Déploiement du prototype MAS4DATA

Quentin Baert

Déploiement du prototype MAS4DATA

Avant de pouvoir déployer le prototype sur le cluster et ainsi pouvoir lancer un job MapReduce, il est nécessaire de suivre les instructions suivantes.

1. Préambule

Ici, on appelle `Monitor` la machine "maître", c'est-à-dire celle à partir de laquelle sera lancée l'exécution et sur laquelle se trouveront les résultats de l'exécution.

1.1. Prévoir le nom qui sera donné à l'`ActorSystem` déployé lors de l'exécution

Lors de l'exécution MapReduce, les agents mappers et reducers distants sont créés dans un `ActorSystem` commun. Cet `ActorSystem` devra être nommé à l'exécution mais son nom est nécessaire pour l'étape de configuration préalable. Par la suite, nous l'appellerons `RemoteAS`.

Il est également nécessaire de prévoir le port qui sera dédié à l'`ActorSystem` pour communiquer avec les autres machines distantes. Ici, nous utiliserons le port `6666`.

1.2. Créer un fichier qui contient l'ensemble des adresses IP des machines distantes utilisées lors de l'exécution

Ce fichier doit contenir l'ensemble des adresses IP des machines distantes utilisées lors de l'exécution MapReduce. Il contient une adresse IP par ligne. Par la suite, nous appellerons ce fichier `mes_ip.txt`.

1.3. Créer deux fichiers qui répartissent les mappers et les reducers sur les différentes machines

Lors de l'exécution, il faut que le `Monitor` puisse déterminer comment répartir les agents parmi les différentes machines utilisées. Pour cela, créer deux fichiers que nous appellerons `remote_mappers.txt` et `remote_reducers.txt`.

Pour indiquer que des agents doivent être déployer sur un `ActorSystem`, indiquer l'adresse de l'`ActorSystem` suivie d'un espace puis du nombre d'agent à y déployer.

Par exemple, pour déployer 5 mappers sur la machine `172.18.12.221`, indiquer la ligne suivante dans le fichier `remote_mappers.txt` :

```
akka.tcp://RemoteAS@172.18.12.221:6666 5
```

Si le fichier de configuration indique plus d'agents à déployer qu'il n'y en a de renseignés dans les fichiers `remote_mappers.txt` et `remote_reducers.txt`, les agents restant seront déployés localement sur la machine du `Monitor`.

1.4. Découper les données et les distribuer sur les machines mappers
Il est préférable de découper les données d'entrée et de distribuer chaque fragment sur la machine du mapper qui est censé l'exécuter.

Un fragment doit se nommer `mapperX.txt`, avec `X` le numéro du fragment.

Il est commun de créer un fragment par mapper. Cependant, s'il existe `X` fragments et `Y` mappers et que `X > Y` alors le fragment `x` est attribué au mapper `x % Y`.

Les fragments doivent tous se trouver au même endroit sur les machines mapper. Ainsi si un dossier `Data` se trouve à la racine de chaque machine mapper, le fragment `x` se trouve dans le dossier `Data` du mapper `x % Y`. Par exemple, pour 10 mappers et 12 fragments, on devrait retrouver la trace suivante :

```
mapper1 ~$ cd Data
mapper1 ~/Data$ ls
mapper1.txt    mapper11.txt
```

Pour cela, utiliser le script `uploadfile.sh` qui permet de télécharger un fichier sur une machine distante.

```
deployement/uploadfile.sh <login> <nodeIP> <fileToUpload>
<pathOnTheNode>
```

- `<login>` : le login de l'utilisateur sur les machines distantes ;
- `<nodeIP>` : l'adresse IP de la machine distante sur laquelle uploader le fichier ;
- `<fileToUpload>` : le chemin local vers le fichier à télécharger sur la machine distante ;

- `<pathOnTheNode` : le chemin auquel télécharger le fichier sur la machine distante.

