Master Side:

-1. Initialization step

0. Clean import step

1. Input splitting step

2. Send Mapping Orders step

3. Shuffling step

4. Send Reducing Orders step

5. Result step

main:

|-Utils.java

|-Main.java

|-Config.java

# Utils.java

package main;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Utils {

public static Integer getId(String rawName) {

Pattern r = Pattern.compile("(\\d+)");

Matcher m = r.matcher(rawName);

m.find();

String id = m.group(0);

return Integer.parseInt(id);

}

public static void printBeautiful(String toPrint) {

String line = "---------------------------------";

System.out.println(line);

System.out.println(toPrint.toUpperCase());

System.out.println(line);

}

public static void printDiffTime(long stepTime) {

System.out.println("---Time: "+(System.currentTimeMillis() - stepTime)+" ---");

}

}

# main.java

package main;

import java.io.IOException;

import java.nio.file.Path;

import java.nio.file.Paths;

import network.NetworkConfig;

import wordcount.WordCount;

public class Main {

public static void main(String[] args) throws IOException, InterruptedException {

// Check passed arguments

if (args.length != 1){

System.out.println("Please pass (only) one argument to specify input file location relative to jar file, or absolute path.");;

System.exit(0);

}

Path inputPath = Paths.get(args[0]);

// Set config

Config config = new Config();

config.setInputLocation(inputPath);

NetworkConfig networkConfig = new NetworkConfig();

// Launch wordcount

Utils.printBeautiful("Launch programm on " + inputPath);

WordCount wordcount = new WordCount(config, networkConfig);

wordcount.computeWordCount();

}

}

# Config.java

package main;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.Arrays;

import java.util.List;

public class Config {

public List<String> filteredWords;

public Path slaveJarLocation; // SLAVE.jar location, relative (to ssh arrival point), or absolute

public Path inputFilePath; // input file location, relative (to MASTER.jar), or absolute

public Integer linesPerSplit;

public Path masterJobsPath; // generated files location, relative (to MASTER.jar), or absolute

public Path folderSx;

public Path folderUmx;

public Path folderSmx;

public Path folderRmx;

public Path folderResult;

public Path machinesToTestPath;

public Path machinesRespondingPath;

public Config(){

this.filteredWords = Arrays.asList("le","la","les","je","du","de","des","ou","nous","vous","leur","eux","et","ne","l","à","en","par","ses","ce","son","un","une");

this.slaveJarLocation = Paths.get("/cal/homes/lbinet/workspace/Sys\_distribue/");

this.linesPerSplit = 1;

this.masterJobsPath = Paths.get("/cal/homes/lbinet/workspace/Sys\_distribue/Jobs/");

this.folderSx = this.masterJobsPath.resolve("Sx/");

this.folderUmx = this.masterJobsPath.resolve("Umx/");

this.folderSmx = this.masterJobsPath.resolve("Smx/");

this.folderRmx = this.masterJobsPath.resolve("Rmx/");

this.folderResult = this.masterJobsPath.resolve("Result/");

this.machinesToTestPath = Paths.get("liste\_machines.txt");

this.machinesRespondingPath =Paths.get("liste\_machines\_OK.txt");

}

public void setInputLocation(Path inputPath){

this.inputFilePath = inputPath;

}

}

network

|--CheckMachinesUp.java

|--NetworkConfig.java

|--SshCommand.java

|--StreamReader.java

# CheckMachinesUp.java

package network;

import java.io.IOException;

import java.nio.charset.Charset;

import java.nio.file.Files;

import java.nio.file.Path;

import java.util.ArrayList;

import java.util.List;

import main.Utils;

public class CheckMachinesUp {

private List<String> machinesToTest;

private ArrayList<String> respondingMachines;

private NetworkConfig networkConfig;

public CheckMachinesUp(NetworkConfig networkConfig){

this.networkConfig = networkConfig;

}

public void readMachinesToTest(Path path) throws IOException{

this.machinesToTest = Files.readAllLines(path, Charset.forName("UTF-8"));

