


# Systemes répartis



**Rapport MapReduce**

Martial de Jurquet

05/12/2024

## Introduction :

Nous allons explorer dans ce rapport un programme réalisant un mapreduce sur un ensemble de données volumineux. Le programme s'articule autour de deux fichiers principaux, le deploy (bash) et le script (python). J'ai choisi dans cette version de réaliser les échanges entre machines via ssh plutôt que par des sockets, par commodité de lecture, plus aisé en bash selon moi.

Trois opérations principales sont réalisées à travers ces deux fichiers ; un mapping, un shuffle et un reduce.

Le mapping consiste à créer des lignes clé-valeurs, représentant le couple « mot » 1, permettant par la suite d'agréger les valeurs.

Le shuffle répartira les couples clés-valeurs selon une logique de hachage et les enverra à chaque machine. L'objectif est de créer des parties dans lesquelles le même mot sera toujours destiné à la même machine après l'envoi des shuffle\_part\_<x>. Autrement dit, après répartition, aucun mot ne sera traité par deux machines différentes.

Le reduce additionne les valeurs par occurrence du mot, après avoir regroupé les différentes parts\_shuffle reçues.

Cet enchaînement d'étape nécessite plusieurs paramètres prérequis indispensables : Tout d'abord, la gestion des connexions ssh avec un nombre fluctuant de machines. Nous allons réaliser des tests de connexion, et établir une liste de machines stables. Attention à établir une première connexion en amont sur les machines utilisés pour faciliter les connexions par la suite.

Ensuite, nous devons définir des id de machine, de manière à pouvoir se repérer dans la répartition à la phase de shuffle, définir les différents chemins de stockage, pour enfin répartir les blocs sur chaque machine, et être prêt à lancer le mapping.

Nous allons voir rapidement les détails techniques derrière ces fichiers et étapes, en commençant par le deploy, puis le script, pour finir par analyser les performances du programme.

## I) Le deploy

Étape	Description technique détaillée
Initialisation	Le script établit les bases en définissant les chemins locaux et distants, les tailles de blocs, et les paramètres de traitement. Il vérifie les connexions SSH en testant chaque machine via une commande <code>`ssh`</code> . Les machines valides sont ajoutées à une liste dynamique.
Téléchargement	La machine local liste les fichiers disponibles dans le répertoire source distant ( <code>`sourceFolder`</code> ) via SSH. Les fichiers sont téléchargés par <code>`scp`</code> dans le dossier local ( <code>`localFolder`</code> ), et leur taille est calculée avec <code>`stat`</code> .
Découpage	Les fichiers téléchargés sont divisés en blocs de taille fixe (64 Mo, spécifié par <code>`blockSize`</code> ) à l'aide de la commande <code>`split`</code> . Chaque bloc est renommé en un format séquentiel ( <code>`part1`</code> , <code>`part2`</code> , etc.) pour garantir une identification unique dans les étapes suivantes.
Distribution	Les blocs découpés sont répartis uniformément entre les machines valides. Chaque machine reçoit un sous-ensemble des blocs via <code>`scp`</code> . La distribution se fait de manière cyclique pour équilibrer la charge, en utilisant l'indice des machines modulo le nombre total de blocs.
Mapping	Chaque machine exécute localement le script python ( <code>`script.py`</code> ) pour traiter les blocs qu'elle a reçus. Le script produit des fichiers de sortie au format <code>`mapping_&lt;index&gt;_&lt;bloc&gt;.txt`</code> .
Shuffle	Les données issues du mapping sont redistribuées entre les machines pour regrouper les données similaires (par clé). Cette phase utilise des transferts via SSH pour aligner les données selon les mots spécifiques à traiter par chaque machine. Plusieurs tentatives sont gérées en cas d'échec.
Réduction (Reduce)	Sur chaque machine, un fichier de données consolidé est généré ( <code>`reduce_input_&lt;index&gt;.txt`</code> ) à partir des blocs reçus via le shuffle. Le script Python effectue des agrégations sur ces données pour produire un fichier final local ( <code>`reduce_machine&lt;index&gt;.txt`</code> ). Gestion des erreurs incluse.

