CentraleSupélec

Big Data : Informatique pour les données et calculs massifs

5 – Technologies d'Hadoop

Stéphane Vialle

CentraleSupélec université PARIS-SACLAY | ÉCOLE DOCTORALE Sciences et technologies de l'information et de la communication (STIC) | RISEGrid | Grand Est

Stephane.Vialle@centralesupelec.fr
http://www.metz.supelec.fr/~vialle

CentraleSupélec

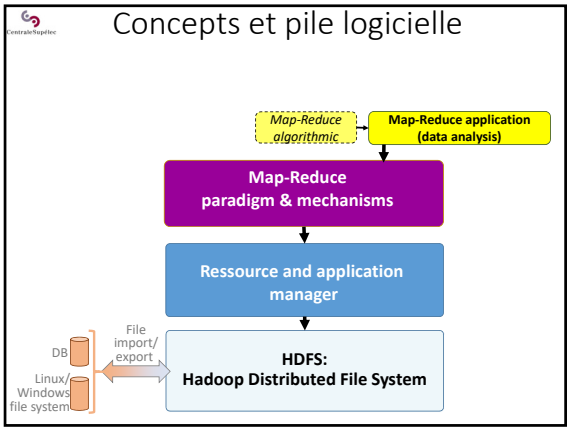
Technologies d'Hadoop

1. Principaux composants d'Hadoop
2. Principes du Map-Reduce d'Hadoop
3. Système de fichiers distribué d'Hadoop : HDFS
4. Allocation et gestion de ressources d'Hadoop …
   …pour un Map-Reduce

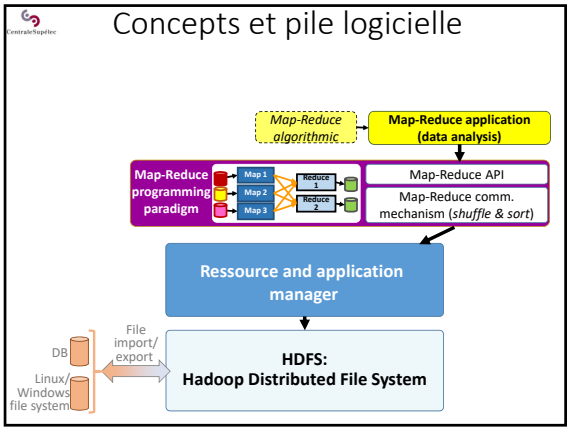
CentraleSupélec

1 - Principaux composants d'Hadoop

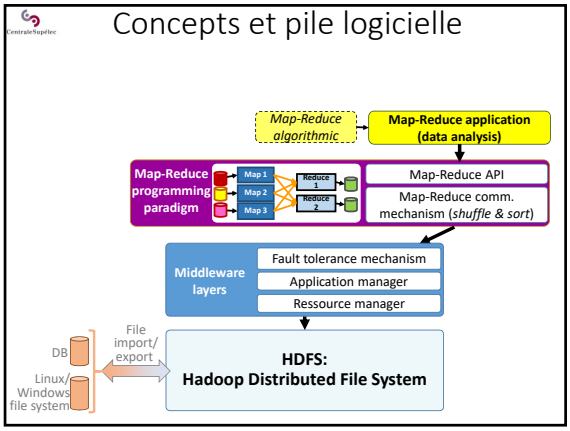
Concepts et pile logicielle



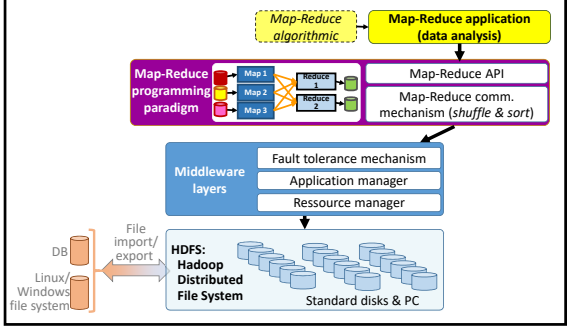
Concepts et pile logicielle



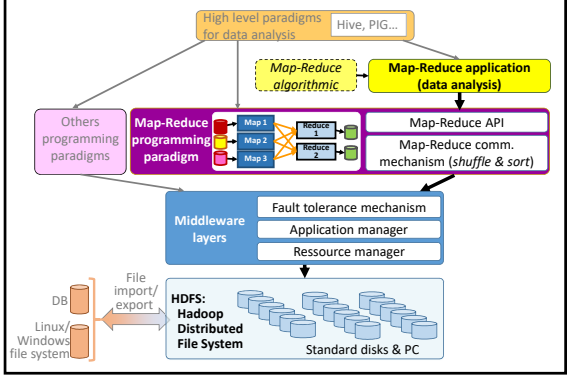
Concepts et pile logicielle



Concepts et pile logicielle



Concepts et pile logicielle



Concepts de « localité des données et des traitements »