}

public void writeRespondingMachines(Path path) throws IOException{

Files.write(path, this.respondingMachines, Charset.forName("UTF-8"));

}

public void test\_Machines\_Up() throws IOException, InterruptedException{

Utils.printBeautiful("Machine test");

ArrayList<SshCommand> sshTestList = new ArrayList<SshCommand>();

for (String machineToTest : this.machinesToTest) {

SshCommand sshCommand = new SshCommand(this.networkConfig, machineToTest,"test ssh", true);

sshCommand.start();

sshTestList.add(sshCommand);

}

ArrayList<String> respondingMachines = new ArrayList<String>();

for (SshCommand sshTest : sshTestList) {

sshTest.join();

if (sshTest.isConnectionOK()) {

respondingMachines.add(sshTest.getMachine());

}

}

this.respondingMachines = respondingMachines;

System.out.println(respondingMachines.size()+" machine(s) responding out of "+this.machinesToTest.size()+".");

}

public ArrayList<String> get\_Machines\_Up(){

return this.respondingMachines;

}

}

# NetworkConfig.java

package network;

import java.nio.file.Path;

import java.nio.file.Paths;

public class NetworkConfig {

public Integer maxThreadsPerMachine;

public Integer timeout;

public Integer test\_timeout; // when testing machines

public Path machinesToTestPath;

public Path machinesRespondingPath;

public String sshUser;

public String sshKeyPath;

public NetworkConfig(){

this.maxThreadsPerMachine = 3;

this.timeout = 2;

this.test\_timeout = 1;

this.machinesToTestPath = Paths.get("liste\_machines.txt");

this.machinesRespondingPath =Paths.get("liste\_machines\_OK.txt");

this.sshUser = "lbinet";

this.sshKeyPath = "/cal/homes/lbinet/.ssh/intra\_telecom";

}

}

# SshCommand.java

package network;

import java.io.IOException;

import java.util.ArrayList;

import java.util.concurrent.ArrayBlockingQueue;

import java.util.concurrent.TimeUnit;

public class SshCommand extends Thread{

// Network config

private NetworkConfig networkConfig;

// Other

private String machine;

private String command;

private Boolean sshTest;

// Reponse

private ArrayBlockingQueue<String> standard\_output = new ArrayBlockingQueue<String>(1000);

private ArrayBlockingQueue<String> error\_output = new ArrayBlockingQueue<String>(1000);

private ArrayList<String> response;

private boolean connection\_status = false;

public String getMachine() {

return machine;

}

public void setConnectionOK(boolean connection\_status) {

this.connection\_status = connection\_status;

}

public boolean isConnectionOK() {

return connection\_status;

}

public SshCommand(NetworkConfig networkConfig, String machine, String command, boolean sshTest){

this.sshTest = sshTest;

this.machine = machine;

this.networkConfig = networkConfig;

if (sshTest){

this.command = "echo OK";

}

else {

this.command = command;

}

this.response = new ArrayList<String>();

}

public void show(String texte){

System.out.println("[SSH "+machine+"] "+texte);

}

public void run(){

Integer timeout;

if (this.sshTest){

timeout = this.networkConfig.test\_timeout;

}

else{

timeout = this.networkConfig.timeout;

}

try {

String[] fullCommand = {

"ssh",

"-i",this.networkConfig.sshKeyPath,

"-o","StrictHostKeyChecking=no",

this.networkConfig.sshUser+"@"+this.machine,

this.command};

ProcessBuilder pb = new ProcessBuilder(fullCommand);

Process p = pb.start();

StreamReader fluxSortie = new StreamReader(p.getInputStream(), standard\_output);

StreamReader fluxErreur = new StreamReader(p.getErrorStream(), error\_output);

new Thread(fluxSortie).start();

new Thread(fluxErreur).start();

String s = standard\_output.poll(timeout, TimeUnit.SECONDS);

while(s!=null && !s.equals("ENDOFTHREAD")){

if (this.sshTest){

if(s.contains("OK")){

connection\_status = true;

