TP SYSTÈMES DISTRIBUÉS POUR LE BIG DATA

A. Première étapes

1. Faire un programme séquentiel non parallélisé qui compte l'occurrence des mots dans un fichier.

```
public static TreeMap<String, Integer> get_word_count(String filename) throws FileNotFoundException {
    Scanner input = new Scanner(new File(filename));
    Set<String> stopWordsSet = new HashSet<String>();
    Collections.addAll(stopWordsSet, stop_words);
    TreeMap<String, Integer> wordCounts = new TreeMap<String, Integer>();
    while (input.hasNext()) {
        String next = input.next().toLowerCase()
                .replace( target: ".",  replacement: "")
                .replace( target: ",",  replacement: "")
                .replace( target: "0",  replacement: "")
                .replace( target: "1",  replacement: "")
                .replace( target: "2",  replacement: "")
                .replace( target: "3",  replacement: "")
                .replace( target: "4",  replacement: "")
                .replace( target: "5",  replacement: "")
                .replace( target: "6",  replacement: "")
                .replace( target: "7",  replacement: "")
                .replace( target: "8",  replacement: "")
                .replace( target: "9",  replacement: "")
                .replace( target: "'",  replacement: " ");
        if(!stopWordsSet.contains(next))
        {
            if (!wordCounts.containsKey(next)) {
                wordCounts.put(next, 1);
            } else {
                wordCounts.put(next, wordCounts.get(next) + 1);
            }
        }
    }
    return wordCounts;
```

2. Trier le nombre d'occurrences.

```
public static TreeMap<String,Integer> get_occurences_sorted(TreeMap<String, Integer> wordCounts) throws FileNotFoundException {
    ValueComparator bvc = new ValueComparator(wordCounts);
    TreeMap<String,Integer> sorted_map = new TreeMap<String,Integer>(bvc);
    sorted_map.putAll(wordCounts);
    return sorted_map;
}
```

```
import java.util.Comparator;
import java.util.Map;

class ValueComparator implements Comparator<String> {

    Map<String, Integer> base;

    public ValueComparator(Map<String, Integer> base) { this.base = base; }

    public int compare(String a, String b) {
        if (base.get(a) >= base.get(b)) {
            return -1;
        } else {
            return 1;
        }
    }
}
```

3. Deuxième tri alphabétique pur.

```java
public static void show_alphabetic_sort(TreeMap<String, Integer> wordCounts) throws FileNotFoundException {
    Map<String, Integer> map = new TreeMap<String, Integer>(wordCounts);
    Set set = map.entrySet();
    Iterator iterator = set.iterator();
    while(iterator.hasNext()) {
        Map.Entry me = (Map.Entry) iterator.next();
        System.out.print(me.getKey() + "\t" + me.getValue() + "\n");
    }
}
```

4. Test du programme séquentiel sur le code forestier de Mayotte.

Le programme a bien fonctionné du premier coup.

5. Repérer les erreurs séquentiel.

Cette étape a déjà été affichée dans l'étape 1 notamment avec les '.replace'.
Les stopwords utilisés pour les différents fichiers étudiés après sont:

```java
public static String[] stop_words = {"%", "à", ":", "–", "le", "la", "de", "des", "et", "les",
        "ou", "ne", "que", "qui", ";", "a", "ii", "°", "au", "aux", "du", "en", "où", "se", "sil",
        "y", "v", "sy", "en", "il", "est", "°", "", "”", "”v", "€", "", "r", "r–", "un", "une",
        "pour", "pas", "par", "l", "l–", "ce", "ces", "ainsi", "sa", "dans", "avec", "son", "sont",
        "sur", "leur", "si", "ses", "sous", "iii", "ier", "elle", "tout", "toute"};
```

6. Les 50 mots les plus fréquents.

```java
public static TreeMap<String,Integer> get_top_50(TreeMap<String,Integer> sorted_map, TreeMap<String, Integer> wordCounts) {
    ValueComparator bvc = new ValueComparator(wordCounts);
    TreeMap<String, Integer> myNewMap = sorted_map.entrySet().stream()
            .limit(50)
            .collect(TreeMap::new, (m, e) -> m.put(e.getKey(), e.getValue()), Map::putAll);
    TreeMap<String, Integer> top_50_sorted_map = new TreeMap<String, Integer>(bvc);
    top_50_sorted_map.putAll(myNewMap);
    return top_50_sorted_map;
}
```

7. On filtre les pronoms et les conjonctions.

On a simplement ajouté les pronoms et les conjonctions dans nos stopwords.

8. On filtre les mots bizarres.

Pareil que pour la question précédente, on ajoute les mots dans notre liste de stopwords.

