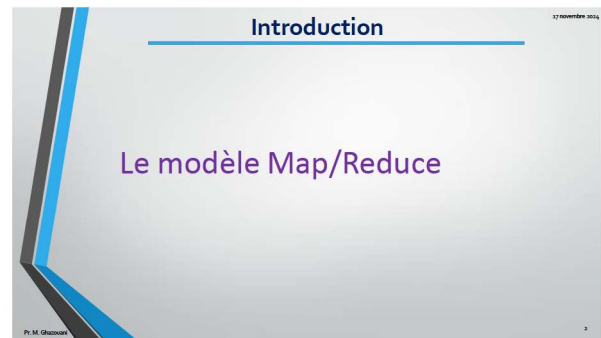
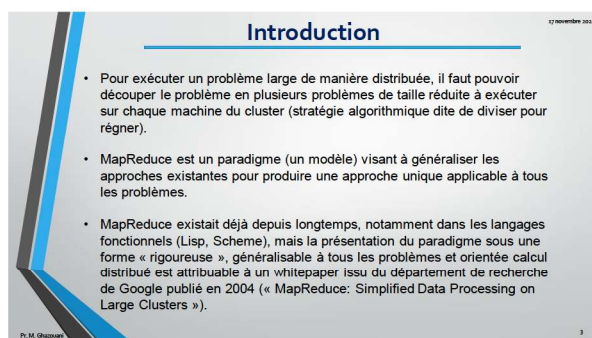


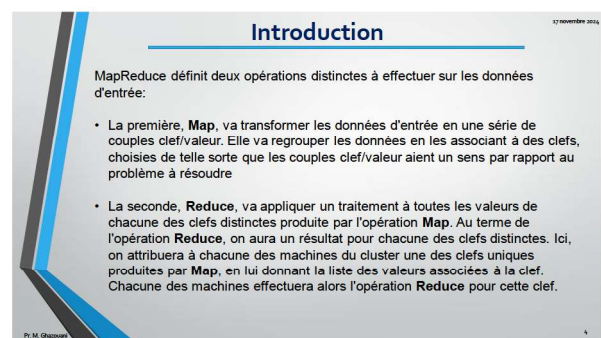
1



2



3



4

Introduction

Dear Bear River
Car Car River
Deer Car Bear

→

Bear, 2
Car, 3
Deer, 2
River, 2

Pr. M. Ghannem

5

5

Introduction

On distingue donc 4 étapes distinctes dans un traitement MapReduce:

- Découper (split) les données d'entrée en plusieurs fragments.
- Mapper chacun de ces fragments pour obtenir des couples (clef ; valeur).
- Grouper (shuffle) ces couples (clef ; valeur) par clef.
- Réduire (reduce) les groupes indexés par clef en une forme finale, avec une valeur pour chacune des clefs distinctes.

En modélisant le problème à résoudre de la sorte, on le rend parallélisable – chacune de ces tâches à l'exception de la première seront effectuées de manière distribuée.

Pr. M. Ghannem

6

6

Introduction

On distingue donc 4 étapes distinctes dans un traitement MapReduce:

The diagram illustrates the MapReduce workflow. It starts with an **Input** box containing 'Dear Bear River', 'Car Car River', and 'Deer Car Bear'. This input is processed through four stages: **Splitting** (dividing into 'Dear Bear River', 'Car Car River', and 'Deer Car Bear'), **Mapping** (transforming into key-value pairs like 'Dear, 1', 'Car, 1', 'River, 1'), **Shuffling** (grouping by key into buckets like 'Dear (1,1)', 'Car (1,1)', 'River (1,1)'), and **Reducing** (aggregating values for each key like 'Bear, 2', 'Car, 3', 'River, 2'). The final output is the **Final Result** box containing 'Bear, 2', 'Car, 3', 'Deer, 2', and 'River, 2'.

