Supélec - Campus de Metz
2^{eme} **année - Année 2013-2014**
Mars-Juin 2014

Projet de conception
Exploration d'Hadoop

Alexandre Careil
Juan Manuel Muñoz Pérez

Table des matières

I Introduction

à compléter

II Hadoop Distritubted File System

HDFS pour *Hadoop Distributed File System* est le système de fichiers propre à Hadoop. C'est le composant en charge du stockage des données dans un cluster Hadoop.

A Introduction

HDFS est un système de fichiers distribué, c'est-à-dire que contrairement à un système de fichiers local, il permet de stocker des fichiers de manière répartie sur plusieurs machines physiques. Du point de vue de l'utilisateur, l'ensemble des machines physiques n'est visible que sous la forme d'un espace de stockage géant.

> A distributed system is a collection of independent computers that appear to the users of the system as a single computer.
>
> Distributed Operating System. A. Tanenbaum, Prentice Hall, 1994

Comparé aux autres systèmes de fichiers distribués, il partage beaucoup de points communs, mais présente toutefois des différences majeures, non négligeables.

HDFS est prévu pour du "commodity hardware", c'est-à-dire du matériel de grande distribution. Sa haute tolérance aux pannes lui permet de gérer du matériel "low cost", peu fiable, d'où sa popularité actuelle.

Pour finir cet aperçu, il est bon de noter que la réplication de données à travers les machines physiques rend l'accès aux données rapide, en plus de rendre le stockage résistant aux pannes.

B Hypothèses de fonctionnement

1 Panne de matériel

HDFS considère les pannes comme des événements qui peuvent se produire à tout moment. En effet, imaginons un cluster de mille noeuds "low cost", il paraît tout à fait concevable qu'une machine tombe en panne (ou même plus). Cette supposition rend HDFS très résistant à ces pannes, et les mécanismes de réaction sont très rapides. La détection et l'adaptation rapides de ces pannes est un des principaux atouts de HDFS.

2 Vitesse d'accès, gros volumes de données

Un système de fichiers classique est censé s'adapter à une utilisation moyenne. Ainsi, un exemple comme ext2 s'adapte à une taille de fichier moyenne inférieure à 3ko. Cela est un bon compromis en termes de rapidité d'accès moyenne et d'espace gaspillé par les clusters pas entièrement remplis. D'autre part, un ensemble de fonctionnalités permet de répondre à une demande hétérogène (POSIX). Mais HDFS est prévu pour de grosses quantités de données (Big Data...), il serait ainsi ridicule de rechercher une grande performance de calcul, si elle est de toute façon limitée par la vitesse de lecture des données. C'est pour cela que HDFS déplace le compromis vers un point plus adapté, et permet ainsi une vitesse d'accès très élevée. Certaines fonctionnalités de systèmes de fichiers traditionnels ont été omises, du fait de leur inutilité pour ce cas d'utilisation précis, rendant HDFS encore plus performant dans son domaine.

3 Modèle de cohérence simple

Comme dit dans le paragraphe précédent, on recherche les cas d'utilisation de HDFS et on supprime toutes les fonctionnalités habituelles inutiles dans ce cas afin d'accéder à une plus grande performance. Une autre adaptation est donc celle du cycle de vie d'un fichier. Sur HDFS, on va créer un fichier (Write), puis y accéder massivement (Read), puis le fermer, mais on ne le modifiera jamais... Cette hypothèse simplifie beaucoup le problème de la cohérence des données, et rend l'accès encore plus rapide. Cela est donc particulièrement adapté aux applications MapReduce.

Un projet de fonctionnalité permettant d'ajouter des données en fin de fichier est en cours.

4 "Moving Computation is Cheaper than Moving Data"

HDFS répond à un besoin classique du calcul distribué. Il s'agit simplement de dire qu'il est plus rapide d'effectuer un calcul sur les données là où elles sont stockées, et non de les déplacer. Cela permet de ne pas encombrer le réseau, et aussi de gagner en rapidité. HDFS met donc à disposition des interfaces pour déplacer les applications sur les noeuds qui contiennent les données (typiquement la fonction Map).

5 Portabilité

HDFS est prévu pour être installé sur une gamme très large de machines différentes
Alex je te laisse le soin de détailler HDFS...

C Fonctionnement

Le fonctionnement de HDFS s'appuie sur plusieurs démons :

- le NameNode (NN) : c'est le noeud maître disposant d'une machine dédiée ;
- le SecondaryNameNode (SNN) : tout comme le NN c'est aussi un noeud maître disposant d'une machine dédiée. Ce noeud vient comme son nom l'indique seconder le NN en cas de panne majeure afin de ne pas perdre l'arborescense des données stockées dans le HDFS ;
- le DataNode (DN) : c'est un noeud esclave contenant les données du HDFS et implanté sur chaque machine du cluster.

