Projet systèmes concurrents et communicants
Implantation et utilisation du modèle Map-Reduce

2 SN

21 novembre 2020

Ce projet propose une première expérience sur le thème des applications concurrentes pour le calcul intensif et le traitement de masses de données. Pour cela, il s'agira

- de réaliser une plateforme comportant un système de fichiers adapté au traitement concurrent de masses de données, ainsi qu'un noyau d'exécution axé sur l'ordonnancement et la gestion de tâches selon le schéma « diviser pour régner » (*map-reduce*). L'architecture et les fonctionnalités de cette plateforme reprendront (de manière simplifiée) celles de la plateforme Hadoop.
- de permettre de tester et évaluer cette plateforme
  - de développer des outils d'instrumentation et de supervision de l'exécution des applications tournant sur la plateforme.
  - au travers de son utilisation par des applications concurrentes utilisant de manière intensive les ressources mémoire et/ou de calcul

Le projet sera organisé en deux étapes principales :

- La première étape consistera à développer une version minimale de la plateforme, qui fournira les services de base en termes de gestion des accès à des fichiers fragmentés, et d'ordonnancement pour les tâches map-reduce. La plateforme développée sera validée sur le plan fonctionnel, au travers d'un jeu d'essai fourni, d'un test final et par le développement d'applications élémentaires.
- La deuxième étape proposera une nouvelle version de la plateforme, axée sur l'amélioration des performances, avec la mise en œuvre et l'évaluation de stratégies adaptées pour la gestion des accès et des tâches dans un environnement parallèle et réparti. Cette version sera validée au travers d'un jeu d'essai fourni, et d'un test final, axés sur des mesures de performance.

Note : Seule la première étape devra être réalisée dans le cadre de l'UE « Systèmes Concurrents et Communicants ». L'étape suivante sera développée au semestre 8, dans le cadre de l'UE « Applications Concurrentes et Communicantes ».

Plan

- La section 1 détaille le contenu attendu de la plateforme.
- La section 2 précise les éléments de l'environnement attendu pour la plateforme : applications, supervision, évaluation.
- La section 3 donne les modalités pratiques : organisation, échéances, livrables....
- Les fournitures, ainsi que des documents complémentaires, sont disponibles sur la **page Moodle** de la matière :
  - **classes utiles (formats. . .) et interfaces requises** ;
  - code de l'application de comptage de mots (versions itérative et map-reduce) ;
  - **jeux de données utilisés pour l'évaluation** ;
  - architecture de principe et fonctionnalités essentielles de la plateforme Hadoop ;
  - **règles et éléments requis pour le suivi et la conduite du projet en groupe** ;
  - exemples d'applications
  - ...

1 Mise en œuvre de la plateforme

1.1 Le schéma MapReduce

Le schéma MapReduce permet d'effectuer en parallèle (sur une grappe (*cluster*) de machines) des traitements sur un grand volume de données. Les données sont découpées en fragments qui sont répartis (stockés) sur les machines de traitement. Les fragments sont alors traités en parallèle sur les différentes machines, et les résultats partiels issus du traitement des fragments sont alors agrégés pour donner le résultat final.

Ainsi, une application Map-Reduce spécifie essentiellement

— une procédure map(), qui est exécutée en parallèle sur les machines de traitement sur tous les fragments (auxquels on accède donc localement),

— et une procédure reduce() qui exploite le résultat des map pour obtenir le résultat final.

1.2 Hidoop

Cette implantation sera composée de deux services :

— un service HiDFS (Hidoop Distributed File System). Il s'agit d'un système de gestion de fichiers répartis dans lequel un fichier est découpé en fragments, chaque fragment étant stocké sur un des nœuds du cluster.

— un service Hidoop contrôlant l'exécution répartie et parallèle des traitements map, la récupération des résultats et l'exécution du reduce.

Chacun de ces services devra proposer, outre une implémentation de l'API, une interface en ligne de commande pour la gestion des fichiers et des tâches.

Pour cette version,

— on pourra considérer que la plateforme est réservée à l'exécution d'une application unique.

— La tolérance aux pannes ne sera pas prise en compte : les blocs gérés par HDFS ne seront pas dupliqués, et les pannes de nœuds ne seront pas gérées.

