

Travaux pratiques: implémentation de Hadoop MapReduce “from scratch” en Java.

Etape 1: faire un programme séquentiel non parallélisé qui compte le nombre d'occurrences des mots dans un fichier.

1. Premier comptage en séquentiel pur

Implémentez un logiciel en java qui compte le nombre d'occurrences des mots d'un fichier d'entrée de manière non parallélisée (monothread, une seul thread), en utilisant un seul processeur.

J'implémente une classe Wordcount (code disponible en annexe) qui permet le comptage des mots dans un fichier de texte donné en entrée. En vue des questions suivantes, la classe permet aussi de sélectionner en option le type de tri des mots (sans tri, tri séquentiel pur, tri séquentiel et alphabétique), ainsi que le type d'affichage des résultats (pas d'affichage, affichage de n mots, affichage de tous les mots). La classe implémente également le chronométrage des deux opérations (comptage et tri).

Quelle structure de donnée est la plus pertinente pour stocker les résultats: List, HashMap ou HashSet ou une autre ? Pour quelle raison ?

La structure la plus pertinente est le HashMap, car il permet d'organiser facilement les résultats du comptage en paires de clés (mots identifiés) et valeurs (nombre d'occurrences pour le mot concerné). La liste est possible mais elle complique la tâche car elle implique de créer une liste d'entrées de type `<String, Integer>`. Le HashSet n'est pas adapté car il n'admet pas les valeurs dupliquées, or il y aura clairement des mots dont le nombre d'occurrences sera identiques.

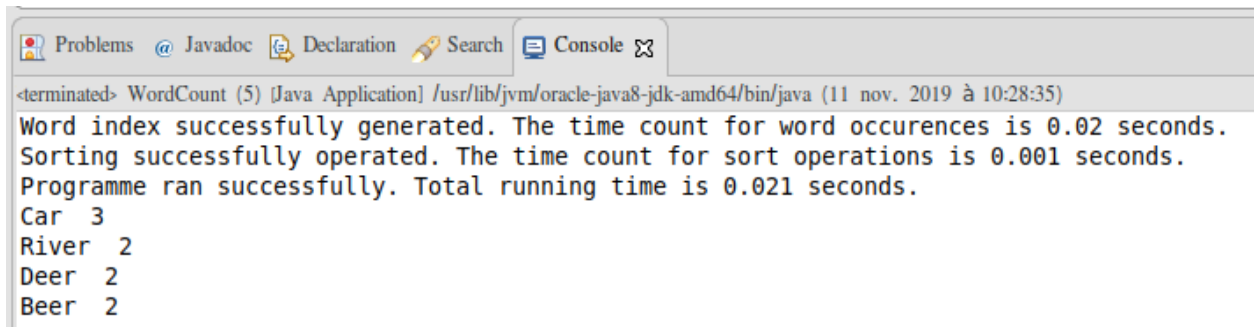
Testez votre programme avec un fichier d'entrée *input.txt* avec comme contenu:

```
Deer Beer River  
Car Car River  
Deer Car Beer
```

Résultat:

```
Deer 2  
Beer 2  
River 2  
Car 3
```

Le programme fonctionne ; j'utilise l'option d'absence de tri (ce qui explique le temps de tri proche de 0, aucun tri n'étant en réalité opéré).



```
<terminated> WordCount (5) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (11 nov. 2019 à 10:28:35)
Word index successfully generated. The time count for word occurrences is 0.02 seconds.
Sorting successfully operated. The time count for sort operations is 0.001 seconds.
Programme ran successfully. Total running time is 0.021 seconds.
Car 3
River 2
Deer 2
Beer 2
```

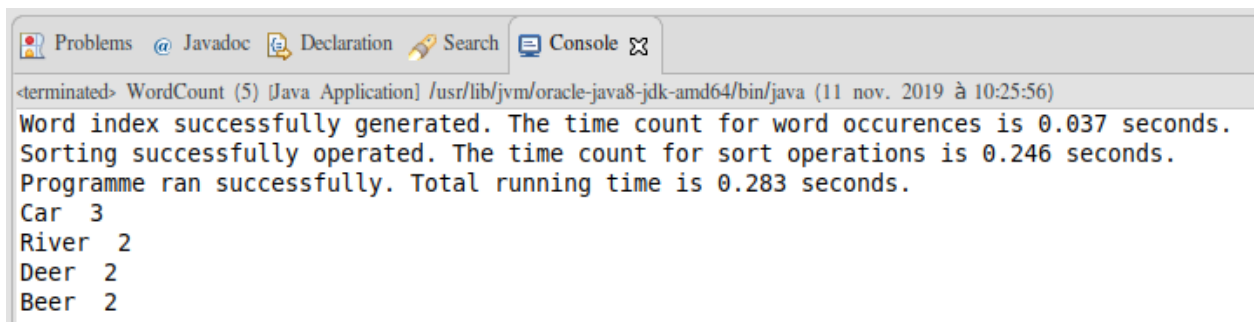
2. Premier tri en séquentiel pur

Modifiez votre programme pour trier par nombre d'occurrences:

Résultat:

Car 3
Deer 2
Beer 2
River 2

Le programme fonctionne ; j'utilise l'option de tri par occurrence.



```
<terminated> WordCount (5) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (11 nov. 2019 à 10:25:56)
Word index successfully generated. The time count for word occurrences is 0.037 seconds.
Sorting successfully operated. The time count for sort operations is 0.246 seconds.
Programme ran successfully. Total running time is 0.283 seconds.
Car 3
River 2
Deer 2
Beer 2
```

3. Deuxième tri alphabétique en séquentiel pur

Modifiez le programme pour trier alphabétiquement pour les mots à égalité du nombre d'occurrences:

Résultat:

Car 3
Beer 2
Deer 2
River 2

Le programme fonctionne ; j'utilise l'option de tri par occurrence puis alphabétique.

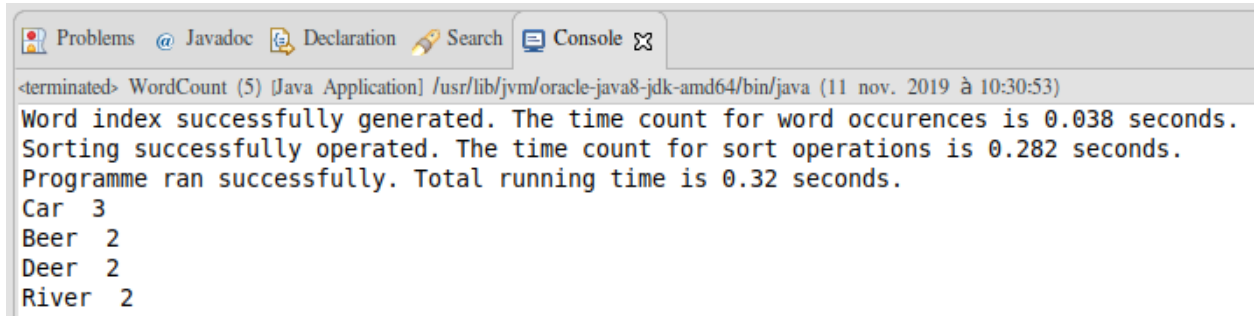
4. Test du programme séquentiel sur le code forestier de Mayotte

Testez ensuite votre programme avec le code forestier de Mayotte disponible sur github *forestier_mayotte.txt* :

<https://github.com/legifrance/Les-codes-en-vigueur>

Votre programme a-t-il fonctionné du premier coup ?

Le programme fonctionne ; j'utilise l'option de tri par occurrence puis alphabétique.



```
<terminated> WordCount (5) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (11 nov. 2019 à 10:30:53)
Word index successfully generated. The time count for word occurrences is 0.038 seconds.
Sorting successfully operated. The time count for sort operations is 0.282 seconds.
Programme ran successfully. Total running time is 0.32 seconds.
Car 3
Beer 2
Deer 2
River 2
```

Vérifiez en ouvrant le fichier texte qu'il contient bien du texte et non du code HTML.

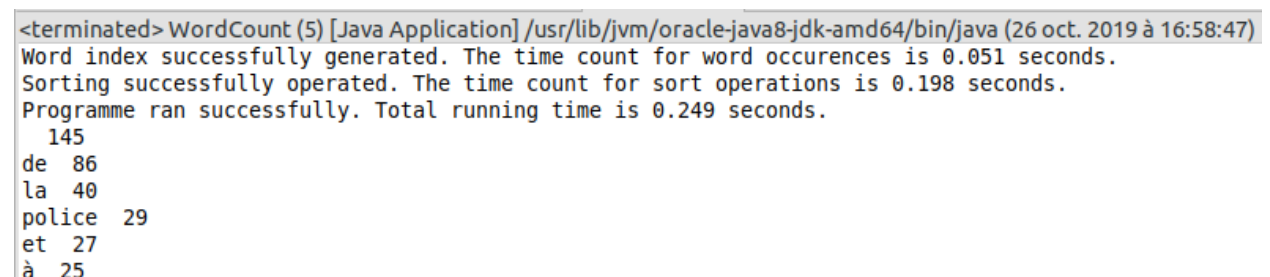
Ne perdez pas de temps à corriger les éventuelles erreurs dues aux caractères spéciaux ou à des mots suspects ou illisibles (de toutes façons par la suite il y aura du chinois dans le texte).

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

Testez votre programme avec le code de déontologie de la police nationale disponible sur github *deontologie_police_nationale.txt* : <https://github.com/legifrance/Les-codes-en-vigueur>

De même ne perdez pas de temps à filtrer les caractères spéciaux ou autres mots bizarres.

Pourquoi ? Car nous travaillerons ensuite sur des textes en chinois, japonais, arabe et d'autres langues. Si vous implémentez une étape de filtrage ici en français elle ne servira à rien par la suite. Quels sont les 5 premiers mots (qui ressemblent à des mots) parmi les 50 premiers de la liste triée résultat ? Gardez la réponse pour l'intégrer au rapport.



```
<terminated> WordCount (5) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (26 oct. 2019 à 16:58:47)
Word index successfully generated. The time count for word occurrences is 0.051 seconds.
Sorting successfully operated. The time count for sort operations is 0.198 seconds.
Programme ran successfully. Total running time is 0.249 seconds.
145
de 86
la 40
police 29
et 27
à 25
```

Les cinq premiers mots qui ressemblent à des mots sont donc : « de », « la », « police », « et », « à ».

6. Les 50 mots du code du domaine public fluvial

Testez votre programme avec le code du domaine public fluvial *domaine_public_fluvial.txt*.

Quels sont les 5 premiers mots (qui ressemblent à des mots) parmi les 50 premiers de la liste triée résultat ? Gardez la réponse pour l'intégrer au rapport.

```
<terminated> WordCount (5) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (26 oct. 2019 à 17:06:26)
Word index successfully generated. The time count for word occurrences is 0.23 seconds.
Sorting successfully operated. The time count for sort operations is 0.242 seconds.
Programme ran successfully. Total running time is 0.472 seconds.
de 621
 585
le 373
du 347
la 330
et 266
```

Les cinq premiers mots qui ressemblent à des mots sont donc : « de », « le », « du », « la », « et ».

7. Les 50 mots du code de la santé publique

Testez votre programme avec le code de la santé publique *sante_publique.txt*.

Quels sont les 5 premiers mots (qui ressemblent à des mots) parmi les 50 premiers de la liste triée résultat ? Gardez la réponse pour l'intégrer au rapport.

```
<terminated> WordCount (5) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (26 oct. 2019 à 17:08:50)
Word index successfully generated. The time count for word occurrences is 2.2 seconds.
Sorting successfully operated. The time count for sort operations is 0.294 seconds.
Programme ran successfully. Total running time is 2.494 seconds.
de 189699
 110806
la 74433
des 66705
à 65462
et 60940
```

Les cinq premiers mots qui ressemblent à des mots sont donc : « de », « la », « des », « à », « et ».

8. Chronométrage du programme séquentiel

Chronométrer votre programme sur le code de la santé publique.

Chronométrage possible avec:

```
long startTime = System.currentTimeMillis();
```

```
...
```

```
long endTime = System.currentTimeMillis();
```

```
long totalTime = endTime - startTime;
```

Combien de temps faut-il pour chacune des étapes:

- Compter le nombre d'occurrences
- Tri (par nombre d'occurrences et alphabétique)

Gardez la réponse pour l'intégrer au rapport.

Comme le montre la capture d'écran de l'étape 1.7, le temps nécessaire pour compter le nombre d'occurrences est de 2,2 secondes environ. Le temps pour opérer le tri est de 0,294 secondes environ.