A titre d'exemple, dans un cluster de 50 machines, vous trouverez trois noeuds maîtres correspondant au NameNode, SecondaryNameNode et JobTracker (confère partie suivante sur MapReduce). Il ne reste plus que 47 noeuds esclaves contenant chacun une copie du DataNode et du TaskTracker (confère partie suivante sur MapReduce). Détaillons les noeuds maîtres.

1 Le NameNode

à compléter

2 Le SecondaryNameNode

à compléter

III MapReduce

à compléter

IV Pré-requis de configuration

A Généralités

Sortie le 1er août 2013, la version 1.2.1 de Hadoop est considérée stable. Ce document donne les étapes nécessaires à l'installation de ce framework Java. Installation et compréhension des concepts ne sont pas à porter de mains. C'est pourquoi, ce guide ne prétend pas être exhaustif, le lecteur sera amené à chercher des informations par lui-même.

Pour de plus amples renseignements sur cette version, rendez-vous sur : *http ://hadoop.apache.org/docs/r1.2.1/*

B Matériel utilisé

L'installation effectuée dans le cadre de notre projet a été réalisée sur un MacBook Pro sous OSX 10.9.2 Mavericks datant de fin 2011 et doté d'un processeur 2,4 GHz Intel Core i5.

C Création d'un nouvel utilisateur

L'installation de Hadoop sur votre ordinateur est effectuée dans une première partie en mode *Single Node*, autrement dit Hadoop déploiera ses calculs que sur votre machine, contrairement au mode *Multi-Node Cluster* dans lequel plusieures machines sont agrégées dans le but d'augmenter la puissance de calcul.

Configurer Hadoop fait appel à l'écriture sur des fichiers susceptibles, en cas de mauvaise utilisation, d'altérer le bon fonctionnement de votre machine. Il est par conséquent conseillé de créer un nouvel utilisateur. De ce fait, Hadoop fait référence tout au long de ce document à l'utilisateur/session dédié à Hadoop. Celle-ci devra posséder les droits administrateur.

D Installation de Java

Une fois le nouvel utilisateur Hadoop créé, ouvrez la session. Afin que Hadoop fonctionne correctement il est nécessaire d'avoir une version de Java supérieure à la version 1.5.

Pour connaître la version installée sur votre machine, ouvrez votre Terminal et tapez :

```
$ java -version
```

Vous obtiendrez une réponse de ce style. Ci-dessous vous pouvez constater que la version Java installée est la 1.7.

```
java version "1.7.0_21"
Java(TM) SE Runtime Environment (build 1.7.0_21-b12)
Java HotSpot(TM) 64-Bit Server VM (build 23.21-b01, mixed mode)
```

Dans le cas où une version de Java en-dessous de la 1.5 est installée, rendez-vous sur http ://www.java.com/fr et suivez les instructions de téléchargement et d'installation.

E Configuration du SSH

L'accès aux différents noeuds via Hadoop nécessite de configurer le SSH (*Secure Shell*). Dans notre mode de fonctionnement, seul la configuration du localhost est nécessaire. Dans le mode cluster, chaque noeud sera accédé par SSH. Pour ce faire, vous devez générer une clé SSH pour votre session Hadoop. Sur votre terminal, tapez :

```
ordinatrthidiff:~ Hadoop$ su - Hadoop
ordinatrthidiff:~ Hadoop$ ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/Hadoop/.ssh/id_rsa) :
Created directory '/Users/Hadoop/.ssh' .
Your identification has been saved in /Users/Hadoop/.ssh/id_rsa.pub.
80:37:49:f6:47:35:d1:86:85:00:39:d1:f0:8e:d3:8a Hadoop@Ordinateur-ThiDiff.local
The key's randomart image is:
+--[ RSA 2048]----+
|      o +*oo+*.  |
|     + ooo. o.o  |
|    . = ..o  .   |
|     . o =       |
|        S o      |
|       . o       |
|      E .        |
|                 |
|                 |
+-----------------+
Ordinateur-ThiDiff:~ Hadoop$
```

Tapez sur la touche Entrée. La seconde ligne du code *bash* ci-dessus génère une clé RSA avec un mot de passe vide. De façon général ceci n'est pas conseillé mais étant donné que nous travaillons que en local actuellement, cela ne pose aucun problème de sécurité. Ceci évite d'entrer la mot de passe à chaque connexion.

Désormais nous devons autoriser l'accès SSH sur notre machine à l'aide de la clé RSA préalablement créée. Pour ce faire, vous devez autoriser cette clé en la plaçant dans le fichier contenant la liste des clés autorisées. Pour ce faire :

```
ordinatrthidiff:~ Hadoop$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

La connexion SSH sur votre machine est théoriquement possible. Pour rappel, votre machine est nomée localhost.

