Indexation et recherche de contenu utilisant MapReduce de Google

Rapport final

Elodie Corbel, Kévin M'Ghari,
Mickaël Olivier, Clarisse Renou

Encadrant : Alexandru Costan

Résumé

Ce rapport a pour but de mettre en avant le travail effectué par 4 élèves-ingénieurs en troisième année informatique à l'INSA de Rennes. Nous avons effectué un moteur de recherche assez simple avec une interface graphique et un système d'indexation. Nous allons donc vous expliquer comment nous nous sommes organisés pour effectuer ce projet. Ce projet comporte aussi une présentation du modèle de programmation MapReduce de Google que nous avons étudié avant de réaliser le moteur de recherche et que nous avons utilisé pour l'indexation.

Remerciements

Nous tenous tout d'abord à remercier Alexandru Costan, professeur à l'INSA de Rennes, nous encadrant lors de ce projet. Il a sû nous donner de précieux conseils et nous guider tout au long de l'année aussi bien que nous procurer un sujet d'étude riche et intéressant. Nous remercions aussi le personnel de l'INRIA toujours souriant et très aimable, nous aillant accueilli lorsque nous allions rendre visite à Monsieur Costan.

Introduction

Dans le cadre des études pratiques, en troisième année au département informatique à l'INSA de Rennes, nous avons été amené à réaliser un projet tout au long de l'année. Celui que nous avons choisi porte sur l'indexation et la recherche de contenu utilisant MapReduce de Google[1]. Nous avons choisi ce sujet car la recherche de contenu et Google sont pour nous quelque chose d'inévitable à partir du moment où un utilisateur utilise un ordinateur. En effet, Google, connu surtout pour son très populaire moteur de recherche, est un phénomène à lui tout seul, il représente 6,4% du traffic Internet mondial en 2010[3]. Connaître et utiliser un modèle de programmation tel que MapReduce conçu par Google était donc pour nous très motivant.

Le principe de la recherche de contenu est assez simple. Il existe des documents structurés ou non dans lesquels des personnes souhaitent effectuer une requête auprès d'un serveur. Le serveur renvoie donc les documents dans lesquels se trouvent les mots sur lesquels porte la requête. L'indexation permet d'améliorer la rapidité et les performances d'une recherche de contenu. En effet, celle-ci identifie les éléments significatifs

du document afin de permettre un accès plus rapide à ceux-ci en créant un index. Lors d'une recherche, le moteur de recherche va d'abord chercher les informations dans l'index puis rend les documents à l'utilisateur.

Nous avons donc pour mission de faire un moteur de recherche avec son système d'indexation utilisant MapReduce de Google

Dans un premier temps, nous allons vous expliquer les solutions que nous avons choisi et la façon dont nous avons décomposé le travail. Puis, nous vous expliquerons comment nous avons conçu notre moteur de recherche. Et enfin, nous expliciterons les résultats obtenus.

1 Etude du projet

Afin de déterminer ce que nous devions faire, dans une première partie de l'année, nous avons étudier la littérature existante sur le fonctionnement de MapReduce de Google afin de mieux comprendre notre objectif. Puis, nous avons découpé notre travail. Nous allons donc dans une première partie expliquer le fonctionnement de MapReduce et la solution choisie pour implémenter ce modèle de programmation. Puis, nous parlerons de la répartition du travail.

1.1 Présentation de MapReduce

MapReduce est un modèle de programmation popularisé par Google. Il est utilisé pour l'indexation de contenu. Il se repose sur deux fonctions une fonction `Map` et une fonction `Reduce`.

La fonction `Map` prend en entrée une clé et des valeurs associées. Par exemple, pour un fichier, la clé peut être un numéro de ligne et la valeur le texte de la ligne. Cette fonction ensuite rend une clé intermédiaire et une valeur intermédiaire. Dans l'exemple du Wordcount (voir *figure 1*), dans laquelle on veut compter le nombre de chaque mot, la clé intermédiaire peut être un mot et la valeur 1.