9. Travailler sur des plus gros fichiers

Testez votre programme sur un cas réel: un extrait de toutes les pages internet transformées au format texte brut (format WET). Toutes les pages sur internet au format texte sont disponibles sur <http://commoncrawl.org/the-data/get-started/> : chaque mois, environ 3 milliards de pages web, soit 250 To de données sont stockées. Ces données sont disponibles par tranche de moins d'1Go environ, vous travaillerez sur une tranche de 380Mo.

J'ai choisi une tranche en particulier pour avoir une comparaison entre nous (vous pouvez tester sur d'autres tranches si vous voulez). Téléchargez cette tranche ici:

<https://commoncrawl.s3.amazonaws.com/crawl-data/CC-MAIN-2017-13/segments/1490218189495.77/wet/CC-MAIN-20170322212949-00140-ip-10-233-31-227.ec2.internal.warc.wet.gz>

Décompressez et obtenez le fichier CC-MAIN-20170322212949-00140-ip-10-233-31-227.ec2.internal.warc.wet

Il s'agit d'une tranche contenant un ensemble de sites internet au format texte brut (WET).

Testez votre programme avec ce fichier en entrée. Chronométrez-le.

Gardez la réponse pour l'intégrer au rapport.

```
<terminated> WordCount (5) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (26 oct. 2019 à 17:44:24)
Word index successfully generated. The time count for word occurrences is 66.267 seconds.
Sorting successfully operated. The time count for sort operations is 24.477 seconds.
Programme ran successfully. Total running time is 90.744 seconds.
the 578397
to 434094
and 432126
- 379506
of 376300
a 374839
```

J'obtiens d'abord une erreur de type `java.lang.OutOfMemoryError`, réglée en augmentant la mémoire de la machine virtuelle Java avec les options `-Xms1024M` et `-Xmx2048M`. Le programme tourne ensuite correctement. On obtient alors un temps de comptage de 66,267 secondes, et un temps de tri de 24,477 secondes.

10. Préparez un document de travail qu'il faudra rendre à la fin de l'unité d'enseignement.

Créez un document dans lequel vous allez expliquer votre implémentation au fur et à mesure. Vous obtiendrez également par la suite (dans les prochaines étapes) les mesures de chronométrage et calculerez le speedup (l'accélération) et la portion de code parallèle (le taux de parallélisation) à chaque fois avec vos nouvelles mesures. Pour comparer deux mesures, il faudra bien faire attention d'avoir le même jeu de données en entrée et le même résultat en sortie... Vous devez prouver que vous obtenez le même résultat en sortie en utilisant un autre logiciel (qui fait de la recherche du nombre de mots) et piocherez au hasard des mots pour savoir si dans le cas séquentiel et dans le cas réparti vous avez les mêmes résultats.

Etape 2: travailler avec plusieurs ordinateurs en réseau.

1. Nom court, nom long

Quel est le nom COURT de votre ordinateur (le nom simple sans le domaine) ? Quel est le nom LONG de votre ordinateur (le nom avec le domaine) ? Comment les connaître en ligne de commande ? Sur les ordinateurs de l'école, est-il possible d'obtenir ces noms autrement qu'en ligne de commande ? Ajoutez les réponses à votre rapport.

La commande pour obtenir le nom court de mon ordinateur est :

`hostname`

```
romain@romain-HP-Laptop:~$ hostname  
romain-HP-Laptop
```

La commande pour obtenir le nom long (avec le nom de domaine) est :

`hostname --long`

```
romain@romain-HP-Laptop:~$ hostname --long  
romain-HP-Laptop
```

Sur les ordinateurs de l'école, on peut également connaître ces noms en regardant les étiquettes collées sur les unités centrales.

2. Adresse ip

Comment connaître les adresses (plusieurs) IP de votre ordinateur en ligne de commande ? Autrement (en passant par un site internet par exemple) ? Ajoutez les réponses à votre rapport.

On peut obtenir toutes les adresses IP d'une machine en tapant la commande :

`hostname -I`

```
romain@romain-HP-Laptop:~$ hostname -I  
192.168.1.107 137.194.120.17 2001:660:330f:120::100f
```

Alternativement, on peut utiliser la commande `ifconfig` qui affiche les statistiques sur les paquets transmis et reporte les adresses IP :

`/sbin/ifconfig`

Ces commandes transmettent les adresses IP privées. Il est également possible d'obtenir son adresse publique (celle utilisée sur internet) par la commande :

`curl ifconfig.me`

```
romain@romain-HP-Laptop:~$ curl ifconfig.me  
137.194.120.17romain@romain-HP-Laptop:~$
```

On peut enfin obtenir son IP publique par internet, avec des sites tels que ip4.me :

This page shows your IPv4 or IPv6 address

You are connecting with an IPv4 Address of:

137.194.120.17

[IPv4 only Test](#)

[Normal Test](#)

[IPv6 only Test](#)

3. Du nom vers l'IP

Comment à partir du nom d'un ordinateur, obtenir les adresses IP en ligne de commande ?
Ajoutez les réponses à votre rapport.

On peut obtenir l'IP depuis le hostname par diverse commandes, par exemple :

`grep romain-HP-Laptop /etc/hosts`

```
romain@romain-HP-Laptop:~$ grep romain-HP-Laptop /etc/hosts
127.0.1.1      romain-HP-Laptop
```

Alternativement :

`nslookup romain-HP-Laptop`

```
romain@romain-HP-Laptop:~$ nslookup romain-HP-Laptop
Server:      127.0.0.53
Address:     127.0.0.53#53

Non-authoritative answer:
Name:   romain-HP-Laptop
Address: 127.0.1.1
```

`host romain-HP-Laptop`

```
romain@romain-HP-Laptop:~$ host romain-HP-Laptop
romain-HP-Laptop has address 127.0.1.1
```

On obtient à chaque fois l'IP 127.0.1.1.

4. De l'IP vers le nom

Comment, à partir d'une adresse IP, obtenir les noms associés en ligne de commande ?
Ajoutez les réponses à votre rapport.

On peut obtenir les nom associés à une IP avec les commandes :

`nslookup 192.168.1.107`

```
romain@romain-HP-Laptop:~$ nslookup 192.168.1.107
107.1.168.192.in-addr.arpa      name = romain-HP-Laptop.
107.1.168.192.in-addr.arpa      name = romain-HP-Laptop.local.
```

`host 192.168.1.107`

```
romain@romain-HP-Laptop:~$ host 192.168.1.107
107.1.168.192.in-addr.arpa domain name pointer romain-HP-Laptop.
```

Dans les deux cas, le hostname retourné est romain-HP-Laptop.

5. Ping pong à l'intérieur!

Testez la communication avec d'autres ordinateurs (pas le vôtre) depuis le réseau de l'école en utilisant la commande ping (pour arrêter le ping faire CTRL + C). suivi du nom court, du nom long, de l'IP. Les trois méthodes fonctionnent-elles ? Ajoutez les réponses à votre rapport.

Avec le nom court :

ping c45-15

```
c45-14% ping c45-15
PING c45-15.enst.fr (137.194.34.206) 56(84) bytes of data.
64 bytes from c45-15.enst.fr (137.194.34.206): icmp_seq=1 ttl=64 time=0.190 ms
```

Avec le nom long :

ping c45-15.enst.fr

```
c45-14% ping c45-15.enst.fr
PING c45-15.enst.fr (137.194.34.206) 56(84) bytes of data.
64 bytes from c45-15.enst.fr (137.194.34.206): icmp_seq=1 ttl=64 time=0.187 ms
```

Avec le nom long :

ping 137.194.34.206

```
c45-14% ping 137.194.34.206
PING 137.194.34.206 (137.194.34.206) 56(84) bytes of data.
64 bytes from 137.194.34.206: icmp_seq=1 ttl=64 time=0.192 ms
```

6. Ping pong à l'extérieur

Si vous effectuez le ping depuis un réseau différent, il est possible que celui ne fonctionne pas (filtrage des accès vers le réseau de l'école depuis un réseau extérieur), contactez la DSI pour mettre en place une connection VPN / OpenVPN afin d'être sur le même réseau que les machines en salle de TP.

Je commence par connecter en VPN ma machine personnelle sur le réseau de l'école. Je peux ensuite tester la connexion avec les machines de l'école (par exemple la machine 3 en salle c128) avec la commande :

ping c128-03

```
romain@romain-HP-Laptop:~$ ping c128-03
PING c128-03.enst.fr (137.194.35.80) 56(84) bytes of data.
64 bytes from c128-03.enst.fr (137.194.35.80): icmp_seq=1 ttl=62 time=77.8 ms
64 bytes from c128-03.enst.fr (137.194.35.80): icmp_seq=2 ttl=62 time=68.1 ms
64 bytes from c128-03.enst.fr (137.194.35.80): icmp_seq=3 ttl=62 time=68.7 ms
64 bytes from c128-03.enst.fr (137.194.35.80): icmp_seq=4 ttl=62 time=64.7 ms
```

On peut voir qu'ici, la machine répond.

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

Comment lancer un calcul en ligne de commande sur votre ordinateur (par exemple $2 + 3$) ? Parmi les multiples réponses possibles, lesquelles permettent de lancer le calcul et d'obtenir le résultat en appuyant une seule fois sur la touche <Entrée> ? Ajoutez les réponses à votre rapport.

Pour lancer un calcul en local (par exemple $2+3$, on peut utiliser les commandes `echo`, `calc`, et `bc` :

`echo $((2+3))`

```
romain@romain-HP-Laptop:~$ echo $((2+3))
5
```

`calc 2+3`

```
romain@romain-HP-Laptop:~$ calc 2+3
5
```

`bc -l` , puis `2+3`

```
romain@romain-HP-Laptop:~$ bc -l
bc 1.07.1
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free Software
Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
2+3
5
```

Les deux premières ne nécessitent d'appuyer qu'une seule fois sur <Entrée> alors que la dernière impose deux commandes.

8. Calculer en ligne de commande sur un ordinateur distant

Comment lancer un calcul (par exemple $2 + 3$) en ligne de commande sur un autre ordinateur (à distance) ? Il faudra certainement vous authentifier avec un mot de passe. Comment obtenir le résultat du calcul immédiatement après avoir tapé son mot de passe ? Ajoutez les réponses à votre rapport.

Pour lancer un calcul à distance, il faut d'abord se connecter sur la machine à distance, par exemple :

`ssh rlegrand@c128-03`

```
romain@romain-HP-Laptop:~$ ssh rlegrand@c128-03
Warning: Permanently added 'c128-03,137.194.35.80' (ECDSA) to the list of known
hosts.
Linux c128-03 4.9.0-11-amd64 #1 SMP Debian 4.9.189-3+deb9u1 (2019-09-20) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
```

Cela requiert une authentification par mot de passe. Une fois connecté sur la machine, on peut réaliser le calcul par la commande simple :

`calc 2+3`

```
c128-03% calc 2+3
5
```

Alternativement, on peut obtenir le résultat du calcul directement après entré le mot de passe en envoyant la requête de calcul directement lors de la commande ssh :

```
ssh rlegrand@c128-03 calc 2+3
```

9. Calculer à distance sans mot de passe

Comment lancer un calcul à distance en utilisant SSH sans taper le mot de passe et en une seule ligne de commande (c'est à dire qu'on appuie sur <Entrée> et on a le résultat directement)?

Pour cela, il faut d'abord générer un jeu de clés (publique et privée) sur machine locale, puis copier la clé publique sur une machine distante du serveur :

```
ssh-keygen
```

```
ssh-copy-id rlegrand@c128-03
```

On peut ensuite désactiver le ssh Host Key Checking pour tous les hôtes. On obtient cela en éditant le fichier config dans le répertoire ~/.ssh :

```
cd ~/.ssh
```

```
nano config
```

Puis, dans le fichier à éditer :

```
Host *
```

```
StrictHostKeyChecking no
```

```
UserKnownHostsFile=/dev/null
```

On peut désormais répéter la commande de calcul à distance, qui s'exécutera sans qu'on ait besoin de rentrer de mot de passe :

```
ssh rlegrand@c128-03 calc 2+3
```

```
romain@romain-HP-Laptop:~$ ssh rlegrand@c128-03 calc 2+3
Warning: Permanently added 'c128-03,137.194.35.80' (ECDSA) to the list of known
hosts.
5
```

Etape 3: travailler avec des fichiers locaux ou sur un serveur NFS.

1. Chemin absolu

Quel est le chemin absolu de votre répertoire personnel, votre *home directory* ?

Après une connexion à distance :

```
ssh rlegrand@c128-03
```

On utilise les commandes :

```
cd
```

```
pwd
```

```
c128-03% cd
c128-03% pwd
/cal/homes/rlegrand
```

2. Un fichier dans le répertoire personnel

Créez un fichier fperso.txt contenant le texte "bonjour" dans votre répertoire personnel (sur un ordinateur de l'école).