— L'espace de nommage des fichiers HDFS sera plat : en d'autres termes, la notion de répertoire ne sera pas implantée/proposée.

— La taille des blocs sera variable. Elle pourra être gérée de telle sorte qu'un enregistrement ne se trouve jamais à cheval sur 2 blocs. Egalement, la taille des blocs pourra être adaptée de sorte qu'un document soit fragmenté en N blocs distribués sur N noeuds avec 1 bloc par noeud.

— Les métadonnées du système de fichiers seront conservées et gérées via un simple fichier implanté sur la machine contrôlant l'exécution de l'application (c.-à-d. le service Hidoop).

— Les applications ne comporteront pas de shuffle et n'utiliseront donc qu'une tâche reduce.

1.2.1 La gestion de format

Chaque bloc (*chunk*) contient une suite d'enregistrements, dont le format est spécifique à l'application. Pour chaque format, une classe doit être définie, qui permet de lire et écrire dans un fichier dans ce format. Ces classes de lecture/écriture de formats sont très importantes, car étant donné que les fichiers peuvent être coupés en fragments par HDFS, il faut qu'HDFS sache faire une coupure cohérente. En particulier, un fichier de structures ne peut pas être coupé au milieu d'une structure. Dans ce qui suit, on ne considère que 2 formats : le format texte (LINE) et le format Clé-Valeur (KV), mais d'autres formats peuvent être envisagés et introduits par la suite.

Une classe implantant un format implante l'interface Format.

```
public interface Format extends FormatReader, FormatWriter, Serializable {
        public enum Type { LINE, KV };
        public enum OpenMode { R, W };
        public void open(OpenMode mode);
        public void close();
        public long getIndex();
        /* position courante (octet) sur le fichier traite */
        public String getFname();
        public void setFname(String fname);
        /* fname identifie (dans le systeme de fichiers local)
         * le fichier/fragment sur lequel opere le Format */
}
public interface FormatReader {
        public KV read();
}
```

```
public interface FormatWriter {
        public void write(KV record);
}
```

Le format KV correspond au format des données intermédiaires produites durant l'exécution de l'application MapReduce.

```
public class KV {
        public static final String SEPARATOR = "<->";
        public String k;
        public String v;
        public KV() {}
        public KV(String k, String v) {
                super();
                this.k = k;
                this.v = v;
        }
        public String toString() {
                return "KV [k=" + k + ", v=" + v + "]";
        }
}
```

Une implantation des classes KVFormat et LineFormat réalisant les formats texte et clé-valeur est fournie.

1.2.2 Le service HDFS

Le système de fichier est composé d'un démon (HdfsServer) qui doit être lancé sur chaque machine. Une classe HdfsClient permet de manipuler les fichiers dans HDFS. Voici un squelette de cette classe :

```
public class HdfsClient {
  public static void HdfsDelete(String hdfsFname) {...}
  public static void HdfsWrite(Format.Type fmt, String localFSSourceFname,
     int repFactor) {...}
  public static void HdfsRead(String hdfsFname, String localFSDestFname) {...}
  public static void main(String[] args) {
    // appel des methodes precedentes depuis la ligne de commande
  }
}
```

La classe HdfsClient peut être utilisée depuis la ligne de commande ou directement depuis du code pour lire, écrire ou détruite des fichiers :

- `HdfsWrite(Format.Type fmt, String localFSSourceFname, int repFactor)` permet d'écrire un fichier dans HDFS. Le fichier localFSSourceFname est lu sur le système de fichiers local, découpé en fragments, et les fragments sont envoyés pour stockage sur les différentes machines. fmt est le format du fichier (Format.Type.LINE ou Format.Type.KV). repFactor est le facteur de duplication des fragments ; pour cette version il sera considéré comme valant 1 (pas de duplication).
- `HdfsRead(String hdfsFname, String localFSDestFname)` permet de lire un fichier de nom hdfsFname à partir de HDFS. Les fragments du fichier sont lus à partir des différentes machines, concaténés et stockés localement dans un fichier de nom localFSDestFname.
- `HdfsDelete(String hdfsFname)` permet de supprimer les fragments d'un fichier stocké dans HDFS.