La fonction `Reduce` ensuite prend les clés et valeurs intermédiaires données par la fonction `Map` et aggrège le résultat afin de donner une clé finale et un résultat final. Dans l'exemple du Wordcount, la clé serait un mot et le résultat serait le nombre d'occurences de ce mot.[2]

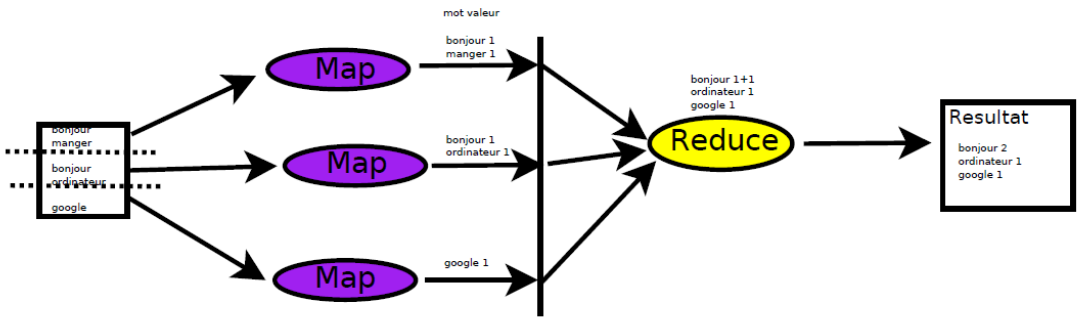

FIGURE 1 – Schema illustratif MapReduce

Il est très efficace pour le traitement de données importantes mais le résultat n'est pas immédiat. Il est donc utilsé en tâche de fond. Pour une même tâche MapReduce, plusieurs fonctions `Map` peuvent s'exécuter en même temps sur différentes parties de données ce qui rend l'algorithme plus efficace. Il en va de même pour les fonctions `Reduce`.

MapReduce est très utilisé, des implémentations en ont donc été faites, la plus utilisée est Hadoop de Apache. C'est un framework qui utilise le langage de programmation java et qui a son propre système de fichiers sous lequel doivent être placés les fichiers à traiter. En fait, pour utiliser Hadoop, il suffit juste de le configurer, de l'installer et d'écrire des classes effectuant les fonctions `Map` et `Reduce`. Nous avons donc décider d'utiliser ce framework pour notre projet afin de réaliser la partie indexation.

1.2 Décomposition du travail

Après le choix de Hadoop, nous avons décidé de découper notre moteur de recherche en trois parties :

- l'interface graphique, où l'utilisateur entre ce qu'il veut rechercher
- le moteur de recherche, qui prend la requête entrée par l'utilisateur et effectue les calculs
- et l'index, réalisé à l'aide d'Hadoop, qui réalisera l'indexation du contenu.

Le contenu en lui-même est constitué de fichiers textes, des livres libres de droit trouvés sur internet. Le résultat rendu sera les lignes dans lesquelles apparaissent les mots entrés par l'utilisateur. Nous avons décidé d'utiliser le langage Java d'une part car Hadoop est en Java, d'autre part parce que tous les membres du groupe connaissant bien le langage, cela permet une meilleure cohésion.

Comme la mise en place d'Hadoop et de l'index nous semblait une tâche difficile, nous avons décidé d'attribuer 2 personnes sur l'indexation, une personne sur le moteur de recherche et une autre personne sur l'interface graphique afin de faire fonctionner au début les modules séparément puis ensuite d'unifier le travail.

2 Elaboration du projet

Très tôt, nous avons décidé de nous pencher sur l'architecture de notre projet, et tout au long du projet, nous avons suivi et rectifié son évolution. Voici ci-dessous notre architecture finale.

