Revue du travail réalisé par le binôme complémentaire

Armelle Jezequel
Loïs Roth
Paul Lacaille
Yannis Benabbi
Alexandre Chatain

15 janvier 2020

Table des matières

2ème année - Sciences du numérique

1 Revue de la partie Hidoop

1.1 Préambule

la partie hidoop du projet rendu au 03/12 est largement incomplète, et donc inexploitable. Dans ce cadre, cette revue sera principalement destiné à faire une point sur ce qui a été fait et sur ce qu'il reste à faire.

1.2 Partie technique

1.2.1 Avancement du code

Job :

- épurer le constructeur
- Dans la classe startJob : finir la 2e partie, c'est-à-dire lancer les runMaps sur les Daemon, attendre le Callback, récuperer les données avec hdfs et lancer l eereduce

DaemonImpl

- lancer le thread quelque part (il manque juste un t.start() dans runMap)

Callback

- ajouter un mutex sur la décrémentation .

1.2.2 Tests

Le code n'étant pas opérationnel, il n'est pas testable. Toutefois, nous avons tout de même défini les tests qu'il aurait fallu exécuter sur le code.

tests unitaires sur les classes : pour vérifier que chacune fait bien ce qu'elle est sensée faire :
–callback : tester que l'accès du callback est exclusif.
–DaemonImpl : tester que le thread secondaire tel que programmé soit correct (potentiel probleme avec l'appel à mapInterne).
–DaemonImpl/Job : tester les naming/rebind. Tester également que les runmap soient bien exécutés concurrentiellement, et non séquentiellement.
–callback/Job : tester que le notify/wait fonctionne bien.

tests du fonctionnement de l'ensemble de la partie hidoop Une fois que le code est écrit et compile, il faut tester au moins 2 scénarios simples .

scénario 1 : (permet de vérifier la bonne gestion des fichiers à travers les différentes étapes du map/reduce) :

1. lancer les Demons sur les bon ordinateurs avec les bon IDs (l'indice du nom de son ordinateur dans le tableau Projet. ? ? ?)
2. créer artificiellement des fragments de fichiers avec les bons numéros à la fin (numéro correspondant à l'ID du deamon) vide
3. lancer le startJob sur le serveur principal
4. vérifier que l'on crée un fichier vide avec le bon nom

scénarios 2 (permet de vérifier la bonne communication entre les démons et le job. On suppose que le RunMap fonctionne et n'est pas à tester, car il est fourni avec le sujet.) :

1. idem
2. créer artificiellement des fragments de fichiers avec les bons numéros à la fin, contenant respectivement :
   - fichier.txt-1 : " <motTest, 2> "

— fichier.txt-2 : “ <motTest, 3> ”

3. idem
4. vérifier que l'on crée un fichier contenant " <motTest, 5>" avec le bon nom : fichier.text-res

1.3 Synthèse

1.3.1 Evaluation du travail

Correction : Le produit ne fonctionne pas, puisqu'il manque beaucoup de code. On ne peut donc pas évaluer si les résultats sont justes ou non.

Complétude : Le binôme hidoop a décidé d'utiliser le protocole de communication RMI (comme proposé) pour la communication entre son serveur principal et les noeuds de son cluster. Pour que ce protocole puisse être opérationnel, chaque classe a besoin d'être "interfacée". Le dossier ordo contient donc :

1. Des interfaces (fournies, sauf pour le Callback) :
   — Daemon
   — JobInterface
   — jobx (fournie mais non utilisée)
   — Callback
2. Une implémentation pour chacune de ces interfaces (créé par le binôme) dans l'ordre :
   — DaemonImpl
   — Job
   — CallbackImpl

Sur chaque ordinateur du cluster , il faut lancer un processus DaemonImpl, et sur le serveur principal, il faut lancer un processus Job (comme décrit dans le sujet) . Dans un premier temps de développement du projet, nous avons arbitrairement décidé que le cluster serait composé de 2 ordinateurs : Leia et Luke (noms stockés dans un tableau attribut de la classe Projet), il faut donc lancer les DaemonImpl en SSH. Le serveur principal peut être n'importe quel ordinateur.

Pour autant que nous puissions en juger, cette architecture correspond à la spécification demandée. Elle n'est toutefois pas suffisamment implémentée pour que nous puissions juger des détails.

Pertinence : Voyons le déroulé du programme. Lorsqu'on veut lancer un mapReduce sur un ficher stocké dans le cluster, il faut lancer un startJob sur le Job du serveur principal. Ensuite, voilà ce qu'il se passe (si on prend en compte les commentaire dans le code) :

- Le Job crée un Callback (Remote object), avec en paramètre le nombre de DaemonImpl appelé.
- Le Job demande à chaque DeamonImpl d'exécuter le map sur leur bout du fichier, et leur donne un lien vers le Callback.
- Chaque DaemonImpl commence par créer un thread secondaire pour rendre la main au Job immédiatement (afin qu'il appelle les DaemonImpl suivants), pendant que le thread secondaire effectue le map.
- Job attend que le Callback le réveille.
- Quand les DeamonImpl ont fini le map, ils le signalent au Callback.
- Quand tout les DeamonImpl se sont signalés au Callback, celui-ci réveille le Job.
- Le Job lance le reduce et produit le fichier résultat final.

On a bien une exécution concurrentielle des map sur les différents noeuds du cluster, l'attente du callback, mais il manque la récupération des fichiers fragment de résultat avec hdfs. Globalement, on est bien dans un schéma map/reduce (parallélisation du travail sur differentes machines) comme voulu dans le projet.

Cohérence : La conception nous semble dans l'ensemble assez claire. Même si certaines choses auraient gagnées à être faites autrement (cf partie suivante).

1.3.2 Pistes d'amélioration

parallélisation des maps : en l'état, la façon de faire est mauvaise. En effet, le thread secondaire est créé en redéfinissant la méthode run.
Une piste d'amélioration serait de : soit définir le thread d'une autre manière, soit de gérer toute la parallélisation dans le job, avec un système de barrières par exemple.

Gestion des noeuds du cluster : Dans la version actuelle, le lancement des threads a de grandes chances d'être source d'erreurs : il faut lancer le thread sur la bonne machine (en SSH éventuellement), en lui passant en paramètre le bon ID (l'indice de son nom dans le tableau Project.nomDeamon).
Une piste d'amélioration serait de commencer avec un tableau vide de taille suffisamment grand, et les Daemon indiqueraient au moment de leur création dans Project.nomDeamon le nom de la machine sur lequel ils sont. Avec un accès au tableau en exclusion mutuelle pour ne pas qu'il y ait pas de problème d'écritures simultanées.