Pr. M. Ghannem

7

7

Introduction

Pour résoudre un problème via la méthodologie MapReduce avec Hadoop, on devra donc:

- Choisir une manière de découper les données d'entrée de telle sorte que l'opération MAP soit parallélisable.
- Définir quelle CLEF utiliser pour notre problème.
- Écrire le programme pour l'opération MAP.
- Écrire le programme pour l'opération REDUCE.

... et Hadoop se chargera du reste (problématiques calcul distribué, groupement par clef distincte entre MAP et REDUCE, etc.).

Pr. M. Ghannem

8

8

Exemple concret

Imaginons qu'on nous donne un texte écrit en langue Française. On souhaite déterminer pour un travail de recherche quels sont les mots les plus utilisés au sein de ce texte.

Ici, nos données d'entrée sont constituées du contenu du texte.

Première étape: déterminer une manière de découper (split) les données d'entrée pour que chacune des machines puisse travailler sur une partie du texte.

Notre problème est ici très simple – on peut par exemple décider de découper les données d'entrée ligne par ligne. Chacune des lignes du texte sera un fragment de nos données d'entrée.

17 novembre 2014

9

9

Exemple concret

Nos données d'entrée (le texte):

Celui qui croyait au ciel
Celui qui n'y croyait pas
[...]
Fou qui fait le délicat
Fou qui songe à ses querelles

(Louis Aragon, *La rose et le Réséda*, 1943, fragment)

Pour simplifier les choses, on va avant le découpage supprimer toute ponctuation et tous les caractères accentués. On va également passer l'intégralité du texte en minuscules.

17 novembre 2014

10

10

Exemple concret

Nos données d'entrée (le texte):

celui qui croyait au ciel

celui qui n y croyait pas

fou qui fait le delicat

fou qui songe a ses querelles

• ... on obtient 4 fragments depuis nos données d'entrée.

17 novembre 2014

11

11

Exemple concret

On doit désormais déterminer la clef à utiliser pour notre opération MAP, et écrire le code de l'opération MAP elle-même.

Puisqu'on s'intéresse aux occurrences des mots dans le texte, et qu'à terme on aura après l'opération REDUCE un résultat pour chacune des clefs distinctes, la clef qui s'impose logiquement dans notre cas est: le mot-lui même.

Quand à notre opération MAP, elle sera elle aussi très simple: on va simplement parcourir le fragment qui nous est fourni et, pour chacun des mots, générer le couple clef/valeur: (MOT ; 1). La valeur indique ici l'occurrence pour cette clef - puisqu'on a croisé le mot une fois, on donne la valeur « 1 ».

17 novembre 2014

12

12

Exemple concret

17 novembre 2013

Le code de notre opération MAP sera donc (ici en pseudo code):

POUR MOT dans LIGNE, FAIRE :
GENERER COUPLE (MOT; 1)

Pour chacun de nos fragments, les couples (clef; valeur) générés seront donc:

celui qui croyait au ciel	➡	(celui;1) (qui;1) (croyait;1) (au;1) (ciel;1)
celui qui ny croyait pas	➡	(celui;1) (qui;1) (ny;1) (croyait;1) (pas;1)
fou qui fait le delicat	➡	(fou;1) (qui;1) (fait;1) (le;1) (delicat;1)
fou qui songe a ses querelles	➡	(fou;1) (qui;1) (songe;1) (a;1) (ses;1) (querelles;1)

Pr. M. Ghazouan

13

13

Exemple concret

17 novembre 2013

Une fois notre opération MAP effectuée (de manière distribuée), Hadoop groupera (shuffle) tous les couples par clef commune.

Cette opération est effectuée automatiquement par Hadoop. Elle est, là aussi, effectuée de manière distribuée en utilisant un algorithme de tri distribué. Après son exécution, on obtiendra les 15 groupes suivants:

(celui;1) (celui;1)	(fou;1) (fou;1)	(fait;1) (le;1)
(qui;1) (qui;1) (qui;1) (qui;1)	(delicat;1)	(songe;1)
(croyait;1) (croyait;1)	(a;1)	(ses;1)
(au;1) (ciel;1) (ny;1) (pas;1)	(querelles;1)	

Pr. M. Ghazouan

14

14

Exemple concret

17 novembre 2013

Il nous reste à créer notre opération REDUCE, qui sera appelée pour chacun des groupes/clef distincte.

