Rapport finale - Partie 1

David Diochot, Romain Égelé

November 24, 2017

1 Introduction

Tous notre service Hdfs est fonctionnel, c'est à dire que toutes les commandes sont disponibles et les serveurs gèrent la robustesse à différents type d'erreurs qui pouraient normalement arrêter leur exécution. Nous n'avons mis dans ce compte rendu ce qui nous semblait pertinent par rapport à l'utilité de notre service Hdfs pour le service Hidoop (ie. la fonction write). Nous avons rendu le service Hdfs disponible sur plusieurs machines. En effet la version expliquée dans notre rapport précédent décrivait un service Hdfs utilisant plusieurs serveurs mais ceci sur une même machine (les serveurs étaient alors différenciés par les ports).

2 Utilisation du service Hdfs

Pour lancer le service Hdfs, vous devez commencer par configurer le fichier de configuration (ie. src/config/config-serveurs.txt), il suffit de mettre le nombre de lignes correspondant au nombre de machines à lancer et sur chaque ligne indiquer le nom de l'hôte puis après un espace le numéro du port (exemple : carbone 8090). Une fois le fichier de configuration fait nous avons prévu un script bash pour faciliter le lancement du serveur, lancer le serveur en faisant à la racine du projet :

```
./run-hdfsserveur
```

Une fois que cette commande est lancée vous retrouverez la création de nouveaux dossiers à votre racine utilisateur (ie. pwd après un cd). Le premier dossier sera "hdfs" et à l'intérieur de ce dossier sont créés les dossiers "files-[hostname]" dans lesquels travaillent nos HdfsServeurs. Nous créons ces dossiers avec cette nomenclature car le système de fichier réplique les fichiers sur toutes les machines où nous sommes connectés.

Maintenant que les serveurs Hdfs sont initialisés nous pouvons utiliser le service via notre client. Pour lancer en ligne de commande il faut avoir compilé

les fichiers (avec eclipse de préférence pour que les .class se placent automatiquement dans le bin/, sinon remplacer dans la suite de l'explication, "bin" par "src"). Se placer dans le "./bin/" du projet. Vous pouvez alors lancer les commandes :

```
java hdfs.HdfsClient write [line|kv] [filename]
/* ou */
java hdfs.HdfsClient read [filename]
/* ou */
java hdfs.HdfsClient delete
```

Le "filename" indique le nom d'un fichier se trouvant dans le dossier "./data/" comme indiqué par le sujet.

3 Performance

Nous avons fait particulièrement attention à la performance de notre service Hdfs. Celui-ci servant à l'export d'un fichier en plusieurs fraguements afin d'appliquer une application de type MapReduce et ceci afin d'améliorer la performance du comptage de mots séquenciel fourni (ie. Count). Il est donc impératif que notre fonction "write" soit suffisament bien parallèlisée pour que sa seule execution soit bien inférieure au temps de calcul total de Count. Nos mesures sont les suivantes pour le fichier filesample.txt fourni dans les sources :

- Count : $t \approx 9$ms
- Write : $t \approx 1$ms

Pour atteindre cette performance nous avons fait très attention :

- aux délais de connexion des sockets que nous avons parallèlisés dans chaque Thread, ainsi le délai total de connexion à tous les sockets est quasiment égale au délai de connexion d'un unique socket.
- à la façon de compter les lignes du fichier source pour le fraguementer en différents chunks de même taille. Nous avons remarqué que suivant la fonction java utilisée (car la librairie Java fournie différentes possibilités) le temps de calcul augmente parfois énormément (jusqu'à 14ms pour compter le nombre de lignes du fichier ce qui rend le service hdfs inutile).
- à la bonne parallèlisation de l'envoie des données via les sockets.

Pour prendre les temps de calcul nous avons également fait attention au système de compilation à la volée de la JVM.