2.1 Indexation

En ce qui concerne l'indexation, nous avons commencé par installer Hadoop sur une distribution Linux : Ubuntu. Puis, nous avons décidé du format que l'on devait adopter pour la sortie de l'index obtenue à partir des fichiers en entrée.

Comme expliqué plus haut, MapReduce se repose essentiellement sur 2 fonctions : une fonction `Map` et une fonction `Reduce`. Nous avons donc dû décider des clés intermédiaires obtenues à la sortie de la fonction `Map` et en entrée de la fonction `Reduce`.

Nous avions en entrée de la fonction `Map` une clé, l'offset du premier mot de la ligne et une valeur, le texte de la ligne, les fichiers en entrée étant lu ligne à ligne. En sortie de cette fonction, nous avons pour clé intermédiaire, le mot et le nom du fichier dans lequel il se trouve et en valeur le numéro de la ligne (*pour un exemple concret, voir figure 2*).

La fonction `Reduce` agrège ensuite ces informations pour nous donner en sortie un fichier texte, la clé étant le mot et le nom du fichier dans lequel il se trouve et la valeur la liste des numéros de ligne de ce mot dans le fichier associé.

Afin d'avoir le numéro de ligne dans le fichier, Hadoop ne donnant que par défaut l'offset du premier mot de la ligne par rapport au début du fichier, nous avons aussi entrepris d'attribuer à un thread effectuant la fonction `Map`, un fichier entier.

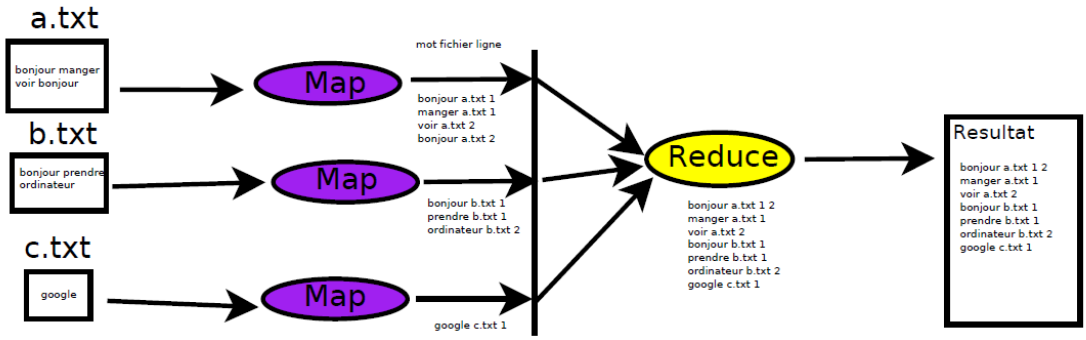

FIGURE 2 – Schema illustratif du processus MapReduce pour notre projet

Nous avons aussi choisi d'éliminer les mots non significatifs en entrée afin d'améliorer les performances de notre indexation. Nous avons donc exclu les les articles, les mots de liaisons et les pronoms de la langue française.

Une fois le traitement sur le texte pur effectué via MapReduce, on obtient donc un fichier texte qui contient ligne par ligne les entrées suivantes, séparées par des espaces : mot fichier suite de numéros de lignes associés.

Il faut donc reconstituer à partir de ce fichier beaucoup moins lourd que le texte initial un index dans la mémoire vive. Pour celà, nous avons opté pour une table de hachage.

Il s'agit en fait d'une table de hachage contenant elle-même une autre table de hachage ayant en clé principale un entier (identifiant unique pour chaque entrée de la table de hachage) auquel on associe une autre table de hachage, dont la clé (secondaire) est une chaîne de caractères qui décrit le mot auquel est associée l'entrée.Enfin, on associe à ce mot le nom de fichier dans lequel il se trouve, et une liste qui contient les numéros de lignes où l'on trouve le mot dans le fichier décrit.

