Pasquet Emmanuel

Thoumin Pauline

**Compte rendu Shavadoop**

**Note : La documentation technique, concernant le lancement du programme et les exemples d'Output à partir de fichiers input sont référencés dans le README du Git.**

1. **– Fonctionnement du programme (jusqu’à la question 37)**
2. RAPPEL : Les 2 modes successifs avec attente de terminaison

Le code du master se divise en 6 phases principales :

* Inventaire des ressources de traitement disponibles
* Répartition de l’input sur les ressources disponibles (Splitting)
* Map :

La connexion ssh « map » qui envoie au slave le nom d’un fichier Sx contenant un String. Le slave répond par une liste de mot-clé.

* Shuffle :

Récupération des fichiers contenant chaque clé et nouvelle répartition sur les machines.

* Reduce :

La connexion ssh « reduce » qui envoie au slave un mot-clé, le nom d’un fichier SMx, et la liste des fichiers UMx qui contiennent ce mot-clé. Le slave renvoie le mot-clé avec le nombre d’occurrence total.

* Assemblage

1. Architecture distribuée du traitement :

Pour gérer la distribution des traitements sur différentes machines esclaves :

* On distribue un fichier exécutable (le JAR slave) sur toutes les machines esclaves (en réalité, le système de fichiers partagés dispense de la transmission réelle du fichier, Cf. c) ).
* On coordonne l’exécution des différentes tâches par l’utilisation de threads. Les threads permettent d’orchestrer des tâches simultanées de durées différentes. Cette orchestration permet d’assurer l’exécution successive des différentes phases du programme.

Exemple : dans le cas présent, on attend que tous les threads de mapping soient terminés afin de de récupérer l’exhaustivité des informations de la première phase (les Unsorted Maps) avant de lancer la deuxième phase qui dépend de ce résultat.

* Le lancement de chaque thread est encapsulé par un processus qui consolide l’output du thread dans la sortie standard avec l’identifiant de la machine utilisée. Ainsi, lors du map, on sait rattacher les clés obtenues aux fichiers qui les contiennent, ce qui permet d’assurer le shuffle.

*Implémentation : pour récupérer la réponse du slave, un attribut supplémentaire (« sortie ») et son getter (« getSortie() ») dans la classe LaunchSlaveShavadoop ont été rajoutés. Le slave transfère ainsi une sortie sous forme d’ArrayList de String au master.*

1. Méthode d’appel des slaves par le master

L’exécution des programmes slaves par le master s’effectue en construisant une ligne de commande sous forme de String qui sera exécutée par la machine slave. La ligne de commande contient le nom du fichier exécutable ainsi que les arguments nécessaires pour l’exécuter. Le master la transmet aux slaves via ssh.

1. Méthode du splitting du fichier d’input.

Créer un fichier Sx par ligne du fichier d’input peut poser des problèmes de volumétrie de fichiers ainsi que de trop nombreuses connexions ssh. Pour éviter cela, le fichier d’input a été divisé équitablement en autant de parties qu’il y a de machines disponibles.

On réalise donc une seule connexion ssh par slave, qui s’occupe d’une fraction de texte.

Afin de simplifier le code du slave et que celui-ci n’ait qu’un split sur les espaces à effectuer, on crée le fichier Sx de manière à ce qu’il ne contienne qu’un String sans saut de lignes.

1. - Problèmes et solutions
2. -les optimisations et les singularités de votre projet

Peu d'optimisations ont été réellement apportées en cet état d'avancement du projet. Les aspects qui ont été identifiés sont les suivants :

* Accès disques (écriture et lecture) trop nombreux : à réduire en ne lançant pas un thread par clé dans la phase reduce mais un fichier de clés par thread.
* Connexions SSH trop nombreuses : à réduire
* Threads sans mécanisme de reprise sur panne (cas d'un slave qui tombe pendant le traitement)
* Implémenter une fonction pour effacer les fichiers créés
* Factorisation du code (créer des modules réutilisables)
* Généralisation du code à tout type de texte : fonctions de nettoyage, regex, etc.