Tout est fini
```

Figure 8: Code Domaine Public Fluvial

```
c45-18% java -jar master.jar domaine_public_fluvialClean.txt  /cal/homes/mkalantari/Shavadoop_files liste_machines.txt
4 3 mkalantari /cal/homes/mkalantari/SHAVA/bin

Début du split du fichier Input
Fin du split du fichier Input => Exécution en 27ms

Début du test sur la connection
mkalantari@c127-05.enst.fr's password:
mkalantari@c127-05.enst.fr's password: Fin du test sur la connection des machines => Exécution en 19826ms

Début du mapping en UM
Fin du mapping en UM => Exécution en 1833ms

Début du mapping en RSM
[TestConnectionSSH c130-01.enst.fr] Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
[TestConnectionSSH c130-01.enst.fr]    at Slave.main(Slave.java:216)
[TestConnectionSSH c130-01.enst.fr] zsh:1: parse error near `)'
[TestConnectionSSH c130-02.enst.fr] zsh:1: bad pattern: (avec
Fin du mapping en RSM => Exécution en 342797ms
Tout est fini
```

4.3 Optimisation and Project's Singularity

- Data Cleaning: before using shavadoop program, *input.txt* is cleaned, getting rid of empty lines, special characters, pronouns, very common words... We create a cleaned input file called *inputClean.txt*.
- Our program allows the user to specify the number of workers and reducers.

4.4 Problem encountered

MapReduce operations are performed using Telecom Paristech's network sharing.We haven't implemented a reading via system.out.
Furthermore, files'reading is highly time consumming, communication via scp between master and slave would be a way to optimize the program.

4.5 Technical Documentation

Javadoc is located in the doc folder. We have documented every classes and their corresponding private members (even if it's against java's philosophy :)).

4.6 Github Link

Our project is stored at `https://github.com/MahzadK/SHAVADOOP-BGD`