Pour l'indexation, le principal problème que nous avons rencontré fut des problèmes dans l'installation d'Hadoop. En effet, Hadoop nécessite Java et ce dernier n'est pas toujours correctement installé sur le système d'exploitation que nous utilisons. Par ailleurs, nous n'avons pas réussi à installer Hadoop sur Windows étant conçu pour tourner sur Linux/Unix, des problèmes de droit nous ont empêché de pouvoir lancer Hadoop.

2.2 Moteur de recherche

Le moteur de recherche va parcourir la table de hachage reconstituée à partir du fichier texte produit par MapReduce afin d'en tirer des informations intéressantes.

Tout d'abord, nous avons pensé à une recherche simple, d'une suite de mots. Il s'agissait là de découper l'expression passée par l'utilisateur afin d'accomplir une recherche mot par mot. Sur ce découpage, nous avons choisi d'éliminer certains mots courants de la langue française, à savoir les articles, les mots de liaisons et les pronoms jugés non pertinents pour une recherche.

Ensuite, la partie chargée de traiter l'expression, passe la main à un autre module qui va se charger de récupérer les informations pour chaque mot (noms des fichiers dans lesquels ils se trouvent, numéros des lignes). A partir de ces informations, on renvoie pour affichage le résultat associé, c'est-à-dire le mot recherché, le fichier dans lequel il se trouve, le numéro de la ligne et le contexte dans lequel le mot se trouve c'est-à-dire le texte des trois lignes autour du fichier.

En ce qui concerne la récupération du texte des lignes dans les fichiers, nous avons décidé de découper les fichiers d'origine en plusieurs parties afin que la recherche des lignes soit plus rapide.

Pour finir, nous avons envisagé d'introduire quelques prédicats de base, à savoir `AND`, `OR` et `NOT`. Ils permettent respectivement de rechercher un fichier contenant deux mots, de rechercher des fichiers ne comprenant pas les deux mots passés en argument en même temps, et d'éliminer un mot de la recherche. Ils sont cummulables, et leur grammaire est de la forme suivante :

```
AND mot mot
OR mot mot
NOT mot
```

Pour mettre en place le `AND` comme le `OR`, il a fallu parcourir le résultat associés aux deux mots et éliminer les résultats ne correspondant pas - à savoir les fichiers ne contenant pas les deux mots dans le cas du `AND`, et inversement dans le cas du `OR`, ceux contenant les deux mots.

Pour le `NOT`, un cas plus particulier, il a fallu créer une deuxième liste qui contient les informations à éliminer. Après avoir traité tous les mots, il nous suffit alors d'enlever les instances de cette liste à la liste de recherche principale. Dans le cas où aucun `NOT` ne serait utilisé, cette liste est vide donc la méthode fonctionne dans tous les cas.

Les seuls problèmes que nous avons rencontrés dans cette partie sont purement des problèmes d'algorithme, d'organisation des structures de données et de lecture du texte passé en amont par l'utilisateur.

2.3 Interface graphique

Enfin, pour l'interface graphique, nous avons vite pensé à utiliser une applet. Java met à dispositions des classes qui permettent simplement de réaliser un applet avec la bibliothèque graphique Swing. Il suffit d'intégrer l'applet dans une page html qui étoffe son apparence pour avoir une interface simple et élégante, constituée entre autres d'un champ de caractère et d'un bouton.

Après la recherche entrée comme spécifiée plus haut, l'applet renvoie tout simplement un affichage des lignes des textes contenant les mots clés nous intéressants, que nous avons mis en avant en les mettant en gras. Si jamais l'utilisateur rentre des informations aberrantes, comme un mot-clé n'existant pas ou un prédicat mal formalisé, l'applet en informe aussi l'utilisateur, en lui proposant même une correction dans le second cas.