La liaison entre les noms de fragments (chunks) dans le système de fichiers local et le nom du fichier HDFS (`NameNode`)peut être implantée

- par une convention de nommage documentée et réalisée « en dur » directement dans le code des méthodes des classes du service HDFS (`HdsfsClient` notamment) et du service Hidoop (`Job` notamment) ;
- par un fichier accessible aux services et aux applications ;
- par un serveur spécifique.

Nous recommandons d'utiliser les sockets en mode TCP pour implanter la communication entre HdfsClient et HdfsServer. Chaque interaction entre HdfsClient et HdfsServer sera alors initiée par l'envoi d'un message spécifiant la commande à exécuter par HdfsServer. Une syntaxe minimale pour ces messages pourrait être :

— le code commande,
— suivi de la taille du premier nom de fichier,
— suivi du premier nom de fichier
— (suivi éventuellement (pour une version ultérieure) du facteur de réplication, ou de la taille du second nom de fichier, et du second nom de fichier)

Le code commande pourra être défini par un type énuméré comme :

```
public enum Commande {CMD_READ, CMD_WRITE, CMD_DELETE};
```

1.2.3 Le service Hidoop

Le service Hidoop fournit le support pour l'exécution répartie. Un démon (Worker) doit être lancé sur chaque machine. Nous proposons d'utiliser RMI pour la communication entre ce démon et ses clients. L'interface du démon est interne à Hidoop, donc libre à vous de la définir. Voici toutefois une proposition d'interface du démon, **pour la gestion des tâches Map** :

```
public interface Worker extends Remote {
  public void runMap (Mapper m, Format reader, Format writer, CallBack cb)
                throws RemoteException;
}
```

Les paramètres sont les suivants :

— Mapper m : il s'agit du programme map à appliquer sur un fragment hébergé sur la machine où s'exécute le démon.
— Format reader : il s'agit du fichier (dans un format donné par la classe de reader) sur la machine où s'exécute le démon, contenant le fragment sur lequel doit être appliqué le map.
— Format writer : il s'agit du fichier (dans un format donné par la classe de writer) sur la machine où s'exécute le démon, dans lequel les résultats du map doivent être écrits.
— CallBack cb : il s'agit d'un objet de rappel, appelé lorsque l'exécution du map est terminée.

Pour lancer un calcul parallèle, Hidoop fournit une classe Job, implantant une interface JobInterface et fournissant en particulier une méthode startJob :

```
public class Job {
        public void setInputFormat(Format.Type format) {...}
        public void setInputFname(String fname) {...}
        public void startJob (MapReduce mr) {...}
}
```

— MapReduce mr : correspond au programme MapReduce à exécuter en parallèle (voir modèle de programmation),
— Format.Type format : indique le format du fichier en entrée (Format.Type.LINE ou Format.Type.KV),
— String fname : le nom du fichier HDFS contenant les données à traiter.

Le comportement de startJob est de lancer des map (avec runMap) sur tous les démons des machines, puis attendre que tous les map soient terminés. Les map ont généré des fichiers locaux sur les machines qui sont des fragments. On peut alors récupérer le fichier global avec HDFS, puis appliquer le reduce localement. Le résultat final est un fichier dont le nom est dérivé de fname (par exemple par l'ajout du suffixe "-res".)

1.2.4 Le modèle de programmation

Le modèle de programmation MapReduce permet de définir une classe implémentant l'interface MapReduce et fournissant les méthodes map() et reduce(). Une procédure map() lit les données (des KV) provenant de son fragment local avec le paramètre reader, et écrire ses résultats (des KV) dans un fragment local avec le paramètre writer. Une procédure reduce() lit également des KV avec reader et écrit des KV sur writer, mais Hidoop la fait opérer sur un fichier unique et local à la machine de lancement du programme (ce fichier est le résultat des map extrait de HDFS).

```
public interface Mapper extends Serializable {
        public void map(FormatReader reader, FormatWriter writer);
}

public interface Reducer extends Serializable {
        public void reduce(FormatReader reader, FormatWriter writer);
}

public interface MapReduce extends Mapper, Reducer {
}
```

1.2.5 Exemple d'application

L'annexe B est un exemple d'application de comptage des occurrences de mots dans un fichier, programmée par MapReduce. En annexe A, vous trouvez le même programme en itératif.