```
ordinatrthidiff:~ Hadoop$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
RSA key fingerprint is 6c:8b:6f:43:42:ee:f3:27:ca:af:28:ea:51:08:a4:21.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
Last login: Sat May 31 19:00:12 2014 from localhost
Ordinateur-ThiDiff:~ Hadoop$
```

Lors de votre première connexion, votre machine est enregistrée dans la liste des hôtes connus. Ainsi, à la prochaine connexion elle sera reconnue. Pour vous déconnecter, rien de plus simple :

```
Ordinateur-ThiDiff:~ Hadoop$ logout
Connection to localhost closed.
```

V Hadoop 1.2.1

A Préparation d'Hadoop

Actuellement il est possible de trouver sur le serveur Apache toutes les versions de Hadoop. Cependant il est préférable de travailler avec des versions considérées comme stables. C'est notamment le cas pour les versions 1.2.x, 2.2.x et 2.3.x. Notre projet est basé sur la version 1.2.1.

Sur le lien ci-dessous, téléchargez le fichier hadoop-1.2.1.tar.gz (61 Mb) :
http ://apache.mirrors.multidist.eu/hadoop/common/hadoop-1.2.1/

Une fois téléchargé, décompressez le fichier, renommez-le en hadoop et placez le dans /usr/local. Il est de plus très important de changer le propriétaire de ce fichier :

```
Ordinateur-ThiDiff:~ Hadoop$ cd /Users/Hadoop/Downloads
Ordinateur-ThiDiff:~ Hadoop$ mv hadoop-1.2.1 hadoop
Ordinateur-ThiDiff:~ Hadoop$ mv hadoop /usr/local
Ordinateur-ThiDiff:~ Hadoop$ sudo chown Hadoop hadoop
```

Vérifiez que le nouveau propriétaire du fichier hadoop est bien votre session Hadoop :

```
Ordinateur-ThiDiff:~ Hadoop$ cd /usr/local
Ordinateur-ThiDiff:local Hadoop$ ls -lisa | grep hadoop
26284444  0 drwxr-xr-x@  29 Hadoop   admin    986  3 mai 14:12 hadoop
```

Hadoop téléchargé, il faut désormais configurer quelques fichiers propres à Hadoop et qui font sa spécificité. Par conséquent, on ne parle par d'installation mais plutôt de configuration car il s'agit tout au long de définir les bonnes propriétés.

B Débriefing du dossier hadoop

Le dossier hadoop contient de nombreux sous-dossier qui ne sont pas tous aussi utiles les uns que les autres. En voici la liste :

```
CHANGES.txt docs ivy.xml
LICENSE.txt hadoop-ant-1.2.1.jar lib
NOTICE.txt hadoop-client-1.2.1.jar libexec
README.txt hadoop-core-1.2.1.jar logs
bin hadoop-examples-1.2.1.jar sbin
build.xml hadoop-minicluster-1.2.1.jar share
c++ hadoop-test-1.2.1.jar src
conf
```

Nous nous attardons que sur les dossiers bin et conf qui se sont montrés essentiels lors de notre étude d'Hadoop, laissant ainsi le lecteur s'intéresser de plus près aux autres fichiers.

A titre anecdotique, les fichiers hadoop-*.jar contiennent les librairies nécessaires à la compilation et à l'exécution des exemples contenus notamment dans le fichier hadoop-examples-1.2.1.jar (WordCount, etc...).

Ci-dessous quelques fichiers contenus dans le dossier bin que vous aurez à utiliser sans cesse par la suite afin de lancer Hadoop.

```
start-all.sh stop-all.sh
```

Ces scripts shell vont permettent de lancer ou d'arrêter les différents démons d'Hadoop à savoir : le NameNode (NN), le DataNode, le SecondaryNameNode, le JobTracker et le TaskTracker. Pour de plus amples détails référez-vous à la partie détaillant chaque démon. Par conséquent, il sera possible de tout lancer simultanément ou de lancer au fur et à mesure chaque démon à l'aide des autres scripts.

Le dossier conf est le plus important de tous car il contient les fichiers permettant de configurer Hadoop. Quatre fichiers ont leur grande importance et qui seront détaillés par la suite :

```
hadoop-env.sh mapred-site.xml
core-site.xml hdfs-site.xml
```

C Le fichier .bashrc

Le fichier .bashrc fait parti des fichiers lus lorsque le shell est invoqué comme shell interactif sans fonction de connexion. Il est essentiel de définir quelques variables de l'environnement Hadoop dans ce dernier. Celui-ci se trouve dans $HOME, si ce n'est pas le cas il suffit de le créér et devra contenir les informations ci-dessous.