9. Les 50 mots du code de la déontologie de la police nationale.

Même processus que pour les premières données.

10. Les 50 mots du code du domaine public fluvial.

Même processus.

11. Les 50 mots du code de la santé publique.

Même processus

12. Chronométrage du programme séquentiel.

Pour chronométrer du code sous JAVA:

```
long startTime_allWork = System.currentTimeMillis();
TreeMap<String, Integer> wordCounts = get_word_count( filename: "data/sante_public.txt");
System.out.println(wordCounts);
long endTime_allWork = System.currentTimeMillis();
long totalTime_allWork = endTime_allWork - startTime_allWork;
System.out.println(totalTime_allWork);
```

13. Travail sur des plus gros fichiers.

On charge le fichier en question (400MO) et nous exécutons notre programme séquentiel pour avoir un wordcount simple avec nettoyage.
Le temps d'exécution est de: 3.5 minutes 2.9

B. Seconde étape

14. Nombre de processeurs.

Sur mac, pour connaître le nombre de processeurs il faut exécuter:
$ sysctl -n hw.ncpu

15. Nombre de coeur de calcul.

Sur mac, pour connaître le nombre de coeurs de calculs, il faut exécuter:
§

16. Parallélisation du calcul.

Pour paralléliser notre calcul on va utiliser des threads.
Dans notre cas, on en a utilisé 3. Voici comment nous avons implémenté notre code.

À l'intérieur du main de notre classe Main:

```
// THREADS ! //
long startTime_allWork = System.currentTimeMillis();
Scanner input = new Scanner(new File( pathname: "data/CC-MAIN-20170322212949-00140-ip-10-233-31-227.ec2.internal.warc.wet"));
ConcurrentHashMap<String, Integer> wordCounts = new ConcurrentHashMap<String, Integer>();


Thread t1 = new Share(wordCounts, input);
Thread t2 = new Share(wordCounts, input);
Thread t3 = new Share(wordCounts, input);

t1.run();
t2.run();
t3.run();

try {
    t1.join();
} catch (InterruptedException e) {
    e.printStackTrace();
}
try {
    t2.join();
} catch (InterruptedException e) {
    e.printStackTrace();
}
try {
    t3.join();
} catch (InterruptedException e) {
    e.printStackTrace();
}

System.out.println(wordCounts);
long endTime_allWork = System.currentTimeMillis();
long totalTime_allWork = endTime_allWork - startTime_allWork;
System.out.println("\n\n" + totalTime_allWork + "ms");
```

Notre classe Thread qu'on a appelé Share:

```
public class Share extends Thread {

    public static String[] stop_words = {"»", "à", ":", "–", "le", "la", "de", "des", "et", "les",
            "ou", "ne", "que", "qui", ",", "a", "ii", "°", "au", "aux", "du", "en", "où", "se", "sil",
            "y", "v", "sy", "en", "il", "est", "°°", "", "°°", "°v", "€", "", "r", "r–", "un", "une",
            "pour", "pas", "par", "l", "l–", "ce", "ces", "ainsi", "sa", "dans", "avec", "son", "sont",
            "sur", "leur", "si", "ses", "sous", "iii", "ier", "elle", "tout", "toute"};

    public Scanner input = null;
    public ConcurrentHashMap wordcount = null;

    public Share(ConcurrentHashMap<String, Integer> wordcount, Scanner input) {
        this.wordcount = wordcount;
        this.input = input;
    }

    public void run() { this.find(); }

    public void find() {
        Set<String> stopWordsSet = new HashSet<>();
        Collections.addAll(stopWordsSet, stop_words);
        while (input.hasNext()) {
            String next = input.next().toLowerCase()
                    .replace( target: ".",  replacement: "")
                    .replace( target: ",",  replacement: "")
                    .replace( target: "0",  replacement: "")
                    .replace( target: "1",  replacement: "")
                    .replace( target: "2",  replacement: "")
                    .replace( target: "3",  replacement: "")
                    .replace( target: "4",  replacement: "")
                    .replace( target: "5",  replacement: "")
                    .replace( target: "6",  replacement: "")
                    .replace( target: "7",  replacement: "")
                    .replace( target: "8",  replacement: "")
                    .replace( target: "9",  replacement: "")
                    .replace( target: "'",  replacement: " ");
            if (!stopWordsSet.contains(next)) {
                if (!this.wordcount.containsKey(next)) {
                    this.wordcount.put(next, 1);
                } else {
                    this.wordcount.put(next, (Integer) this.wordcount.get(next) + 1);
                }
            }
        }
    }
```

17. Chronométrage du programme parallèle.

Après parallélisation de notre programme, le temps d'exécution pour un wordcount sur le fichier de 400 MO passe de 3.5 minutes à 2.9 minutes.