2 Utilisation et évaluation

2.1 Configuration

Il sera sans doute utile de développer des outils permettant d'initialiser Hidoop à partir d'un fichier de configuration fournissant les nœuds et ports utilisés par les différents serveurs.

2.2 Outils de supervision

Les applications tournant sur la plateforme Hidoop, et la plateforme Hidoop elle-même, devront être testées et évaluées de manière suffisamment complète. A cette fin, il sera utile de développer un certain nombre d'outils *simples*, permettant :

- de jouer des scénarios interactifs (interpréteur de commandes) ou prédéfinis (simulateur),
- d'évaluer (sommairement) les performances des applications exécutées ;
- de consulter et de fixer des paramètres relatifs aux ressources utilisées comme la taille (maximale ou courante) de l'espace utilisé pour les calculs intermédiaires, la taille des files d'attente, le nombre de tâches map ou reduce actives ou échues.
- ...

Notez que

- cette liste n'est qu'indicative : elle n'est ni exhaustive, ni prescriptive. Cependant, il est attendu qu'un certain nombre de ces outils soient développés et utilisés.
- l'interface Worker fournit une interface ascendante (la classe Callback) qui peut être utilisée à des fins d'instrumentation, même si ce n'est pas son usage principal.

2.3 Applications

La plateforme servira de support à l'exécution de différentes applications concurrentes. L'objectif sera d'une part d'évaluer différents choix de conception et d'architecture pour les applications concurrentes et d'autre part de permettre de comparer et valider les différentes versions de Hidoop.

2.3.1 Tests élémentaires

La validation de la plateforme s'appuiera sur des tests et scénarios de base afin d'en évaluer les aspects fonctionnels et les performances. Ces tests mettront en jeu des threads réalisant différentes configurations d'accès, sans que ces accès aient forcément une signification particulière.

Les *tests fonctionnels* reposeront sur la méthodologie et les outils de test – comme Junit – vus en première année, dans le cadre des enseignements de programmation, ainsi que sur la définition d'un ensemble de scénarios d'intégration suffisamment représentatif de situations d'utilisation mettant en jeu des activités concurrentes. L'objectif de ces scénarios est de valider la cohérence et la vivacité des opérations d'accès aux données.

La validation du projet pour ce ce semestre reposera pour l'essentiel sur cette classe de tests.

Les *tests de performances* visent à permettre de comparer les différentes versions développées et d'évaluer leurs limites en termes (quantitatifs) de volumes de données traitées ou conservées, de débit d'opérations concurrentes, de nombre de processus concurrents possibles, de temps d'exécution... Ces tests reposeront sur un jeu d'applications de référence *simples*, qui pourront être inspirées de ce qui a été fait dans le cadre des TPs sur les threads ou la concurrence.

Pour cette phase, le développement d'outils *simples* de supervision ou d'instrumentation paraît particulièrement approprié.

La suite du projet, menée au **semestre suivant**, se concentrera sur cette classe de tests.

2.3.2 Schémas et problèmes de base

Une bonne manière de se familiariser avec le modèle de programmation Map-Reduce consiste à réaliser dans ce modèle des exercices classiques traités précédemment avec d'autres outils :

— comptage de mots dans un texte
— analyses statistiques sur des journaux, des mesures...
— tri externe
— recherche d'expressions régulières (grep)

3 Modalités pratiques

Note : Les informations qui suivent sont susceptibles d'évoluer (légèrement), quelques précisions pouvant être apportées, en fonction des circonstances et/ou des questions/demandes des différents groupes. Ces modifications éventuelles seront alors publiées sur la **page Moodle** de la matière.

3.1 Structure du projet

Le projet à réaliser pour ce semestre comporte 2 lots :
- lot A : HDFS
- lot B : Hidoop

Outre les fonctionnalités de base, chacune de ces parties devrait voir ses fonctionnalités accessibles via une interface en ligne de commande, et offrir/prévoir des possibilités d'instrumentation.

3.2 Constitution des groupes

Projet réalisé en binôme de binômes. Les binômes sont constitués au sein d'un même groupe de TD. Les binômes d'un groupe de TD sont répartis en un nombre égal de binômes "A" et de binômes "B" [1]. Les "A" et les "B" sont enfin appariés en groupes. Un groupe est constitué par un "A" et un "B", avec la contrainte que l'effectif du groupe ne peut être supérieur à 5.

