

# HADOOP

**HADOOP  
FEUILLE  
DE ROUTE**

# Préface

*Nous avons profité de l'intervention de Doug Cutting aux Cloudera Sessions Paris en octobre 2014 pour l'interviewer. Doug Cutting n'est rien de moins que le papa d'Hadoop, et le Chief Architect de Cloudera. Quel meilleur choix pour une préface qu'une transcription de cette interview ?*

*Merci à Cloudera France de nous avoir permis cet échange.*

► **Quel effet cela fait-il de voir qu'Hadoop est en train de devenir la plateforme de référence pour le stockage et le calcul de données, dans les grandes entreprises ?**

Objectivement, c'est une très bonne chose. C'est une technologie qui est faite pour cela. Au plan émotionnel, c'est très satisfaisant, mais je dois admettre que j'ai eu beaucoup de chance. J'étais au bon endroit au bon moment, et c'est tombé sur moi. Si cela n'avait pas été moi, quelqu'un d'autre l'aurait fait depuis le temps.

► **C'est drôle, car hier vous mentionniez les papiers que Google a publiés sur GFS et sur MapReduce, et vous sembliez surpris que personne d'autre ne les ait mis en œuvre. Comment l'expliquez-vous ? Les personnes concernées étaient-elles découragées par l'immensité de la tâche, ou bien... ?**

J'ai l'avantage d'avoir déjà contribué à des logiciels open source. J'ai travaillé sur des moteurs de recherche ; je pouvais apprécier l'apport des technologies, je comprenais le problème, et comment ces deux aspects se combinent. Je pense avoir été dans l'industrie logicielle suffisamment longtemps pour savoir ce que cela signifie de construire un projet utile et utilisable. À mon avis, personne d'autre n'avait alors assez de recul sur tous ces aspects. J'ai pu profiter de ces papiers et les mettre en œuvre, en open source, pour en faire profiter les autres. C'est comme ça que je vois les choses. Ce n'était pas planifié.

► **Vous attendiez-vous à un tel impact ?**

Non, pas du tout.

► **Une question un peu plus technique maintenant : vous parliez hier de tous ces outils qui se créent à partir d'Hadoop, apportant de nouvelles technologies et de nouveaux usages pour la donnée. Comment percevez-vous l'évolution d'Hadoop, en termes d'architecture, pour qu'il offre de plus en plus de capacités dans le futur ?**

C'est une architecture très générique. À plus d'un titre c'est, comme je le disais, un système d'exploitation. J'expliquais qu'un système d'exploitation propose du stockage, un ordonnanceur, des mécanismes de sécurité... Le premier défi, c'est de pouvoir supporter des applications très diverses. Je pense que l'architecture actuelle [d'Hadoop] n'est pas loin de permettre ce large spectre d'applications.

► **Tout comme les systèmes d'exploitation qui n'ont pas vraiment changé depuis Windows ?**

Oui, pas dans leurs fondamentaux, depuis les années 60. Ces fonctionnalités de base constituent une plateforme sur laquelle vous pouvez développer plusieurs applications différentes, qui dans un certain sens partagent le matériel. En fait, c'est un "système d'exploitation Java" qui sait se faire discret et laisse les applications se partager le matériel.

► **Qui fournit des abstractions, comme le rappelait Jim Baum.**

Exactement.

► **Pour faire face à la complexité.**

C'est là, je crois, un rôle qu'Hadoop joue de plus en plus. Je suis conscient que cela demandera un changement radical d'architecture. Peut-être verra-t-on émerger des systèmes de fichiers alternatifs [à HDFS]... On verra.

► **On voit arriver des outils, comme Kafka pour l'agrégation de logs sur plusieurs data centers, Storm pour le traitement de flux, et tant d'autres. Voyez-vous d'autres usages de la donnée qui n'ont pas encore vu le jour ? On peut déjà la rechercher, l'indexer, la diffuser et la traiter en temps réel...**

Nous pensons qu'il y a beaucoup d'opportunités pour des applications plus verticales, très spécifiques, au sein des différentes industries. Des outils capables de traiter des images, des données changeantes... Il reste beaucoup de domaines qui ne sont pas encore outillés. Sans parler des applications intégrées pour l'assurance, la banque, etc.

Certaines entrevoient des modèles commerciaux, d'autres des modèles open source. Pour l'instant, ce que les gens voient ce sont surtout des outils de bas niveau que l'on peut combiner.

Je pense que de plus en plus, des outils de haut niveau vont profiter d'implémentations open source, rendant universelle la plateforme sous-jacente. Le processus est déjà commencé.

# Table des matières



# Qu'est-ce qu'Hadoop ?

*Hadoop est né dans un contexte particulier, celui de l'émergence du Big Data. Porté par les Géants du Web, il se démarque radicalement des logiciels et des bases de données historiques du marché.*



# Les origines

Nous sommes en 2002. Deux ingénieurs, Doug Cutting et Mike Cafarella, décident de s'attaquer à un défi de taille : rendre Lucene, le célèbre moteur de recherche open source d'Apache, capable d'absorber le contenu des milliards de pages du web. C'est le projet **Nutch**<sup>1</sup>. Leur inspiration s'est portée tout naturellement vers un modèle évident : Google, dont les labs ont publié en 2003 un papier fondateur : *The Google File System*<sup>2</sup>. Sur cette base, ils ont construit NDFS, le **Nutch Distributed File System**. NDFS, comme son ancêtre GFS, est capable de stocker de grandes quantités de données sur un grand nombre de serveurs, de manière fiable.

Le problème du stockage résolu, restait à transformer le système en un véritable moteur de recherche, et donc à lui ajouter un mécanisme d'indexation. La solution est venue peu de temps après, en 2004, avec ce qui allait devenir le deuxième papier fondateur d'Hadoop : **MapReduce : Simplified Data Processing on Large Clusters**<sup>3</sup>. Les auteurs y décrivent l'algorithme au cœur du moteur d'indexation web de Google, à savoir MapReduce.

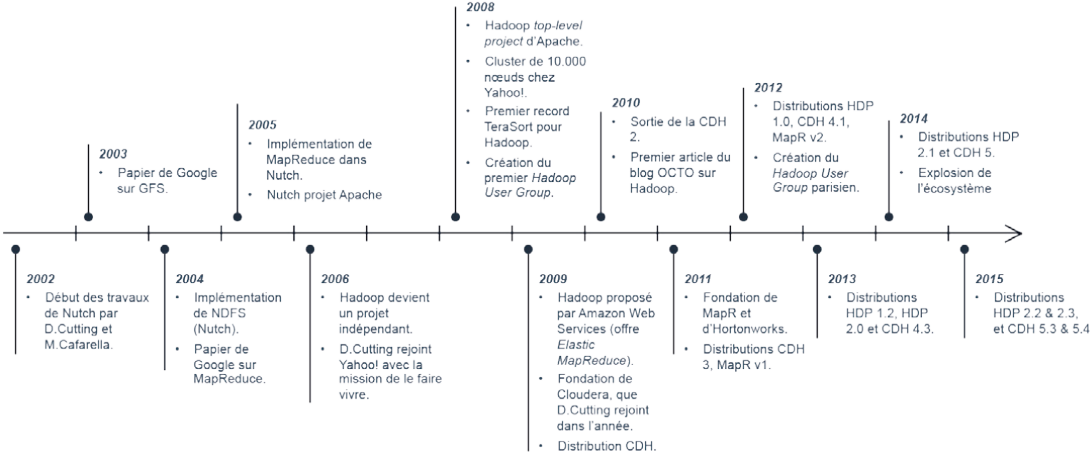
<sup>1</sup> Nutch est toujours actif : <http://nutch.apache.org/>  
<sup>2</sup> <http://static.googleusercontent.com/media/research.google.com/fr/archive/gfs-sosp2003.pdf> (S.Ghemawat, H.Gobioff, S-T.Leung)  
<sup>3</sup> <http://static.googleusercontent.com/media/research.google.com/fr/archive/mapreduce-osdi04.pdf> (J.Dean, S.Ghemawat)

L'architecture sous-jacente de Nutch – NDFS et MapReduce – s'est avérée assez générique pour justifier la création d'une plateforme à part entière, dont Nutch ne serait qu'une application parmi d'autres. C'est ainsi qu'en 2006, le projet Apache Hadoop a vu le jour. Doug Cutting a entre-temps rejoint Yahoo!, qui est un très gros utilisateur et contributeur d'Hadoop, et fait tourner en production certains des plus gros clusters Hadoop mondiaux. Doug est aussi membre du comité de direction de la fondation Apache.