Vérifiez le contenu du fichier avec cette commande exactement:

cat ~/fperso.txt

On commence par ouvrir et éditer le fichier texte :

nano ~/fperso.txt

Puis, dans le fichier à éditer :

bonjour

On utilise enfin la commande :

cat ~/fperso.txt

```
c128-03% nano ~/fperso.txt
c128-03% cat ~/fperso.txt
bonjour
```

3. Ou se trouve le fichier dans le répertoire personnel

Ce fichier est-il sur le disque dur de l'ordinateur ou autre part ? Comment savoir où est stocké physiquement ce fichier, à l'aide de quelle commande ?

Le fichier vient d'être créé, il se trouve donc normalement toujours dans le répertoire personnel.

Pour le confirmer, on peut utiliser la commande :

sudo find / -iname fperso.txt

```
c45-14% find /cal/homes/rlegrand -iname fperso.txt
/cal/homes/rlegrand/fperso.txt
```

On peut ensuite vérifier simplement la présence du fichier en se plaçant dans le répertoire indiqué et en vérifiant son contenu :

cd /cal/homes/rlegrand

ls

```
c45-14% cd /cal/homes/rlegrand
c45-14% ls
Desktop      fperso.txt      hadoop_tp2      script.sh      tmp
Downloads    gulliver2.txt   Kafka           Téléchargements TP Hadoop.odt
```

4. Un dossier et un fichier dans le répertoire temporaire

Créez un dossier /tmp/<votre nom d'utilisateur> en remplaçant <votre nom d'utilisateur> (ne pas mettre les caractères < et >).

On commence par créer le dossier /tmp/<votre nom d'utilisateur> :

cd

mkdir -p tmp/rlegrand

Créez un fichier ftemp.txt dans le répertoire /tmp/<votre nom d'utilisateur> .
Vérifiez le contenu du fichier avec cette commande **exactement**:
cat /tmp/<votre nom d'utilisateur>/ftemp.txt

On utilise les même commandes que pour la question 3.2 :

nano tmp/rlegrand/ftemp.txt

bonjour

cat tmp/rlegrand/ftemp.txt

Ce dossier et ce fichier sont-ils sur le disque dur de l'ordinateur ou autre part ? Comment savoir où sont stockés physiquement ces éléments, à l'aide de quelle commande ?

Le fichier est présent sur le disque dur local, ce qu'on confirme, une fois encore, en utilisant la commande :

sudo find / -iname ftemp.txt

Note : je dois sauter les questions 5 et 6. Elles requièrent en effet un serveur NFS. Pour monter un tel serveur, il est nécessaire de posséder les droits sudo (administrateur) sur au moins deux machines., or je ne dispose pas de ces droits sur les machines de Telecom.

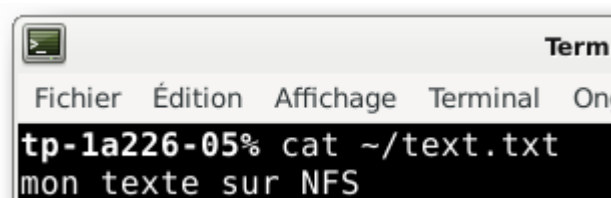
5. Trois ordinateurs A B C. On commence avec A. Utilisation du serveur NFS.

Pour les questions suivantes, utilisez trois ordinateurs: A, B C.

Connectez vous physiquement (avec un clavier, une souris et un écran) sur l'ordinateur A. Sur A, créez un fichier text.txt contenant le texte "mon texte sur NFS" dans votre répertoire personnel.

Vérifiez que le fichier existe et vérifiez son contenu. Pour cela, sur A, utilisez la commande :
cat ~/text.txt

Le fichier existe et son contenu est conforme au fichier créé :

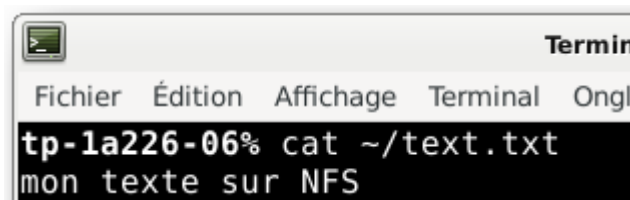


6. Trois ordinateurs A B C. On continue sur B et sur C. Utilisation du serveur NFS.

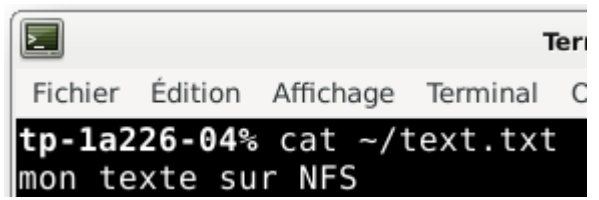
Connectez-vous à B (physiquement ou à distance) et vérifiez que le fichier text.txt est également présent dans votre répertoire personnel. Pour cela, sur B, utilisez la commande :
cat ~/text.txt

De même, connectez-vous à C et vérifiez que text.txt est aussi présent.

Remarquez que vous n'avez pas copié le fichier mais qu'il est présent sur A, B et C grâce au serveur NFS.



```
tp-1a226-06% cat ~/text.txt
mon texte sur NFS
```



```
tp-1a226-04% cat ~/text.txt
mon texte sur NFS
```

On constate que sans avoir créé de copie, le fichier existe aussi sur les deux autres machines.

7. Trois ordinateurs A B C. On commence avec A. Utilisation des disques locaux.

Déconnectez vous de B et de C et revenez sur l'ordinateur A.

Sur A, créez un dossier /tmp/<votre nom d'utilisateur> et un fichier local.txt contenant le texte "mon texte sur disque local" dans ce dossier /tmp/<votre nom d'utilisateur>.

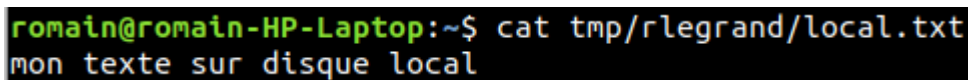
Je réutilise mon ordinateur personnel en tant que machine A. On utilise les commandes :

```
cd
mkdir -p tmp/rlegrand
nano ~/tmp/rlegrand/local.txt
mon texte sur disque local
```

Vérifiez que le fichier existe et vérifiez son contenu. Pour cela, sur A, utilisez la commande :
cat /tmp/<votre nom d'utilisateur>/local.txt

On utilise la commande :

```
cat tmp/rlegrand/local.txt
```



```
romain@romain-HP-Laptop:~$ cat tmp/rlegrand/local.txt
mon texte sur disque local
```

8. Trois ordinateurs A B C. On continue sur B et sur C. Utilisation des disques locaux.

Connectez-vous à B et C (physiquement ou à distance) et vérifiez que le dossier <votre nom d'utilisateur> ainsi que le fichier local.txt **ne sont pas** présent dans /tmp . Pour cela vérifiez avec la commande:

```
ls /tmp
```

On se connecte à distance sur la machine c128-03. On utilise alors la commande :

```
ls /tmp
```



```
c128-03% ls /tmp
hsperfdata_root
krb5ccmachine_ENST.FR
krb5cc_pam_W2fjYY
pulse-PKdhtXMmr18n
python2.7-virtualbox.tgz
systemd-private-6eb56b0fb40b45c686e6279c2adfa7e8-lldpd.service-ZJXUI0
vboxapi
vboxapi-1.0.egg-info
vboxapi.tgz
virtualbox
xrand.log
```

9. Depuis A, copier de A vers B avec les disques locaux.

Comment, à **partir de A**, transférer le fichier /tmp/local.txt **sur B** (dans /tmp/<votre nom d'utilisateur>/local.txt) en utilisant scp ? Vérifiez que le fichier est bien présent sur B. Attention: si vous avez une erreur "no such file or directory" (ou l'équivalent français), vous devez d'abord créer le répertoire /tmp/<votre nom d'utilisateur>/ avec la commande `mkdir -p` associée à un ssh pour l'ordinateur distant.

On commence par créer à distance le répertoire tmp/rlegrand, depuis la machine A (romain-HP-Laptop) vers la machine B (c128-03) :

`ssh rlegrand@c128-03 mkdir -p /cal/homes/rlegrand/tmp/rlegrand`

On peut ensuite transférer à distance le fichier tmp/rlegrand/local.txt par la commande scp:

`scp ~/tmp/rlegrand/local.txt rlegrand@c128-03:/cal/homes/rlegrand/tmp/rlegrand`

On vérifie enfin la présence du fichier sur le répertoire avec la commande :

`ssh rlegrand@c128-03 ls /cal/homes/rlegrand/tmp/rlegrand`

```
romain@romain-HP-Laptop:~$ ssh rlegrand@c128-03 ls /cal/homes/rlegrand/tmp/rlegrand
Warning: Permanently added 'c128-03,137.194.35.80' (ECDSA) to the list of known hosts.
ftemp.txt
local.txt
```

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

Comment, à **partir de A**, transférer le fichier **de B** (depuis /tmp/<votre nom d'utilisateur>/local.txt) **vers C** (dans /tmp/<votre nom d'utilisateur>/local.txt) ? Vérifiez que le fichier est bien présent sur C. De même que la question précédente, vous devez créer les répertoires /tmp/<votre nom d'utilisateur>/ correspondants.

On commence par créer depuis A dans C le répertoire tmp/rlegrand :

`ssh rlegrand@c128-08 mkdir -p /cal/homes/rlegrand/tmp/rlegrand`

On peut ensuite transférer à distance le fichier tmp/rlegrand/local.txt par la commande scp:

`scp -3 rlegrand@c128-03:/cal/homes/rlegrand/tmp/rlegrand/local.txt rlegrand@c128-08:/cal/homes/rlegrand/tmp/rlegrand`

On vérifie enfin la présence du fichier sur le répertoire avec la commande :

`ssh rlegrand@c128-08 ls /cal/homes/rlegrand/tmp/rlegrand`

```
romain@romain-HP-Laptop:~$ ssh rlegrand@c128-08 ls /cal/homes/rlegrand/tmp/rlegrand
Warning: Permanently added 'c128-08,137.194.35.85' (ECDSA) to the list of known hosts.
ftemp.txt
local.txt
```

Etape 4: lancer des programmes java à distance manuellement.

J'implémente une classe Slave (code disponible en annexe) qui exécute un programme différent selon les arguments transmis par args (de la classe main). Cela permet d'obtenir un jar qui accepte des arguments en ligne de commande, et de combiner en une seule classe cohérente les différentes mises à jour du programme Slave. Les programmes proposés sont le programme de calcul 3+5 de l'étape 4.1 (args[0]=-2), le programme de calcul 3+5 avec attente de 10 secondes de l'étape 6.1 (args[0]=-1), le programme de map de l'étape 10.2 (args[0]=0), le programme de shuffle de l'étape 10.2 (args[0]=1), et le programme de reduce de l'étape 12.1 (args[0]=2).

Un premier programme SLAVE sous Eclipse

Faire un programme Java nommé "SLAVE" qui calcule 3+5, affiche le résultat, sous Eclipse (Pour lancer Eclipse: Menu applications>développement), lancer ce programme dans Eclipse (flèche verte "exécuter")

Le programme fonctionne.

