|  |  |
| --- | --- |
|  | Version 0.0.1 ● |
| PROJET SHAVADOOP  Hervin KASA & Sidoine KAKEUH FOSSO | |
|  | |

Table Des matières

1 Description du logiciel 5

1.1 Diagramme de classe 5

1.2 Fonctionnement du logiciel 5

1.2.1 Démarrage 6

1.2.2 Splitting 6

1.2.3 Mapping 6

1.2.4 Shuffling & Reducing 7

1.2.5 Assembling 8

1.2.6 Résultats 9

1.3 Diagramme de séquence 9

2 Lancement du Logiciel 9

3 Etude des performances 10

3.1 Impact du paramètre *‘size\_split’* 10

3.2 Impact du nombre de machines disponible 10

4 Problèmes rencontrés solutions apportées 11

5 Optimisation 11

6 Annexe 12

Préface

1. Version du document

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Auteurs (s)** | **Objectif** | **Date** |
| 0.0.1 | Sidoine KAKEUH FOSSO / Hervin KASA | Documentation du projet | 24/11/16 |

1. Documentation associée

|  |  |  |  |
| --- | --- | --- | --- |
| Titre du document | Auteurs | Description du document | Date de dernière |
| MASTER\_SHAVADOOP\_ZIP | Sidoine KAKEUH FOSSO / Hervin KASA | * Code Source * Fichier .jar | 24/11/2016 |
|  |  |  |  |

1. Materiels / outils

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **OS** | **Logiciel** | **Version** |
| Langage de programmation | Linux | Java | 1.7 |
| IDE | Linux | Eclipse | Juno |

Introduction

Le but de ce document est de donner une description du projet Shavadoop. Le projet Shavadoop consiste à implémenter une version MapReduce afin d’implémenter l’algorithme du ‘**MapReduce** **Wordcount’,** programme qui permet de compter les mots contenu dans un document (exemple : implémentation dans le système de fichier distribué **Hadoop)**.

L’implémentation de cette algorithme en Java s’est appuyer sur le support de Travaux pratique du

de l’UE INF727 : Système repartis et distribués qui se trouve en annexe.

Nous utiliserons comme infrastructures matérielles les machines des salles de TP disponible de l’école comme système distribué, l’IDE Eclipse, ainsi que le Shell Linux.

Notre programme Java comportera deux projets principaux : MASTER\_SHAVADOOP, SLAVE\_SHAVADOOP dont nous expliciterons plus bas, ainsi que les fonctions annexes de gestion

des threads / affichages de flux.

# Description du logiciel

Le projet est organisé en deux sous projets principaux : **MASTER\_SHAVADOOP**  et **SLAVE\_SHAVADOOP :**

* MASTER\_SHAVADOOP : programme maitre chargé d’effectuer des traitements mineurs et principalement de coordonner l’ensemble des traitements. Il peut être vu comme le **Namenode** dans le système de gestion de fichiers distribués dans Hadoop.
* SLAVE\_SHAVADOOP : programme chargé de réalisé des traitements majeurs (écriture et lecture des données). Il peut être vu comme le **Datanode** dans le système de gestion de fichiers distribués dans Hadoop.

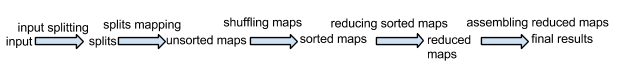
## Diagramme de classe

Le MASTER\_SHAVADOOP  contient trois classes principales à savoir : **MASTER (Main),** **Runner\_slave, AfficheurFlux** qui contient des fonctions utiles pour l’affichage des flux. La classe MASTER (Main)lance les différents traitements selon deux mode (**modeSxUMx** et **modeUMxSMx**), sur SLAVE\_SHAVADOOP et ceci à travers la classe Runner\_Slave.

Le SLAVE\_SHAVADOOP contient une classe Slave (**Main)** + qui exécute les différentes tâches demandé par le MASTER pour les deux modes **modeSxUMx** et **modeUMxSMx**.