```
# Hadoop environment variables
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.7.0_21.jdk
  /Contents/Home
export HADOOP_HOME=/usr/local/hadoop

# Add Hadoop bin/ dir to PATH Hadoop autocompletion commands
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
```

Hadoop a recours à Java, c'est pourquoi JAVA_HOME est défini. Les autres variables définies permettent de définir les dossiers sur lesquels Hadoop devra être amené à travailler.

D Le fichier hadoop-env.sh

Ce fichier contient les variables d'environnement nécessaires à Hadoop. Pour notre installation, seule la variable JAVA_HOME doit être précisée. Pour l'obtenir, tapez la commande cat $(/usr/libexec/java_home). Votre fichier doit contenir :

```
# The java implementation to use.  Required.
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.7.0_21.jdk
  /Contents/Home
```

E Les fichiers *-site.xml

Les trois fichiers *-site.xml permettent de configurer le dossier où Hadoop stock les fichiers de données, les ports utilisés, le nombre de réplication des blocs, etc.

1 core-site.xml

Ce fichier XML contient les informations générales d'Hadoop. Entre autres, il définit le dossier dans lequel Hadoop stockera les données temporaires lors de l'exécution des tâches. Il définit aussi le nom et le port du système de fichiers qui a rendu célèbre Hadoop : HDFS (*Hadoop Distributed File System*).

```
<configuration>

<property>
<name>hadoop.tmp.dir</name>
<value>/app/hadoop/tmp</value>
<description>A base for other temporary directories.</description>
</property>

<property>
   <name>fs.default.name</name>
   <value>hdfs://localhost:54310</value>
 </property>

</configuration>
```

/app/hadoop/tmp est le dossier contenant les données temporaires stockées par Hadoop aussi bien pour le système de fichier local que HDFS. Créez ce dossier et donnez-lui tous les droits d'accès :

```
Ordinateur-ThiDiff:~ Hadoop$ sudo mkdir -p /app/hadoop/tmp
Ordinateur-ThiDiff:~ Hadoop$ sudo chown Hadoop /app/hadoop/tmp
Ordinateur-ThiDiff:~ Hadoop$ sudo chmod 750 /app/hadoop/tmp
```

Ne pas donner l'intégralité des droits à la session Hadoop produira une exception java.io.Exception lors du formatage du NameNode (confère partie suivante).

2 mapred-site.xml

Rappelons-le, Hadoop a deux principales composantes : HDFS et MapReduce. Ce fichier permet de configurer le deuxième point et plus particulièrement le JobTracker qui est le service d'Hadoop permettant de suivre les tâches. Pour cela on lui attribut un port sur lequel, à travers une interface web comme nous verrons plus loin, allons venir lire les informations concernant les jobs.

```
<configuration>

<property>
<name>mapred.job.tracker</name>
    <value>localhost:54311</value>
</property>

</configuration>
```

3 hdfs-site.xml

Ce dernier fichier définit les propriétés relatives au système de fichiers propre à Hadoop : HDFS. On y définit particulièrement le nombre de réplications des blocs. Nous ne nous attarderons pas d'avantage sur ce point à l'heure actuelle.

```
<configuration>

<property>
<name>dfs.replication</name>
<value>1</value>
</property>

</configuration>
```

La configuration basique d'Hadoop en mode *Single Node* ne nécessite que la modification des fichiers mentionnés précédemment. Pour le mode *Multi Node* nous verrons que les fichiers /conf/masters et /conf/slaves auront leur importance.

F Formatage du NameNode

La dernière étape restante avant de pouvoir lancer des jobs Hadoop est de formater le NameNode. Ce service Hadoop est la pièce maîtresse de HDFS en ce qu'il conserve l'arborescence de tous les fichiers HDFS au sein des DataNodes. Il contient par conséquent l'ensemble des métadonnées des fichiers HDFS.

Le formatage du NameNode permet d'effacer l'ensemble des données de HDFS. C'est équivalent à un RAZ. Cette étape n'est à réaliser qu'une seule fois lors de la mise en place de votre cluster Hadoop sous risque de perdre toutes les données HDFS présentes sur votre cluster.