C'est toujours l'accès aux données qui coute cher, pas le calcul lui-même une fois les données dans le cœur de calcul

HPC :

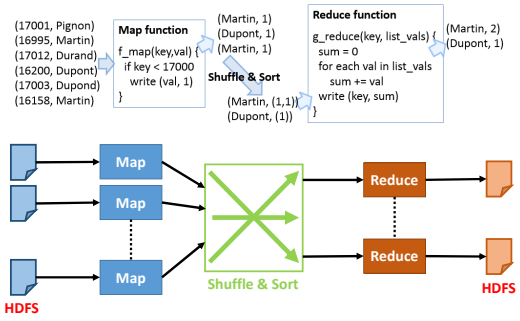
- Charger les données en RAM (utiliser assez de nœuds de calcul)
- Calculer tout ce qui est possible avec des données locales stockées dans la mémoire cache (calcul par bloc/tuile)
- Eviter les défauts de cache

Big Data :

- **Amener les codes de traitements aux données**
  → Transformer momentanément les nœuds de stockage en nœuds de traitement
- Eviter de déplacer des données (très volumineuses)

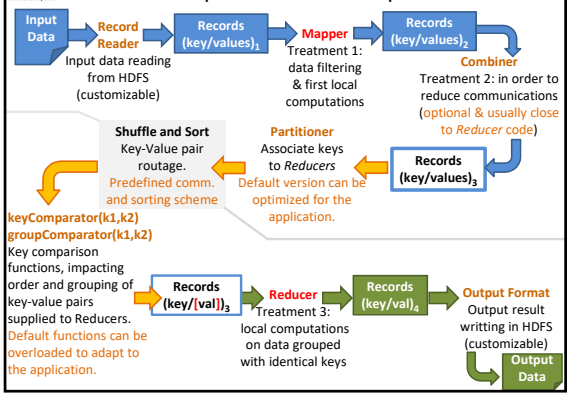
2 - Principes du Map-Reduce d'Hadoop

Chaîne d'opérations Map-Reduce

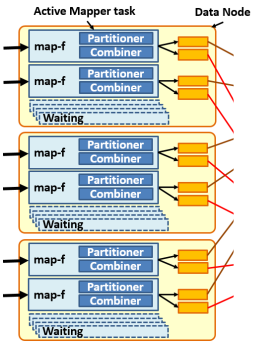


Suite de transformations de paires « clé-valeur »… voir plus loin

Chaîne d'opérations Map-Reduce



Déploiement des tâches



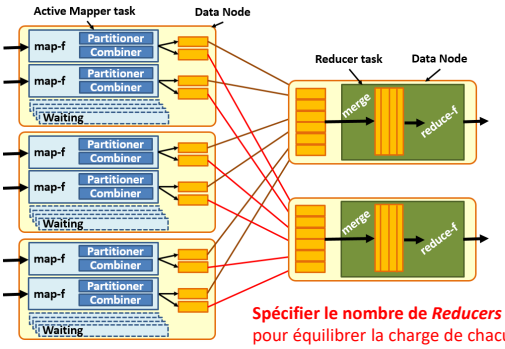
Hadoop installe des *Mappers* sur les nœuds où sont les données accédées

Hadoop crée un *Mapper* par bloc de fichier à lire (bloc classique : tranche de fichier de 64 Mo/128Mo)

Possibilité de limiter le nombre de *Mappers* actifs simultanément sur un PC en fonction de ses caractéristiques matérielles

Possibilité de limitations spécifiques à chaque nœud

Déploiement des tâches

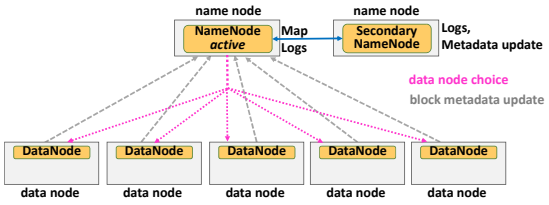


Spécifier le nombre de *Reducers* pour équilibrer la charge de chacun et limiter la RAM nécessaire