C. Troisième étape

18. Nom court, nom long.

Nom court: $ hostname
Nom long: $ hostname -f

19. Adresse IP.

Connaître adresse IP: $ ifconfig

20. Du nom vers l'IP.

Entrer: $ nslookup <ip.long> C133-10.enst.fr

21. De l'IP vers le nom.

Entrer: $ dig

22. Ping pong à l'intérieur.

Entrer: $ ping google.com

23. Ping pong à l'extérieur.

Pour se connecter depuis notre ordinateur personnel à une machine extérieur, il faut commencer par générer une clé:
$ ssh-keygen
Cette commande va créer deux fichier. Le premier (~/.ssh/id_rsa) est la clé privée, qu'il ne faut ni toucher, ni communiquer. Le second (~/.ssh/id_rsa.pub) est la clé publique qu'il faut copier sur la machine extérieur en utilisant la commande:
$ ssh-copy-id -i id_rsa.pub usernameTelecomParisTech@machineDeLecoleAllumee
Désormais plus besoin de spécifier le mot de passe à chaque connection.
Pour se connecter il faut exécuter la commande:
$ ssh 'tailleur@c130-30.enst.fr'
Une fois connecté on peut faire un ping de la même façon que dans la question précédente.

24. Calculer en ligne de commande sur l'ordinateur local.

Exécuter: $ calc 2+3

25. Calculer en ligne de commande sur un ordinateur distant.

Exécuter: $ ssh ‘tailleur@c130-30.enst.fr’ calc 2+3

26. Calculer à distance sans mot de passe.

Grâce à la manipulation effectuée lors de la question 23, plus besoin de taper de mot de passe. La réponse à cette question est donc la même que celle donnée à la question 23.

D. Quatrième étape

27. Chemin absolu.

Exécuter: $ pwd
Le chemin absolu de mon projet est: ‘/Users/clementtailleur/Documents/Telecom/Distributed_Systems’

28. Un fichier dans le répertoire personnel.

Exécuter: $ nano ~/fperso.txt
Puis écrire “bonjour” et ctrl+x

29. Où se trouve le fichier dans le répertoire personnel ?

On peut utiliser la commande ‘df’ pour savoir où est stocké physiquement notre fichier.

30. Un dossier et un fichier dans le répertoire personnel.

Exécuter: $ mkdir /tmp/tailleur
Puis: $ nano /tmp/tailleur/ftemp.txt
Grâce à la commande ‘df’, on sait que le fichier est stocké sur le disque local.

31. Trois ordinateurs A, B et C. On commence avec A. Utilisation du serveur NFS.

Exécuter: $ nano ~/text.txt
Puis écrire ‘mon texte sur NFS’ et quitter avec ctrl+x

32. Trois ordinateurs A, B et C. On continue sur B et sur C. Utilisation du serveur NFS.

Après connexion sur une machine B puis C, on retrouve bien dans le répertoire personnel, le fichier text.txt créé dans la machine A et son contenu ‘mon texte sur NFS’.

33. Trois ordinateurs A, B et C. On commence avec A. Utilisation des disques locaux.

On commence par créer un répertoire tailleur dans /tmp/. On y ajoute alors un fichier local.txt qui contient le texte ‘mon texte sur disque local’.
Exécuter: $ mkdir /tmp/tailleur
Puis: $ nano /tmp/tailleur/local.txt
Et écrire ‘mon texte sur disque local’ avant de quitter avec ctrl+x.

34. Trois ordinateurs A, B et C. On commence sur B et sur C. Utilisation des disques locaux.

Comme prévu, le dossier et le fichier créés dans la question précédente ne sont pas accessible depuis les machines B et C.

35. Depuis A, copier A vers B avec les disques locaux.

Pour copier depuis la machine A le fichier local.txt vers la machine B on se connecte sur A puis on exécute la commande suivante en utilisant scp.
Exécuter: $ scp -r -p /tmp/tailleur/local.txt tailleur@c130-28:/tmp/tailleur/local.txt

36. Depuis A, copier B vers C avec les disques locaux.

De même, pour copier le contenu de B vers C en étant connecté sur A il faut exécuter la commande suivante:
$ scp -r -p tailleur@c130-30.enst.fr:tmp/tailleur/local.txt tailleur@c130-28.enst.fr:/tmp/tailleur/

E. Cinquième étape

37. Faire un programme JAVA nommé SLAVE qui calcule 3+5.