```
ordinatrthidiff:~ Hadoop$ $HADOOP_HOME/bin/hadoop namenode -format
14/06/01 00:18:33 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = ordinatrthidiff/192.168.1.66
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 1.2.1
STARTUP_MSG:   build = https://svn.apache.org/repos/asf/hadoop/common/branches/branch
STARTUP_MSG:   java = 1.7.0_21
************************************************************/
14/06/01 00:18:35 INFO util.GSet: Computing capacity for map BlocksMap
14/06/01 00:18:35 INFO util.GSet: VM type       = 64-bit
14/06/01 00:18:35 INFO util.GSet: 2.0% max memory = 1013645312
14/06/01 00:18:35 INFO util.GSet: capacity      = 2^21 = 2097152 entries
14/06/01 00:18:35 INFO util.GSet: recommended=2097152, actual=2097152
2014-06-01 00:18:35.649 java[10479:1b03] Unable to load realm info from SCDynamicStor
14/06/01 00:18:36 INFO namenode.FSNamesystem: fsOwner=Hadoop
14/06/01 00:18:36 INFO namenode.FSNamesystem: supergroup=supergroup
14/06/01 00:18:36 INFO namenode.FSNamesystem: isPermissionEnabled=true
```

```
14/06/01 00:18:36 INFO namenode.FSNamesystem: dfs.block.invalidate.limit=100
14/06/01 00:18:36 INFO namenode.FSNamesystem: isAccessTokenEnabled=false accessKeyUpd
14/06/01 00:18:36 INFO namenode.FSEditLog: dfs.namenode.edits.toleration.length = 0
14/06/01 00:18:36 INFO namenode.NameNode: Caching file names occuring more than 10 ti
14/06/01 00:18:38 INFO common.Storage: Image file /app/hadoop/tmp/dfs/name/current/fs
14/06/01 00:18:38 INFO namenode.FSEditLog: closing edit log: position=4, editlog=/app
14/06/01 00:18:38 INFO namenode.FSEditLog: close success: truncate to 4, editlog=/app
14/06/01 00:18:38 INFO common.Storage: Storage directory /app/hadoop/tmp/dfs/name has
14/06/01 00:18:38 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at ordinatrthidiff/192.168.1.66
************************************************************/
```

Hadoop est désormais configuré. L'étape suivante consiste à lancer les différents services Hadoop.

G Lancement des services Hadoop

A chaque fois que vous souhaitez lancer les services Hadoop vous devrez suivre les étapes suivantes :

- se connecter via SSH au localhost ;
- forcer la lecture du fichier .bashrc à l'aide de la commande "source .bashrc" ;

Il suffit ensuite de lancer le script shell fournit dans le dossier Hadoop (/bin/start-all.sh) :

```
ordinatrthidiff:~ Hadoop$ start-all.sh
starting namenode, logging to /usr/local/hadoop/libexec/../logs/hadoop-Hadoop-namenod
localhost: starting datanode, logging to /usr/local/hadoop/libexec/../logs/hadoop-Had
localhost: starting secondarynamenode, logging to /usr/local/hadoop/libexec/../logs/h
starting jobtracker, logging to /usr/local/hadoop/libexec/../logs/hadoop-Hadoop-jobtr
localhost: starting tasktracker, logging to /usr/local/hadoop/libexec/../logs/hadoop-
```

On voit sur la sortie que le NameNode, le DataNode, le SecondaryNameNode, le JobTracker et le TaskTracker se sont lancés. Cependant il est bon de vérifier leur état à l'aide de la commande jps (*Java Virtual Machine Process Status Tool*).

```
ordinatrthidiff:~ Hadoop$ jps
10649 DataNode
10891 TaskTracker
10560 NameNode
10803 JobTracker
11087 Jps
10737 SecondaryNameNode
```

Nous sommes désormais sûrs que les services Hadoop se sont lancés correctement. En effet, il se peut que NameNode ou le DataNode ne se lancent pas. Dans ce cas la solution la plus radicale et la plus efficace consiste à formater à nouveau le NameNode. Pour ce faire, vous devez arrêter les services et suivre la démarche détaillée précédemment.

H Arrêt des services Hadoop

L'arrêt des services Hadoop s'effectue avec :