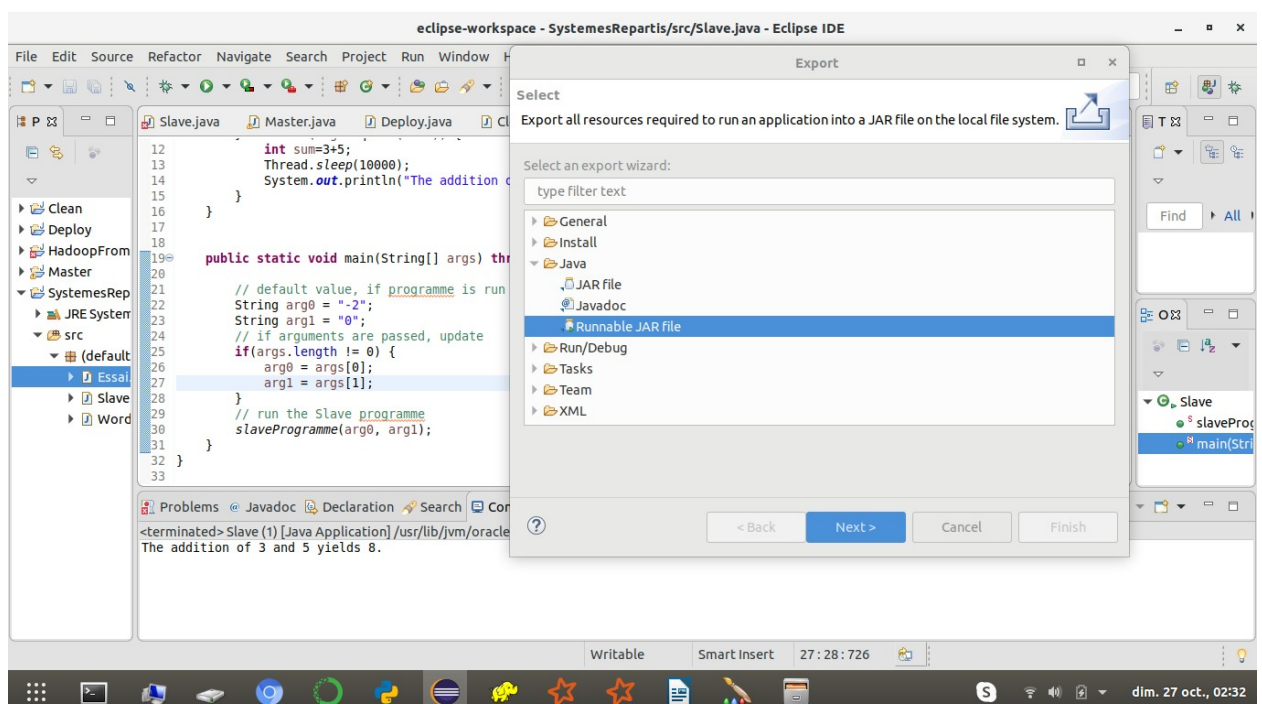
```
<terminated> Slave (1) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (27 oct. 2019 à 09:34:43)
The addition of 3 and 5 yields 8.
```

1. Exportation en JAR

Exporter le programme Java en slave.jar exécutable (Java ARchive dite Runnable) sous Eclipse. Attention de bien vérifier que le JAR est de type "Runnable"/"exécutable".

Pour exporter un fichier Java en .jar sous Eclipse :

File -> Export -> Java -> Runnable JAR file -> Finish



2. Exécution sur disque dur local

Créez le répertoire /tmp/<votre nom d'utilisateur>/

Copiez slave.jar exécutable dans le répertoire /tmp/<votre nom d'utilisateur>/

Testez et Lancez le slave.jar en ligne de commande sur votre ordinateur local.

Une fois le répertoire créé et le fichier Slave.jar placé dedans, on peut l'exécuter en ligne de commande avec :

```
java -jar /home/romain/tmp/rlegrand/Slave.jar
```

```
romain@romain-HP-Laptop:~$ java -jar /home/romain/tmp/rlegrand/Slave.jar
The addition of 3 and 5 yields 8.
```

3. Copie du JAR et exécution distante

Depuis la machine A contenant /tmp/<votre nom d'utilisateur>/slave.jar

Créez à distance sur la machine B (s'il n'existe pas) un répertoire /tmp/<votre nom d'utilisateur>/

Copiez slave.jar sur la machine B dans le répertoire /tmp/<votre nom d'utilisateur>/

Exécutez à distance (depuis A sur la machine B) le slave.jar.

Quelle est la commande tapée pour effectuer cette dernière action ?

On transfère à distance le fichier tmp/rlegrand/slave.jar par la commande scp:

```
scp ~/tmp/rlegrand/Slave.jar rlegrand@c128-03:/cal/homes/rlegrand/tmp/rlegrand
```

On exécute ensuite le programme à distance en ssh avec les mêmes commandes qu'en local:

```
ssh rlegrand@c128-08 java -jar /cal/homes/rlegrand/tmp/rlegrand/Slave.jar
```

```
romain@romain-HP-Laptop:~$ ssh rlegrand@c128-08 java -jar /cal/homes/rlegrand/tmp/rlegrand/Slave.jar
Warning: Permanently added 'c128-08,137.194.35.85' (ECDSA) to the list of known hosts.
The addition of 3 and 5 yields 8.
```

Etape 5: lancer des programmes en ligne de commande depuis java et afficher la sortie standard et la sortie d'erreur.

J'implémente une classe Master (code disponible en annexe) qui exécute des méthodes différentes correspondant aux différents programmes Master à mettre en œuvre dans le TP, selon les étapes. Cela permet de combiner en une seule classe cohérente les différentes mises à jour du programme. Les étapes concernées par la classe sont les étapes 5.1, 5.2, 5.3, 6.2, 9.2, 10.1, 10.3, 11.1, 11.4 et 12.2. Au niveau du fonctionnement, la classe repose sur l'utilisation de process builder (pour exécuter des commandes) et buffered reader (pour lire les flux de sortie et voir si le programme répond correctement).

1. Un programme MASTER java qui lance un autre programme en ligne de commande!

Ecrire un programme java nommé "MASTER" qui lance la commande suivante en utilisant ProcessBuilder:

```
ls -al /tmp
```

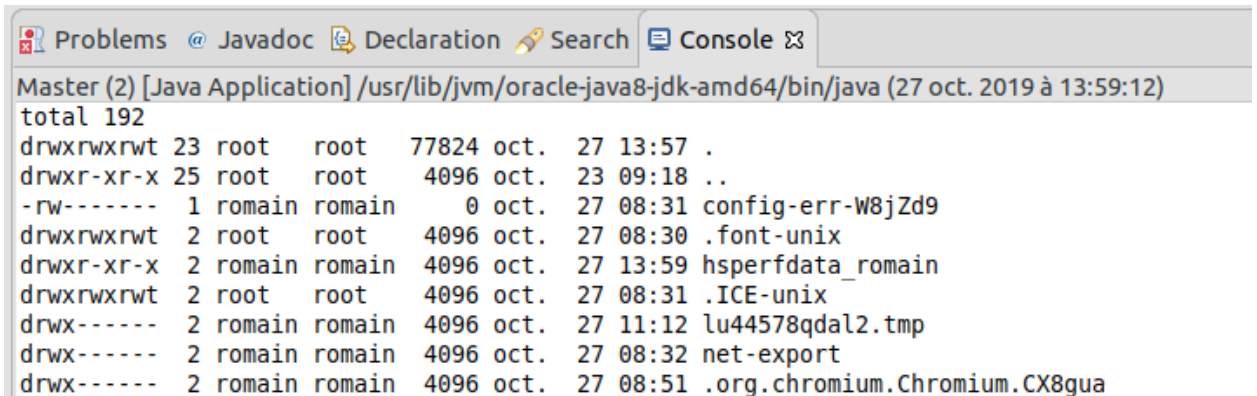
(vous pouvez aussi tester cette commande dans un terminal avant)

Récupérer et afficher la sortie de cette commande.

Vous devez utiliser ProcessBuilder de cette façon:

```
ProcessBuilder pb = new ProcessBuilder("ls", "-al", "/tmp");
```

On lance le programme Master pour cette commande. On obtient :



```
Master (2) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (27 oct. 2019 à 13:59:12)
total 192
drwxrwxrwt 23 root root 77824 oct. 27 13:57 .
drwxr-xr-x 25 root root 4096 oct. 23 09:18 ..
-rw----- 1 romain romain 0 oct. 27 08:31 config-err-W8jZd9
drwxrwxrwt 2 root root 4096 oct. 27 08:30 .font-unix
drwxr-xr-x 2 romain romain 4096 oct. 27 13:59 hsperfdata_romain
drwxrwxrwt 2 root root 4096 oct. 27 08:31 .ICE-unix
drwx----- 2 romain romain 4096 oct. 27 11:12 lu44578qdal2.tmp
drwx----- 2 romain romain 4096 oct. 27 08:32 net-export
drwx----- 2 romain romain 4096 oct. 27 08:51 .org.chromium.Chromium.CX8gua
```

2. Un programme MASTER java qui gère les erreurs de lancement d'un autre programme en ligne de commande.

Modifiez votre programme "MASTER" pour qu'il affiche la sortie d'erreur en cas d'erreur lors de l'exécution de la commande. Testez la sortie d'erreur avec une commande qui échoue avec un sortie d'erreur. Essayez par exemple d'exécuter la commande "ls /jesuisunhero".

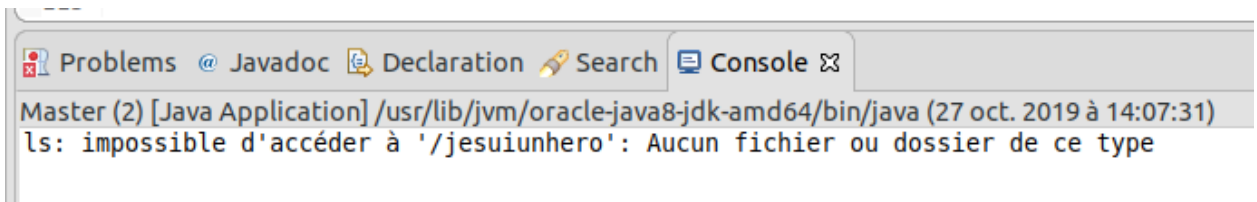
Explications: si on tape la commande "ls /jesuiunhero", le dossier /jesuisunhero n'existant pas, on aura une erreur de type "impossible d'accéder à /jesuisunhero: aucun fichier ou dossier de ce type." qui s'affiche dans la sortie d'erreur. En effet, il y a deux sorties: les sorties standards (sans erreur) et les sorties d'erreurs.

Vous devez utiliser ProcessBuilder de cette façon:

```
ProcessBuilder pb = new ProcessBuilder("ls", "/jesuisunhero");
```

Sur pb, vous pouvez récupérer le flux de la sortie standard et le flux de la sortie d'erreur.

On lance le programme Master pour cette commande. On obtient :

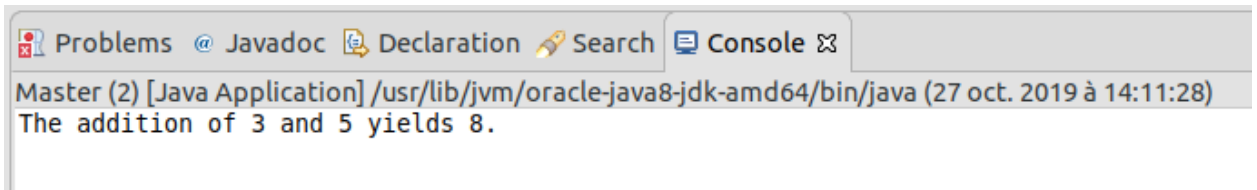


```
Master (2) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (27 oct. 2019 à 14:07:31)
ls: impossible d'accéder à '/jesuiunhero': Aucun fichier ou dossier de ce type
```

3. Un programme MASTER java qui lance un slave.jar en ligne de commande.

Modifiez votre programme "MASTER" pour qu'il lance "SLAVE", c'est à dire slave.jar situé sur la même machine que "MASTER" dans le dossier /tmp/<votre nom d'utilisateur>/slave.jar

On lance le programme Master pour qu'il exécute Slave.jar, comme dans l'étape 4.2. On obtient :



```
Problems @ Javadoc Declaration Search Console
Master (2) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (27 oct. 2019 à 14:11:28)
The addition of 3 and 5 yields 8.
```

Etape 6: gérer les timeout du MASTER.

1. Un SLAVE qui simule un calcul de 10 secondes.

Modifiez votre programme SLAVE pour qu'il simule une attente de 10 secondes avant d'afficher le résultat du calcul 3+5. Pour cela utilisez

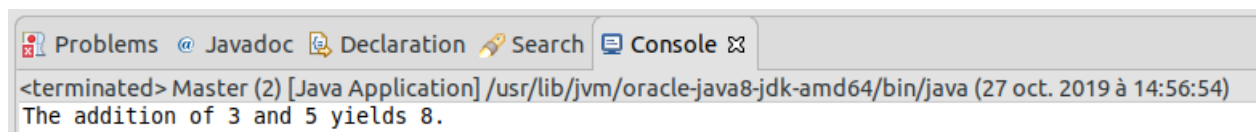
```
Thread.sleep(10000);
```

Vérifiez le bon fonctionnement du SLAVE et constatez qu'il y a 10 secondes entre le démarrage du SLAVE et l'affichage du résultat. Attention de ne rien afficher avant les 10 secondes pour que la question suivante fonctionne correctement.

Générez de nouveau le slave.jar. Copiez-le en écrasant le slave.jar dans le dossier /tmp/<votre nom d'utilisateur>/slave.jar

Testez le lancement à partir de MASTER.