Dans notre cas, elle va simplement consister à additionner toutes les valeurs liées à la clef spécifiée:

TOTAL=0
POUR COUPLE dans GROUPE, FAIRE:
TOTAL=TOTAL+1
RENVoyer TOTAL

Pr. M. Ghazouan

15

15

Exemple concret

17 novembre 2013

Une fois l'opération REDUCE effectuée, on obtiendra donc une valeur unique pour chaque clef distincte. En l'occurrence, notre résultat sera:

qui : 4
celui : 2
croyait : 2
fou : 2
au : 1
ciel : 1
ny : 1
pas : 1
fait : 1
[...]

On constate que le mot le plus utilisé dans notre texte est « qui », avec 4 occurrences, suivi de « celui », « croyait » et « fou », avec 2 occurrences chacun.

Pr. M. Ghazouan

16

16

Exemple concret

Notre exemple est évidemment trivial, et son exécution aurait été instantanée même sur une machine unique, mais il est d'ores et déjà utile: on pourrait tout à fait utiliser les mêmes implémentations de MAP et REDUCE sur l'intégralité des textes d'une bibliothèque Française, et obtenir ainsi un bon échantillon des mots les plus utilisés dans la langue Française.

L'intérêt du modèle MapReduce est qu'il nous suffit de développer les deux opérations réellement importantes du traitement: MAP et REDUCE, et de bénéficier automatiquement de la possibilité d'effectuer le traitement sur un nombre variable de machines de manière distribuée.

17

Schéma général

Données d'entrée Splitting (fragmentation) MAP Shuffling (mélange/n) REDUCE Résultat final

celui qui croyait
ciel ciel croyait
celui ciel qui

18

Schéma général

Données d'entrée Splitting (fragmentation) MAP Shuffling REDUCE Résultat final

celui qui croyait
ciel ciel croyait
celui ciel qui

(celui : 1)
(qui : 1)
(croyait : 1)

(ciel : 1)
(ciel : 1)
(croyait : 1)

(celui : 1)
(ciel : 1)
(qui : 1)

(croyait : 1)
(croyait : 1)
(croyait : 1)

(qui : 2)
(ciel : 3)
(celui : 2)

qui : 2
ciel : 3
celui : 2
croyait : 2

19

Exemple concret: Statistiques web

Un autre exemple: on souhaite compter le nombre de visiteurs sur chacune des pages d'un site Internet. On dispose des fichiers de logs sous la forme suivante:

```
/index.html [19/05/2017:18:45:03 +0200]  
/contact.html [19/05/2017:18:46:15 +0200]  
/news.php?id=5 [24/05/2017:18:13:02 +0200]  
/news.php?id=4 [24/05/2017:18:13:12 +0200]  
/news.php?id=18 [24/05/2017:18:14:31 +0200]  
...etc...
```

Notre clef ?

Ici, notre clef sera par exemple l'URL d'accès à la page, et nos opérations MAP et REDUCE seront exactement les mêmes que celles qui viennent d'être présentées: on obtiendra ainsi le nombre de vue pour chaque page distincte du site.

20

Avantages MapReduce

- Scalabilité : (on peut ajouter autant qu'on veut des PC en parallèle sans arrêt, une fois connecté le système les configurent automatiquement)
- Cost effective : puisque on peut mettre des PC ordinaires qui coûtent pas chère.
- Flexibilité : on peut programmer avec java, python, R, scala, etc. On peut faire des traitements sur des fichiers texte, Excel, CSV, etc.
- Rapide : puisque le travail se fait en parallèle sur plusieurs machine au même temps.
- Haute disponibilité : grâce à la tolérance aux pannes, (en outre, si le name node tombe en panne on le standby name node qui prend le relais).