1.5. Créer un fichier de configuration valide

1.5.1. Entrées du fichier de configuration

Entrée du fichier de configuration	Type	Valeur à associer
nb-mapper	INT	Nombre de mappers
nb-reducer	INT	Nombre de reducers
pb	STRING	Job MapReduce exécuté
task-bundle-management-strategy	STRING	Stratégie de sélection de tâches
threshold	DOUBLE	Seuil pour considérer une délégation de tâche comme socialement rationnelle
bidder-max-auction	either STRING(none) or INT	Nombre maximum d'enchère dans lesquelles s'engage un bidder
partition-strategy	STRING	Méthode de répartition des clés aux reducers par le `Partitioner`
task-cost-strategy	STRING	Fonction de coût
inform-contribution-frequency	INT	Fréquence de prise en charge du message `InformContribution` par le `Manager`
init-file	STRING	Chemin vers le fichier qui contient les données d'entrées
init-chunks	BOOLEAN	Si `false`, c'est au `Monitor` de créer les fragments à partir des données d'entrée
init-chunks-path	STRING	Chemin vers les fragments (par exemple `~/Data`)
init-chunks-number	INT	Nombre de fragments disponibles
chunk-size	INT	Taille des shunts créés par les mappers
result-path	STRING	Chemin vers le dossier qui contiendra les résultat de

		l'exécution
initiator-timeout	INT	Timeout (en ms) utilisé par les initiateurs
bidder-timeout	INT	Timeout (en ms) utilisé par les bidders
contractor-timeout	INT	Timeout (en ms) utilisé par un bidder en tant que contractor
acknowledgment-timeout	INT	Timeout (en ms) utilisé par les agents en attente d'un accusé de réception
pause-mili	INT	Pause (en ms) appliquée par le worker entre le traitement de deux clés
pause-nano	INT	Pause (en ns) appliquée par le worker entre le traitement de deux clés
debug-reducer	BOOLEAN	Si `true` imprime les traces du reducer lors d'une exécution
debug-manager	BOOLEAN	Si `true` imprime les traces du manager lors d'une exécution
debug-broker	BOOLEAN	Si `true` imprime les traces du broker lors d'une exécution
debug-rfh	BOOLEAN	Si `true` imprime les traces de l'agent `RemoteFileHandler` lors d'une exécution
debug-monitor	BOOLEAN	Si `true` imprime les traces du `Monitor` lors d'une exécution
remote	BOOLEAN	Si `true`, le `Monitor` cherche à lancer l'exécution de manière distribuée
remote-mappers	STRING	Chemin vers le fichier qui contient la distribution des mappers (par exemple `remote_mappers.txt`)
remote-reducers	STRING	Chemin vers le fichier qui contient la distribution des reducers (par exemple `remote_reducers.txt`)
gnuplot-max-taskdone-number	INT	Estimation du nombre maximum de tâches exécutées

		par un reducer (permet une génération correcte des courbes *post run*)
gnuplot-title	STRING	Titre donné à la figure qui présentera les contributions des reducers lors de l'exécution
gnuplot-output-filename	STRING	Nom du fichier qui présentera les contributions des reducers lors de l'exécution
gnuplot-output-format	STRING	Format du fichier de résultat (généralement `png` ou `pdf`)
task-monitor	BOOLEAN	Si `true` affiche en temps réel les contributions des reducers au cours de l'exécution
monitor-task-scale	STRING	Dans le cas d'un affichage en temps réel des contribution, échelle initiale pour la représentation des tâches
monitor-task-scale-step	INT	Valeur utilisée pour faire croître ou décroître l'échelle de la représentation des tâches

Certains des champs du fichier de configuration demandent l'entrée de mots clés. Le tableau suivant indique où trouver les mots clés pour chaque entrée qui le nécessite.

Entrée du fichier de configuration	**Classe qui contient les mots clés**
pb	`utils.config.JobHandler`
task-bundle-management-strategy	`utils.config.TaskBundleHandler`
partition-strategy	`utils.config.PartitionStrategyHandler`
task-cost-strategy	`utils.config.TaskCostStrategyHandler`

1.5.2. Fichier config/configLocation.txt

Le fichier `config/configLocagion.txt` indique quel est le chemin du fichier de configuration à considérer lors de la prochaine exécution. Il permet de pouvoir faire coexister plusieurs fichier de configurations et de passer de l'un à l'autre sans avoir à faire de copier-coller et ainsi risquer de perdre une configuration précédemment définie.

1.6. Indiquer la bonne adresse IP dans le fichier `application.conf`

Dans le fichier `src/main/resources/application.conf`, indiquer l'adresse IP du `Monitor` pour le champ `akka.remote.netty.tcp.hostname`.

1.7. Ouvrir les ports sur les machines

Sur l'ensemble des machines utilisées lors de l'exécution, s'assurer que le port `6666` est ouvert.

