Evaluation du Lot B

FARAH Othmane

OUALI ALAMI Anass

Partie technique :

Résultats des tests : Calcul du nombre d'occurrences des mots dans des fichiers de tailles différentes (Lancement en local) :

Les tests sont réalisés avec des données relativement petites (pas très volumineuses). Du coup, on ne peut pas observer le boost de performance qu'on peut obtenir avec ce type de service.

Nombre de lignes	468425	1536885	3090990	9272971
Version itérative : Count	503	1255	2057	6185
MyMapReduce	1970	3097	3772	9654

Correction (validation des fonctionnalités) :

Validation du contenu du répertoire ordo qui contient les fichiers concernant la partie Hidoop.

- **`WorkImpl.java` :** La méthode `runMap` fonctionne correctement. Cette méthode commence par l'ouverture des deux fichiers (supposés déjà créés) , applique la méthode map sur eux, informe que le map est fini à travers la méthode `FinishMap` de l'objet `CallBack` puis ferme les fichiers.

- **`CallBack.java` et `CallBackImpl.java` :** L'interface `CallBack` et son implémentation fonctionnent correctement. Elle fournit deux méthodes : `FinishMap` qui permet de savoir si un traitement map est terminé et `waitForRunMap` qui permet d'attendre le traitement de tous les map.

- **`Job.java` :** Cette classe fonctionne correctement. Elle nous permet de façon correcte de lancer les `runMap` sur tous les serveurs et puis d'appliquer reduce afin d'obtenir le résultat final.

Performances :

- On remarque que les performances s'améliorent avec des fichiers de taille plus grande. En effet, à partir d'un nombre de lignes très grand et un nombre de Worker correct, on pourra observer une amélioration en performances significative par rapport à la version itérative.

- Il serait préférable de lancer le `runMap` en parallèle en utilisant des threads. Dans la version actuelle, le worker se retrouve bloqué. Cela permettra également d'améliorer les performances.

Qualité du code :

- Le code n'est pas très aéré et n'est pas très bien commenté et documenté.

- Le code de cette version n'est pas très organisé et peut être factorisé.

Synthèse :

Correction :

- Comme nous l'avons vu précédemment dans la section **Correction** de la **partie technique**, tout semble fonctionner correctement.

- Cependant, cette version ne gère pas les exceptions (par exemple : fichier inexistant, erreur sur le nom …).

Complétude :

- Il était demandé de réaliser une application répartie sur plusieurs machines. Le service Hidoop réalisé pour cette version est fait pour fonctionner en local sur une seule machine, ainsi les workers sont lancés sur une seule machine ce qui n'est pas idéal au niveau des performances.
  Cependant, le fonctionnement réparti est bien simulé par cette version car chaque Worker est identifié par un numéro de port différent.

- Le démon Worker a été implémenté (Classe `WorkerImpl.java`) en utilisant RMI pour sa communication avec les clients.

- Le Callback utilisé dans Worker a été également implémenté en utilisant un sémaphore.

Pertinence :

Le travail présenté répond à ce qui est demandé :

- Chaque Worker lance un runMap sur fichier et nous donne un résultat sur un autre fichier

- Job lance dans chaque Worker un runMap.

- Callback nous permet de vérifier que tous les runMap sont terminés pour pouvoir faire un reduce.

Cohérence :

Le travail réalisé respecte l'architecture déjà définie et réalise le travail demandé.

Améliorations :

- Avoir du parallélisme dans le lancement des `runMap`.

- Les fichiers générés par les Map ne sont pas supprimés à la fin du reduce.

- Les tests fonctionnent en implémentant l'interface `JobInterface`. Il serait mieux de réaliser une version qui implémente `JobInterfaceX` qui contient des méthodes plus complexes comme par exemple : `setNumberOfReduces` … et qui pourraient apporter de nouvelles fonctionnalités.