3 - Système de fichiers distribué d'Hadoop : HDFS

3.1 - Distribution et réplication des fichiers sous HDFS

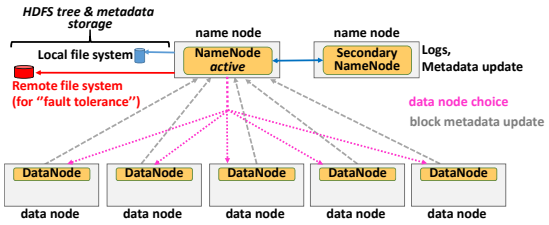
Mécanismes d'HDFS



Le *NameNode* conserve la cartographie du HDFS + les évolutions (les « logs »)
→ Permet de savoir où sont les fichiers

Le *secondary NameNode* stocke les logs et met à jour la cartographie (qd bcp de logs)
→ Sur un nœud à part pour pouvoir calculer la mise à jour sans ralentir le système

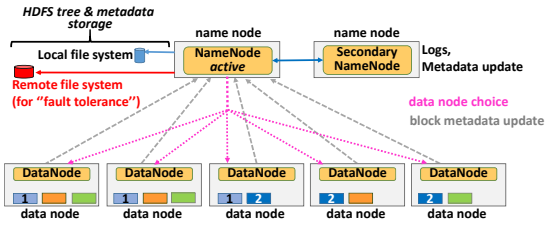
Mécanismes d'HDFS



Les metadonnées du HDFS sont stockés sur le file system classique, localement.

Un stockage distant permet de renforcer la tolérance aux pannes
(sans ses metadonnées, le HDFS est inexploitable)

Mécanismes d'HDFS

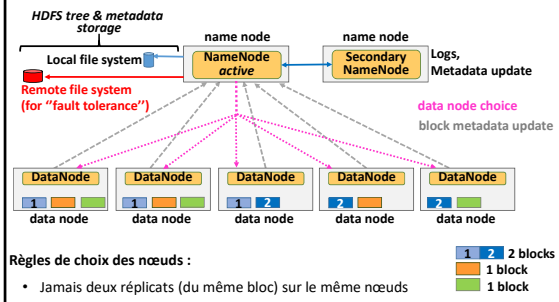


Chaque fichier est découpé en blocs de 64 ou 128 Mo

- **Distribués sur les différents nœuds**
  pour le passage à l'échelle en taille,
  pour la vitesse d'accès
- **Répliqués en n exemplaires (recopiés d'un nœud à un autre)**
  pour la tolérance aux pannes (habituellement n = 3).

1 2 2 blocks
1 block
1 block

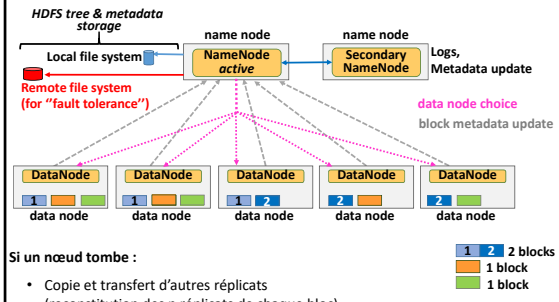
Mécanismes d'HDFS



Règles de choix des nœuds :

- Jamais deux réplicats (du même bloc) sur le même nœuds
- Des réplicats dans des « raks » différents (grands systèmes d'ensembles de raks)

Mécanismes d'HDFS



Si un nœud tombe :

- Copie et transfert d'autres réplicats (reconstitution des n réplicats de chaque bloc)
- Envoie des mise à jour des nœuds vers les NameNode

Mécanismes d'HDFS

Pourquoi des blocs de 64 Mo ?

Temps de « seek » : temps de positionnement au début du fichier sur le disque (disque standard, rotatif)

Tseek = 10ms

Bande passante disque std : Bw = 100 Mo/s

Tread = Q / Bw

On veut : Tseek < 1% Tread

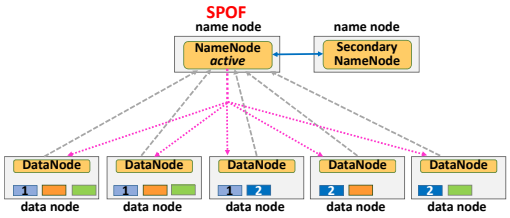
$\Leftrightarrow 10.10^{-3}$ s < (1/100).(Q/100) s

