Shavadoop

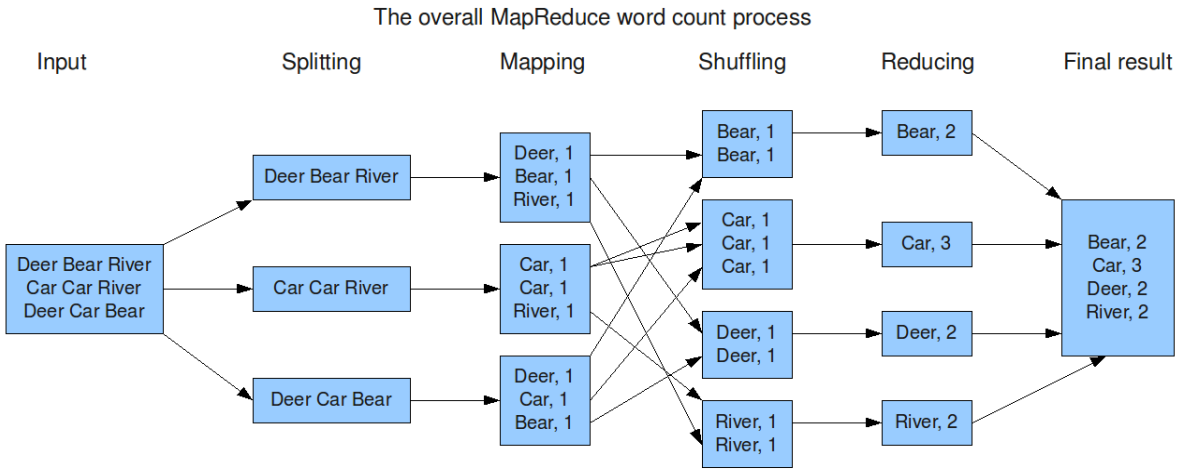
Une implémentation de MapReduce en Java
L'exemple du wordcount
par François Blas

Table des matières

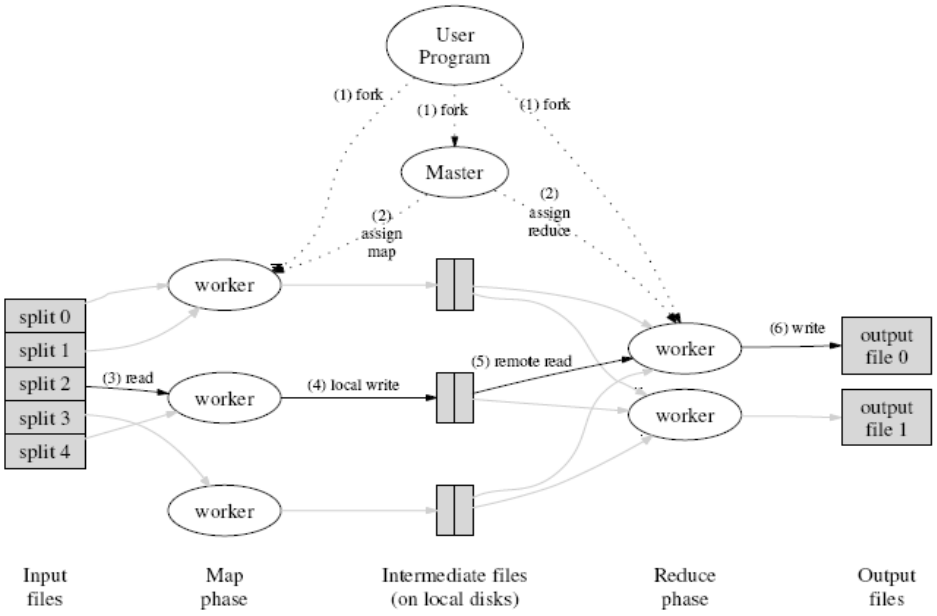
1. Contexte et objectifs

Le but du projet Shavadoop est d'implémenter en Java un « word count » en MapReduce, à la manière de Hadoop. Pour rappel, le « word count » est un programme permettant de compter le nombre d'occurrences de chaque mot présent dans un fichier et l'implémentation en MapReduce permet de faire ces opérations en calculs distribués sur un cluster de machines.

Le MapReduce est composé de 2 fonctions principales : La fonction de mappage et la fonction de reducer. Ici la fonction de reducer est composé de sous fonctions : celle de sorting et celle de shuffling.



L'opération de MapReduce s'effectue avec une architecture réseau très simple : un master et plusieurs slave (workers). Le rôle du master est de superviser les opérations et de lancer les programmes à distance, celui des workers est d'exécuter les opérations lancées par le master et de notifier ce dernier en cas de problème.



La communication entre master et workers se fait via SSH. Ce type de communication est très utile surtout quand il s'agit de lancer des commandes sur des machines distantes.