}

}

// affiche(s);

this.response.add(s);

s = standard\_output.poll(this.networkConfig.timeout, TimeUnit.SECONDS);

}

s = null;

s = error\_output.poll(timeout, TimeUnit.SECONDS);

while(s!=null && !s.equals("ENDOFTHREAD")){

if (! this.sshTest){

// do not print errors for ssh tests

show(s);

}

this.response.add(s);

s = error\_output.poll(this.networkConfig.timeout, TimeUnit.SECONDS);

}

} catch (IOException e) {

e.printStackTrace();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public ArrayList<String> get\_response(){

return this.response;

}

public String get\_command(){

return this.command;

}

}

# StreamReader.java

package network;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.util.concurrent.ArrayBlockingQueue;

class StreamReader implements Runnable {

private final InputStream inputStream;

ArrayBlockingQueue<String> output;

StreamReader(InputStream inputStream, ArrayBlockingQueue<String> output) {

this.inputStream = inputStream;

this.output = output;

}

private BufferedReader getBufferedReader(InputStream is) {

return new BufferedReader(new InputStreamReader(is));

}

@Override

public void run() {

BufferedReader br = getBufferedReader(inputStream);

String ligne = "";

try {

while ((ligne = br.readLine()) != null) {

output.put(ligne);

}

output.put("ENDOFTHREAD");

} catch (IOException | InterruptedException e) {

e.printStackTrace();

}

}

}

WordCount

|--CleanImport.java

|--Split.java

|--MapOrder.java

|--ReduceCommandsPreparation.java

|--ReduceOrder.java

|--ResultMerge.java

|-- WordCount.java

# WordCount.java

package wordcount;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.HashSet;

import java.util.List;

import main.Config;

import main.Utils;

import network.CheckMachinesUp;

import network.NetworkConfig;

import wordcount.ResultMerge.ReduceResult;

public class WordCount {

private Config config;

private NetworkConfig networkConfig;

private ArrayList<String> input\_content;

private ArrayList<String> sx\_list;

private HashMap<String, ArrayList<String>> umx\_keys;

private HashMap<String, HashSet<String>> key\_umxs;

private ArrayList<List<String>> machine\_command;

private ArrayList<ReduceResult> rmx\_final;

public WordCount(Config config, NetworkConfig networkConfig) {

this.config = config;

this.networkConfig = networkConfig;

}

public void computeWordCount() throws IOException, InterruptedException {

// Initialize processing times

long startTime = System.currentTimeMillis();

long startStepTime;

long totalTime;

// Check responding machines

startStepTime = System.currentTimeMillis();

CheckMachinesUp checkMachines = new CheckMachinesUp(this.networkConfig);

checkMachines.readMachinesToTest(this.config.machinesToTestPath);

checkMachines.test\_Machines\_Up();

checkMachines.writeRespondingMachines(this.config.machinesRespondingPath);

ArrayList<String> liste\_machines\_ok = checkMachines.get\_Machines\_Up();

Utils.printDiffTime(startStepTime);

// Import and clean file

startStepTime = System.currentTimeMillis();

this.input\_content = CleanImport.cleanImport(this.config.inputFilePath);

System.out.println("---Time: " + (System.currentTimeMillis() - startStepTime) + " ---");

// Split file (parts to be processed to slaves)

startStepTime = System.currentTimeMillis();

this.sx\_list = Split.split(this.config.folderSx, this.input\_content, this.config.linesPerSplit);

Utils.printDiffTime(startStepTime);

// Send map order to slaves

startStepTime = System.currentTimeMillis();

MapOrder mapOrder = new MapOrder(this.sx\_list, liste\_machines\_ok, this.networkConfig);

mapOrder.setSlaveLocation(this.config.slaveJarLocation);

this.umx\_keys = mapOrder.send();

Utils.printDiffTime(startStepTime);