$\Leftrightarrow$ 100 (Mo) < Q

→ Des blocs de 64 Mo ou 128 Mo permettent de masquer les temps de seek

→ Au-delà : pas plus de gain, mais moins de distribution des fichiers, moins de vitesse de lecture

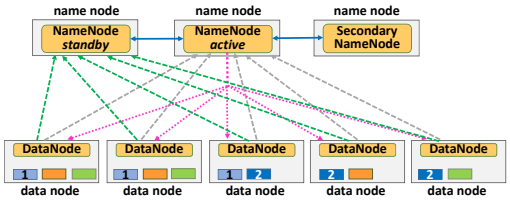
Mécanismes d'HDFS : « haute disponibilité »



Le NameNode est un SPOF : Single Point Of Failure

→ Si on perd le NameNode alors le File System d'Hadoop ne marche plus !

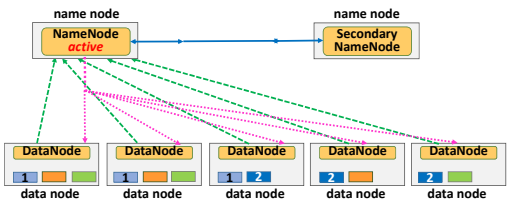
Mécanismes d'HDFS : « haute disponibilité »



Si on perd le NameNode alors le File System d'Hadoop ne marche plus !

→ On duplique le NameNode

Mécanismes d'HDFS : « haute disponibilité »



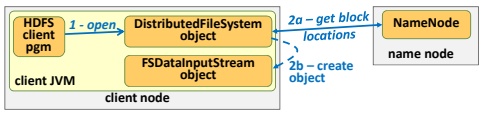
- Le NameNode en *standy* est passé *actif* « très rapidement »
- Il n'y a plus de SPOF

→ **On « ne sent plus passer la panne » : Haute Disponibilité**

3 - Système de fichiers distribué d'Hadoop : HDFS

3.2 - Lecture de fichiers sous HDFS

Mécanismes de lecture d'HDFS



0 – Création d'un objet permettant de s'interfacer à HDFS (un stub/proxy d'HDFS)

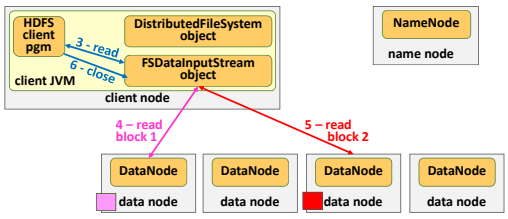
1 – Demande d'ouverture d'un fichier HDFS en lecture (« open »)

2 – Demande de localisation du fichier

2a - Le proxy interroge le NameNode : pour savoir quels blocs lire et où les lire
Le NameNode répond au proxy

2b - Le proxy crée et retourne un objet lecteur spécialisé sur le fichier ciblé

Mécanismes de lecture d'HDFS



3 – Le client exploite le lecteur du fichier

4 – Le lecteur accède au data node du premier bloc et récupère ce bloc

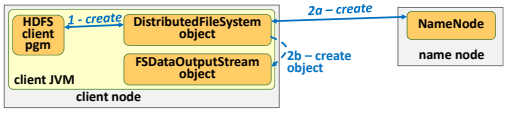
5 – Le lecteur accède au data node du second bloc et récupère ce bloc

6 – Le client referme le fichier en s'adressant au lecteur du fichier

3 - Système de fichiers distribué d'Hadoop : HDFS

3.3 - Ecriture de fichiers sous HDFS

Mécanisme d'écriture



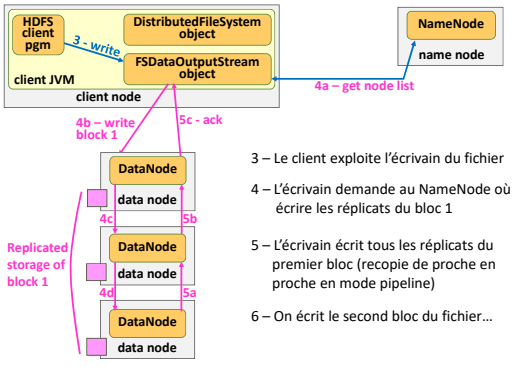
0 – Création d'un objet permettant de s'interfacer à HDFS (un stub/proxy d'HDFS)

1 – Demande d'ouverture d'un fichier HDFS en écriture (« create »)