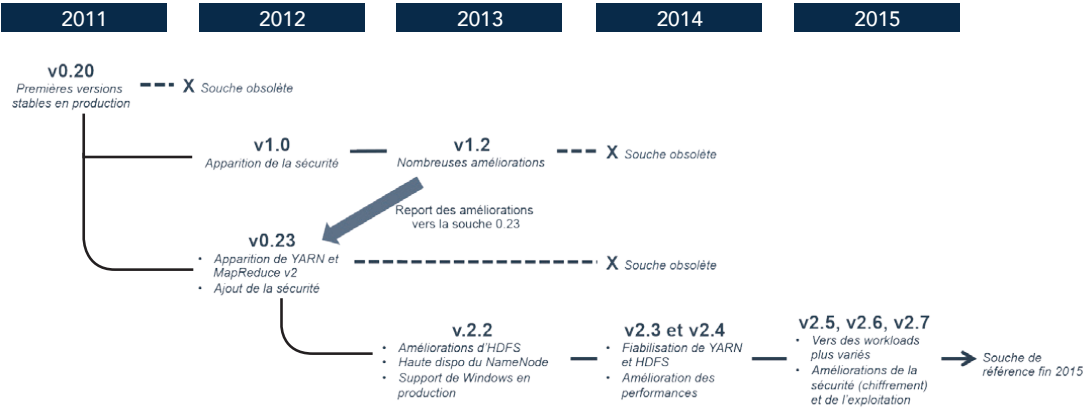
## Mais d'où vient le nom si curieux de ce logiciel ?

Mais d'où vient le nom si curieux de ce logiciel ? De la mythologie ? De la science-fiction ? Non, d'un babillage d'enfant. Doug Cutting raconte que son fils avait une peluche en forme d'éléphant jaune, qu'il avait baptisée... Hadoop. L'enfant devenu grand doit sourire en voyant le nom et le logo d'Hadoop repris frénétiquement par toute la sphère Big Data, et en imaginant les DSI de la terre entière sucer leur pouce en faisant des rêves chatoyants peuplés de gentils éléphants jaunes. Les lecteurs chez qui Hadoop suscite des cauchemars pourraient bien retrouver le sommeil à la lecture de ce papier. Quant à ceux qui voient des éléphants roses, ils peuvent poser le livre et le reprendre un peu plus tard.

## Hadoop : les grandes dates clefs



## Généalogie des versions d'Apache Hadoop



# Hadoop et Big Data, une question d'opportunités

Si son nom est celui d'une peluche (voir encadré précédent), Hadoop est pourtant loin d'être un jouet. Il motive une part significative des développements open source. En effet, plus de 10% des projets mis en avant par la fondation Apache (les *top-level projects*) font partie de la galaxie Hadoop. La galaxie est en continuelle évolution, et nous sommes bien placés pour le savoir car nous suivons l'actualité de près ! La R&D open source d'Hadoop est portée par des sociétés comme Yahoo!, Facebook, eBay, Criteo, et une myriade d'éditeurs. Ils utilisent Hadoop au quotidien pour analyser toutes les empreintes de navigation que nous laissons sur leurs sites et ceux de leurs partenaires, pour nous recommander des produits, du contenu, pour prédire ce qui est le plus susceptible de nous plaire, en optimisant des dizaines de milliers de paramètres. Parfois pour nous espionner, aussi. Les plus gros clusters Hadoop dans le monde frôlent aujourd'hui les 10000 nœuds. En France, c'est Criteo qui semble mener la course avec plus de 1000 nœuds.

Web, mobile, réseaux sociaux, *machine to machine* et bientôt Internet des objets :

les entreprises, portées par l'exemple des réussites de compagnies comme Google et Facebook, prennent conscience que savoir traiter et faire parler ses données est un véritable **avantage compétitif** et ouvre la voie à de **nouveaux business models**. Ce monde, régi par l'analyse de dizaines de téra-octets, voire de péta-octets, est celui du Big Data.

## Big data ?

Ce n'est pas qu'une question de taille. Néanmoins, on considère que le seuil à partir duquel on « fait du Big Data » est celui à partir duquel les approches classiques ne sont plus praticables à coût raisonnable. En pratique, on juge qu'à partir de 10 To de données, on est dans le Big Data (ce nombre peut varier selon les caractéristiques des données à traiter). Plus le volume stocké augmente, plus Hadoop devient économiquement intéressant face aux appliances MPP<sup>4</sup> commerciales.

Tous les cas d'utilisation d'Hadoop illustrent une motivation essentielle du Big Data : exploiter plus de données, plus vite, qu'elles soient déjà connues, ou issues de nouvelles sources à combiner aux existantes pour leur donner un contexte plus riche. Souvent, les données disponibles ne suffisent plus. Les nouvelles sources de données sont pour cette raison souvent externes : externes au département ou à l'organisation, voire complètement étrangères au domaine (météo, web, réseaux sociaux, données de géolocalisation, etc.).

Qui dit nouvelles informations, ou informations plus fines, dit plus de calculs et une combinatoire plus importante. Les idées nouvelles appellent des besoins d'infrastructure plus importants, et en retour, l'innovation technique suscite de nouvelles idées par les possibilités accrues qu'elle offre. Une dynamique se met en place, et il faut une plateforme qui sache grandir petit à petit, à partir de besoins modestes. Il n'est plus question de déclencher une dépense pharaonique d'un coup sur un gros investissement censé subvenir aux besoins des 10 ans à venir... Il est irréaliste de prévoir ces besoins aujourd'hui, tant ces derniers sont devenus volatiles !

Traditionnellement, dans les SI, de telles quantités de données sont conservées dans des armoires de stockage (des SAN), ou des appliances de bases de données MPP<sup>4</sup> comme

**Teradata.** Cependant, ces solutions exigent des investissements importants dès lors que le volume dépasse les quelques dizaines de téra-octets. Les nouveaux besoins suscités par Big Data engendrent des quantités plus grandes encore, plusieurs centaines de téra-octets ou plus, et qui vont en augmentant. C'est là qu'Hadoop rentre en jeu, avec ses capacités de scalabilité linéaire de stockage à partir de matériel courant.

*Le premier enjeu d'Hadoop est de stocker et d'analyser toute cette masse de données en maîtrisant son budget.*

## Quelques objectifs d'une démarche Big Data et leurs grandes étapes

## MIEUX CONNAÎTRE SES CLIENTS

- Réconciliation parcours digitaux / CRM
- Segmentation comportementale
- Vision 360° du client

## ANALYSER DES SIGNAUX FAIBLES ET PRÉDIRE

- Collecte de données exogènes
- Recherche de corrélations internes/ externes
- Modèle prédictif

## FAIRE AUSSI BIEN (VOIRE MIEUX) AVEC MOINS DE MOYENS

- Mise en place d'architectures mixtes
- Augmentation de profondeur d'historique
- Data Lake

④ MPP : Massive Parallel Processing. Le terme s'applique aux bases de données distribuées sur des nœuds indépendants ; lorsqu'une requête s'exécute, elle est découpée en tâches indépendantes envoyées aux unités de calcul. Un nœud de coordination se charge de rassembler les résultats et de les envoyer au client.



# Les gènes d'Hadoop

*Hadoop propose une nouvelle forme d'architecture capable de traiter de gros volumes de données. En cela, il se distingue des bases de données couramment utilisées.*

## ○ Autoriser des formats de données flexibles

Hadoop est principalement une technologie orientée vers le traitement massif de fichiers, et en cela s'apparente à une plateforme de batch. Les usages typiques des bases de données (accès transactionnels) s'y sont ajoutés par la suite.

Pour comprendre la nouveauté apportée par Hadoop, expliquons le fonctionnement des bases de données. Une base de données relationnelle structure les données de plusieurs façons, qui constituent ce que l'on appelle un schéma :

- elle contraint le format de la donnée,
- elle associe les éléments constitutifs de la donnée sous forme de tables,
- elle lie les tables entre elles par le biais des contraintes d'intégrité.

Avec Hadoop, le fonctionnement est différent. Le parti pris est de stocker les fichiers sans contraintes, **pour en interpréter la structure au moment de les lire**. On parle de *schema on read*. Ce principe, simple en apparence, est une évolution importante dans l'approche de la donnée. En effet, il offre plusieurs avantages :

- La donnée reste dans son format natif, on n'est pas obligé de laisser de côté les fragments qui ne rentrent pas dans un schéma fixé *a priori*.
- Quand le schéma doit évoluer (parce que l'on sait exploiter de nouveaux champs que l'on avait négligés, par exemple), il n'est pas nécessaire de reprendre le stock longuement accumulé avec de coûteux traitements de migration.
- On peut conserver dans un même espace de stockage de la donnée structurée (bases de données) et non structurée (logs, texte, etc.) – qui peut le plus peut le moins !

La contrepartie, bien sûr, c'est que le programme de traitement devra faire face à la diversité de formats accumulés au gré des sources, des époques de collecte.