```
ordinatrthidiff:~ Hadoop$ stop-all.sh
stopping jobtracker
localhost: stopping tasktracker
stopping namenode
localhost: stopping datanode
localhost: stopping secondarynamenode
```

VI Hadoop 2.2

A partir des versions , Hadoop vient avec un nouveau composant appelé Yarn.

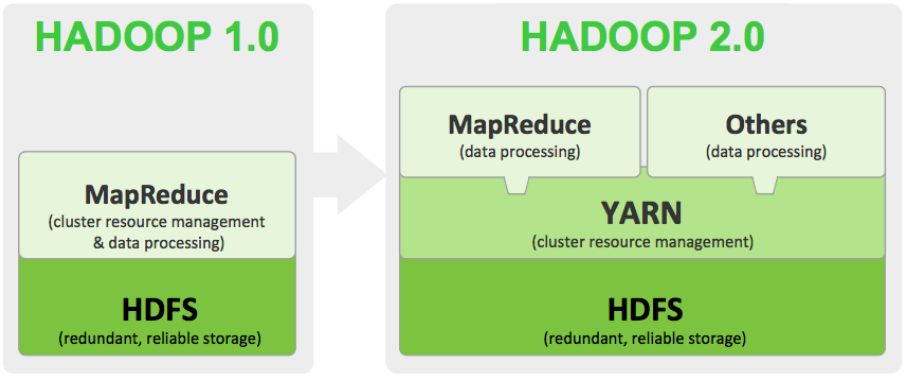

Figure 1 – Différences entre Hadoop 1.x et Hadoop 2.x

Yarn est responsable de la gestion des ressources, et utilise des processus similaires pour leur gestion. Le NodeManager vient remplacer le TaskTracker, qui s'occupe de faire le lien entre le DataNode et le ResourceManager localisé sur le master (NameNode). Le ResourceManager déploie les applications sur les noeuds grâce à sa communication avec les NodeManagers.

–> Schéma Hadoop Yarn

A Connexion sur un ordinateur à Supélec

La connexion avec un compte utilisateur précis est particulière, dans la mesure où on peut accéder à son dossier personnel, correspondant à des identifiants de connexion, depuis n'importe quelle machine. En effet, une machine est responsable de la gestion des logins, et contient les données personnelles correspondantes. Ces données transitent lors d'une connexion, ce qui peut faire croire que ces données sont présentes sur tous les ordinateurs.

Il est important de garder à l'esprit ce mécanisme, car c'est grâce à celui-ci que nous pourront simplifier la configuration de Hadoop sur le cluster Skynet. Malgré ces données qui ne sont pas propres à une machine, nous devons également pouvoir accéder à la mémoire de la machine physique, afin de pouvoir les utiliser comme DataNodes. Dans un premier temps, nous utiliserons le dossier `/tmp` accessible en lecture écriture à n'importe qui (donc peu sécurisé, mais adapté à notre installation de test).

B Connexion SSH

Lors d'une connexion SSH, le fichier `.bash_profile` est executé. Lors d'un login, c'est le fichier `.bashrc` qui est exécuté. Comme toutes les connexions via hadoop se font par ssh, nous allons les lier tous les deux. Pour cela, on créera un fichier `.bash_profile` contenant au moins :

```
1  if [ -f .bashrc ]; then
2    . ~/.bashrc
3  fi
```

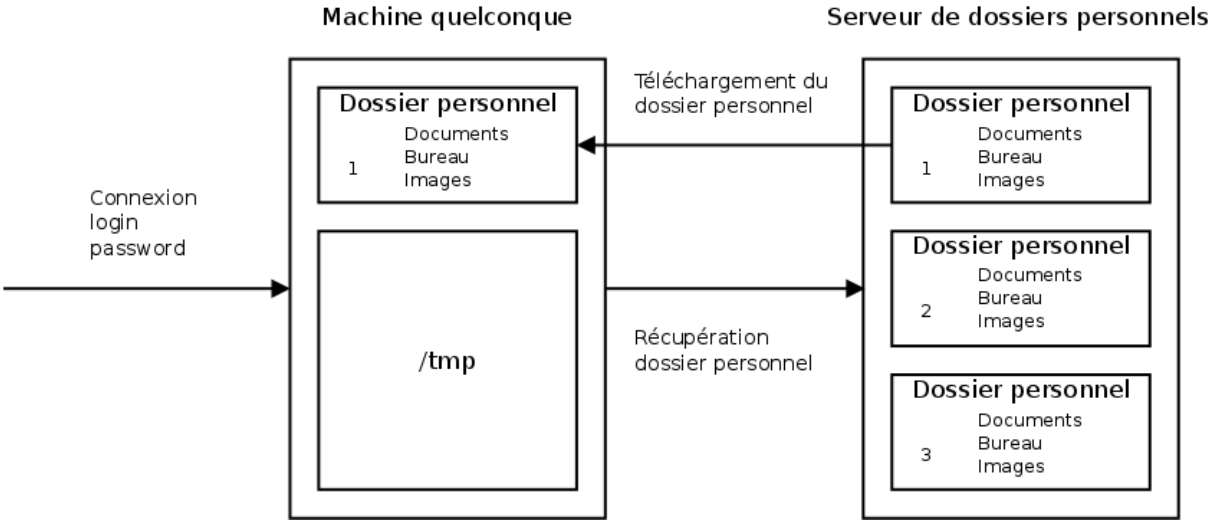

FIGURE 2 – Exemple d'une connexion sur une machine quelconque

De cette manière, si le fichier `.bashrc` existe, on le charge, sinon on ne fait rien.
On se connecte successivement sur `Ghome`, puis `Term2` :