2 – Demande de localisation des nœuds d'accueil des futurs blocs du fichier
2a - Le proxy interroge le NameNode : pour savoir où écrire les blocs
Le NameNode répond au proxy

2b - Le proxy crée et retourne un objet écrivain spécialisé sur le fichier ciblé

Mécanisme d'écriture



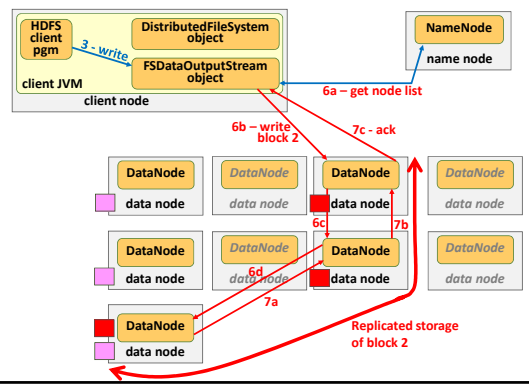
3 – Le client exploite l'écrivain du fichier

4 – L'écrivain demande au NameNode où écrire les réplicats du bloc 1

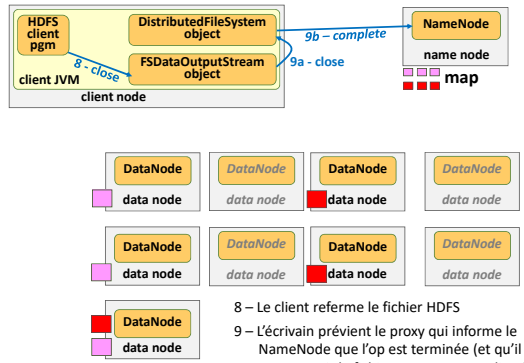
5 – L'écrivain écrit tous les réplicats du premier bloc (recopie de proche en proche en mode pipeline)

6 – On écrit le second bloc du fichier…

Mécanisme d'écriture



Mécanisme d'écriture



8 – Le client referme le fichier HDFS

9 – L'écrivain prévient le proxy qui informe le NameNode que l'op est terminée (et qu'il peut rouvrir le fichier pour un autre client)

3 - Système de fichiers distribué d'Hadoop : HDFS

3.4 - Bilan des accès aux fichiers sous HDFS

Bilan des accès aux fichiers HDFS

Le client crée un objet « proxy » (ou « stub ») local qui fait toute l'interface avec l'architecture HDFS (NameNode et DataNodes) :

- SIMPLE
- **Pb 1 : Tous les accès (R/W) sont séquentiels** : bloc après bloc, et séquentiel au sein d'un bloc !
- **Pb 2 : Le client n'aura pas assez de RAM ou même de disque** pour rapatrier en local un gros fichier HDFS !

→ OK pour importer/exporter des données vers d'autres File Systems ou BdD (sera long mais fonctionnera)

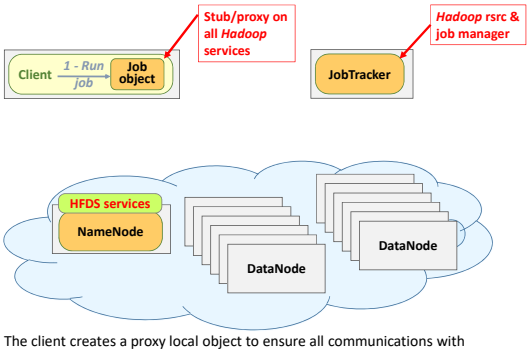
→ **PAS la bonne stratégie pour traiter des fichiers HDFS (des Big Data) : il manque la localisation Data/Calculs**

4 - Allocation et gestion de ressources d'Hadoop…

… pour un Map-Reduce

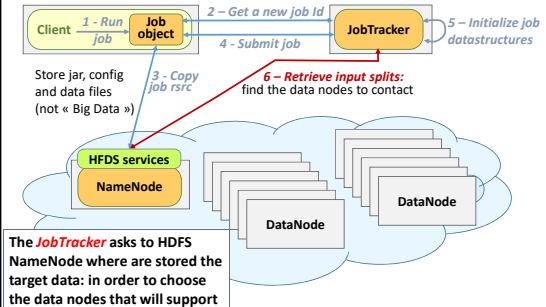
Version 1

Gestion des ressources v1



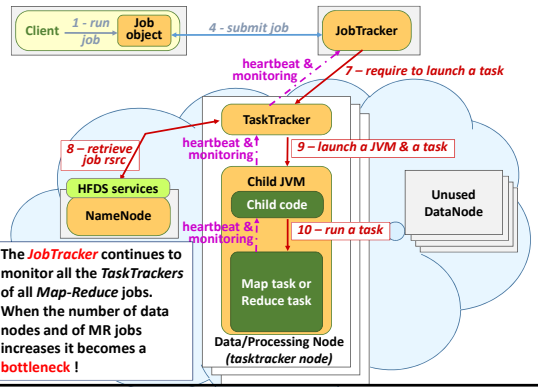
The client creates a proxy local object to ensure all communications with *Hadoop rsrc & job* manager, and with *HDFS* services