La constitution des binômes/trinômes, des groupes et le choix des lettres sont libres, sous réserve du respect des contraintes précédentes. Les cas de blocage ou l'absence de choix seront réglés par tirage au sort.

Les "A" mèneront le lot A, et évalueront le lot B
Les "B" évalueront le lot A, et mèneront le lot B
La version finale sera réalisée par le groupe complet.

3.3 Déroulement

Le projet se déroule en trois temps :

1. Chaque binôme réalise une première version de son lot. A l'issue de ce travail, il remet :
   — un rapport provisoire succinct présentant l'architecture, les algorithmes des opérations essentielles, une explication claire des points délicats et de leur résolution.
   — le code complet de la partie réalisée.
2. Ce travail n'est pas noté mais il est transmis, pour évaluation et validation, à l'autre binôme/trinôme du groupe. Cette étape d'évaluation consiste à :
   — valider le code fourni et en évaluer les performances par des tests élémentaires ;
   — réaliser une revue critique du code et des choix de conception effectués ;
   — proposer des améliorations, ou non. Les propositions d'amélioration, ou la validation du code et des choix existants doivent être étayés dans tous les cas.

   Cette évaluation aboutira à un rapport qui sera remis et transmis à l'autre binôme/trinôme du groupe. Ce rapport est noté.
3. Enfin, le groupe devra produire la version finale, qui intègre les 2 parties en tenant compte (ou non) des remarques et propositions issues de l'évaluation. A l'issue de ce travail, le groupe remet une archive contenant
   — un rapport succinct présentant l'architecture, les algorithmes des opérations essentielles, une explication claire des points délicats et de leur résolution, ainsi qu'une comparaison de performances visant à entre déterminer le nombre de map à partir duquel la version MapReduce de l'application Wordcount est plus rapide que sa version séquentielle.
   — le code complet de la partie réalisée.

   Ce travail est noté.

Remarque importante Si les lots sont clairement séparés, il est tout aussi clair qu'ils sont interdépendants. Les binômes d'un même groupe auront donc tout intérêt à **communiquer souvent** (et pas uniquement au moment de l'évaluation), pour faciliter l'intégration de leur travail au sein de la version finale. Dans tous les cas, chaque binôme peut progresser en définissant si besoin une interface « factice » dont l'implémentation est composée de méthodes vides, pour

1. Comme pour le bridge, cela suppose que l'effectif du groupe de TD est multiple de 4. Si ce n'est pas le cas, un nombre de trinômes égal au reste de la division entière par 4 de l'effectif du groupe de TD sera constitué.

représenter les fonctions attendues de l'autre binôme. Cette interface factice peut constituer une base pour la communication et l'intégration ultérieure.

3.4 Echéancier

21/11 constitution des groupes, binômes...
29/11 remise de l'architecture de principe
10/12 remise des rapports et livrables provisoires
17/12 remise des évaluations
17/1 remise des rapports et livrables finaux
19/1 et 20/1 présentation/échange avec le groupe sur le travail réalisé

3.5 Séances de suivi

Plusieurs séances de suivi sont programmées, avec (par défaut) les objets suivants :
25/11 : validation des groupes, questions sur le sujet ;
2/12 : validation de l'architecture de principe, questions sur la mise en œuvre ;
9/12 : questions sur la mise en œuvre ;
18/12 : retours sur la première version ;
6/1 et 7/1 : questions sur la mise en œuvre ;

Pour chaque groupe, Il est demandé de fournir un **point d'avancement** avant chaque séance de suivi (sauf pour la séance du 25/11). Le contenu et les modalités sont précisées sur la **page Moodle** de la matière.

3.6 Fournitures

Les différents documents et fournitures utiles sont (ou seront) mis en ligne sur la page Moodle de l'enseignement, au fur et à mesure de la progression du projet, et notamment :

- les attentes quant au fonctionnement et à l'organisation des groupes ;
- le format et le contenu attendu pour vos différents livrables : points d'avancement, évaluation de lot, code et rapport final ;
- un générateur de jeux d'essais pour l'application de comptage de mots ;
- une archive contenant en particulier
  - l'arborescence de fichiers dans laquelle devront se trouver les livrables ;
  - le code source des interfaces, classes et squelettes de base, exemples élémentaires ;

3.7 Notation

Les notes des évaluations seront attribuées par binôme/trinôme, et la note finale sera attribuée au groupe, sauf dysfonctionnement avéré [2].