On lance le programme Master pour qu'il exécute le Slave.jar modifié avec 10 secondes d'attente. Après 10 secondes, on obtient bien :



```
Problems @ Javadoc Declaration Search Console
<terminated> Master (2) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (27 oct. 2019 à 14:56:54)
The addition of 3 and 5 yields 8.
```

2. Gérer les timeout au niveau du MASTER.

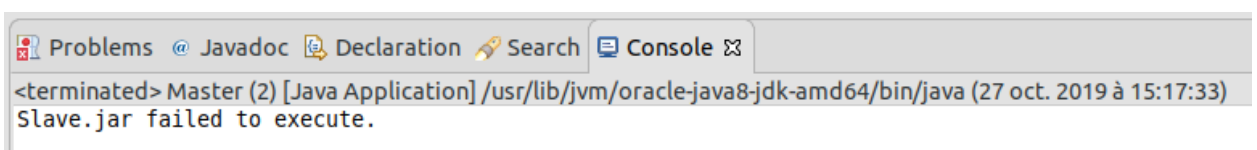
Modifier le MASTER pour qu'il attende que quelque chose soit écrit dans la sortie standard (sans erreur) ou dans la sortie d'erreurs du SLAVE pendant un certain temps maximum. Au bout du temps imparti le MASTER considère un timeout.

Il arrête les éventuels threads (si vous utilisez des threads - non obligatoire) s'occupant des sorties et/ou stoppe le processus créé avec ProcessBuilder.

Vous avez la solution ci-dessous pour implémenter les TESTs suivants:

TEST1 : Testez le bon fonctionnement du timeout en lançant le SLAVE avec un timeout de 2 secondes sur les sorties (standard et d'erreur). Le timeout étant plus court (au niveau du MASTER 2 secondes) que le temps de calcul du SLAVE (10 secondes), le MASTER doit arrêter les éventuels threads (si vous en utilisez) et le processus.

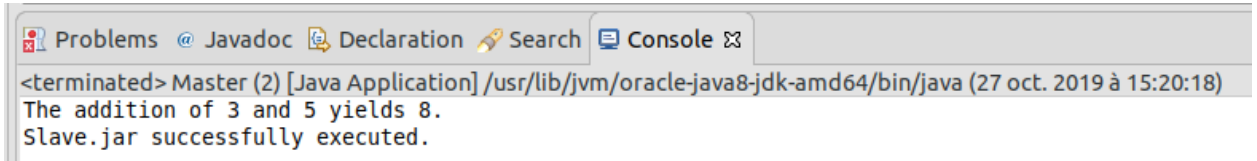
En imposant 10 secondes d'attente sur le Slave et seulement 2 secondes de timeout sur le Master, on obtient en effet un échec du processus :



```
Problems @ Javadoc Declaration Search Console
<terminated> Master (2) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (27 oct. 2019 à 15:17:33)
Slave.jar failed to execute.
```

TEST 2: Testez ensuite avec un timeout de 15 secondes, il ne devrait pas y avoir de timeout.

En allongeant le timeout sur le Master à 15 secondes, on évite l'arrêt du processus qui se conduit à son terme :



```
<terminated> Master (2) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (27 oct. 2019 à 15:20:18)
The addition of 3 and 5 yields 8.
Slave.jar successfully executed.
```

TEST 3: Testez ensuite en changeant le SLAVE pour qu'il écrive non plus dans la sortie standard (sans erreur) mais dans la sortie d'erreur. Pour cela, remplacez dans le Slave les `System.out.print...` par `System.err.print...`

Dans mon programme Slave, la sortie d'erreur est redirigée vers la sortie standard. Il n'y a donc pas de différence entre le TEST1 et le TEST3.

Etape 7: déployer automatiquement le programme SLAVE sur un ensemble de machines.

J'implémente une classe Deploy (code disponible en annexe) qui reprend l'essentiel des éléments de la classe Master. En particulier, la classe utilise la même stratégie de process builder/ buffered reader pour trouver quelles machines répondent et quelles machines ne répondent pas.

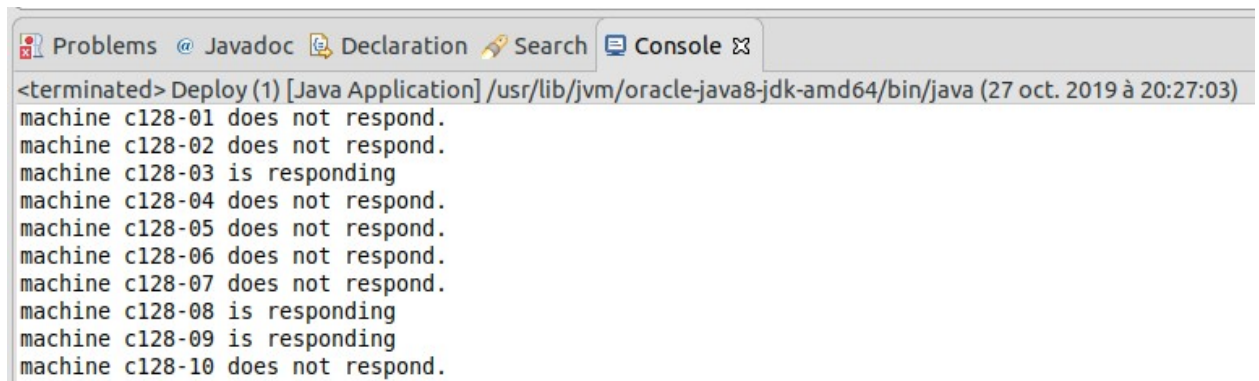
1. Un programme DEPLOY : Test de connexion SSH multiple

Créer un fichier texte à la main contenant : les adresses IP et/ou les noms des machines que nous voulons utiliser pour notre système réparti (par exemple toutes les machines de cette salle de TP), avec un nom ou une IP par ligne dans le fichier.

Créer un nouveau programme java DEPLOY qui lit ce fichier ligne par ligne et teste si la connexion SSH fonctionne bien sur chacune des machines. Attention, il s'agit bien d'un nouveau programme qui est séparé de MASTER ou SLAVE, vous ne l'exécuterez qu'en cas de mise à jour du SLAVE. Ceci permet de vérifier qu'une machine n'est pas éteinte ou qu'il y a un problème de connexion (par exemple).

Pour vérifier que la connexion fonctionne bien, vous pouvez faire afficher le nom de la machine distante (en exécutant la commande `hostname` à distance) et vérifier que l'affichage a effectivement lieu, sans erreurs. Réutilisez des parties de codes de la cinquième étape.

On lance le programme en testant la connexion SSH sur l'ensemble de la liste des machines du fichier texte. On obtient :



```
<terminated> Deploy (1) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (27 oct. 2019 à 20:27:03)
machine c128-01 does not respond.
machine c128-02 does not respond.
machine c128-03 is responding
machine c128-04 does not respond.
machine c128-05 does not respond.
machine c128-06 does not respond.
machine c128-07 does not respond.
machine c128-08 is responding
machine c128-09 is responding
machine c128-10 does not respond.
```

Votre programme DEPLOY lance-t-il les connections de manière séquentielle (les unes après les autres) ou de manière parallèle?

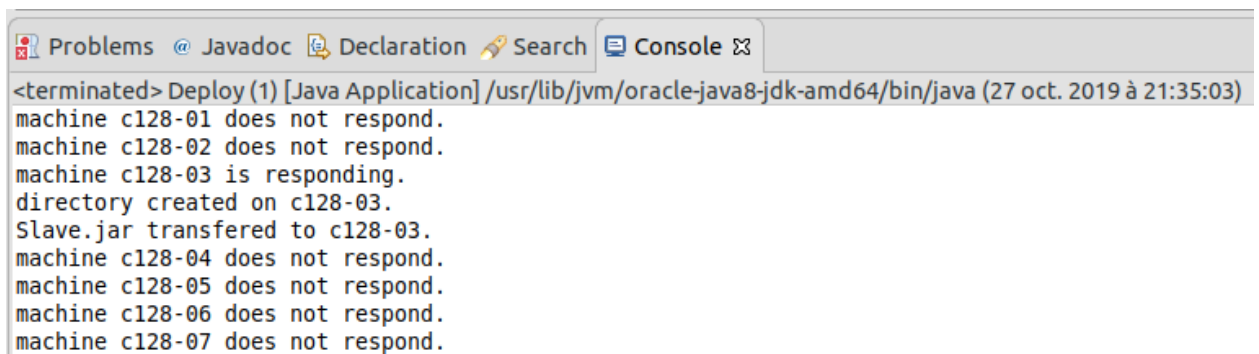
Le programme est séquentiel : il boucle sur toutes les machines les unes après les autres pour tester leurs réponses.

2. Un programme DEPLOY : copie de slave.jar multiple

Modifier votre programme DEPLOY pour qu'il copie le slave.jar dans /tmp/<votre nom d'utilisateur>/ sur les ordinateurs dont la connection SSH fonctionne.

Pour cela, vous devez utiliser la commande `mkdir -p` pour créer les répertoires dans /tmp s'ils n'existent pas déjà, attendre que le mkdir se termine puis copier avec `scp` le fichier slave.jar. Comment faites-vous pour attendre que le mkdir se termine correctement?

On lance le programme qui crée sur les machines à distance qui répondent un nouveau répertoire et y transfère le fichier Slave.jar. On obtient :



```
<terminated> Deploy (1) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (27 oct. 2019 à 21:35:03)
machine c128-01 does not respond.
machine c128-02 does not respond.
machine c128-03 is responding.
directory created on c128-03.
Slave.jar transfered to c128-03.
machine c128-04 does not respond.
machine c128-05 does not respond.
machine c128-06 does not respond.
machine c128-07 does not respond.
```

Pour s'assurer que le mkdir est terminé, on lui alloue suffisamment de temps (10 secondes de timeout). Si malgré tout le fichier n'est pas créé, on récupère le flux d'erreur et annule le transfert de Slave.jar vers la machine concernée.

Vérifiez ensuite manuellement que le fichier a bien été copié sur les ordinateurs distants. Attention de bien attendre la fin du mkdir avant de lancer le scp (on ne veut pas avoir de copie avant que le dossier soit effectivement créé).

Les fichiers ont bien été copiés sur l'ordinateur distant :

```
c128-08% pwd
/cal/homes/rlegrand/tmp/rlegrand
c128-08% ls
ftemp.txt  local.txt  Slave.jar
```

Lors des copies, faites attention au caractère “ / ” à la fin d'un chemin :

/tmp/toto est un chemin vers un fichier nommé toto

/tmp/toto/ est un chemin vers un dossier nommé toto.

Votre programme DEPLOY lance-t-il les copies de manière séquentielle (les unes après les autres) ou de manière parallèle?

Le programme est séquentiel : il boucle sur toutes les machines les unes après les autres pour tester leurs réponses, puis crée le répertoire et copie Slave.jar pour chaque machine qui répond.

Etape 8: nettoyer un ensemble de machines avec CLEAN.

J'implémente une classe Clean (code disponible en annexe) qui tout comme Deploy reprend l'essentiel des éléments de la classe Master. En particulier, la classe utilise la même stratégie de process builder/ buffered reader pour trouver quelles machines répondent et quelles machines ne répondent pas.

1. Un programme “CLEAN” qui nettoie les machines distantes.

Créez un nouveau programme CLEAN (différent de MASTER, SLAVE ou DEPLOY) qui efface votre dossier /tmp/<votre nom d'utilisateur>/sur les ordinateurs dont la connexion SSH fonctionne. Pour cela, vous utiliserez le même fichier texte écrit à la main et utilisé par DEPLOY contenant : les adresses IP et/ou les noms des machines que nous voulons utiliser pour notre système réparti (par exemple toutes les machines de cette salle de TP), avec un nom ou une IP par ligne dans le fichier.

CLEAN lit ce fichier ligne par ligne et efface sur chacune des machines votre dossier créé dans le dossier temporaire, en lançant la commande à distance `rm -rf /tmp/<votre nom d'utilisateur>/`. Attention de bien attendre la fin de la commande `rm -rf` pour être sûr que l'effacement a bien été effectué.

On lance le programme qui supprime le répertoire /tmp/rlegrand sur les machines à distance qui répondent. On obtient :

```
Problems @ Javadoc Declaration Search Console
<terminated> Clean (1) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (27 oct. 2019 à 22:33:58)
machine c128-01 does not respond.
machine c128-02 does not respond.
machine c128-03 is responding.
directory suppressed on c128-03.
machine c128-04 does not respond.
machine c128-05 does not respond.
```


Votre programme CLEAN lance-t-il les commande d'effacement de manière séquentielle (les unes après les autres) ou de manière parallèle?

Le programme est là encore séquentiel : il boucle sur toutes les machines les unes après les autres pour tester leurs réponses, puis supprime le répertoire sur chaque machine qui répond.

