Indexation et recherche de contenu utilisant MapReduce de Google

Rapport final

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

Encadrant : Alexandru Costan

Résumé

Le résumé est limité à 10 lignes au maximum. A faire à la fin.

1 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 donner un sujet d'étude riche et intéressant. Nous remercions aussi le personnel de l'INRIA, nous aillant accueilli lorsque nous allions rendre visite à Monsieur Costan.

2 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 lesquelles 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'inxation 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.

3 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.

3.1 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 Map 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 aggrégège le résultat afin de donner une clé finale et un résultat final. Dans l'exemple du Wordcount, la clé serait un moteur 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, il en a donc été fait des implémentations, 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 Map et Reduce. Nous avons donc décider d'utiliser ce framework pour notre projet avec de réaliser la partie indexation.

3.2 Décomposition du travail

Après le choix de Hadoop, nous avons donc 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 appraissent 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.

4 Elaboration du projet

Très tôt, nous avons décidé de se 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 dans les grandes lignes.

4.1 Indexation

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 virgules : mot, fichier, lignes associées.
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 vons choisi d'utiliser une Map<Integer, HashMap<String, Informations ».
Il s'agit en fait d'une Hashmap d'Hashmap contenant en clé principale un integer (pseudo hashcode permettant de parcourir simplement la structure) auquel on associe une autre HashMap, dont la clé (secondaire) est un String qui décrit le mot auquel est associée l'entrée. Enfin, on associe à ce String une classe Information, qui possède comme attributs un autre String, qui est un fichier dans lequel on trouve ce mot, et une ArrayList<Long> qui contient les numéros de lignes où l'on trouve le mot dans le fichier décrit.

4.2 Moteur de recherche

Le moteur de recherche va parcourir la hashmap 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 lange française, à savoir les articles, les mots de liaisons et les pronoms.

Ensuite, la classe chargée de traiter l'expression, nommée search, passe la main à la classe seeker qui va pour chaque mot vérifier s'il s'agit d'une clé de la hashmap constituant l'index puis, si tel est le cas, renvoyer les entrées correspondantes (fichiers, numéros de lignes, nombre d'occurences). A partir de ces informations, on renvoie un affichage de chaque instance de ligne où le mot clé est trouvé, en donnant les numéros de ligne associés. L'ordre de l'affichage se base sur un critère de relevance qui concerne le nombre d'occurences de chaque mot clé pour chaque fichier. On réaffichera des parties d'un même fichier pour chaque mot, ce qui nous permet d'obtenir le contexte dans lequel ce mot a été trouvé.
Pour finir, l'idée nous est venue d'introduire quelque 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 fallut 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 dans la classe Seeker qui contienne les informations à éliminer. Après avoir traitéer tous les mots, dans la classe Search, 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 marche dans tous les cas.
Les seuls problèmes que nous avons rencontré dans cet partie sont purement des problèmes d'algorithme, d'organisation des structures de données et de lecture du texte passé en amont par l'utilisateur.

4.3 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 une classe fenêtre, que l'on peut ensuite exporter sous forme d'applet. Il suffit d'intégrer celle-ci dans une pâge html qui étoffe son apparence pour avoir une interface simple etélégante, 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'éxécution mais aussi l'accès aux données de l'applet.

Titre de paragraphe Exemple de

4.4 Encore un titre de sous-section

Exemple de liste à puces :

- ligne de remplissage pour visualiser la mise en page. Ligne de remplissage pour visualiser la mise en page ;
- ligne de remplissage pour visualiser la mise en page. Ligne de remplissage pour visualiser la mise en page.

Ligne de remplissage pour visualiser la mise en page. Ligne de remplissage pour visualiser la mise en page.

5 Conclusion

LaTeX c'est facile pour produire des documents standard et nickel ! Et BibTeX pour les références, c'est le pied.

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`.