Pour ce qui est des parties de réalisation, la notation portera sur le contenu du rapport final et sur nos propres évaluations, lectures et tests du code fourni. A ce propos, **il est impératif que l'API qui vous est fournie soit strictement respectée**, afin que nos tests puissent fonctionner. Ainsi, le code des exemples qui vous sont fournis doit compiler et s'exécuter correctement (c.-à-d. donner le même résultat que celui fourni) *sans que vous y touchiez le moindre caractère.*

Pour ce qui est des parties d'évaluation, les deux éléments pris en compte seront

- le caractère complet de la démarche de validation
- la pertinence de la revue critique et des propositions d'amélioration présentées, par rapport à notre propre évaluation du rapport provisoire, dont nous disposerons : si les rapports provisoires ne sont pas notés en eux-mêmes, ils servent de base pour noter les évaluations.

2. En cas (exceptionnel) d'anomalies importantes constatées dans le fonctionnement/travail du groupe ou du binôme/trinôme, les notes seront individualisées.

Annexe A (comptage de mots en itératif)

```
public class Count {
        public static void main(String[] args) {
                try {  long t1 = System.currentTimeMillis();
                        HashMap<String,Integer> hm =
                          new HashMap<String,Integer>();
                        LineNumberReader lnr =
                          new LineNumberReader(
                            new InputStreamReader(new
                               FileInputStream(Project.PATH+"data/"+args[0])));
                        while (true) {
                                String l = lnr.readLine();
                                if (l == null) break;
                                String tokens[] = l.split(" ");
                                for (String tok : tokens) {
                                        if (hm.containsKey(tok))
                                          hm.put(tok, hm.get(tok).intValue()+1);
                                        else hm.put(tok, 1);
                                }
                        }
                        BufferedWriter writer = new BufferedWriter(
                                        new OutputStreamWriter(new
                                            FileOutputStream("count-res")));
                        for (String k : hm.keySet()) {
                                writer.write(k+"<->"+hm.get(k).toString());
                                writer.newLine();
                        }
                        writer.close();
                        lnr.close();
                        long t2 = System.currentTimeMillis();
                        System.out.println("time in ms ="+(t2-t1));
                } catch (Exception e) { e.printStackTrace(); }
        }
}
```

Annexe B (comptage de mots en MapReduce)

```
public class MyMapReduce implements MapReduce {
        private static final long serialVersionUID = 1L;

        // MapReduce program that computes word counts
        public void map(FormatReader reader, FormatWriter writer) {
                Map<String,Integer> hm = new HashMap<>();
                KV kv;
                while ((kv = reader.read()) != null) {
                                String tokens[] = kv.v.split(" ");
                                for (String tok : tokens) {
                                if (hm.containsKey(tok)) hm.put(tok,
                                    hm.get(tok).intValue()+1);
                                else hm.put(tok, 1);
                        }
                }
                for (String k : hm.keySet()) writer.write(new
                    KV(k,hm.get(k).toString()));
        }

        public void reduce(FormatReader reader, FormatWriter writer) {
                Map<String,Integer> hm = new HashMap<>();
                KV kv;
                while ((kv = reader.read()) != null) {
                        if (hm.containsKey(kv.k)) hm.put(kv.k,
                            hm.get(kv.k)+Integer.parseInt(kv.v));
                        else hm.put(kv.k, Integer.parseInt(kv.v));
                }
                for (String k : hm.keySet()) writer.write(new
                    KV(k,hm.get(k).toString()));
        }

        public static void main(String args[]) {
                Job j = new Job();
                j.setInputFormat(Format.Type.LINE);
                j.setInputFname(args[0]);
                long t1 = System.currentTimeMillis();
                j.startJob(new MyMapReduce());
                long t2 = System.currentTimeMillis();
                System.out.println("time in ms ="+(t2-t1));
                System.exit(0);
        }
}
```