// Shuffling: reverse index and prepare job dispatch

startStepTime = System.currentTimeMillis();

this.key\_umxs = ReverseIndex.reverseIndex(umx\_keys);

this.machine\_command = ReduceCommandsPreparation.prepare\_job\_dispatch(this.key\_umxs, liste\_machines\_ok);

Utils.printDiffTime(startStepTime);

// Send reduce order to slaves

startStepTime = System.currentTimeMillis();

ReduceOrder reduceOrder = new ReduceOrder(this.networkConfig,this.machine\_command, liste\_machines\_ok);

reduceOrder.setSlaveLocation(this.config.slaveJarLocation);

this.rmx\_final = reduceOrder.send(this.networkConfig.timeout, this.networkConfig.maxThreadsPerMachine);

Utils.printDiffTime(startStepTime);

startStepTime = System.currentTimeMillis();

ResultMerge result = new ResultMerge(rmx\_final);

result.writeResult(this.config.folderResult);

result.set\_filtered\_words(config.filteredWords);

System.out.println("\nResult: \n" + result.get\_rmx\_ordered().toString());

System.out.println("\nFiltered result: \n" + result.getFilteredResults().toString());

Utils.printDiffTime(startStepTime);

totalTime = System.currentTimeMillis() - startTime;

System.out.println("---TOTAL TIME: " + totalTime + " ---");

}

}

# CleanImport.java

package wordcount;

import java.io.IOException;

import java.nio.charset.Charset;

import java.nio.file.Files;

import java.nio.file.Path;

import java.util.ArrayList;

import java.util.List;

import main.Utils;

public class CleanImport {

public static ArrayList<String> cleanImport(Path inputPath) throws IOException {

Utils.printBeautiful("Clean import");

List<String> lignes = new ArrayList<String>();

ArrayList<String> lignes\_clean = new ArrayList<String>();

lignes = Files.readAllLines(inputPath, Charset.forName("UTF-8"));

for (String ligne : lignes) {

// get rid of empty lines

if (ligne.length() != 0) {

// get rid of special characters

ligne = ligne.replaceAll("[^\\p{L}\\p{Z}]", "");

ligne = ligne.trim();

ligne = ligne.toLowerCase();

if (ligne.length() != 0) {

lignes\_clean.add(ligne);

}

}

}

return lignes\_clean;

}

}

# Split.java

package wordcount;

import java.io.IOException;

import java.nio.charset.Charset;

import java.nio.file.Files;

import java.nio.file.Path;

import java.util.ArrayList;

import main.Utils;

public class Split {

public static ArrayList<String> split(Path sxFolder, ArrayList<String> inputContent, Integer linesPerSplit) throws IOException{

Utils.printBeautiful("SPLIT");

System.out.println("File to send contains "+inputContent.size()+" lines.");

ArrayList<String> sxList = new ArrayList<String>();

Integer i = 0;

ArrayList<String> bloc = new ArrayList<String>();

for (int k=0; k< inputContent.size(); k++){

String ligne = inputContent.get(k);

// System.out.println(ligne);

bloc.add(ligne);

if ( ((k+1) % linesPerSplit) == 0 || k==inputContent.size()-1){

// line blocks

Path sxi = sxFolder.resolve("S"+i);

Files.write(sxi, bloc, Charset.forName("UTF-8"));

sxList.add("S"+i);

i += 1;

bloc = new ArrayList<String>();

}

}

System.out.println("Created "+sxList.size()+" splits to send.");

return sxList;

}

}

# MapOrder.java

package wordcount;

import java.nio.file.Path;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

import network.NetworkConfig;

import network.SshCommand;

import main.Utils;

public class MapOrder {

private ArrayList<String> splitsToSend;

private ArrayList<String> machineList;

private Path slaveJarLocation;

private NetworkConfig networkConfig;

public MapOrder(ArrayList<String> splitsToSend, ArrayList<String> machineList,NetworkConfig networkConfig ){

this.splitsToSend= splitsToSend;

this.machineList = machineList;

this.networkConfig = networkConfig;

}

public void setSlaveLocation(Path slaveJarLocation){

this.slaveJarLocation = slaveJarLocation;