Agrégation finale	Les fichiers de réduction générés par chaque machine sont rapatriés localement via `scp`. Ils sont concaténés en un fichier unique (`resultat_final.txt`) à l'aide de `cat`. Une fois l'agrégation terminée, les fichiers intermédiaires locaux et distants sont supprimés pour libérer l'espace.
-------------------	---

## II) Le script

### **map\_phase(input\_file, output\_file) :**

Cette fonction réalise la phase de mapping, qui crée les couples clé-valeur en les initiant au format <mot> 1. La fonction prend en entrée un argument input\_file, qui correspond aux parties reçues de la machine maître. L'output sera un fichier mapper que l'on utilisera au shuffle.

### **shuffle\_phase(machine\_id, input\_folder, output\_folder, num\_machines) :**

Cette fonction réalise la phase de shuffle. Pour cela, elle a besoin d'un input\_file (le mapping), du nombre de machine (variable num\_machine), et l'id de la machine. Le principe est de répartir les mots selon une logique de hachage qui garantit l'uniformité du résultat. Nous utilisons la fonction MD5 de la bibliothèque python Haslib. Les output se matérialisent sous la forme de fichier

« shuffle\_<machine\_target>\_from\_machine\_<id\_machine>, ils seront réparti entre les machines à l'étape reduce.

### **reduce\_phase(input\_file, output\_file) :**

Cette étape finalise l'utilisation du cluster. Une fois les fichiers shuffles envoyés par chaque machine, et reçus par les destinataires, nous regroupons ces derniers dans un fichier reduce. Nous consolidons ensuite les valeurs de chaque mot pour afficher le nombre d'occurrence du mot. Etant donné que chaque mot n'est présent que sur une machine, grâce à la répartition en hachage, les calculs d'occurrences des reduces correspondent déjà au total d'occurrence de chaque mot dans le dataset de base. Nous rapatrions ensuite les différents reduce en local, pour les concatener.

### III) Performance et loi d'Amdahl

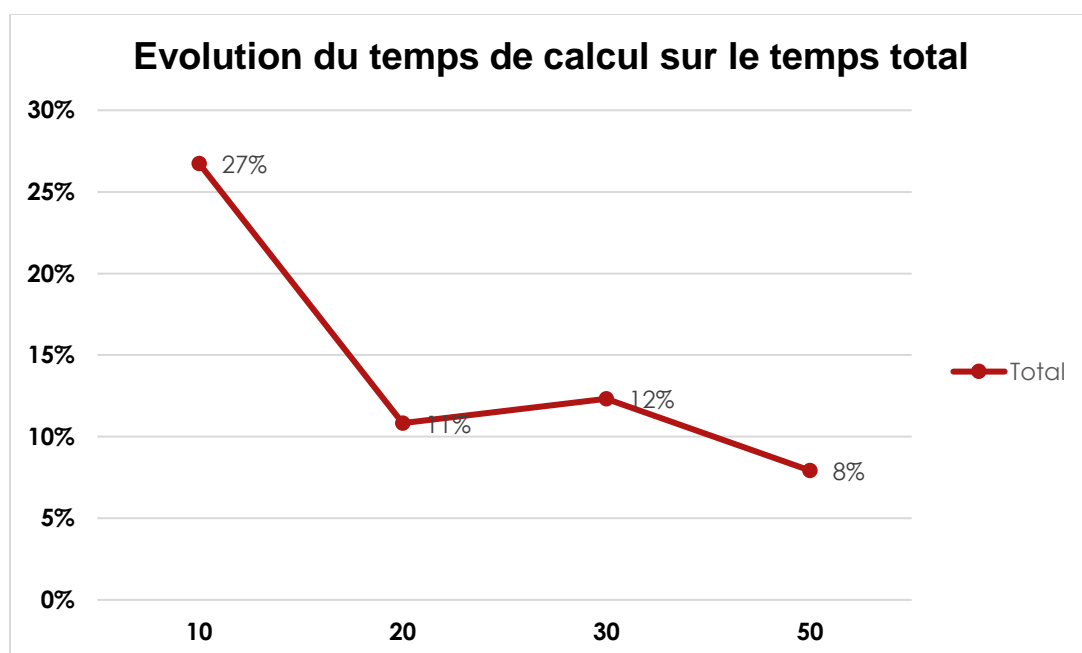
Nous allons nous pencher sur un problème décrit par la loi d'Amdahl qui concerne l'utilisation de cluster pour réaliser des opérations gourmandes en ressources, tel que notre MapReduce. Le problème consiste à remarquer que le cluster aide bien à réaliser des calculs plus rapides en divisant la charge, mais génère aussi des temps d'opérations incompressibles, lié aux tâches « de transition », comme le transfert de fichier, le regroupement de fichier... Ces tâches incompressibles augmentent fatalement avec le nombre de machine, tel qu'à un certain point, cette augmentation concurrence le gain de puissance de calcul, jusqu'à l'annuler. Nous avons alors un point où augmenter le nombre de machine n'augmentera pas la vitesse du processus.