```
1  Alex > ssh ghome
2  Last login: Sat Jun 14 22:00:38 2014 from 193.48.225.99
3  Alex [gserver] > ssh term2
4  Last login: Sat Jun 14 20:52:34 2014 from gserver.grid.metz.supelec.fr
5  Alex [term2] >
```

A partir de `Term2`, on peut se connecter à chacune des machines du cluster Skynet, dont les alias permettent de simplifier les opérations. Ainsi au lieu de

```
ssh careil_ale@term2.grid.metz.supelec.fr
```

on peut simplement faire

```
ssh careil_ale@term2
```

car nous sommes dans le domaine `grid.metz.supelec.fr`.
Il est inutile de spécifier le nom d'utilisateur lors de la connexion puisque, comme expliqué précédemment, notre compte existe sur toutes les machines (pas physiquement, mais il est chargé à chaque fois). Donc au lieu de se connecter avec la commande précédente, on peut simplement faire :

```
ssh term2
```

Cependant, il faut effectuer une manoeuvre afin de pouvoir se connecter facilement sans mot de passe. En effet, on va générer une clé SSH avec la commande `ssh-keygen` (qu'on appellera `id_rsa_test`). Vu qu'on ne veut pas utiliser de mot de passe pour se connecter, on omettra la `Passphrase` lors de la création de la clé.

```
Alex > ssh-keygen
Generating public/private rsa key pair.
```

```
Enter file in which to save the key (/home/alexandre/.ssh/id_rsa): id_rsa_test
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_rsa_test.
Your public key has been saved in id_rsa_test.pub.
The key fingerprint is:
61:e0:38:43:34:aa:fc:7b:6e:e3:04:79:a2:f6:14:1a alexandre@gentooalex
The key's randomart image is:
+--[ RSA 2048]----+
|   .+ .          |
|   o + .         |
|  . + . o        |
|..  .o . .       |
|.E = .  S        |
|  = =            |
| + o .           |
|. o o+           |
|   o=o.          |
+-----------------+
```

Maintenant, allons dans le dossier `.ssh` contenant notre clé publique, et notre clé privée afin de créer le fichier `authorized_keys` s'il n'existe pas déjà. S'il n'existe pas, tapez la commande `touch authorized_keys` afin de le créer.

On va ajouter notre clé publique à l'ensemble des clés autorisées :

```
cat id_rsa_test.pub >> authorized_keys
```

On peut désormais vérifier que cela fonctionne bien en se connectant par exemple à `sh00`. Normalement, aucun mot de passe n'est demandé.

Un dernier problème peut empêcher une connexion directe en ssh, celui de la vérification d'authenticité de l'hôte sur lequel on veut se connecter :

```
Juan [master] > ssh munozperez_jua@ghomE.metz.supelec.fr
The authenticity of host 'ghome.metz.supelec.fr (193.48.224.129)'
can't be established.
RSA key fingerprint is c1:41:c3:6d:f1:21:a2:2a:e6:30:b0:60:f7:de:22:b1.
Are you sure you want to continue connecting (yes/no)? YES
Warning: Permanently added 'ghome.metz.supelec.fr,193.48.224.129'
(RSA) to the list of known hosts.
Last login: Sun Apr  6 21:20:11 2014 from 193.48.225.99
```

Pour enlever ce message, il faut se connecter manuellement à chaque noeud pour qu'il soit ajouté à la liste des hôtes connus (listés dans le fichier `/.ssh/known`$_h$*osts*).

VII Applications

La configuration Hadoop fournit des exemples plus ou moins complexes dans le but de se familiariser avec cet environnement. A titre d'exemple, dans l'archive hadoop-examples-1.2.1.jar nous retrouvons l'algorithme MapReduce permettant d'estimer la valeur de π. Afin de mieux comprendre le patron d'architecture MapReduce traitons un cas concret de multiplication de matrices.

A Multiplication de matrices

Dans le cadre de l'électif STAGVOD (*Stockage et accès à de gros volumes de données*) nous avons été une première fois initiés à Hadoop. L'enseignant s'est particulièrement intéressé à l'algorithme MapReduce permettant de multiplier deux matrices qui est un très bon exemple de compréhension du fonctionnement des méthodes *map* et *reduce*. Vous trouverez en annexe le fichier Mm.java contenant l'implémentation des fonctions.

1 La fonction *main*

map prend comme entrée un fichier contenant les matrices. Compte tenu de l'implémentation de l'algorithme, il est indispensable que fichier en entrée soit de la forme suivante :

A : [0 1 2 3 4] [5 6 7 8 9]
B : [0 1 2] [3 4 5] [6 7 8] [9 10 11] [12 13 14]

Ci-dessous vous trouverez la fonction main :