## ○ Limiter les déplacements de données

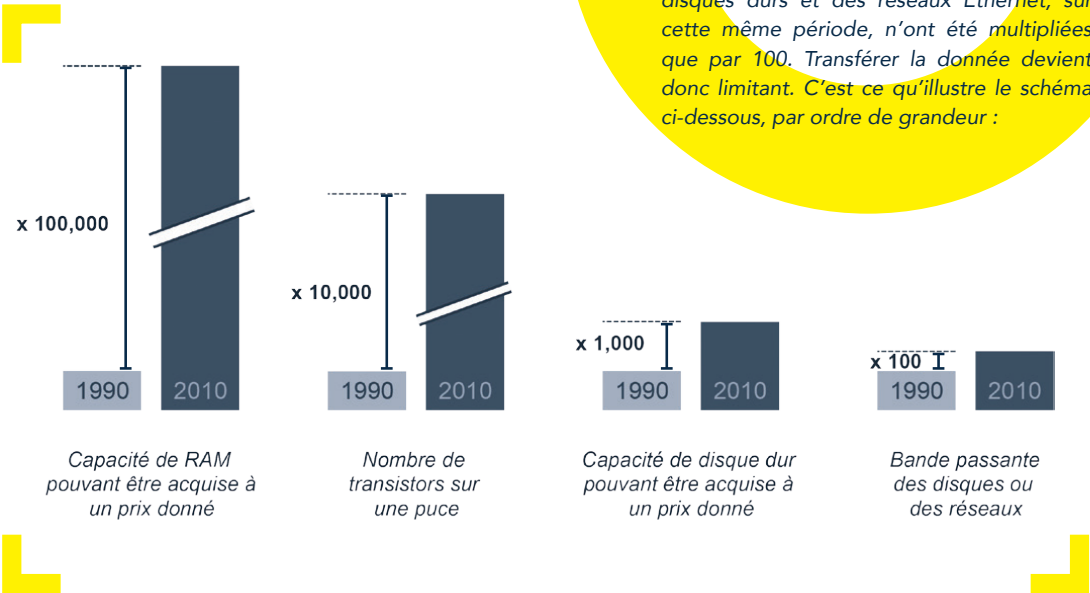
L'architecture de la plateforme a été influencée dès le début par un constat : **le goulet d'étranglement, dans le traitement de données**, est au niveau de l'accès à ces données, et non pas, comme on pourrait le penser naïvement, au niveau de la puissance machine (CPU). En effet, avec les progrès techniques, les débits de lecture et d'écriture des disques augmentent bien moins vite que la fréquence des microprocesseurs.

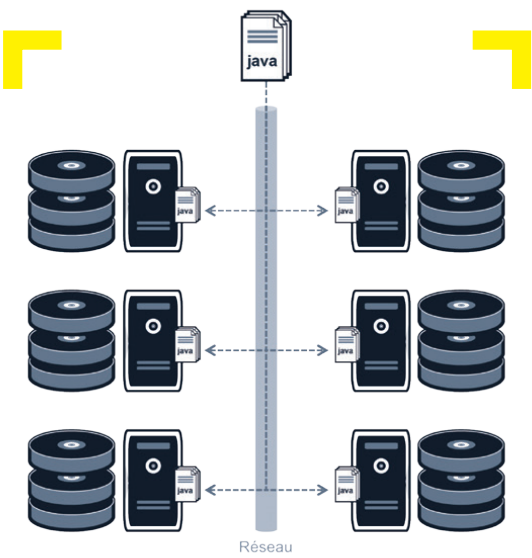
Hadoop a ainsi fait de nécessité vertu. D'une part, la distribution des données sur plusieurs serveurs, rendue nécessaire par le volume à manipuler, permet d'augmenter le nombre de « tuyaux » par lesquels la donnée est récupérée des disques.

D'autre part, Hadoop est conçu pour limiter au maximum le déplacement de données à travers le réseau : **Hadoop fait transiter les traitements plutôt que les données**. C'est ce que l'on appelle la colocalisation traitements/données, mise en œuvre en particulier avec le modèle MapReduce (cf. chapitre 2).

### Quelques observations relatives à la loi de Moore

Entre 1990 et 2010... le coût du Go de RAM a été divisé par 100 000, celui du Go de disque dur par 1 000. Le nombre de transistors sur un processeur Intel a été multiplié par 10 000. Face à ces progrès, les bandes passantes des disques durs et des réseaux Ethernet, sur cette même période, n'ont été multipliées que par 100. Transférer la donnée devient donc limitant. C'est ce qu'illustre le schéma ci-dessous, par ordre de grandeur :





Il est plus rapide d'envoyer le code d'un programme (quelques Mo), que de rapatrier plusieurs Go ou To de données à travers le réseau sur un serveur d'exécution

### ◉ Pouvoir adapter la puissance au besoin

Contrairement aux appliances de bases de données<sup>③</sup>, Hadoop n'a pas été pensé comme une plateforme monolithique sous stéroïdes censée répondre immédiatement à tous les besoins. En effet, un cluster Hadoop s'agrandit au fil du temps, par ajout incrémental de ressources de stockage et de calcul, c'est-à-

dire par ajout de serveurs. Cette stratégie a été inscrite dès le départ dans l'architecture même d'Hadoop.

Les serveurs que l'on ajoute au fil du temps n'ont pas besoin d'être haut de gamme: Hadoop s'accommode très bien de serveurs «normaux», ce qui facilite les investissements.

Dans le cas simplifié où tous les serveurs sont identiques (même CPU, même mémoire, même capacité de disque dur), la puissance de traitement et de stockage croît linéairement avec le nombre de serveurs. Pour cette raison, **on parle de scalabilité linéaire**.

### Le commodity hardware

On entend souvent parler de commodity hardware pour désigner ces serveurs «normaux» pouvant être utilisés avec Hadoop. Cela ne veut pas pour autant dire que n'importe quel serveur d'entrée de gamme fait l'affaire : la RAM et les performances des I/O (disques et réseaux) sont essentielles pour obtenir des performances acceptables. Ce sont typiquement les serveurs physiques qui sont déjà au catalogue de l'entreprise.

### ◉ Garantir une fiabilité de fonctionnement

La fiabilité de fonctionnement est essentielle dès que l'on parle de traitements distribués. En effet, la probabilité qu'une machine du cluster subisse un incident à un instant  $t$  croît avec la taille dudit cluster – il faut vivre avec.

*Pour illustrer ce point, considérons des machines avec une disponibilité de 99 % chacune. Lorsque l'on les regroupe dans un cluster de 100 machines, la probabilité qu'à un instant  $t$  une machine au moins soit tombée est de 63 % ( $1 - 0,99^{100}$ ). On a donc à tout moment environ 2 chances sur 3 pour qu'une machine du cluster soit défaillante.*

Hadoop a pris le parti de garantir une fiabilité quasi complète de son fonctionnement sans dépendre de la disponibilité totale du cluster. Le mécanisme qui a été retenu consiste à répliquer  $N$  fois les données,  $N$  pouvant être configuré au cas par cas. Par exemple, si on prend un nombre couramment utilisé de 3 répliques, cela signifie que l'on peut perdre 2 nœuds du cluster stockant une donnée, sans perdre cette donnée. Dès qu'Hadoop constate la perte de ces serveurs, il recrée les répliques manquantes sur d'autres serveurs, rétablissant

le facteur de réplication configuré. **On parle de design for failure<sup>④</sup>**, c'est-à-dire la capacité de l'architecture à fonctionner normalement même en présence de pannes physiques.

### ◉ Proposer une plateforme ouverte

Hadoop est aujourd'hui une véritable plateforme ouverte. D'une part, il existe de nombreux composants qui viennent se brancher au-dessus d'Hadoop pour lui offrir des interfaces de haut niveau, dont nous parlerons en détail dans le chapitre 3.

D'autre part, il est désormais possible de substituer à HDFS et MapReduce des alternatives spécifiques lorsque les besoins l'exigent. Hadoop s'occupe alors de fournir les API et la tuyauterie permettant de lier le tout harmonieusement ; on entend parfois parler de «système d'exploitation Big Data». Et c'est de plus en plus vrai.

### ◉ Open source

Hadoop est porté par l'Apache Software Foundation. Son code source est ouvert à tous, de même que la liste des améliorations et des bugs : libre à chacun de participer. La communauté est solide ; les principaux contributeurs au code d'Hadoop proviennent des éditeurs (Hortonworks, Cloudera, Microsoft...), et des Géants du Web (Yahoo!, Facebook, LinkedIn, Twitter...).

③ Une appliance de bases de données est une solution clefs en main comprenant un logiciel de bases de données, pré-installé sur une plateforme matérielle conçue et optimisée exprès pour lui.

④ Une explication en vidéo du principe de design for failure se trouve ici : <https://www.youtube.com/watch?v=RO6Glnqa-gU>



# Le socle + technique d'Hadoop

*Grâce à ce chapitre, vous comprendrez tout d'HDFS, de MapReduce et de YARN, qui constituent le socle fondamental d'Hadoop. Toute la philosophie de l'architecture d'Hadoop se retrouve dans ce triplet d'outils !*



# HDFS, un système de fichiers distribué

HDFS – Hadoop Distributed FileSystem – est la couche de stockage native d'Hadoop. Il s'agit d'un système de fichiers logique, avec la notion de répertoires contenant des fichiers. Physiquement, les fichiers sont découpés en blocs d'octets, et répartis sur les nœuds d'un cluster. Ces nœuds de stockage portent le nom de **data nodes**.

Comme nous l'avons vu précédemment, les blocs de chaque fichier sont répliqués afin d'assurer la redondance en cas de panne, et de faciliter la colocalisation des traitements et des données. La réplication se fait de manière intelligente, en tenant compte de la topologie réseau configurée : les répliques d'un même bloc sont autant que possible hébergées par des nœuds « éloignés », typiquement dans des racks ou des sous-réseaux différents. L'accès aux fichiers complets est ainsi garanti, même lorsqu'un équipement réseau tombe. Malin.

L'autre élément clef de l'architecture d'Hadoop est le **name node**. Les fichiers étant découpés

en blocs répartis sur les *data nodes*, il faut bien stocker quelque part la liste des blocs composant un fichier donné, et leur localisation – pour les lire ou pour appliquer la stratégie de colocalisation lorsqu'un traitement se lance. Ces services d'annuaire sont ainsi assurés par le *name node*.