Voyons maintenant si cette loi est vérifiée dans notre projet !

J'ai réalisé le calcul sur 5 fichiers prélevés sur common/crawl des ordinateurs telecom, avec 10, 20, 30 machines. Le script fonctionne sur de plus grandes quantités, mais par soucis de gain de temps, 5 fichiers sont suffisants pour remarquer l'effet de la loi d'Amdahl.

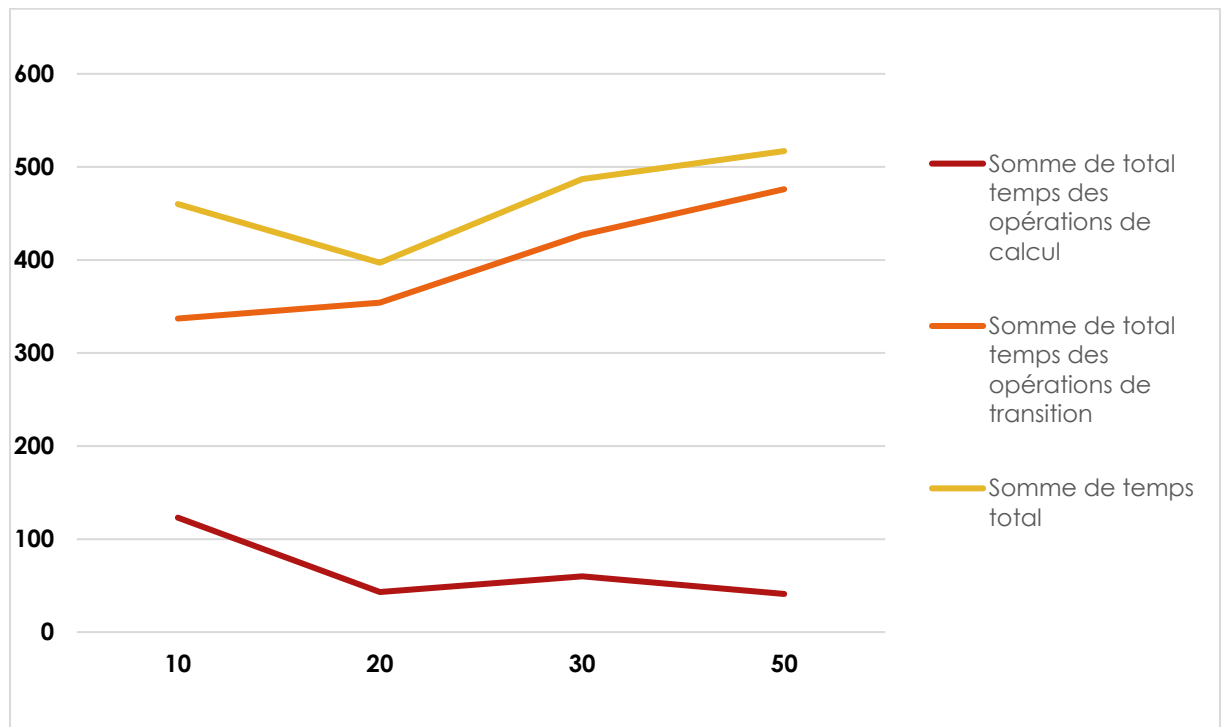
Nous allons voir le détail des performances selon le nombre de machine et les opérations. Ces dernières seront classées en deux catégories, les opérations de calcul et les opérations de transition. Par calcul, je désigne les opérations qui sont déclenchées en parallèle sur les machines du cluster. Celles dites de transitions correspondent aux opérations non répartissables, tel que les transferts, regroupement.\*