21

L'architecture Hadoop

On distingue deux architectures:

- La première, constitué du JobTracker et du TaskTracker, correspond à la première version du moteur d'exécution map/reduce (« MRv1 », Hadoop 1.x) et est présentée à titre informatif et pour des raisons historiques.
- A partir de la version 2.x, Hadoop intègre un nouveau moteur d'exécution: Yarn (« MRv2 »); cette architecture d'exécution, l'actuelle, est également présentée dans un second temps.

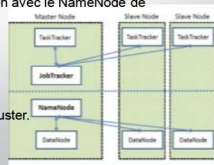


22

MRv1

La gestion des tâches de Hadoop se basait jusqu'en version 2 sur deux services (des daemons):

- Le JobTracker, qui va directement recevoir la tâche à exécuter (un .jar Java), ainsi que les données d'entrées (nom des fichiers stockés sur HDFS) et le répertoire où stocker les données de sortie (toujours sur HDFS). Il y a un seul JobTracker sur une seule machine du cluster Hadoop. Le JobTracker est en communication avec le NameNode de HDFS et sait donc où sont les données.
- Le TaskTracker, qui est en communication constante avec le JobTracker et va recevoir les opérations simples à effectuer (MAP/REDUCE) ainsi que les blocs de données correspondants (stockés sur HDFS). Il y a un TaskTracker sur chaque machine du cluster.

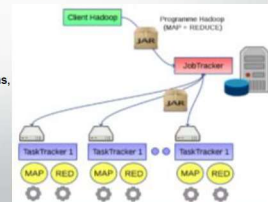


23

MRv1-Le JobTracker

Le déroulement de l'exécution d'une tâche Hadoop suit les étapes suivantes du point de vue du JobTracker:

1. Le client (un outil Hadoop console) va soumettre le travail à effectuer au JobTracker une archive java .jar implémentant les opérations Map et Reduce. Il va également soumettre le nom des fichiers d'entrée, et l'endroit où stocker les résultats.
2. Le JobTracker communique avec le NameNode HDFS pour savoir où se trouvent les blocs correspondant aux noms de fichiers donnés par le client.
3. Le JobTracker, à partir de ces informations, détermine quels sont les nœuds TaskTracker, c'est à dire ceux qui contiennent les données.
4. TaskTracker envoient régulièrement un « heartbeat ».



24

YARN (MapReduce 2)

YARN (Yet-Another-Resource-Negotiator) est aussi appelé MRv2 (MapReduce 2). Ce n'est pas une refonte mais une évolution du framework MapReduce.

Ce nouveau moteur d'exécution :

- YARN gère les ressources du cluster (comme DD, la mémoire et le CPU) pour les différentes applications qui s'exécutent dans Hadoop. Il permet également de gérer et de planifier les tâches pour une utilisation optimale des ressources.
- Lors de l'exécution de plusieurs applications sur un cluster Hadoop, YARN s'assure que chaque application dispose de suffisamment de ressources, en les allouant de manière équilibrée.

Pr. M. Ghazwan

25

YARN (MapReduce 2)

Le JobTracker a trop de responsabilités.

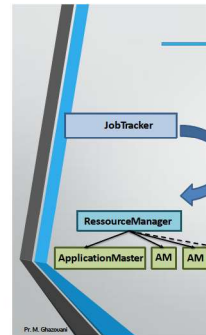
- Gérer les ressources du cluster.
- Gérer tous les jobs :
 - Allouer les tâches et les ordonnancer.
 - Monitorer l'exécution des tâches.
 - Gérer le fail-over.

Re-penser l'architecture du JobTracker.

- Séparer la gestion des ressources du cluster de la coordination des jobs.
- Utiliser les noeuds esclaves pour gérer les jobs.

ResourceManager et ApplicationMaster.

- ResourceManager remplace le JobTracker et ne gère que les ressources du Cluster.
- Une entité ApplicationMaster est allouée par Application pour gérer les tâches.
- ApplicationMaster est déployée sur les noeuds esclaves.