Nous avons rencontré uniquement des difficultés au niveau de l'affichage de l'applet. En effet, sous Linux, il est difficile d'abaisser le niveau de sécurité de Java afin

d'autoriser l'exécution mais aussi l'accès aux données de l'applet. De plus, il faut bien faire attention aux versions avec lesquels on compile l'applet, si on utilise Java 1.7 pour compiler, il faut avoir Java 1.7 d'installé dans le navigateur.

3 Résultats obtenus et améliorations possibles

3.1 Résultat

Au terme de ce projet, nous avons réussi à obtenir un moteur de recherche basique permettant de rechercher les fichiers et les lignes dans lesquelles se trouvent les mots entrés avec un système de prédicats expliqués dans la partie précédente. Pour faire fonctionner tout notre projet ensemble, nous avons rassemblé nos parties. L'indexation s'effectue en tâche de fond par une tâche `cron` Unix qui lance un script toutes les heures permettant d'aller chercher dans un répertoire déterminé à l'avance les fichiers dans lesquels l'utilisateur va chercher des informations. L'interface graphique se constitue d'une simple page web dans laquelle est intégrée une applet Java. Le moteur de recherche se lance lorsque l'utilisateur entre un mot. Vous pouvez avoir un aperçu du rendu donné pour la recherche d'un mot sur la figure 3.

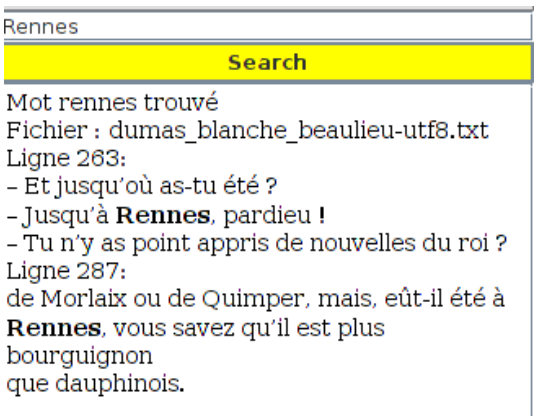

FIGURE 3 – Apercu du resultat donne par une recherche d'un mot

3.2 Améliorations possibles

Quelques améliorations peuvent toutefois être apportées à ce projet. Nous allons vous exposer les améliorations que nous trouvons envisageables mais que nous n'avons pas effectuées par manque de temps.

3.2.1 Relevance

Pour plusieurs mots entrés, il serait possible de trier le résultat donné par ordre de priorité. Par exemple, donner en premier le fichier où les mots entrés sont les plus proches. Cela nécéssite d'utiliser un algorithme de classification automatique.

3.2.2 Nouveau prédicat : expression exacte

Il serait aussi possible d'introduire un nouveau prédicat permettant de rechercher une expression qui doit être trouvée exactement comme elle est rédigée par l'utilisa-

teur. Cependant, selon nous, l'indexation que nous avons mise en place ne serait pas très efficace car cela reviendra à reconstruire le fichier de départ et à mémoriser une trop grand quantité d'informations dans l'index. Il faudrait donc concevoir un nouveau système d'indexation (raison pour laquelle nous avons pas eu le temps de le faire).

Il serait possible aussi de généraliser l'utilisation des prédicats `AND` et `OR` a plus de 2 paramètres.

Conclusion

Ce projet nous a donc permis de mieux comprendre le fonctionnement d'un moteur de recherche tel que Google. Par ailleurs, nous avons pu apprendre à réaliser un projet d'assez longue durée comparé aux projets faits précédemment et de travailler en équipe.

Références

[1] Alexandru Costan : *Sujets des etudes pratiques 2012-2013*, chapitre Sujet 8. Institut National des Sciences Appliquees de Rennes, 2012.

[2] Sanjay Ghemawat Jeffrey Dean : *MapReduce : Simplified Data Processing on Large Clusters*. Operating Systems and Implementations, 2004.

[3] Wikipedia l'encyclopedie libre : Google. `http://fr.wikipedia.org/wiki/Google`.