Par abus de langage, on désigne par les mêmes termes (data node et name node) les services logiciels d'Hadoop (des

démons) et les rôles que les serveurs jouent au sein du cluster.

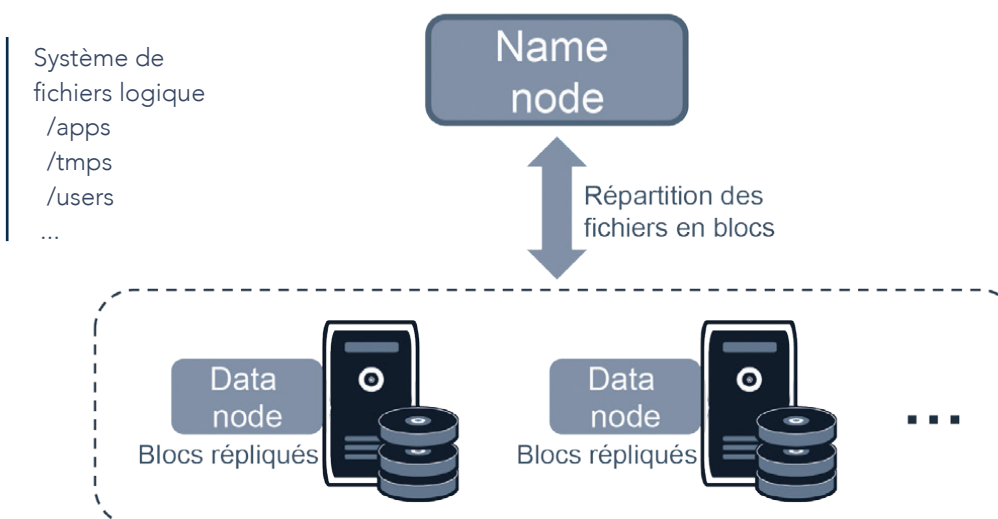
On ne peut quand même pas faire n'importe quoi avec HDFS. Hadoop ayant été conçu avec une logique de batch sur des très gros fichiers, c'est dans ce cas nominal qu'HDFS est le plus efficace car il est optimisé pour les parcourir d'un bout à l'autre. Dans le cas d'un grand nombre de petits fichiers, il est préférable de les regrouper en archives pour les traiter en blocs.

*HDFS a été conçu pour permettre les écritures par écrasement ou ajout en fin de fichier, et non les écritures aléatoires en milieu de fichier.*

En outre, HDFS a été conçu pour permettre les écritures par écrasement ou ajout en fin de fichier, et non les écritures aléatoires en milieu de fichier. Cette limitation sera levée dans le futur. En attendant ces deux caractéristiques font qu'HDFS est un très mauvais candidat pour remplacer un serveur de fichiers documentaire ! Mais il est très bon comme puits de données, structurées et non structurées. Comme un gros (très gros) DVD, il répond au pattern WORM : *Write Once, Read Many*.

Il est important de noter que le *name node* est le composant le plus critique d'un cluster HDFS. S'il est perdu, et non récupérable, alors toutes les données hébergées sur le cluster ne sont qu'une soupe d'octets inutilisable. Il est donc essentiel de le sauvegarder ou de le rendre hautement disponible ; HDFS assure nativement cette sécurité.

## L'architecture logique d'HDFS



# MapReduce, un algorithme distribué

## ◉ Description de l'algorithme

Le principe de MapReduce est simple :

MAP                      REDUCE

Diviser pour conquérir, et rassembler pour mieux régner

L'algorithme fonctionne en deux temps. La première étape, appelée **Map**, consiste à distribuer des traitements au plus proche des données qu'ils utilisent. La deuxième étape, Reduce, est chargée de l'agrégation du résultat final. L'étape de Reduce est facultative, par exemple lorsque aucune agrégation n'est nécessaire.

En pratique, lors de l'étape de Map, l'algorithme interroge le name node pour savoir sur quels data nodes les données d'entrée sont stockées. Cette information prend la forme d'une liste de *splits*, c'est-à-dire d'ensembles de blocs HDFS contigus avec pour chacun le data node hôte. Le code du traitement est envoyé à tous les data nodes hôtes : chacun exécute le code sur les splits qu'il possède. Les traitements de Map sont exécutés en parallèle.

À l'issue de l'étape de Map, on dispose ainsi d'autant de résultats intermédiaires qu'il y a de

splits impliqués. Il reste à les combiner pour obtenir le résultat final du calcul : c'est l'étape de Reduce, qui produit les fichiers résultats.

Dans certains cas, on peut avoir besoin de lancer plusieurs tâches de Reduce en parallèle, afin d'éviter un goulet d'étranglement dans la chaîne de traitement. Le cas échéant, Hadoop s'assure de la bonne affectation des données intermédiaires aux différentes tâches de Reduce, afin de garantir la cohérence des agrégats finaux. Cette étape, réalisée conjointement par les mappers et les reducers, s'appelle le **shuffle & sort**. Pour les amateurs de plomberie algorithmique, l'exemple donné en encart explique cette étape plus en détail.

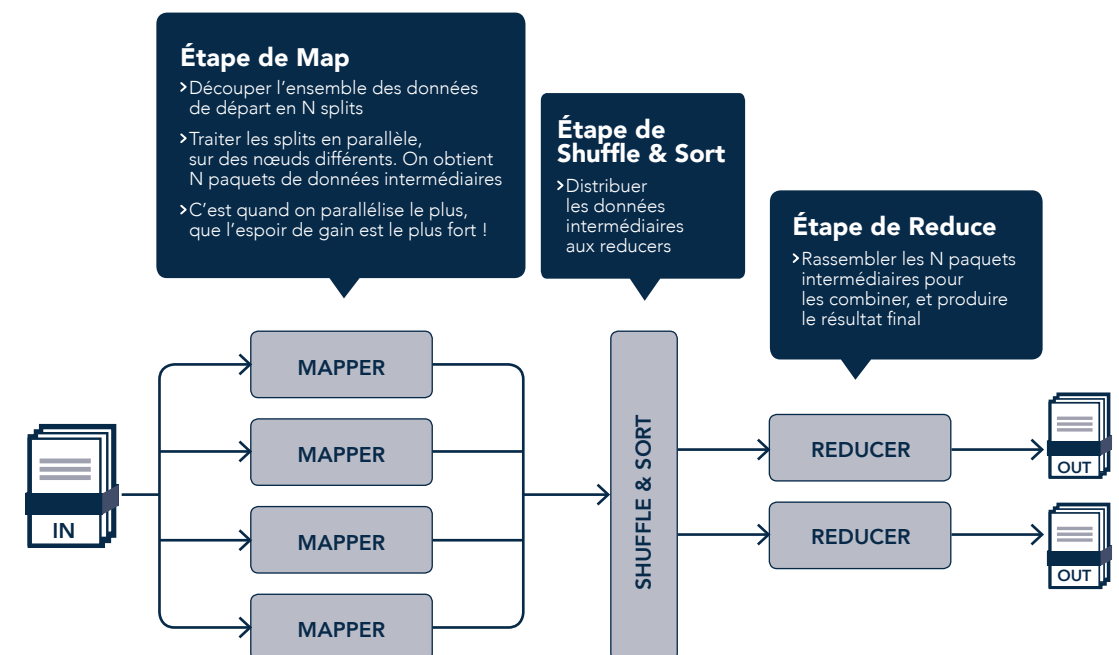
La colocalisation n'est pas possible pour la phase de Reduce car, par construction, les tâches de Reduce doivent récupérer les résultats intermédiaires provenant de toutes les tâches de Map, qui se sont exécutées sur des nœuds différents.

Par conséquent, l'étape de shuffle & sort est susceptible de mettre le réseau à rude épreuve, notamment si toutes les tâches de Reduce interrogent les données intermédiaires simultanément. Pour éviter d'introduire de

nouveaux goulets d'étranglement dans le traitement, il est important que les données intermédiaires ne soient pas trop volumineuses (lorsque c'est possible). Typiquement, dans un traitement comportant une agrégation, il peut être utile de faire des pré-agrégations par lots dès l'étape de Map, afin de faire transiter des agrégats plutôt que de multiples

enregistrements unitaires. Cela est possible par l'intermédiaire de *combiners*, une facilité offerte par le framework de développement. Un combiner est une sorte de mini-reducer qui s'exécute au sein d'un mapper.

## Principe de l'algorithme MapReduce



EXEMPLE

# Un traitement MapReduce

Pour notre exemple, supposons que nous gérons **un site de e-commerce à fort trafic**. Dans ce contexte, nous possédons énormément d’informations sur les visiteurs du site, qui génèrent des logs et dont les comportements de navigation donnent des informations intéressantes sur leurs pratiques de consommation et sur la pertinence du contenu du site.