(\*) En fin de compte, j'ai retiré les téléchargements du compte des opérations de transitions, car cette étape variait beaucoup en fonction de la connexion internet, et fausse ainsi les comparaisons.



Nous remarquons ici, que le temps alloué au calcul diminue quand on augmente le nombre de machine. Nous pourrions voir en détail plus bas, les résultats des phases map, shuffle et reduce. La phase shuffle passe de 51 sec pour 10 machines à 17 sec pour 50.

Pour avoir une vue d'ensemble, nous pouvons regarder l'évolution du temps total :



Le temps est indiqué en seconde à gauche, tandis que le nombre de machine varie sur l'axe des abscisses. Nous remarquons que le temps total augmente à nouveau quand on dépasse 30 machines.

En détail, voici les temps de réalisation de chaque étapes :

Nombre machine	10	20	30	50
Poids total des fichiers traités (mo)	1638	1638	1638	1638
Téléchargement des fichiers	304	383	293	310
Découpage des fichiers	56	36	58	32
Répartition des blocs	323	302	345	407
Phase de mapping	70	24	33	23
Phase de shuffle	51	17	25	17
Phase de réduction	2	2	2	1
Phase d'agrégation	14	16	24	37
total temps des opérations de calcul	123	43	60	41
total temps des opérations de transition	337	354	427	476
temps total	460	397	487	517

Nous constatons donc que l'augmentation du nombre de machine n'est pas accompagnée d'un gain de temps pour le programme. Néanmoins, quelques éléments, comme une connexion ssh défaillante qui demande 4 tentatives pour fonctionner, peuvent fausser les comparaisons entre les performances...

## IV) Conclusion :

Ce projet s'est construit dans une logique « par étage », bloc par bloc, et manque certainement d'optimisation. J'ai préféré laisser le code ainsi, cela permet de bien comprendre les étapes, qui, d'après moi, suivent un déroulement facilement compréhensible. En décomposant ainsi, le projet fût riche d'enseignement, il ouvre une infinité de possibilité de réalisation. J'en retiens trois piliers fondamentaux : les connexions ssh/socket, les transfert scp, et le déclenchement de script à distance via un fichier bash. Avec ces trois outils maîtrisés les projets de ce genre sont fortement facilités.

D'autre part, la gestion des logs et la conception des dossiers de stockage sont primordiaux. Les logs fournissent une indication indispensable, et utile – si les messages de logs sont explicites !- . Le projet tendant à se complexifier assez vite, il m'est arrivé à plusieurs reprises d'emmêler les chemins par besoin de créer de nouveau endroit de stockage. Cette étape est à réaliser en amont, si l'on possède la vision globale nécessaire !

Une dernière étape de tri est demandée dans le sujet, j'ai eu le temps de la réaliser jusqu'au shuffle, mais elle n'est pas fonctionnelle et ne fera donc malheureusement pas partie du rapport.

Ce MapReduce est une manière intéressante de découvrir les clusters et les programmes. Les logiques appliquées lors de ce projet sont des bases applicables dans n'importe quel projet de clustering, et ainsi, seront un excellent enseignement pour la suite.