Pr. M. Ghazwan

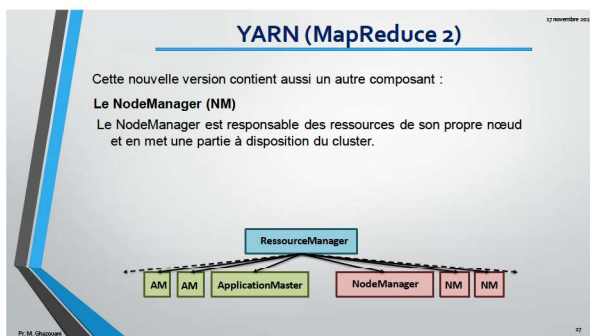
26

YARN (MapReduce 2)

Cette nouvelle version contient aussi un autre composant :

Le NodeManager (NM)

Le NodeManager est responsable des ressources de son propre nœud et en met une partie à disposition du cluster.



Pr. M. Ghazwan

27

YARN (MapReduce 2)

Le JobTracker a disparu de l'architecture, ou plus précisément, ses rôles ont été répartis différemment.

- L'architecture est maintenant organisée autour d'un **ResourceManager** dont le périmètre d'action est global au cluster et à des **ApplicationMaster** locaux dont le périmètre est celui d'un job ou d'un groupe de jobs.
- En terme de responsabilités, on peut donc dire que :
JobTracker = ResourceManager + ApplicationMaster.
- Un **ResourceManager** gère n **ApplicationMaster**, lesquels gèrent chacun n jobs.



Pr. M. Ghazwan

28

YARN (MapReduce 2)

17 novembre 2014

Resource Manager: Sub-components

1a Scheduler 1b Applications Manager 2 Node Manager 3 Container

Client Application Manager Scheduler Node Manager Container

MapReduce Status Job Submission Node Status Resource Request

1- Le **ResourceManager** est le remplaçant du **JobTracker** du point de vue du client qui soumet des jobs à un cluster Hadoop. Il n'a maintenant plus que deux tâches bien distinctes à accomplir :

1a) **Scheduler (ResourceScheduler)**
1b) **ApplicationsManager**

2- **Scheduler**

- Détermine comment les ressources disponibles du cluster seront partagées.
- Le Scheduler est responsable de l'allocation des ressources à des applications tournant sur le cluster.
- Les ressources allouées aux applications par le Scheduler pour leur permettre de s'exécuter sont appelées des **Containers**.
- Un Container désigne un regroupement de mémoire, de cpu, d'espace disque,...

29

YARN (MapReduce 2)

17 novembre 2014

Resource Manager: Sub-components

1a Scheduler 1b Applications Manager 2 Node Manager 3 Container

Client Application Manager Scheduler Node Manager Container

MapReduce Status Job Submission Node Status Resource Request

b- **ApplicationsManager**

L'**ApplicationsManager** est l'autorité qui gère les **ApplicationMaster** du cluster. A ce titre, c'est donc via l'**ApplicationsManager** que l'on peut :

- Superviser l'état des **ApplicationMaster**
- Relancer des **ApplicationMaster**

30

YARN (MapReduce 2)

17 novembre 2014

Resource Manager: Sub-components

1a Scheduler 1b Applications Manager 2 Node Manager 3 Container

Client Application Manager Scheduler Node Manager Container

MapReduce Status Job Submission Node Status Resource Request

2- **NodeManager**

Un **NodeManager** agit sur chaque nœud du cluster d'ordinateurs. C'est le destinataire de commandes du **ResourceManager** et envoie un signe de vie périodique (*heartbeat*). Chaque **NodeManager** est responsable des ressources de son propre nœud et en met une partie à disposition du cluster (**Container**).

3- **ApplicationMaster**

- L'**ApplicationMaster** est le composant spécifique à chaque application, il est en charge des jobs qui y sont associés.
- Lancer et au besoin relancer des jobs
- Négocier les Containers nécessaires auprès du **Scheduler**
- Superviser l'état et la progression des jobs.
- Un **ApplicationMaster**, de ce point de vue, il remplit un rôle de **TaskTracker**.

31

Travail pratique MapReduce

17 novembre 2014

Pr. M. Ghannem

32



33