Concentrons-nous sur les logs produits par les serveurs du site de vente en ligne dont voici un extrait ciblé sur un internaute en particulier :

Adresse IP de l'internaute

N° de session HTTP

ID client (quand il est connu)

Date et heure de la requête

Contenu de la requête HTTP

Interprétation en termes de navigation

81.129.34.188 - 1a89b - - 2013-10-03 08:53:17 - GET /accueil.php

81.129.34.188 - 1a89b - - 2013-10-03 08:53:26 - GET /catalogue.php

81.129.34.188 - 1a89b - - 2013-10-03 08:54:02 - GET /catalogue.php?categorie=57

81.129.34.188 - 1a89b - - 2013-10-03 08:54:31 - GET /catalogue.php?categorie=176

81.129.34.188 - 1a89b - - 2013-10-03 08:54:58 - GET /produit.php?produit=7364

81.129.34.188 - 1a89b - - 2013-10-03 08:55:09 - GET /ajout\_panier.php?produit=7364

81.129.34.188 - 1a89b - - 2013-10-03 08:56:34 - GET /commande.php

81.129.34.188 - 1a89b - - 2013-10-03 08:56:34 - GET /login.php?origine=commande

81.129.34.188 - 1a89b - - 2013-10-03 08:57:06 - POST /login.php?origine=commande

81.129.34.188 - 1a89b - 99183 - 2013-10-03 08:57:07 - POST /commande.php

81.129.34.188 - 1a89b - 99183 - 2013-10-03 09:00:41 - GET /accueil.php

81.129.34.188 - 1a89b - 99183 - 2013-10-03 09:00:41 - GET /ajout\_panier.php?produit=699

>Un visiteur (anonyme pour l'instant) arrive sur le site.

>Clic sur l'onglet « Catalogue ».

>Recherche dans les catégories de produits.

...

>Consultation d'un produit.

>Ajout au panier.

>Déclenchement de la commande.

>Le site exige d'être authentifié: il redirige vers la page de login.

>Le visiteur se signe, on connaît maintenant son ID client.

>La commande est lancée.

>Le client est redirigé vers l'accueil une fois la commande validée.

>Il clique sur un produit mis en avant en page d'accueil.

Afin d’**optimiser nos campagnes marketing**, nous cherchons à mieux connaître les clients qui abandonnent leurs commandes en cours de route. Pour cela, nous pouvons nous appuyer sur les logs.

Pour faire cette analyse, nous choisissons d’utiliser Hadoop et en particulier MapReduce car les logs sont trop volumineux. Nous supposons que ces logs ont été accumulés sur le cluster.

Il nous faut d’abord écrire le **programme d’analyse**. Pour cela MapReduce nous impose un cadre. Hadoop ne saura pas paralléliser « magiquement » n’importe quel programme que nous aurions écrit de manière séquentielle.

Nous pouvons résumer ce cadre en 3 points :

- Le programmeur fournit un programme *mapper*, qui sera exécuté par toutes les tâches de Map en parallèle.
- De même, il peut fournir un programme *reducer*, qui sera exécuté par les tâches de Reduce le cas échéant.
- Les programmes mapper et reducer prennent en entrée et émettent des enregistrements modélisés sous forme de paires (clef, valeur) : avec MapReduce, tout doit rentrer dans ce moule, que ce soit les données d’entrée, les données intermédiaires, et celles produites à la fin en sortie.

Pour être clair, nous distinguons dans le texte les programmes mapper et reducer (uniques et écrits par le programmeur) des tâches Map et Reduce (lancées en plusieurs instances par Hadoop).

## Premier étage : Map

Chaque tâche de Map reçoit une portion de fichier log à analyser, sous forme de paires (clef, valeur). La constitution des clefs et des valeurs est du ressort d’une spécification de format (un *input format*), qu’il faut préciser en paramètre au lancement du traitement. Hadoop vient avec des input formats courants, mais dans notre cas nous en créons un adapté à ce que nous voulons faire. En particulier, il découpe chaque ligne ainsi :

81.129.34.188 - 1a89b - 99183 - 2013-10-03 09:00:41 - GET /accueil.php

Clef d'enregistrement

Valeur

Valeur

## EXEMPLE > Un traitement MapReduce

Ici nous décidons de garder l'IP et le n° de session comme clef : notre modélisation introduit la notion de parcours sur le site. Quant à la valeur, elle est composée de l'ID client et de l'URL visitée. L'ID client ne doit pas faire partie de la clef car un parcours doit rester unique même quand le visiteur s'enregistre au milieu de sa navigation.

Nous n'avons pas besoin du timestamp (date et heure de la requête) car nos règles sont suffisamment simples pour s'en dispenser.

À elles toutes, l'ensemble de nos tâches de Map reçoivent donc les éléments suivants, parmi toutes les autres entrées de logs (la flèche symbolise la structure de donnée qui fait l'association entre une clef et sa valeur):

```
81.129.34.188 - 1a89b → NULL - /accueil.php
81.129.34.188 - 1a89b → NULL - /catalogue.php
81.129.34.188 - 1a89b → NULL - /catalogue.php?categorie=57
81.129.34.188 - 1a89b → NULL - /catalogue.php?categorie=176
81.129.34.188 - 1a89b → NULL - /produit.php?produit=7364
81.129.34.188 - 1a89b → NULL - /ajout_panier.php?produit=7364
81.129.34.188 - 1a89b → NULL - /commande.php
81.129.34.188 - 1a89b → NULL - /login.php?origine=commande
81.129.34.188 - 1a89b → NULL - /login.php?origine=commande
81.129.34.188 - 1a89b → 99183 - /commande.php
81.129.34.188 - 1a89b → 99183 - /accueil.php
81.129.34.188 - 1a89b → 99183 - /ajout_panier.php?produit=699
```

Nous avons mis en évidence un découpage possible en splits par Hadoop – nous ne contrôlons pas ce découpage dans les programmes car il dépend des données d'entrée.

Le mapper, qui prend ces données en entrée, doit émettre en retour des paires de clefs et de valeurs. Dans notre cas, la clef reçue en entrée suffit pour notre besoin, inutile de la transformer. Quant aux valeurs, on peut à ce stade interpréter les URL comme des événements bien identifiés. On peut pour cela s'appuyer sur un référentiel (le CMS du site), ou bien sur une liste codée « en dur » dans le programme.

Dans notre exemple simplifié, on en profite pour filtrer les URL qui n'apportent par d'information utile, comme les images ou les simples consultations du catalogue.

Voici le résultat en sortie des tâches de Map, pour notre sous-ensemble illustratif :

```
81.129.34.188 - 1a89b → NULL - AjoutPanier(produit=7364)
81.129.34.188 - 1a89b → NULL - Commande
81.129.34.188 - 1a89b → 99183 - Commande
81.129.34.188 - 1a89b → 99183 - AjoutPanier(produit=699)
```

Les instances de Map travaillant en parallèle et ne se parlant pas, il n'a pas été possible à ce stade de répercuter l'ID client (99183) sur l'ensemble du parcours : l'instance Map 2 ne le voit pas du tout passer.

Le rôle de l'étape de Reduce sera justement d'analyser les parcours dans leur globalité. Mais on n'y est pas encore tout à fait.

## ◉ Etage caché : le shuffle & sort

Tous les couples (clef, valeur) émis par les Maps sont redistribués par Hadoop vers les Reduces – d'où le terme *shuffle*. Cette collaboration entre tâches de Map et de Reduce se fait automatiquement, sans que nous ayons à la programmer.

Les couples sont redirigés de manière déterministe à partir des clefs: c'est essentiel pour nous, car c'est cela qui va nous permettre de voir ensemble (dans une même tâche de Reduce) tous les événements de chaque parcours, pour décider à la fin s'ils aboutissent ou non à une commande par un client identifié.

Pour cette étape de distribution, Hadoop a besoin de trier les paires selon les valeurs des clefs – d'où le deuxième terme *sort*. C'est une nécessité technique qui dans certains cas s'avère sympathique : les données d'arrivée seront automatiquement triées.

C'est le fonctionnement du shuffle & sort qui explique pourquoi nous devons tout modéliser sous forme de paires (clef, valeur). Ce qui peut sembler contraignant au premier abord est en fait nécessaire à la parallélisation des traitements.

- ◉ Dernier étage : Reduce

Le reducer prend les paires du shuffle & sort en entrée, sous la forme suivante: - clef émise par un mapper, liste de valeurs, la liste en deuxième position étant constituée automatiquement par Hadoop

## EXEMPLE > Un traitement MapReduce

par juxtaposition des valeurs correspondant à une même clef. Chaque clef ne sera donc traitée qu'une fois. Dans notre cas, cela donne, pour notre parcours type (en gras) et quelques autres :

```
131.91.239.12 - 6d193 →
    12440 - AjoutPanier (produit=133)
    12440 - AjoutPanier (produit=78)
81.129.34.188 - 1a89b →
    NULL - AjoutPanier (produit=7364)
    NULL - Commande
    99183 - Commande
    99183 - AjoutPanier (produit=699)
88.166.61.40 - f8ce9 →
    NULL - AjoutPanier (produit=817)
    NULL - Commande
201.12.78.160 - 189a3 →
    NULL - AjoutPanier (produit=51)
    NULL - Commande
```

Comme annoncé, chaque parcours reconstitué sera traité en une fois par une unique tâche de Reduce. Bien sûr, une tâche de Reduce peut traiter plusieurs parcours : tous ceux qui lui ont été attribués par le shuffle & sort.