}

public HashMap<String, ArrayList<String>> send(){

Utils.printBeautiful("Map");

/\* cette méthode créé le dictionnaire Umx-machines

// idéalement on réalise un scp pour envoyer les fichiers (ici on triche)

// Initialisation de notre dictionnaire (qui trace ce que l'on a envoyé)

\*/

Integer maxThreads = this.machineList.size()\*this.networkConfig.maxThreadsPerMachine;

HashMap<String,String> umx\_machine = new HashMap<String,String>();

HashMap<String, ArrayList<String>> machine\_keys = new HashMap<String, ArrayList<String>>();

HashMap<String, ArrayList<String>> umx\_keys = new HashMap<String, ArrayList<String>>();

while (! this.splitsToSend.isEmpty()) {

Integer limit = Math.min(maxThreads, this.splitsToSend.size());

// slave-id dict pour savoir de qui on reçoit les réponses

HashMap<SshCommand,Integer> slaves\_dict = new HashMap<SshCommand,Integer>();

for (int k = 0; k < limit; k++) {

String split = this.splitsToSend.get(k);

Integer id = Utils.getId(split);

String machine = machineList.get(k % this.machineList.size());

umx\_machine.put("Um"+id, machine);

// System.out.println("Send "+split+" to machine "+machine+" => Um"+id);

String command = "cd "+this.slaveJarLocation+";java -jar SLAVE.jar modeSXUMX S"+id;

SshCommand slave = new SshCommand(this.networkConfig, machine, command, false);

slave.start();

slaves\_dict.put(slave,id);

}

for (Map.Entry<SshCommand,Integer> entry:slaves\_dict.entrySet()) {

try {

SshCommand slave = entry.getKey();

Integer id = entry.getValue();

slave.join();

// on attend que le thread soit terminé pour ajouter au dictionnaire

machine\_keys.put(slave.getMachine(), slave.get\_response());

umx\_keys.put("Um"+id, slave.get\_response());

// on supprime de la liste à traiter

this.splitsToSend.remove("S"+id);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

System.out.println("Received "+umx\_keys.size()+" responses.");

return umx\_keys;

}

}

# Split.java

package wordcount;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Map;

public class ReverseIndex {

public static HashMap<String, HashSet<String>> reverseIndex(HashMap<String, ArrayList<String>> umx\_keys){

// INPUT Umx - [keys]

// OUTPUT key - [Umxs]

HashMap<String,HashSet<String>> inversed = new HashMap<String,HashSet<String>>(umx\_keys.size());

for(Map.Entry<String, ArrayList<String>> entry : umx\_keys.entrySet()) {

for(String key : entry.getValue()) { // entry.getValue() est ArrayList<String> (keys)

if(!inversed.containsKey(key)) {

inversed.put(key,new HashSet<String>());

}

inversed.get(key).add(entry.getKey());

}

}

return inversed;

}

}

# ReduceCommandsPreparation.java

package wordcount;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.HashMap;

import java.util.HashSet;

import java.util.List;

import java.util.Map;

import main.Utils;

public class ReduceCommandsPreparation {

public static ArrayList<List<String>> prepare\_job\_dispatch(HashMap<String, HashSet<String>> key\_umxs ,ArrayList<String> machine\_list){

Utils.printBeautiful("Reduce commands preparation");

// à partir du dict machine\_keys on va déterminer à qui on va envoyer quels jobs

// le dictionnaire machine => commande est enregistré en output

// version simple et débile, itération sur les clés

// TODO algo plus intelligent à faire

ArrayList<List<String>> machine\_command = new ArrayList<List<String>>();

Integer i = 0;

Integer nbr\_mach = machine\_list.size();

for (Map.Entry<String, HashSet<String>> key\_values: key\_umxs.entrySet()){

String machine = machine\_list.get(i % nbr\_mach);

String key = key\_values.getKey();

HashSet<String> umxs = key\_values.getValue();

String umx\_concat = "";

for (String umx: umxs){

umx\_concat += " "+umx; // attention, commence par un " "