2. Organisation du code

Le code est organisé autour de 6 packages :

- Utils : Contient toutes les fonctions utiles du projet, comme par exemple la lecture ou écriture dans un fichier. Ce packge contient également la classe Configuration qui permet de régler certains paramètres du MapReduce.
- Ssh : Contient toutes les fonctions liées au réseau et au SSH (lancement de commandes, vérification du réseau et des postes...).
- Master : Contient tout ce qui est lié au master (dictionnaires, split, merge...).
- Slave : Contient tout ce qui est lié aux workers (map, reduce, sort, shuffle...).
- MapReduce : Contient le processus principal qui lance les opérations dans le bon ordre.
- Test : Contient le main pour tester le wordcount.

3. Comment fonctionne le code ?

3.1 Le master

En plus de posséder les 3 dictionnaires, le master se dote également de 3 méthodes qui permettent d'effectuer des opérations utiles au bon fonctionnement du programme général :

- Splitting : cette méthode coupe un fichier ligne par ligne. Indispensable pour le wordcount.
- CleanFolder : cette méthode va supprimer tout les fichiers présent dans le dossier des workers ou sont effectuées les opérations de calculs.
- MultipleMerge : cette méthode permet de faire un « cat » de plusieurs fichiers dans un seul. Très utile pour rassembler de l'info. La fonction « cat » des systèmes UNIX est extrêmement performante et puissante.

3.2 Les workers

Chaque grande étape du MapReduce possède un classe principale dont le nom correspond à l'étape en question , ainsi qu'une seconde classe finissant par un 'x'. Ces dernières sont des patrons de classe destines à être modifiés par les classes principales et copiés sur les workers.

Les grandes classes sont donc : Mapping, UnikWords, Shuffling et Reducing.

Les patrons associés (respectivement) sont : Umx, Uwx, Shx et Rex.

Rq : UnikWords permet de trier les mots et obtenir la liste des mots uniques.

De façon générale les classes principales vont écrire les fichiers sources (à partir des patrons) directement sur les workers, les compilées puis les exécutées. Cette opération est générée via des threads, chaque thread étant invoqué pour une machine en particulier. Par exemple, Mapping va créer un thread pour la machine ayant pour ip 192.168.1.100 et générer à partir du patron Umx, la classe UM0 avec des attributs de classe propre au thread. Le thread ensuite compile et exécute cette classe sur le worker.

Les classes principales ont toutes 3 méthodes qui se ressemblent :

- addEcho : qui permet l'ajout des « echo » pour lancer la commande bash a distance et pouvoir générer le fichier source (.java).
- CreateExevJava : qui permet de modifier les classes « patrons ».

- execEtape: qui permet de compiler et d'exécuter le fichier source sur le worker.

Toutes les étapes (sauf split et merge) sont réunies dans une super classe nomme Slave.

3.3 Le MapReduce

La classe MapReduce rassemble la classe master et la classe slave et exécute les étapes dans le bon ordre à partir d'un fichier texte. Les threads des différentes étapes sont invoqués via des boucles et un « join » est effectué en sortie de boucle afin d'attendre la fin de cette étape, avant de passer à la suivante.

Par ailleurs une recherche de machines sur le réseau est effectuée au début du processus mapreduce puis stockée sous forme de liste. Cette recherche est unique, il n'y a donc pas de vérification par la suite, ce qui peut poser problème.

4. Utilisation du code

4.1 Prérequis

Le code ne requiert qu'une librairie extérieure: Jsch, qui implémente le protocole SSH en Java. Le jar est disponible à cette adresse :
http://www.jcraft.com/jsch/

Par ailleurs, les connexions SSH se font via une clé, il est donc nécessaire d'avoir configurer l'intégralité du réseau utilise avant de lancer le mapreduce (les serveurs SSH).

4.2 Utilisation

Le code est très simple à utiliser. Il faut importer les sources dans un nouveau projet éclipse, puis importer le jar Jsch. Il faut ensuite ouvrir la classe Configuration (du package utils) qui permet la configuration des paramètres du programme. Description des paramètres :

- slavePath : c'est le dossier dans lequel l'ensemble des fichiers sources et compiles vont se trouver. Étant donne que le réseau de l'école possède un système de fichier partage, un seul dossier suffit.
- SshKey : correspond au path pour la clé SSH
- sshUser : correspond au nom d'utilisateur sur le réseau
- sshSubnet : correspond au sous réseau sur lequel s'effectue le mapreduce (adresse ip sans le dernier nombre).
- SshNumberOfHosts : correspond au nombre de machines effectuant le mapreduce.
- Debug : correspond au mode debug. S'il est active, chaque connexion ou opération est affiche en console.

Une fois la configuration terminée, rendez vous dans la classe Test du package test puis remplacer le path du fichier texte par celui que vous voulez mapreducer. Lancer le main de Test et c'est parti.

4.3 Lien Github

https://github.com/franblas/Shavadoop/