Comme nous souhaitons connaître les clients qui abandonnent des commandes, nous appliquons la règle suivante : **SI** la liste d'événements contient au moins un AjoutPanier **MAIS** aucune Commande avec un ID client non nul, **ALORS** émettre en sortie l'ID client et la liste des produits « perdus » pour la commande, sous forme de couples (clef, valeur).

Ici, nous décidons d'ajouter en plus le prix total de la commande perdue, en faisant appel à un référentiel produit :

12440 → 133,78 - 458,00 €  
99183 → 699 - 329,00 €  
NULL → 817 - 54,90 €  
NULL → 51 - 1270,00 €

Nous devons spécifier un format de sortie (*output format*). Ici, nous choisissons de produire du CSV (*Comma-Separated Value*), exploitable par Excel ou R :

```
ID client ; produits ; manque à gagner
12440 ; 133-78 ; 458,00
99183 ; 699 ; 329,00
NULL ; 817 ; 54,90
NULL ; 51 ; 1270,00
```

Faut-il une ou plusieurs tâches de Reduce ? C'est à nous de choisir, en fonction de la manière dont le résultat final sera exploité. Avec un seul Reduce, nous obtenons un fichier résultat unique, trié sur les clefs grâce au shuffle & sort. À l'inverse, avec plusieurs Reduces nous obtenons plusieurs fragments, triés localement mais pas globalement et qu'il faudra combiner pour obtenir le résultat final, ce qui demande un peu plus d'efforts. En revanche, plusieurs Reduces peuvent travailler en parallèle, sans introduire de goulet d'étranglement dans le traitement : à nous de trouver le meilleur compromis. Ici, le tri global ne nous intéresse pas. Nous pouvons nous contenter de plusieurs fichiers que nous pourrions concaténer sans difficulté.

Cet exemple démontre qu'il est finalement simple de traiter des données semi-structurées (logs web) et de relier des événements séquencés dans le temps (détection de l'ID client en cours de parcours).

Grâce à l'architecture Hadoop, ce même programme peut tourner que vous ayez quelques gigaoctets ou plusieurs péta-octets de données.



🕒 L'architecture support de MapReduce

L'algorithme MapReduce s'appuie sur plusieurs fonctions de la plateforme Hadoop. Il a besoin :

- d'ordonnancer les traitements,
- de connaître l'emplacement sur le cluster des fichiers à traiter,
- de distribuer l'exécution sur les nœuds de calcul en fonction de leur disponibilité et des données qu'ils détiennent (colocalisation).

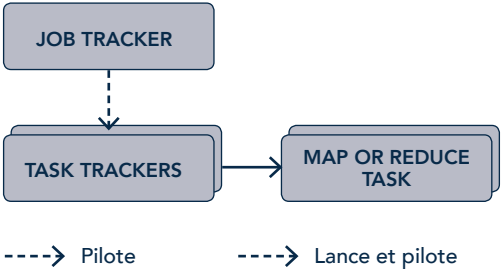
La fonction d'ordonnancement est prise en charge par un service d'Hadoop propre à MapReduce : le **job tracker**. C'est lui qui reçoit tous les jobs, c'est-à-dire le code binaire des traitements et leurs paramètres d'exécution.

Pour connaître l'emplacement des fichiers sur le cluster, le job tracker fait appel aux services du name node d'HDFS, qui lui fournit les emplacements des blocs composant les fichiers d'entrée.

Enfin, pour distribuer les calculs, le job tracker interroge un autre processus qui s'exécute sur chacun des nœuds : le **task tracker**. C'est lui qui assure le lancement et le suivi des tâches de Map ou de Reduce s'exécutant sur son nœud. Concrètement, chaque task tracker reçoit du job tracker le code de traitement et lance une JVM à laquelle il délègue l'exécution de la tâche Map ou Reduce.

Sur chaque nœud, le task tracker dispose d'un nombre limité de *slots*, qui déterminent combien de tâches de Map et de Reduce peuvent s'exécuter en même temps sur ce nœud.

Le ballet des démons de MapReduce



Et la fiabilité ?

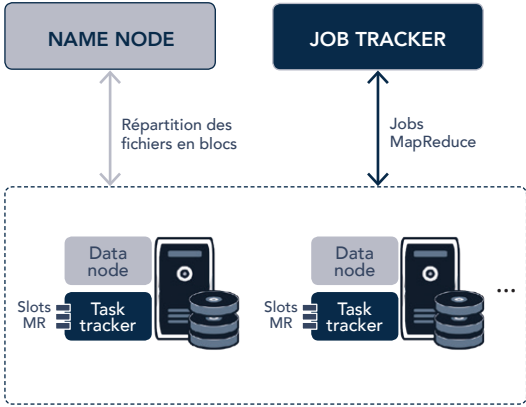
Il existe des mécanismes de surveillance (heartbeat) servant à relancer les portions de traitement en échec, voire à les rediriger vers un nouveau nœud si l'un d'entre eux venait à disparaître complètement du cluster. Ces opérations sont faites de manière transparente par la plateforme ; le développeur n'a pas à s'en préoccuper.

🕒 La programmation des jobs MapReduce

L'API native de MapReduce est écrite en Java. Ainsi, la première manière d'écrire un programme MapReduce est d'implémenter **3 classes du framework**. Ces classes définissent 3 fonctions : une fonction `map()`, une fonction `reduce()`, et une fonction d'initialisation `run()` qui va se charger de la configuration technique. Il y a donc peu de code à écrire. Cela nécessite cependant d'installer un environnement Java complet : IDE, dépendances Maven, compilation, déploiement. On réserve donc cette approche à des phases d'industrialisation.

En phase d'expérimentation, il est plus simple d'écrire les mappers et reducers en langages de script (Python, Ruby, Shell...). Ceci est possible grâce à un mécanisme appelé Hadoop Streaming. Cependant, certaines fonctions avancées (combiners, formats d'entrée/sortie non standard) ne sont pas disponibles dans ce mode, si besoin il faut alors revenir à l'API Java.

L'architecture combinée d'HDFS et de MapReduce



Rappelons qu'il s'agit d'une vue logique des services présents sur le cluster. Le name node et le job tracker peuvent s'exécuter en théorie sur n'importe quel nœud. Il n'est toutefois pas recommandé de les faire cohabiter avec d'autres services pour deux raisons :

- ils sont fortement sollicités et ils ont des exigences de mémoire qui risquent de pénaliser les services concurrents,
- ils ne sont pas soumis aux mêmes contraintes de disponibilité que les autres services, ce qui motive à les héberger sur des serveurs différents, avec des SLA distincts.

# Dépasser HDFS et MapReduce

## De HDFS à l'API de stockage

Sur Hadoop, l'accès aux données est avant tout une API Java, dont HDFS est l'implémentation de base. L'API expose les services que nous avons évoqués : stocker, lire et savoir où sont répartis les morceaux dans le cluster. Un fournisseur de solution de stockage peut, par conséquent, remplacer HDFS en exposant cette API en plus de ses API propres habituelles. Il en existe quelques-uns, mais il faut avoir des besoins très particuliers pour qu'HDFS ne sache pas y répondre !

## YARN et Tez : vers une plateforme de calcul distribué

L'algorithme MapReduce permet d'implémenter de nombreux types de calculs, pour peu qu'ils soient parallélisables. Premier problème cependant, quand on dispose déjà d'un algorithme, il peut être difficile de l'adapter pour le faire rentrer dans le moule MapReduce. Il faut le décomposer en deux phases Map et Reduce, et trouver les bonnes représentations (clef, valeur) pour écrire chaque phase. L'approche est parfois trop contraignante pour donner lieu à une implémentation intuitive. Faites le test :

prenez à un algorithme quelconque, et réfléchissez à la manière dont il pourrait être traduit en MapReduce. On prend les paris que cela vous prendra un certain temps !

MapReduce souffre d'un autre problème, cette fois-ci lié aux performances. Il ne prévoit en tout et pour tout que deux phases de calculs (Map et Reduce), alors que de nombreux algorithmes un peu élaborés se décomposent en plusieurs séquences itératives, chacune utilisant le résultat produit par la précédente. On voudrait parfois enchaîner des séries de Map et de Reduce, mais sans devoir soumettre à chaque fois un nouveau traitement au cluster. En effet, sur Hadoop, le démarrage d'un traitement a un coût : l'instanciation des  $N$  copies des mappers et reducers, et leur synchronisation pour enchaîner les phases. Ce coût est d'autant plus grand, en proportion, que les traitements sont petits.