}

String command = "modeUMXSMX "+key+" SM"+i+umx\_concat;

List<String> temp =Arrays.asList(machine,command) ;

machine\_command.add(temp);

i += 1;

}

System.out.println("Created "+machine\_command.size()+" commands to send.");

return machine\_command;

}

}

# ReduceOrder.java

package wordcount;

import java.nio.file.Path;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import main.Utils;

import network.NetworkConfig;

import network.SshCommand;

import wordcount.ResultMerge.ReduceResult;

public class ReduceOrder {

private ArrayList<List<String>> machine\_command;

private Path slaveJarLocation;

private ArrayList<String> machineList;

private ArrayList<ReduceResult> rmx\_final;

private NetworkConfig networkConfig;

public ReduceOrder(NetworkConfig networkConfig,ArrayList<List<String>> machine\_command, ArrayList<String> machineList){

this.machine\_command = machine\_command;

this.machineList = machineList;

this.networkConfig = networkConfig;

}

public void setSlaveLocation(Path slaveJarLocation){

this.slaveJarLocation = slaveJarLocation;

}

public ArrayList<ReduceResult> send(Integer timeout, Integer maxThreadPerMachine){

Utils.printBeautiful("Reduce");

Integer max\_threads = this.machineList.size()\*maxThreadPerMachine;

ArrayList<List<String>> machine\_command\_to\_compute = this.machine\_command;

ArrayList<List<String>> machine\_command\_failed = new ArrayList<List<String>>();

this.rmx\_final = new ArrayList<ReduceResult>();

while (! machine\_command\_to\_compute.isEmpty()) {

HashMap<SshCommand,List<String>> slaves\_dict = new HashMap<SshCommand,List<String>>();

Integer limit = Math.min(max\_threads, machine\_command\_to\_compute.size());

for (List<String> entry: machine\_command\_to\_compute) {

// System.out.println("Command "+ entry.get(1)+" to machine "+entry.get(0)+".");

SshCommand slave = new SshCommand(

this.networkConfig,

entry.get(0),

"cd "+this.slaveJarLocation+";java -jar SLAVE.jar "+entry.get(1),

false);

slave.start();

slaves\_dict.put(slave,entry);

limit -= 1;

if (limit.equals(0)){

break;

}

}

for (Map.Entry<SshCommand,List<String>> entry:slaves\_dict.entrySet()) {

try {

SshCommand slave = entry.getKey();

slave.join();

// wait for response

if (slave.get\_response().size()!=0){

ReduceResult result = new ReduceResult(slave.get\_response().get(0));

this.rmx\_final.add(result);

machine\_command\_to\_compute.remove(entry.getValue());

}

else {

List<String> mach\_comm = Arrays.asList(slave.getMachine(),slave.get\_command());

machine\_command\_failed.add(mach\_comm);

System.out.println(

"-------\nERROR:"

+ " machine "+slave.getMachine()

+" commande: "+slave.get\_command()

+"\n=> New try"

+"\n------");

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

System.out.println("Received "+rmx\_final.size()+" elements.");

return this.rmx\_final;

}

}

# ResultMerge.java

package wordcount;

import java.io.IOException;

import java.nio.charset.Charset;

import java.nio.file.Files;

import java.nio.file.Path;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import main.Utils;

public class ResultMerge {

private ArrayList<ReduceResult> rmxs;

private List<String> filtered\_words;

public ResultMerge(ArrayList<ReduceResult> rmxs){

this.rmxs = rmxs;

}

public void writeResult(Path folderResult) throws IOException{

Utils.printBeautiful("Results");

Path output = folderResult.resolve("Results.txt");

ArrayList<String> converted = new ArrayList<String>();

for (ReduceResult resultat :this.rmxs){

converted.add(resultat.toString());

}

Files.write(output, converted, Charset.forName("UTF-8"));

}

public ArrayList<ReduceResult> get\_rmx\_ordered(){

Collections.sort(this.rmxs, result\_comp);

return this.rmxs;