2. Vérification du DEPLOY et du CLEAN

Vérifiez manuellement que l'effacement des dossiers a bien lieu.

On vérifie manuellement l'effacement des dossiers. Pour la machine c128-03, on obtient :

```
c128-03% pwd
/cal/homes/rlegrand/tmp
c128-03% cd rlegrand
cd: aucun fichier ou dossier de ce type: rlegrand
```

Ce programme vous permet de nettoyer un ensemble de machines avant de relancer un calcul. A partir de maintenant, vous pouvez déployer votre application en utilisant DEPLOY et vous pouvez nettoyer votre application en utilisant CLEAN. Vérifiez donc que DEPLOY suivi CLEAN fonctionne correctement.

Etape 9: lancer le programme SLAVE sur un ordinateur à distance.

1. CLEAN et DEPLOY

Faites un CLEAN. Faites un DEPLOY sur une seule machine (modifier la liste des machines pour n'en mettre qu'une) . Vous devriez avoir la dernière version de slave.jar déployée sur une seule machine.

On ne retient que la machine c128-03. On fait un Clean puis un Deploy. On obtient effectivement la dernière version de Slave.jar :

```
c128-03% pwd
/cal/homes/rlegrand/tmp/rlegrand
c128-03% ls
Slave.jar
```

2. Master lançant Slave à distance.

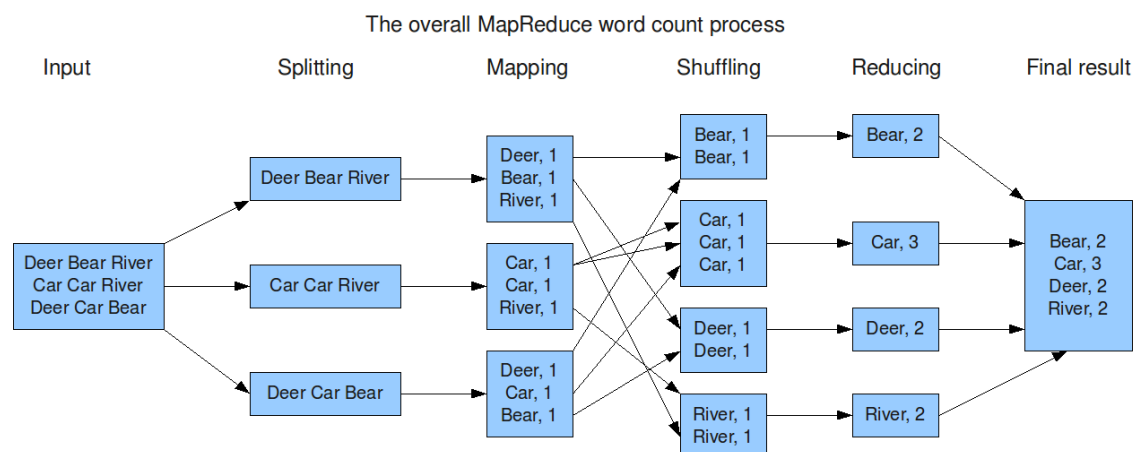
Modifier votre programme "MASTER" pour qu'il lance à distance le programme "SLAVE" déjà déployé avec le programme "DEPLOY" sur la seule machine distante.

Le Master modifié s'exécute correctement sur l'unique machine c128-03. On obtient :

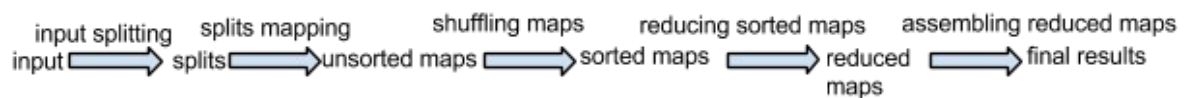
```
Problems @ Javadoc Declaration Search Console
<terminated> Master (2) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (27 oct. 2019 à 23:33:01)
Warning: Permanently added 'c128-03,137.194.35.80' (ECDSA) to the list of known hosts.
c128-03
machine c128-03 is responding.
Warning: Permanently added 'c128-03,137.194.35.80' (ECDSA) to the list of known hosts.
The addition of 3 and 5 yields 8.
Slave.jar executed on c128-03.
```

Etape 10: MapReduce - SPLIT et MAP

-



-



1. Un MASTER qui déploie les splits

Créez trois fichiers correspondants à des "splits" dans le répertoire temporaire. Dans un premier temps, créez ces fichiers manuellement.

/tmp/<votre nom d'utilisateur>/splits

S0.txt S1.txt S2.txt.

S0.txt contient:

Deer Bear River

S1.txt contient:

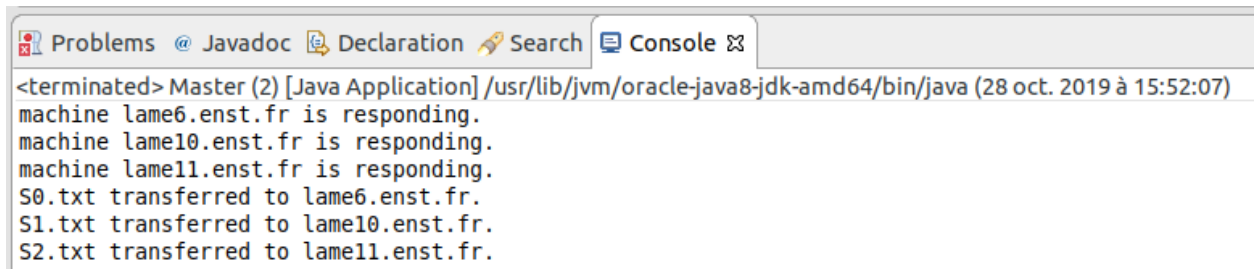
Car Car River

S2.txt contient:

Deer Car Bear

Modifiez votre MASTER pour qu'il copie les 3 fichiers de splits dans 3 ordinateurs différents, en ne copiant qu'un split par machine. Par exemple, s'il existe 3 machines et 3 splits, la première machine aura le premier split, la deuxième le deuxième etc. Pour cela vous utiliserez le fichier précédemment créé qui contient la liste des machines que vous voulez utiliser pour votre projet. Attention, le répertoire /tmp/<votre nom d'utilisateur>/splits doit être créé sur les 3 ordinateurs s'il n'existe pas. Cette création peut se faire de manière automatique (en créant de manière programmatique ces répertoires).

Le Master s'exécute, trouve trois machines qui fonctionnent et copie un split sur chacune d'elle :



```
<terminated> Master (2) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (28 oct. 2019 à 15:52:07)
machine lame6.enst.fr is responding.
machine lame10.enst.fr is responding.
machine lame11.enst.fr is responding.
S0.txt transferred to lame6.enst.fr.
S1.txt transferred to lame10.enst.fr.
S2.txt transferred to lame11.enst.fr.
```

Attention de faire attention de bien attendre que la création des dossiers soit bien effectuée avant de lancer la copie des splits. Comment attendez-vous que la création des dossiers soit bien effectuée avant de copier véritablement les fichiers?

Comme d'habitude, on laisse un temps suffisant (10 secondes), pour que les dossiers soient créés. Si ce temps est insuffisant, le processus sera considéré comme inactif et la création considérée comme un échec.

De la même manière que le programme DEPLOY, le MASTER va copier ces splits vers 3 ordinateurs dont la connexion SSH fonctionne.

Votre programme MASTER lance-t-il les copies de manière séquentielle (les unes après les autres) ou de manière parallèle?

Encore une fois, la création est séquentielle, les splits étant copiés les uns après les autres sur les machines à distance.

2. Un SLAVE qui fait la phase de map

Modifiez le SLAVE pour qu'il calcule un map à partir d'un split.

Pour cela, il prend un mode de fonctionnement en argument: 0 correspond au calcul du map à partir d'un split, puis un nom de fichier "Sx.txt" en entrée depuis le dossier splits et calcule un fichier "UMx.txt" en sortie dans le dossier maps, avec x variant (ici de 1 à 3). De la même manière que précédemment, le dossier maps doit être créé avant de pouvoir écrire des fichiers dedans. Vous devez attendre que le dossier maps soit créé avant de travailler avec. Comment attendez-vous que le dossier maps soit bien créé avant de travailler avec?

Le nom du fichier sera donné comme argument args du main:

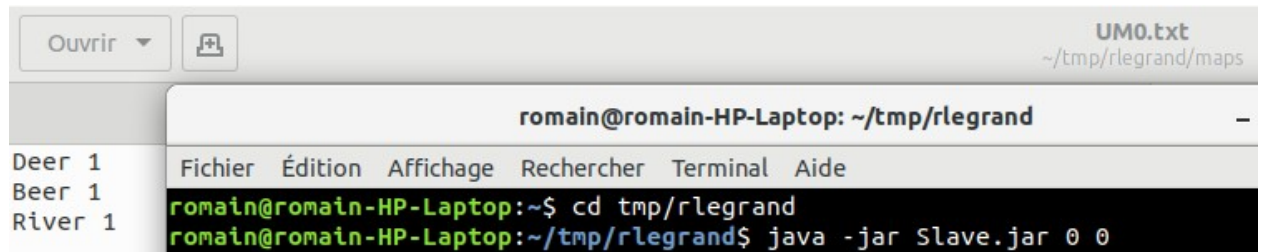
```
public static void main (String[] args)
```

Testez dans un terminal le slave.jar comme suit:

```
cd /tmp/<votre nom d'utilisateur>/
```

```
java -jar slave.jar 0 /tmp/<votre nom d'utilisateur>/splits/S0.txt
```

On teste d'abord le programme pour S0.txt. Il s'exécute correctement. On obtient :



The screenshot shows a file editor window titled 'UM0.txt' with the path '~/tmp/rlegrand/maps'. On the left, a list of files shows 'Deer 1', 'Beer 1', and 'River 1'. The main editor area displays a terminal window with the prompt 'romain@romain-HP-Laptop: ~/tmp/rlegrand'. The terminal shows the command 'cd tmp/rlegrand' and then 'java -jar Slave.jar 0 0'.

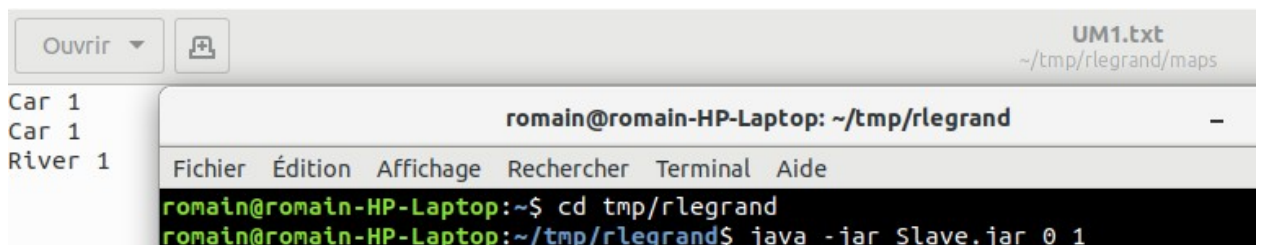
Le fichier /tmp/<votre nom d'utilisateur>/maps/UM0.txt doit être créé contenant
Deer 1
Beer 1
River 1

Testez le fonctionnement de votre SLAVE avec le fichier S1.txt contenant
Car Car River
Testez dans un terminal le JAR comme suit:

```
cd /tmp/<votre nom d'utilisateur>/  
java -jar slave.jar 0 /tmp/<votre nom d'utilisateur>/splits/S1.txt
```

Le fichier /tmp/<votre nom d'utilisateur>/maps/UM1.txt doit être créé contenant
Car 1
Car 1
River 1

On teste maintenant le programme pour S1.txt. Il s'exécute correctement. On obtient :



The screenshot shows a file editor window titled 'UM1.txt' with the path '~/tmp/rlegrand/maps'. On the left, a list of files shows 'Car 1', 'Car 1', and 'River 1'. The main editor area displays a terminal window with the prompt 'romain@romain-HP-Laptop: ~/tmp/rlegrand'. The terminal shows the command 'cd tmp/rlegrand' and then 'java -jar Slave.jar 0 1'.

Attention: Comme votre SLAVE est modifié, utilisez le DEPLOY pour déployer la nouvelle version.

Posez-vous la question: pourquoi retrouvons-nous deux lignes
Car 1
Car 1
Au lieu d'une seule ligne
Car 2 ?

Un indice: la phase de reduce qui arrivera plus tard, fera une addition des valeurs. Dans ce cas, la "fonction" de reduce est très simple: c'est une grande addition. Imaginez une fonction de reduce beaucoup plus complexe qui effectue un algorithme complexe appliqué sur toutes les valeurs...