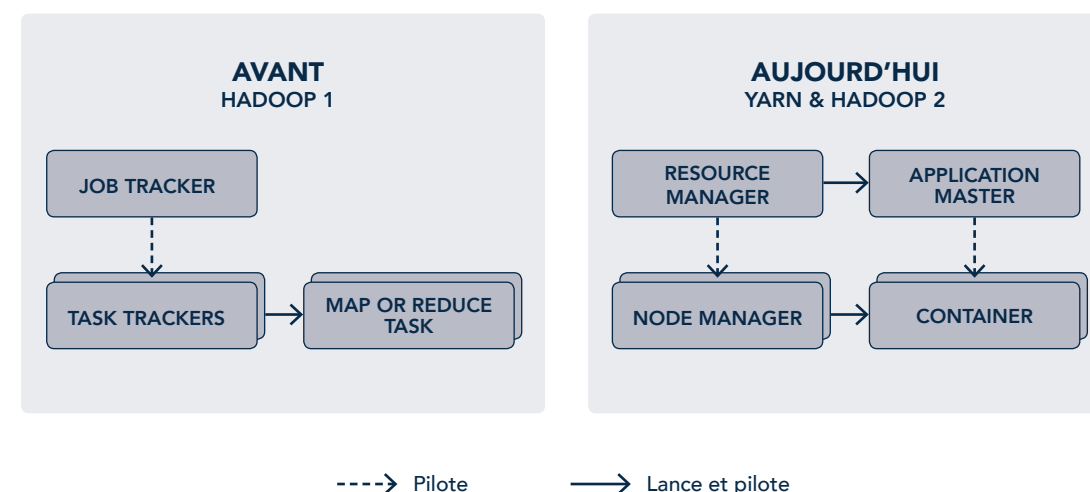
Prenons un exemple simple en apparence, mais que n'aimera pas du tout MapReduce : la recherche d'amis d'amis (d'amis...) dans un réseau social centré autour d'une personne. Cela correspond à l'exploration d'un voisinage dans un graphe. La liste des voisins de premier niveau se calcule avec un premier traitement

MapReduce ; elle sert elle-même à donner les points de départ des voisins de niveau 2, qui serviront pour l'exploration du niveau 3, etc. Le surcoût d'un traitement est payé à chaque couche de voisinage que l'on traverse.

Ces deux problèmes – la rigidité du modèle MapReduce et le surcoût d'enchaînement des jobs – ont conduit la communauté à introduire dans Hadoop 2 deux généralisations de MapReduce : **YARN** et **Tez**.

YARN signifie **Yet Another Resource Negotiator**. C'est un framework qui permet d'exécuter n'importe quel type d'application distribuée sur un cluster Hadoop, application composée d'un processus maître (*application master*) et de processus esclaves (*containers*) s'exécutant sur plusieurs machines. Avec Hadoop 2, MapReduce a été réécrit pour s'exécuter sur YARN ; il peut ainsi être vu comme un cas particulier d'application YARN.

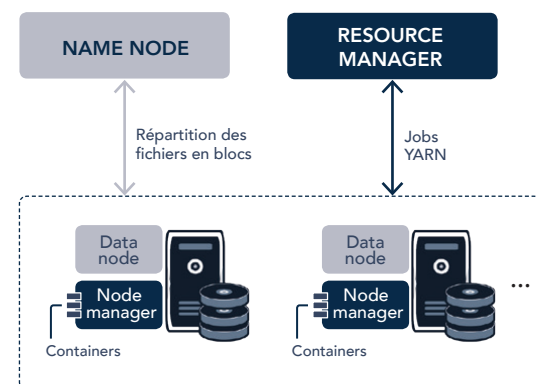
## MapReduce avant et après YARN



Le processus orchestrateur de YARN s'appelle le **resource manager** ; il pilote le cluster par l'intermédiaire de *node managers* qui s'exécutent sur chaque nœud de calcul. Ainsi Le resource manager :

- Lance l'application master.
- Pilote les node managers, qui à leur tour lancent les containers.

*L'architecture combinée  
d'HDFS et de YARN.  
Cela ne vous rappelle rien ?*



YARN est donc bien le rejeton de MapReduce une fois que l'on en a sorti tout ce qui sert à lancer un calcul distribué, indépendamment de l'algorithme. Pour généraliser la notion de calcul distribué, YARN voit le cluster Hadoop comme un gros agrégat de mémoire et de processeurs. Là où MapReduce « canal

historique » g rait des slots de t ches abstraits, qu'il fallait configurer   la main, YARN pilote les traitements en fonction de la disponibilit  des ressources physiques d'une grosse machine virtuelle.

Puisque MapReduce existe toujours, les programmes écrits en Hadoop 1 pourront s'exécuter au prix de quelques modifications de paramétrage mineures.

Tez, quant à lui, est une évolution de MapReduce visant à donner plus de flexibilité au programmeur dans la définition des traitements. **Tez** propose un cadre pour écrire des traitements plus élaborés sous forme de **DAG**, c'est-à-dire de « **graphes acycliques orientés** ». Concrètement, cela signifie des enchaînements d'étapes arbitraires qui se passent des données de l'une à l'autre, sans boucle.

Pour les applications qui en tirent parti, Tez apporte une amélioration de performance importante. Le fait de voir les jobs comme des DAG laisse à la plateforme des leviers pour optimiser globalement leur exécution. Avec MapReduce, c'est au programmeur de le faire, avec la contrainte d'enchaîner des couples Map → Reduce. La diminution des accès disque et l'élimination des tâches inutiles (« passe-plats ») sont encore d'autres leviers de performance.

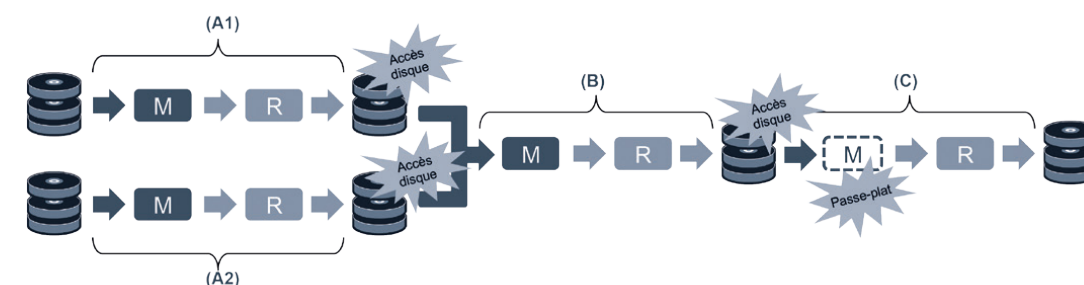
L'exemple du voisinage d'amis, problématique pour MapReduce, serait un bon candidat pour un unique job Tez.

Pour les architectes techniques, il est important de comprendre ces évolutions d'Hadoop,

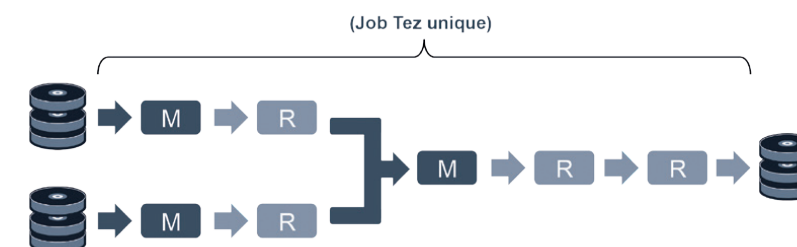
notamment pour définir l'architecture d'une application et optimiser la performance du cluster. Néanmoins, il est assez rare d'avoir à implémenter directement un algorithme avec YARN ou Tez. Nous verrons dans le chapitre

suivant qu'Hadoop propose une panoplie d'outils qui masquent le formalisme de ces frameworks au développeur.

## L'apport du modèle DAG



Exemple de traitement composé souffrant de la rigidité de MapReduce.  
Les jobs (A1) et (A2) préparent 2 sous-ensembles de données, qui sont ensuite joints par le job (B).  
Le job (C) fait une simple agrégation finale, avec un mapper inutile (en pointillés) vers le reducer.



Le même traitement avec Tez, dans le cas favorable où des accès disques intermédiaires ne sont pas nécessaires.  
Le mapper « passe-plats » de l'ex-job (C) a été éliminé, réduisant encore la latence d'exécution.



# L'éco- système

*Hadoop est également riche d'un écosystème en perpétuelle évolution et, disons-le clairement, un peu touffu. Visite guidée dans la galaxie des acteurs et des outils que tout bon architecte ou développeur se doit de connaître.*

# Open source, éditeurs et gouvernance

Aujourd’hui, Hadoop est un des projets phares portés par la fondation Apache. Il faut voir Hadoop comme une plateforme flexible, et par là même capable de mettre en œuvre la plupart des idées qui gravitent autour du Big Data.

Son universalité attire de nombreux acteurs qui contribuent activement à l’évolution et à l’amélioration d’Hadoop, et parmi eux les Géants du Web : ils ont besoin d’Hadoop pour leur cœur de métier, et les contributions en retour à la communauté font partie de leur culture.

À côté de ces acteurs-utilisateurs, il y a également des acteurs-distributeurs. Ils ont choisi de packager la distribution canonique d’Apache à leur manière, en y ajoutant souvent des outils d’administration ou d’intégration au SI. Ils contribuent également au projet communautaire, en reversant une bonne partie de ces outils au monde open source. À ce jour, les 4 grands acteurs-distributeurs du paysage sont **Hortonworks**, **Cloudera**, **Pivotal**<sup>①</sup> et **MapR**. De grandes entreprises de l’IT, comme IBM ou Oracle, ont aussi « leur » distribution Hadoop. Elles sont, pour certaines, basées sur

d’autres distributions – c’est le cas d’Oracle qui embarque Cloudera dans son appliance Big Data. Même si elles sont moins visibles sur le marché, elles permettent à ces acteurs d’offrir des stacks complètes à leurs clients, intégrant des architectures Big Data.