Sur chaque VM du cluster, utiliser la commande suivante :

```
iptables -A INPUT -p tcp --dport 6666
```

2. Créer le .jar à déployer

Depuis la racine.

```
sbt daemon:assembly
```

Au terme de la commande, le .jar à déployer se trouve au chemin suivant `./target/scala-2.11/daemon.jar`.

3. Déployer le .jar sur les machines distantes

```
./deployement/deploy.sh <login> <nodesFile> <daemonJar> <distantPath>
```

Avec :

- `<login>` : le login de l'utilisateur sur les machines distantes ;
- `<nodesFile>` : le chemin vers le fichier qui contient la liste des adresses IP des machines distantes sur lesquels déployer le .jar (`mes_ip.txt`);
- `<daemonJar>` : le chemin vers le .jar `daemon.jar` sur la machine locale ;
- `<distantPath>` : le chemin auquel déposer le .jar sur les machines distantes.

4. Initialiser les `ActorSystem` sur les machines distantes

```
./deployement/lauch.sh <login> <nodesFile> <daemonJar> <asName>
<port>
```

Avec :

- `<login>` : le login de l'utilisateur sur les machines distantes ;
- `<nodesFile>` : le chemin vers le fichier qui contient la liste des adresses IP des machines distantes sur lesquels déployer le .jar ;
- `<daemonJar>` : le chemin vers le .jar `daemon.jar` sur les machines distantes.
- `<asName>` : nom de l'`ActorSystem` (par exemple `RemoteAS`) ;
- `<port>` : port sur lequel écoute l'`ActorSystem`.

5. Lancer l'exécution

5.1. Lancer une exécution unique

Tout d'abord, s'assurer que le fichier `config/configLocation.txt` pointe bien vers le fichier de configuration à utiliser.

Ensuite, générer le .jar exécutable.

```
sbt monitor:assembly
```

Enfin, lancer le .jar crée.

```
java -jar target/scala-2.11/monitor.jar mapreduce.adpative.Monitor
```

5.2. Lancer un ensemble d'exécution à l'aide d'un méta fichier de configuration

5.2.1. Méta fichier de configuration

Il est également possible de lancer un ensemble d'exécution en faisant varier les paramètres des exécutions.

Pour cela, il faut créer un méta fichier de configuration, c'est-à-dire un fichier de configuration à partir duquel générer plusieurs fichiers de configurations différents.

La syntaxe d'un méta fichier de configuration est simple. Pour chaque entrée à faire varier, préfixer la ligne d'un caractère `*` et indiquer les différentes valeurs entre crochets, séparées d'un `;`. Par exemple, pour faire varier la stratégie de sélection de tâches utilisée :

```
* task-bundle-management-strategy : [ownership; (k-eligible-big, 2)]
```

La ligne précédente donnera lieu à deux fichiers de configuration différents. Un premier pour lequel la valeur associée au champ `task-bundle-management-strategy` sera `ownership`. Un second pour lequel la valeur sera `(k-eligible-big,2)`.

Un méta fichier de configuration représente l'ensemble des fichiers de configuration qui correspond au produit cartésien de chacun des champs de configuration marqué du caractère `*`.

5.2.2. Lancement de l'exécution

Il n'existe pour l'instant pas de .jar exécutable pour lancer un ensemble d'exécutions. Il faut donc utiliser la console de `sbt` pour être en mesure d'accéder à la classe `ExperimentsBuilder` du prototype.

```
$ sbt console
> import utils.experiments.ExperimentsBuilder
> val eb = new ExperimentsBuilder(5, "exp",
"config/configLocation.txt", "exp/config.txt")
> eb.runAdaptive
```

L'extrait de terminal précédent provoque les actions suivantes :

1. Ouvrir la console de `sbt`.
2. Importer la classe `ExperimentsBuilder` du prototype.
3. Construire une instance `eb` de `ExperimentsBuilder`. D'après ces paramètres :
   - `eb` lancera 5 exécutions par fichier de configuration généré à partir du méta fichier de configuration `exp/config.txt`.
   - `eb` stockera les résultats de chacune des exécutions dans le dossier `exp`.
   - `eb` écrira successivement le fichier de configuration courant dans le fichier `config/configLocation.txt`.
4. Lancer l'ensemble des exécutions avec `eb.runAdaptive`.