## Fonctionnement du logiciel

L’algorithme se divise en cinq étapes principales (voir annexe) ci-dessous :



**Figure 1** : Etapes principales

L’implémentation de ces étapes va être explicitée ci-dessous aux travers des différents paragraphes

Demarrage, Splitting, mapping, shuffling et reducing et assembling.

## Démarrage

Afin de démarrer notre programme, nous devons d’abord tester l’infrastructure matérielle disponible pour notre système distribué, ce qui nous amène à lister dans un fichier nommé ici **«listemachine.txt»** toutes les machines disponibles. Ensuite, nous faisons un test de connexion réseau entre le master et les slave (liste de machine présent dans le fichier «listemachine.txt» passé en paramètre) avant de commencer notre algorithme à proprement parlé.

. A la fin de cette phase la liste des machines sur lesquelles la connexion a réussi est enregistrée dans un fichier créé («**listedemachineOK»**). Ce fichier est créé dans le répertoire contenant le MASTER\_SHAVADOOP.jar. (workspace) Les machines contenues dans **listedemachineOK** seront utilisées dans la suite pour lancer des traitements en parallèle.

### Splitting

Dans cette phase le Master commence par faire un Splitting (division) ligne par ligne du fichier en entrée. Un ensemble de fichier **Sx** sont créés à ce niveau. Il faut noter ici que le nombre de **Sx** crée dépend du paramètre **size\_split** en entrée

### Mapping

Cette étape marque le début de l’interaction entre les deux sous projet : MASTER\_SHAVADOOP et SLAVE\_SHAVADOOP. Le Master programme commence par créer un ensemble d'objet de la classe Runner\_Slave (classe qui hérite de la classe **Thread**) et crée le dictionnaire **UMx - machines**. Le Master au moment de la création des Runner\_Slave passe trois paramètres importants au constructeur de la classe Runner\_Slave:

* **mode** : le mode d'exécution qui vaut **modeSXUMX** dans ce cas.
* **machine**: l'adresse sur laquelle le Runner\_Slave sera créé
* **Sx** : le fichier à traiter
* **Path**: chemin ou se trouve le fichier  « Sx »

Ces Runner\_Slave sont ensuite lancés par le Master avec la méthode **start** de la classe Thread. Les Runner\_Slave à leur tour grâce à la classe **ProcessBuilder** exécutent le fichier SLAVE\_SHAVADOOP.jar en passantminutieusement les paramètres transmis par le Master.

Le programme SLAVE\_SHAVADOOP.jar dans cette phase génère Un fichier **UMx** associé au fichier **Sx** en paramètre. Le fichier UMx contient pour chaque occurrence d’un mot dans le fichier Sx une association **clé – valeur** ou clé est le mot et valeur est égale à **1**.

Pour une meilleure compréhension des interactions (paramètres des fonctions, …), se référer à la documentation associée MASTER\_SHAVADOOP\_DOC et SLAVE\_SHAVADOOP\_DOC.

### Shuffling & Reducing

Cette phase débute par la réception des clés issus du traitement Mapping des slaves. Le Master peut alors construire le dictionnaire **clés -UMx**

Le Master crée ensuite un ensemble d'objet de la classe Runner\_Slave en passant trois paramètres importants au constructeur de la classe Runner\_Slave:

* **mode** : le mode d'exécution qui vaut **modeUMXSMX** dans ce cas.
* **machine**: l'adresse sur laquelle le Runner\_Slave sera créer
* **key** : la clé (mot) à traiter
* **SMx :** le fichier à créer par le programme SLAVE\_SHAVADOOP\_DOC
* **List UMx :** « motfinal » la liste des fichiers UMx contenant la clé. Pour notre implémentation nous avons créé une chaine de caractères contenant cette liste, chaque UMx séparé par un espace (variable de type **String**).
* **i :** numéro du fichier UMx à traiter pour générer pour générer le SMx
* **Path**: chemin ou se trouve le fichier  « UMx »

Les Runner\_Slave sont ensuite lancés par le Master avec la méthode **start** de la classe Thread. Les Runner\_Slave à leur tour grâce à la classe **ProcessBuilder** exécutent le fichier SLAVE\_SHAVADOOP.jar en passant minutieusement les paramètres transmis par le Master.