3. Un MASTER qui lance les SLAVE pour la phase de map.

Modifiez le MASTER pour qu'il lance la phase de map sur plusieurs machines et affiche "MAP FINISHED". Pour cela vous utiliserez le fichier précédemment créé qui contient la liste des machines que vous voulez utiliser pour votre projet.

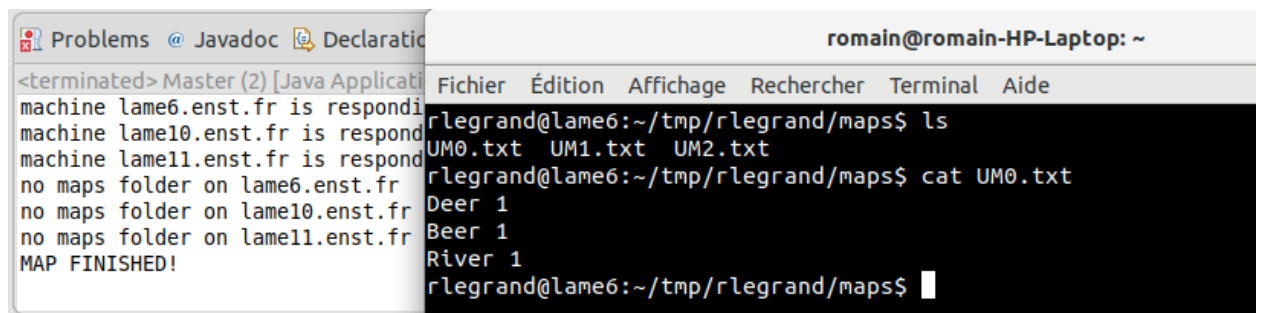
Pour bien synchroniser le MASTER avec les SLAVES, veillez à afficher "MAP FINISHED" qu'une fois tous les SLAVES terminés et uniquement quand tous les SLAVES sont terminés, PAS AVANT! Le MASTER doit donc attendre que SLAVES se terminent correctement. Comment faites-vous pour qu'un process lancé avec ProcessBuilder en Java attende la fin de l'exécution du processus ?

Votre programme MASTER lance-t-il les SLAVE de manière séquentielle (les uns après les autres) ou de manière parallèle?

Jusque là, les différentes phases du programme (création de répertoires et transfert de splits) étaient réalisées dans mon code de manière séquentielle, ce qui n'était pas problématique puisque les exécutions étaient quasi immédiates. Pour la phase de map en revanche, il devient essentiel que les programmes soient parallélisés. Autrement, le concept même de map reduce n'aurait aucun sens puisqu'il reviendrait à exécuter en séquentiel des splits sur différentes machine, sans aucun gain par rapport à du séquentiel pur sur une machine unique.

Pour pouvoir mener des exécutions de maps en parallèle, je crée dans ma classe Master une arraylist de processus (de fonction map). Les processus sont lancés de manière séquentielle (on boucle sur les entrées de l'arraylist), mais comme la boucle n'attend pas qu'un process soit fini pour lancer le suivant, tous les process finissent par tourner en parallèle. Je fais ensuite une autre boucle sur l'arraylist qui vérifie grâce à waitfor que chaque process a terminé son exécution. De cette manière, le Master ne reprend la main que lorsque tous les processus ont fini de tourner.

Le code s'exécute avec succès. On obtient :



The screenshot shows a Java IDE with a 'Problems' window on the left and a 'Terminal' window on the right. The 'Problems' window displays the following output from the Master process:

```
<terminated> Master (2) [Java Application]
machine lame6.enst.fr is responding
machine lame10.enst.fr is responding
machine lame11.enst.fr is responding
no maps folder on lame6.enst.fr
no maps folder on lame10.enst.fr
no maps folder on lame11.enst.fr
MAP FINISHED!
```

The 'Terminal' window shows the following commands and output:

```
romain@romain-HP-Laptop: ~
Fichier Édition Affichage Rechercher Terminal Aide
rlegrand@lame6:~/tmp/rlegrand/maps$ ls
UM0.txt UM1.txt UM2.txt
rlegrand@lame6:~/tmp/rlegrand/maps$ cat UM0.txt
Deer 1
Beer 1
River 1
rlegrand@lame6:~/tmp/rlegrand/maps$
```

Etape 11: MapReduce - SHUFFLE

1. Le MASTER qui prépare les SLAVE à la phase de shuffle.

Modifiez le MASTER pour qu'il envoie le fichier précédemment créé qui contient la liste des machines que vous voulez utiliser pour votre projet à tous les SLAVES utilisés pour la phase de map. Copiez ce fichier dans la cible suivante pour tous les SLAVES:

/tmp/<votre nom d'utilisateur>/machines.txt

Le programme s'exécute, le fichier machines.txt est maintenant présent sur toutes les machines à distance :

```
rlegrand@lame6:~/tmp/rlegrand$ ls
machines.txt  maps  Slave.jar  splits
rlegrand@lame6:~/tmp/rlegrand$ cat machines.txt
lame6.enst.fr
lame10.enst.fr
lame11.enst.fr
```

2. Le SLAVE qui prépare la phase de shuffle.

Modifiez le SLAVE pour qu'il prépare la phase de shuffle en regroupant les clés, en calculant le "hash" pour chacune des clés et en créant un fichier ayant pour nom <hash>-<hostname>.txt dans le dossier shuffles. De la même manière que précédemment, le dossier shuffles doit être créé avant de pouvoir écrire des fichiers dedans. Vous devez attendre que le dossier shuffles soit créé avant de travailler avec. Comment attendez-vous que le dossier shuffles soit bien créé avant de travailler avec?

Le nom du fichier sera donné comme argument args du main:

```
public static void main (String[] args)
```

Attention: si le fichier <hash>-<hostname>.txt existe déjà, il ne faut pas l'écraser mais plutôt continuer d'écrire dedans. Ce nom de fichier correspond au hash, obtenu grâce à la fonction de hachage de la classe String, explications ici

[https://fr.wikipedia.org/wiki/Java_hashCode\(\)#La_fonction_de_hachage_de_la_classe_java.lang.String](https://fr.wikipedia.org/wiki/Java_hashCode()#La_fonction_de_hachage_de_la_classe_java.lang.String) , calculé à partir de la clé; Le nom de la machine, lui, est obtenu grâce à l'instruction java suivante: `java.net.InetAddress.getLocalHost().getHostName()`

Pour cela, votre SLAVE prend un mode de fonctionnement en argument: 1 , qui correspond au calcul du hash , puis un nom de fichier "UMx.txt" en entrée depuis le dossier maps et calcule un fichier "<hash>-<hostname>.txt " en sortie dans le dossier shuffles . Gardez le mode précédent: 0 pour la phase de map.

Testez le fonctionnement de votre SLAVE avec le fichier UM1.txt contenant

Car 1

Car 1

River 1

Testez dans un terminal le JAR comme suit:

```
cd /tmp/<votre nom d'utilisateur>/
```

```
java -jar slave.jar 1 /tmp/<votre nom d'utilisateur>/maps/UM1.txt
```

Les fichiers suivants doivent être créés:

```
/tmp/<votre nom d'utilisateur>/shuffles/67508-c127-12.txt créé contenant
```

Car 1

Car 1

/tmp/<votre nom d'utilisateur>/shuffles/78973420-c127-12.txt créé
contenant
River 1

Comme votre SLAVE est modifié, utilisez le DEPLOY pour déployer la nouvelle version.

Les fichiers sont générés correctement. On obtient :

```
[rlegrand@lame12]~/tmp/rlegrand/shuffles% cat 67508-lame11.enst.fr.txt
Car 1
Car 1
[rlegrand@lame12]~/tmp/rlegrand/shuffles% cat 78973420-lame11.enst.fr.txt
River 1
```

3. Le SLAVE qui exécute la phase de shuffle.

Modifiez le SLAVE pour qu'il exécute la phase de shuffle.

Pour cela, pour chaque clé calculée lors du map, en plus de créer le fichier dans le dossier `shuffles`, il faut envoyer ce fichier dans le dossier `shufflesreceived` d'une des machines du fichier

/tmp/<votre nom d'utilisateur>/machines.txt

Pour savoir sur quelle machine il faut l'envoyer, on utilise le hash calculé à partir de la clé (une valeur entière) et le nombre de machines du fichier. Il suffit de calculer le hash modulo le nombre de machine en considérant que la première machine du fichier a pour numéro 0, la deuxième 1 etc... La formule utilisée pour trouver la machine sur laquelle sera envoyée le shuffle sera donc

$\text{numeroMachine} = \text{hash} \% \text{nbMachines}$

De plus, si le dossier `shufflesreceived` n'existe pas sur la machine réceptrice, il faudra le créer, attendre la fin de sa création, puis envoyer les fichiers dans ce dossier.

Testez le fonctionnement de votre SLAVE avec le fichier UM1.txt contenant

Car 1
Car 1
River 1

et le fichier machine.txt contenant uniquement TROIS machines (on fera donc un modulo 3)

Testez dans un terminal le JAR comme suit:

```
cd /tmp/<votre nom d'utilisateur>/
java -jar slave.jar 1 /tmp/<votre nom d'utilisateur>/maps/UM1.txt
```

Les fichiers suivants doivent être créés:

/tmp/<votre nom d'utilisateur>/shuffles/67508-c127-12.txt créé contenant
Car 1
Car 1

/tmp/<votre nom d'utilisateur>/shuffles/78973420-c127-12.txt créé
contenant
River 1

Normalement, le fichier 67508-c127-12.txt doit être copié dans la machine numéro 2 (67508 modulo 3 = 2) dans le dossier shufflesreceived et le fichier 78973420-c127-12.txt doit être copié dans la machine numéro 1 (78973420 modulo 3 = 1) dans le dossier shufflesreceived

Comme votre SLAVE est modifié, utilisez le DEPLOY pour déployer la nouvelle version.

Je fonctionne avec trois machines (lame10.enst.fr, lame11.enst.fr, lame12.enst.fr). A l'issue du shuffle, je dois donc avoir le fichier 67508-lame11.enst.fr.txt copié dans la machine numéro 2 (soit lame12.enst.fr) et le fichier 78973420-lame11.enst.fr.txt copié dans la machine numéro 1 (soit lame11.enst.fr), dans les dossiers shufflesreceived. C'est ce qu'on obtient :

```
[rlegrand@lame11]~/tmp/rlegrand/shufflesreceived% cat 78973420-lame11.enst.fr.txt
River 1
[rlegrand@lame11]~/tmp/rlegrand/shufflesreceived%
```

```
[rlegrand@lame12]~/tmp/rlegrand/shufflesreceived% cat 67508-lame11.enst.fr.txt
Car 1
Car 1
[rlegrand@lame12]~/tmp/rlegrand/shufflesreceived%
```

4. Un MASTER qui lance et attend la fin de la phase de shuffle.

Modifiez le MASTER pour qu'il lance la phase de shuffle une fois que la phase de map est terminée, sur plusieurs machines et affiche "SHUFFLE FINISHED". Pour cela vous utiliserez le fichier précédemment créé qui contient la liste des machines que vous voulez utiliser pour votre projet.

Pour bien synchroniser le MASTER avec les SLAVES, veillez à afficher "SHUFFLE FINISHED" qu'une fois tous les SLAVES terminés et uniquement quand tous les SLAVES sont terminés, PAS AVANT! Le MASTER doit donc attendre que SLAVES se terminent correctement.

Comment faites-vous pour qu'un process lancé avec ProcessBuilder en Java attende la fin de l'exécution du processus ?

Votre programme MASTER lance-t-il les SLAVE de manière séquentielle (les uns après les autres) ou de manière parallèle?

Je suis pour le shuffle la même stratégie que pour la phase de map : une arraylist de processus qui tournent en parallèle. La phase de shuffle est donc réalisée en exécution parallèle. Pour s'assurer que le Master attende la fin du processus, on utilise un simple waitFor().

Etape 12: MapReduce - REDUCE

1. Le SLAVE qui exécute la phase de reduce.

Modifiez le SLAVE pour qu'il prépare la phase de reduce en regroupant les fichiers avec le même hash, en calculant le reduce pour chacune des clés et en créant un fichier ayant pour nom <hash>.txt dans le dossier reduces. De la même manière que précédemment, le dossier reduces doit être créé avant de pouvoir écrire des fichiers dedans.

Pour cela, votre SLAVE prend un mode de fonctionnement en argument: 2 , qui correspond au calcul du reduce. Gardez les modes précédents: 0 pour la phase de map, 1 pour la phase de shuffle.