Gestion des ressources v1



The *JobTracker* asks to HDFS NameNode where are stored the target data: in order to choose the data nodes that will support Map and Reduce tasks →

Gestion des ressources v1



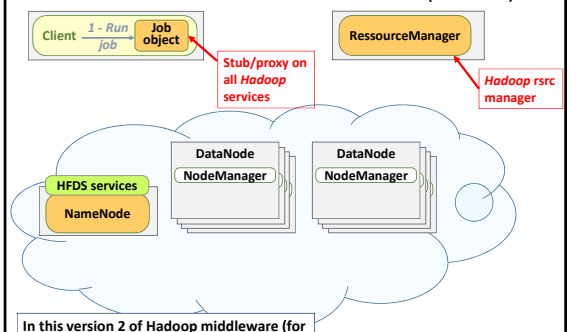
The *JobTracker* continues to monitor all the *TaskTrackers* of all *Map-Reduce* jobs. When the number of data nodes and of MR jobs increases it becomes a bottleneck !

4 - Allocation et gestion de ressources d'Hadoop…

… pour un Map-Reduce

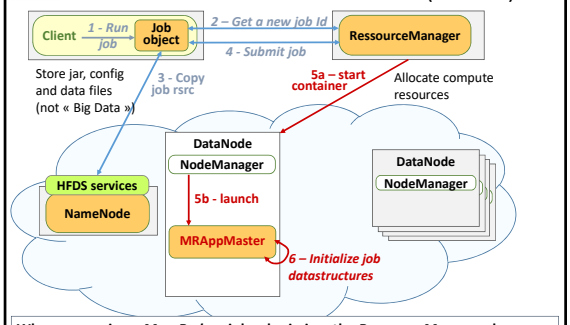
Version 2 (YARN) : passage à l'échelle amélioré

Gestion des ressources v2 (YARN)



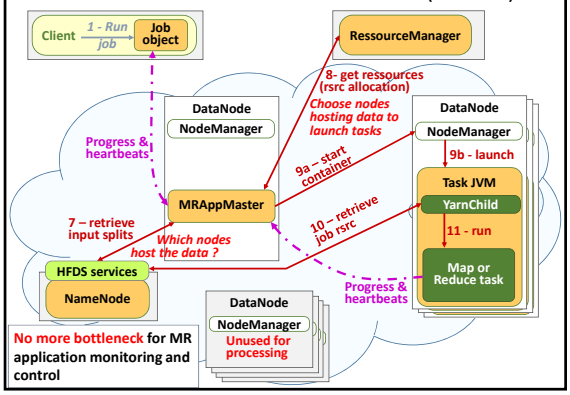
In this version 2 of Hadoop middleware (for Map-Reduce), the RessourceManager is no longer a Job manager (just a rsrc manager)

Gestion des ressources v2 (YARN)



When processing a *Map-Reduce* job submission, the *RessourceManager* chooses a Data Node to launch a *Map-Reduce Application Master* that will manage and monitor the set of MR application tasks.

Gestion des ressources v2 (YARN)



No more bottleneck for MR application monitoring and control

Exécution spéculative

Hadoop lance de nombreuses tâches (map, reduce, …).
Certaines peuvent « planter » ou « ralentir ».

Le TaskTracker (MR v1) ou l'ApplicationManager (MR v2) monitore fortement les exécutions des tâches, et détectent ces « ralentissements ».

Hadoop peut faire une exécution spéculative : il lance de nouvelles instances des tâches « en retard », et dès qu'il obtient des résultats d'une tâche, il tue son doublon.

Mais cette démarche a un coût :
- en charge de calcul (création fréquentes de tâches redondantes)
- en déplacement de données (surtout si on redonde un *reducer*)

On peut débrayer ce comportement (ex : cluster HPC très fiable)

Technologies d'Hadoop