}

public void set\_filtered\_words(List<String> filtered\_words){

this.filtered\_words = filtered\_words;

}

public ArrayList<ReduceResult> getFilteredResults(){

ArrayList<ReduceResult> filtered\_result = new ArrayList<ReduceResult>();

for (ReduceResult entry: this.rmxs){

if (! this.filtered\_words.contains(entry.key)){

filtered\_result.add(entry);

}

}

return filtered\_result;

};

public static class ReduceResult {

private String key;

private Integer value;

public ReduceResult(String rawResult) {

Pattern r = Pattern.compile("(\\D+) (\\d+)");

Matcher m = r.matcher(rawResult);

m.find();

this.key = m.group(1);

this.value = Integer.parseInt(m.group(2));

}

public String toString(){

return this.key +" "+ this.value.toString();

};

public String getKey() { return this.key; }

public Integer getValue() { return this.value; }

}

Comparator<ReduceResult> result\_comp = new Comparator<ReduceResult>() {

@Override

public int compare(ReduceResult a, ReduceResult b) {

return b.getValue().compareTo(a.getValue());

}

};

}

# Mapping.java

package operations;

import java.io.IOException;

import java.nio.charset.Charset;

import java.nio.file.Files;

import java.nio.file.Path;

import java.util.ArrayList;

import java.util.List;

import main.Config;

import main.Utils;

public class Mapping {

private Config config;

private String sxName;

public Mapping(String sxName, Config config){

this.config = config;

this.sxName = sxName;

}

public void launchMapping(){

Path sx\_input = config.folderSx.resolve(this.sxName);

Integer id = Utils.getId(sxName);

Path umx\_output = config.folderUmx.resolve("Um"+id);

List<String> lines;

try {

lines = Files.readAllLines(sx\_input, Charset.forName("UTF-8"));

ArrayList<String> Umx\_write = new ArrayList<String>();

for (String ligne: lines ){

// words are separated by spaces

String[] words = ligne.split(" ");

for (String word: words){

if (word.length()>0){

Umx\_write.add(word + " 1");

System.out.println(word);

}

}

}

Files.write(umx\_output, Umx\_write, Charset.forName("UTF-8"));

} catch (IOException e1) {

e1.printStackTrace();

}

}

}

# Reducing.java

package operations;

import java.io.IOException;

import java.nio.charset.Charset;

import java.nio.file.Files;

import java.nio.file.Path;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.HashSet;

import java.util.List;

import main.Config;

import main.Utils;

public class Reducing {

private Config config;

private String[] args;

public Reducing(String[] args, Config config){

this.config = config;

this.args = args;

}

public void launchReduce(){

String key = this.args[1];

String sm\_name = this.args[2];

Integer id = Utils.getId(sm\_name);

HashSet<String> um\_names = new HashSet<String>();

for(int i=3; i <= args.length -1; i++) {

um\_names.add(args[i]);

}

List<String> allLignes = new ArrayList<String>();

try {

for (String um\_name:um\_names){

Path Umx\_input = config.folderUmx.resolve(um\_name);

List<String> lignes;

lignes = Files.readAllLines(Umx\_input, Charset.forName("UTF-8"));

allLignes.addAll(lignes);

}

ArrayList<String> smx\_to\_write = new ArrayList<String>();

for (String ligne: allLignes ){

// format: word 1

String word = ligne.split(" ")[0];

// append to list if = key

if (word.toLowerCase().equals(key.toLowerCase())){

smx\_to\_write.add(word+" 1");

}

}

Path smx\_output = config.folderSmx.resolve(sm\_name);

Files.write(smx\_output, smx\_to\_write, Charset.forName("UTF-8"));

Path rmx\_output = config.folderRmx.resolve("Rm"+id);

String rmx = key+" "+smx\_to\_write.size();

Files.write(rmx\_output, Arrays.asList(rmx), Charset.forName("UTF-8"));

System.out.println(rmx);

} catch (IOException e1) {

e1.printStackTrace();

}

}

}