Testez le fonctionnement de votre SLAVE avec les fichiers suivants DANS LE DOSSIER SHUFFLERECEIVED:

```
/tmp/<votre nom d'utilisateur>/shufflesreceived/67508-cxxx-12.txt  
contenant  
Car 1  
Car 1
```

```
/tmp/<votre nom d'utilisateur>/shufflesreceived/67508-cxxx-13.txt  
contenant  
Car 1
```

```
/tmp/<votre nom d'utilisateur>/shufflesreceived/78973420-cxxx-12.txt  
contenant  
River 1
```

Testez dans un terminal le JAR comme suit:

```
cd /tmp/<votre nom d'utilisateur>/  
java -jar slave.jar 2
```

Le dossier reduces doit contenir les fichiers suivants:

```
/tmp/<votre nom d'utilisateur>/reduces/67508.txt contenant  
Car 3
```

```
/tmp/<votre nom d'utilisateur>/reduces/78973420.txt contenant  
River 1
```

Comme votre SLAVE est modifié, utilisez le DEPLOY pour déployer la nouvelle version.

Le code fonctionne correctement. On obtient :

```
[rlegrand@lame12]~/tmp/rlegrand/reduce% cat 67508.txt  
Car 3%
```

```
[rlegrand@lame12]~/tmp/rlegrand/reduce% cat 78973420.txt  
River 2%  
[rlegrand@lame12]~/tmp/rlegrand/reduce% █
```

On note que le reduce produit « River 2 » au lieu de « River 1 », comme suggéré dans l'énoncé. Le résultat est toutefois correct, il provient du fait que nos splits contenaient en tout deux fois le mot River (une fois dans S0.txt, une fois dans S1.txt).

2. Un MASTER qui lance et attend la fin de la phase de reduce .

Modifiez le MASTER pour qu'il lance la phase de reduce une fois que la phase de shuffle est terminée, sur plusieurs machines et affiche "REDUCE FINISHED". Pour cela vous utiliserez le fichier précédemment créé qui contient la liste des machines que vous voulez utiliser pour votre projet.

Pour bien synchroniser le MASTER avec les SLAVES, veillez à afficher "REDUCE FINISHED" qu'une fois tous les SLAVES terminés et uniquement quand tous les SLAVES sont terminés, PAS AVANT! Le MASTER doit donc attendre que SLAVES se terminent correctement.

On emploie encore une fois la stratégie de parallélisation développée précédemment, en incluant tous les process de reduce dans une même arraylist. On obtient alors les résultats escomptés, les fichiers de reduce étant répartis sur les différentes machines utilisés par le programme Master. On obtient par exemple, pour le mot Beer :

```
[rlegrand@lame11]~/tmp/rlegrand/reduce% cat 2066512.txt  
Beer 2%  
[rlegrand@lame11]~/tmp/rlegrand/reduce% █
```

3. Un MASTER qui chronomètre les phases.

Modifiez le MASTER pour qu'il chronomètre les différentes phases MAP, SHUFFLE, REDUCE et pour qu'il affiche le temps de chacune des phases.

Testez sur l'exemple de départ avec les trois splits suivants:

S0.txt S1.txt S2.txt.

S0.txt contient:

Deer Beer River

S1.txt contient:

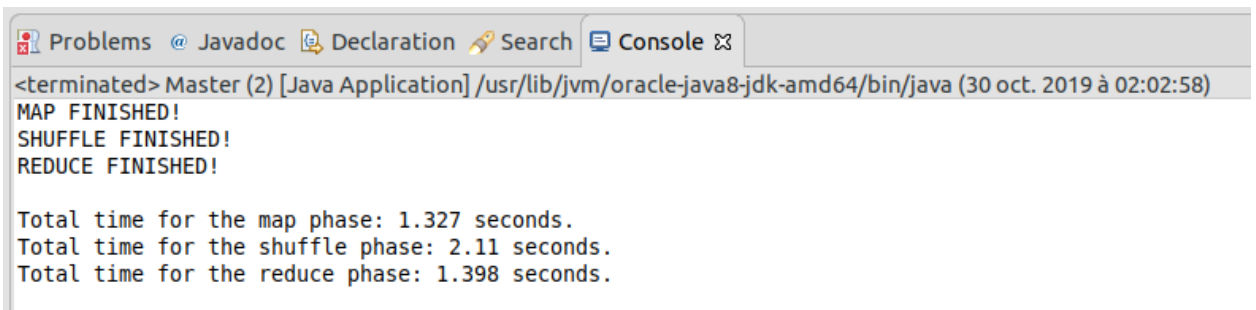
Car Car River

S2.txt contient:

Deer Car Beer

Quelle est la phase qui prend le plus de temps ? Quelle est la phase la plus rapide ?

On effectue un processus map/reduce complet sur les splits de test avec chronométrage. On obtient :

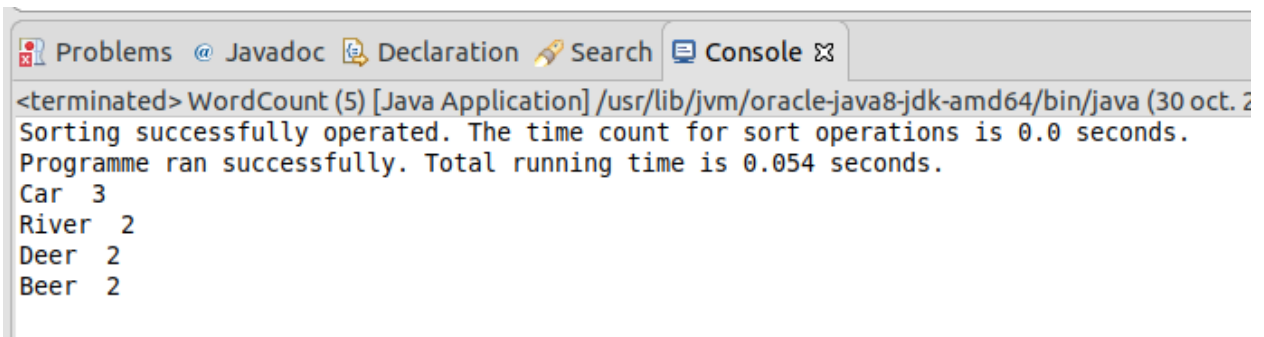


```
<terminated> Master (2) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (30 oct. 2019 à 02:02:58)
MAP FINISHED!
SHUFFLE FINISHED!
REDUCE FINISHED!

Total time for the map phase: 1.327 seconds.
Total time for the shuffle phase: 2.11 seconds.
Total time for the reduce phase: 1.398 seconds.
```

Le phase la plus rapide est celle de map, suivie de près par celle de reduce. La phase de shuffle est quand à elle considérablement plus lente, avec près de 50 % de temps en plus. Le processus total prend près de 5 secondes, ce qui paraît excessivement long pour un programme aussi trivial.

De manière générale, on constate que la méthodologie map/ reduce, pour un cas aussi simple, ne présente pas d'intérêt. En effet, la méthodologie du wordCount en séquentiel pur développée en début de TP permettait d'obtenir, pour ce même fichier, les résultats suivants :



```
<terminated> WordCount (5) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (30 oct. 2019 à 02:02:58)
Sorting successfully operated. The time count for sort operations is 0.0 seconds.
Programme ran successfully. Total running time is 0.054 seconds.
Car 3
River 2
Deer 2
Beer 2
```

Avec un temps total de 0.054 secondes en séquentiel contre 4, 835 secondes en réparti map/reduce, le séquentiel est 90 fois plus rapide. La perte de temps observée est vraisemblablement due au temps de communication entre machines, ainsi qu'à la complexité et au nombre des opérations de map/reduce comparé à un simple wordCount. Pour être tout à fait juste sur l'appréciation qu'on a de la méthodologie, il faudrait l'appliquer à des plus gros fichiers, pour lesquels la parallélisation peut présenter un réel avantage.

Etape 13: MapReduce - finalisations et optimisations

1. Un MASTER qui affiche le résultat à partir d'un fichier input.txt

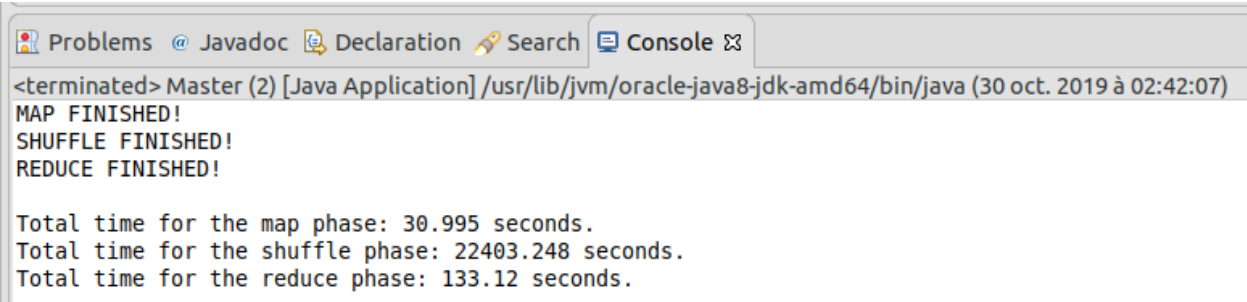
Modifiez le MASTER pour qu'il crée les splits et les déploie sur plusieurs machines à partir d'un fichier input.txt et affiche le résultat final (la combinaison des résultats dans les dossiers reduces des différentes machines).

2. Un projet Hadoop-Mapreduce complet, utilisable en environnement “BigData”

Copiez votre projet tel qu'il est à la question précédente. Ceci vous permet de garder une version de votre projet qui fonctionne avec l'exemple de démo tel que décrit lors des questions précédentes (petit fichier input.txt de démo avec 3 lignes).

Améliorez la copie de votre projet pour qu'il fonctionne avec des plus gros fichiers en input.

Dans l'exercice 12.3, on utilisait un tout petit fichier texte. On obtenait alors que le calcul séquentiel était beaucoup plus rapide que le calcul distribué. Pour être plus juste, il convient toutefois de renouveler l'exercice avec un dataset de plus grande taille. On reprend pour cela le dataset de l'étape 1.9 (extrait de toutes les pages internet). Pour rappel, en séquentiel pur, le programme tournait en 90 secondes. On fait tourner ici le programme en distribué, et on obtient :



```
<terminated> Master (2) [Java Application] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (30 oct. 2019 à 02:42:07)
MAP FINISHED!
SHUFFLE FINISHED!
REDUCE FINISHED!

Total time for the map phase: 30.995 seconds.
Total time for the shuffle phase: 22403.248 seconds.
Total time for the reduce phase: 133.12 seconds.
```

Les résultats sont pour le moins défavorables au système distribué. Les phases de map et de reduce restent de longueur raisonnable : 30 et 130 secondes, respectivement, ce qui implique un total de 160 secondes. C'est plus lent que le séquentiel (d'environ 50 %), mais cela reste raisonnable. La phase de shuffle, en revanche, paraît totalement disproportionnée : 22400 secondes soit 6 heures 12 minutes environ. Je ne peux que formuler des hypothèses sur la raison de cette si grande lenteur :

- des problèmes d'optimisation de mon code. J'ai une expérience limitée en Java et mon code peut être certainement grandement optimisé. Je ne suis toutefois pas sûr que cela puisse expliquer toute la durée de la phase de shuffle.
- le nombre important de communications entre machines, et la lenteur de ces communications selon l'état du réseau. Sur un gros fichier, notre shuffle implique un nombre extrêmement grand de communications : un envoi à distance par mot différent et par fichier UMx.txt qui contient ce mot. Si la communication n'est pas d'une rapidité extrême, le programme passera l'essentiel de son temps à copier des fichiers entre ordinateurs. Je pense que c'est la principale explication.
- le nombre limité de machines sur lequel l'exercice a été distribué. Pour garder l'exercice comparable aux étapes précédentes, le map/reduce n'a été effectué que sur trois machines à distance. Or, nous avons vu en cours que sur un exercice où la part de parallélisation était élevée (ce qui est le cas du word count), paralléliser sur d'avantage de machines amenait à une hausse de performance (même si celle-ci restait soumise à un plafond). Faire tourner mon map/reduce sur plus de machines pourrait donc réduire le temps d'estimation.

3. Une prise en compte des pannes

De la même manière que précédemment, créez une copie de votre projet sur laquelle vous travaillez sur la robustesse: faites en sorte que votre programme fonctionne malgré des pannes inopinées de certaines machines. Pour tester, vous pouvez éteindre une ou plusieurs machines au milieu de calculs. Vous pouvez inventer d'autres pannes vous-même et présenter vos résultats. Comparez avec la méthode de robustesse utilisée par Google :

<https://research.google.com/archive/mapreduce-osdi04.pdf>