Le programme SLAVE\_SHAVADOOP.jar dans cette phase génère Un fichier SMx. Le fichier SMx contient toutes les occurrences contenues dans les fichiers UMx de la clé passée en paramètre avec la valeur 1 associé. Directement à la suite de ce traitement le slave exécute une étape de Reducing (agrégation sur la clé en sommant la valeur associée pour toutes les occurrences). Le slave génère un fichier **RMx** contenant le résultat de cette agrégation et renvoie la paire **clé – valeur** au Master. Ici **clé** représente le mot et **valeur** le nombre d’occurrence du mot dans le fichier.

### Assembling / Resultats

Le Master réceptionne les **RMx** et les fusionnent, c’est la phase de synthèse.

Toutes les sorties des différents fonctions « reducer » sont concaténées et écris dans un fichier « finalresult ». On y retrouve tous les mots présents dans le fichier d’entrée ainsi que leurs fréquences, les temps d’exécution de chaque étape de l’algorithme sont affichés dans le terminal.

Le traitement est terminé.

Le résultat final, le fichier **Finalresult est créé dans le même emplacement que les Sx,UMx,RMx**

Ainsi que les fichiers **listemachine** et les **.jar** .

### Lancement du logiciel

Commande de lancement :

**java – jar MASTER\_SHAVADOOP.jar path list\_machines.txt filename.txt**

*Avec :*

**file** : Le fichier en entrée à traiter.

**path/** : Le chemin ou dossier ou sera créé les fichiers temporaires des traitements.

**list\_machines**: Liste des Adresses à laquelle le master va essayer de se connecter

### Résultats

# Etude des performances

## Impact du nombre de machines disponible

La performance de notre programme (temps d’exécution des phases de l’algorithme) est très dépendante du nombre de machine connecté du système distribué, ainsi que de la taille du fichier texte à analyser en entrée.

# Problèmes rencontrés solutions apportées

* Les problèmes principaux que nous avons rencontré viennent du fait que nous n’avons jamais programmé en Java auparavant, ce qui a rendu notre assimilation un peu plus difficile.
* De même, les notions de thread ont été un challenge à assimiler.

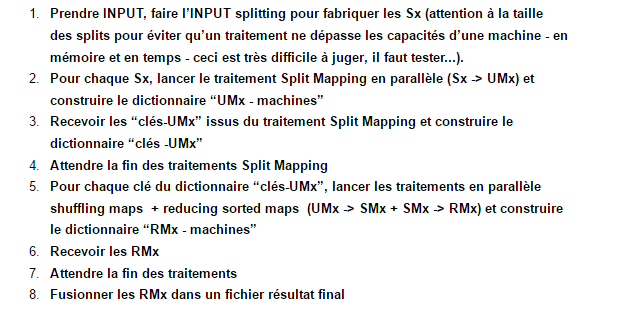
# Optimisation

* Notre programme ne gère pas les Time out, à cause des difficultés que nous avons souvent rencontrées à cause soit de la non disponibilité des machines de l’école (machines éteintes) , soit trop lent à cause de trop d’utilisateurs en même temps.
* Notre programme n’a pas un temps de traitement optimisé, d’où les temps de réponse assez long pour les gros fichiers et très lent lors du test de connexion des machines du système.
* Il subsiste donc plusieurs point à améliorer afin d’obtenir un programme optimisé, en occurrence implémenter les solutions proposé par l’enseignant lors des TP pour les questions de 1 à 22
* Pour les gros fichiers, notre programme à besoin d’autant de machines que de fichier Sx crée, soit le nombre de ligne du fichier d’entrée. Ceci représente un énorme faiblesse qui est à améliorer.

# Annexe



**Figure 10** : Word count pipeline



**Algorithme 1** : Algorithme Word count pipeline

***Annexe1 : Résultat avec fichier contenant l’algorithme ci-dessus :***