```
1  public static void main(String[] args) throws Exception {
2
3          if (args.length < 2) {
4                  System.exit(0);
5          }
6
7          Configuration conf = new Configuration();
8          // A is an L-rows, M-cols matrix
9          // B is an M-rows, N-cols matrix.
10         conf.set("L", "2");
11         conf.set("M", "5");
12         conf.set("N", "3");
13
14         Job job = new Job(conf, "MatrixMultiplication");
15
16         [...]
17 }
```

Tout d'abord il est essentiel d'initialiser la configuration où le contexte viendra puiser ses données en précisant la taille des matrices. Par conséquent, à chaque changement de matrices, il faudra à nouveau définir la taille des matrices. Le job est ensuite créé sur la base de cette configuration.

Le reste de la fonction définit les chemins vers les fichiers .class des fonctions *map* et *reduce*, les formats de la paire (clé, valeur) ainsi que le format des fichiers en entrée et sortie (ici texte).

2 La fonction *map*

Le fichier texte en entrée sera d'abord traité par cette fonction. *map* le lit ligne par ligne le fichier et pour chaque ligne s'exécute une fois. Ce fichier est dans un premier temps splitté au caractère ":" et la lettre A est stockée dans matStruc[0] et les coefficients de cette matrice dans matStrut[1] (valable pour B aussi). A partir de là, un nouveau tableau, matLines, contenant les lignes est créé de telle sorte que :

Pour la matrice A

matLines[0]=[0 1 2 3 4
matLines[1]=[5 6 7 8 9

Pour la matrice B
matLines[0]=[0 1 2
matLines[1]=[3 4 5
matLines[2]=[6 7 8
matLines[3]=[9 10 11
matLines[4]=[12 13 14

Selon que ce soit la matrice A ou B, un traitement différent est opéré. En effet, dans le produit de matrices C=A*B, les coefficients de A et B mis en jeu dans le calcul du coefficient (i,j) de C ne sont pas les mêmes. D'où cette distinction.

Que ce soit pour A ou B, les tableaux à double entrée créés au début sont remplis avec la valeur des coefficients des matrices correspondantes. (A[0][0]=0, A[0][1]=1, etc...). Le reste du code se charge de créer les paires (clé, valeur). La clé correspond au coefficient de la matrice C. La valeur correspond à un des coefficients mis en jeu dans le produit matriciel.

Pour exemple on a :

$(C_{00}, (A, 0, A[0][0]))(C_{10}, (A, 0, A[1][0]))$
$(C_{00}, (A, 1, A[0][1]))(C_{00}, (A, 1, A[1][1]))$
$(C_{00}, (A, 2, A[0][2]))(C_{00}, (A, 2, A[1][2]))$
$(C_{00}, (A, 3, A[0][3]))(C_{00}, (A, 3, A[1][3]))$
$(C_{00}, (A, 4, A[0][4]))(C_{00}, (A, 4, A[1][4]))$

Le fichier mm_mapreduce_explication.txt contient l'ensemble des paires (clé, valeur) des matrices A et B en sortie de la fonction *map*.

3 La fonction *reduce*

"mapper" les matrices revient à parser les coefficients des matrices en leur faisant correspondre le coefficient de C dans lequel il est impliqué. A ce stade, l'ensemble des paires (clé, valeur) nécessaires à la multiplication puis addition pour le calcul des coefficients de C ont été obtenues.

Le rôle de la fonction *reduce* est d'isoler la bonne ligne de A et la bonne colonne de B auxquelles ont assigne la bonne clé. Elle va ainsi récupérer par clé les lots de deux vecteurs (celui associé à A et celui associé à B) correspondant à un élément scalaire de la matrice finale, puis multiplier deux à deux leurs éléments et les sommer.

Par exemple pour C00 on a :

$$
\begin{aligned}
&(\mathtt{C}_{00}, (A, 0, A[0][0]))x(C_{00}, (B, 0, B[0][0])) \\
&+ (C_{00}, (A, 1, A[0][1]))x(C_{00}, (B, 1, B[1][0])) \\
&+ (C_{00}, (A, 2, A[0][2]))x(C_{00}, (B, 2, B[2][0])) \\
&+ (C_{00}, (A, 3, A[0][3]))x(C_{00}, (B, 3, B[3][0])) \\
&+ (C_{00}, (A, 4, A[0][4]))x(C_{00}, (B, 4, B[4][0])) \\
&= C_{00}final
\end{aligned}
$$

Hadoop n'est pas du tout adapté au multiplication de matrices. Ceci est un exemple dans le but unique de comprendre l'algorithme MapReduce. Le fichier de sortie obtenu se trouve en annexe sous le nom $\mathtt{output}_matmult.txt$.

VIII Conclusion

à compléter