Cette multiplicité ne facilite pas le choix d’une distribution pour démarrer un cluster ; nous reviendrons sur cet enjeu dans le chapitre suivant.

*Hadoop est perçu comme le système d’exploitation Big Data open source de référence.*

*Si l’on met de côté les extensions, propriétaires ou non, que chaque distributeur propose, la gouvernance du cœur du projet reste chez Apache. C’est bien la communauté dans son ensemble qui décide des outils à inclure ou non dans la distribution canonique, ainsi que de l’évolution des API que certaines propositions d’extensions peuvent nécessiter.*

<sup>①</sup> Pivotal vient tout juste de céder son offre logicielle Big Data à la communauté open source. Hortonworks reprend le support commercial de la distribution Hadoop. Voir <http://blog.pivotal.io/big-data-pivotal/news-2/pivotal-and-hortonworks-join-forces-on-hadoop> pour les détails.

# Hadoop dans les nuages : les acteurs du cloud

Exécuter un traitement Hadoop sur le cloud peut être une option pertinente pour des raisons économiques. Lorsque l’on a besoin ponctuellement de beaucoup de puissance, il est plus intéressant d’utiliser des ressources de calcul à la demande, plutôt que d’investir dans un cluster complet. Techniquement, l’architecture d’Hadoop lui permet de fonctionner parfaitement sur des clusters de machines hétérogènes. Or sur le cloud, on ne sait jamais vraiment ce qu’on récupère à un instant t. C’est sans doute pourquoi un certain nombre d’acteurs se sont positionnés comme fournisseurs de services Hadoop sur le cloud, donc en **Platform as a Service (PaaS)**.

Ainsi, Amazon Web Services, le cloud d’**Amazon**, met à disposition un service Elastic Map Reduce (EMR) entièrement basé sur Hadoop, offrant le choix entre les distributions Apache et MapR.

Même stratégie chez **Google**, qui offre aussi, avec Google Compute des clusters Hadoop basés sur les distributions Apache et MapR.

**Microsoft** s’est également positionné sur le créneau, avec l’offre HDInsight de sa plateforme Azure, montée en partenariat avec Hortonworks. Le « petit plus » de cette offre est la fourniture de connecteurs ODBC et Excel, qui permettent d’interfacer le cluster avec des applications du SI ou utilisant SQL Server for Azure dans le cloud.

Que des grands d’Outre-Atlantique ? Eh bien non, nous avons aussi « nos » acteurs du cloud. À notre connaissance, à cette date seuls **OVH** et **Orange Business Services** sont positionnés sur le créneau en France.

Outre ces acteurs, qui proposent Hadoop comme un service à part entière, il est bien sûr possible d’utiliser des services **Infrastructure as a Service (IaaS)** sur cloud privé ou public. Le déploiement, la gestion et le paramétrage des clusters sont alors à la charge de l’utilisateur ; ce ne sont pas des tâches triviales !

Dans certaines offres Hadoop en PaaS (notamment Amazon et Google), les données à traiter sont hébergées sur un service de stockage permanent, alors que les clusters de calcul sont éphémères. Cela compromet le principe de colocalisation des traitements et des données : avec ces offres, attention donc à la performance du réseau qui relie le stockage aux machines de calcul. Le réseau ne doit pas être un goulet.

Pour une discussion plus technique sur les enjeux d’Hadoop sur le cloud, nous vous renvoyons à notre article de blog : <http://blog.octo.com/hadoop-in-da-cloud/>.



# L'écosystème logiciel

Hadoop est lui-même un projet chapeau regroupant une certaine quantité de projets logiciels. Nous vous proposons un tour de ces principaux projets, sachant qu'au moment où vous lirez ces lignes il est fort probable qu'ils aient évolué entretemps, tant le dynamisme des initiatives qui gravitent autour d'Hadoop est fort.

On peut distinguer 3 niveaux dans l'écosystème logiciel Hadoop :

- le **cœur** (HDFS, YARN, MapReduce et Tez), que nous avons décrit largement dans la partie précédente,
- la « **constellation** » Hadoop, composée de projets open source gouvernés pour la plupart par la fondation Apache et dont le cycle d'évolution est fortement lié à celui du cœur,
- la « **galaxie** » des initiatives tierces, en général portées par des éditeurs, soit qui enrichissent les fonctionnalités d'Hadoop, soit qui proposent des interfaces pour ouvrir Hadoop au reste du SI.

La tendance actuelle est à l'inflation du nombre d'initiatives. Les éditeurs clefs jouent des coudes pour faire valoir à la fois leur contribution à la constellation, et prendre des positions avec le tampon de la fondation Apache. D'un côté, cela contribue à l'enrichissement

fonctionnel rapide de la plateforme, de l'autre on peut parfois se retrouver avec des initiatives concurrentes, y compris au sein d'Apache, ce qui ne simplifie pas les choix des utilisateurs...

## Stockage de données

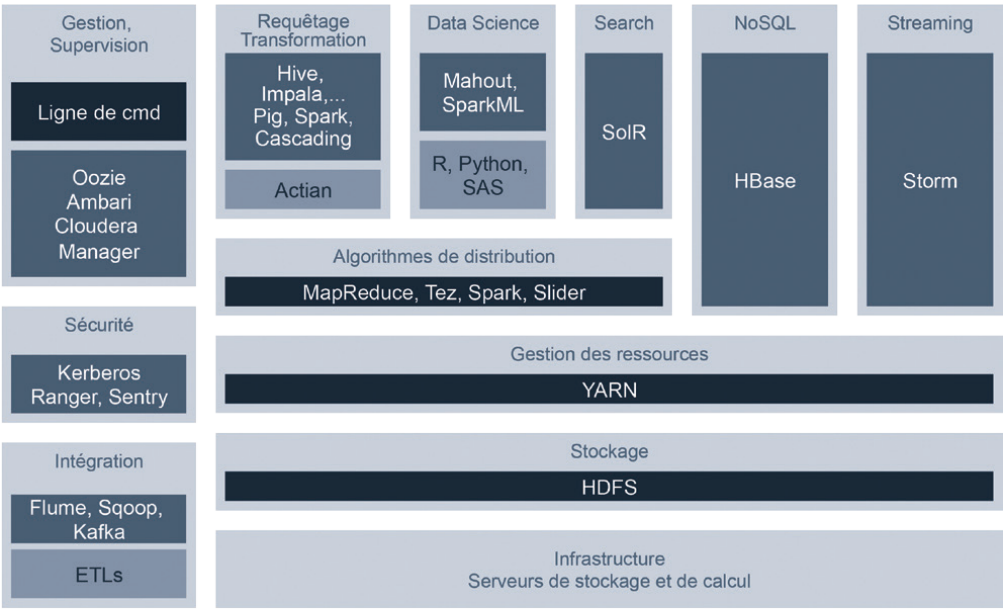
> Des systèmes de fichiers distribués alternatifs à HDFS

Nous avons évoqué plus haut la possibilité de substituer des systèmes de fichiers alternatifs à HDFS. Il existe plusieurs implémentations de l'API HDFS dont voici les exemples les plus visibles.

MapR, dans sa distribution propriétaire, embarque son système de fichiers **MapRFS**. Par rapport à HDFS, MapRFS est à la fois plus complet et plus souple, avec comme fonctionnalités principales le support des écritures aléatoires en milieu de fichier, et la réplication multi-sites en temps réel, utile pour assurer la continuité de service lorsqu'un datacenter est perdu. Ces fonctionnalités avancées vont arriver sur HDFS dans les années à venir, diminuant d'autant l'avance de MapR sur ce créneau.

D'autres systèmes de fichiers proposent une émulation HDFS. Citons **Isilon OneFS** (EMC),

## Quelques outils de l'écosystème Hadoop



Cœur de la plateforme   Constellation Hadoop   Galaxie outils tiers

un système de fichiers distribués servant de support au matériel de stockage NAS Isilon. Bien qu'il n'y ait plus de colocalisation données/traitements, EMC annonce des performances supérieures<sup>8</sup>, mais il est difficile d'affirmer que cela est vrai pour tout type de traitement...

Plus surprenant, la base NoSQL **Cassandra** (dans son édition DataStax Enterprise) implémente l'API d'HDFS au-dessus de son stockage

objet de base. Cette stratégie permet à l'éditeur DataStax d'intégrer la suite logicielle d'Hadoop dans son propre écosystème, par le biais de MapReduce. Il faut le voir comme une opportunité d'ajouter des capacités analytiques au-dessus d'une application développée avec Cassandra transactionnelle (un site web par exemple, ou un agrégateur de logs/capteurs).

<sup>8</sup> <http://bigdatablog.emc.com/2013/07/25/not-sure-which-hadoop-distro-to-use-emc-isilon-provides-big-data-portability/>



## ◉ Intégration avec le reste du SI

Un des gros enjeux d'une intégration d'Hadoop dans le SI, est d'injecter la donnée source dans le cluster, et d'en extraire la donnée produite. Pour cela, on ne manque pas d'outils ! Nous ne présentons ici que les principaux.

Les plus simples sont les **commandes en ligne** d'Hadoop (`hdfs dfs`) : elles reprennent les opérations classiques de manipulation de fichiers (copie, suppression, renommage, etc.), et ajoutent deux opérations, `get` et `put`, qui copient des données entre le système de fichiers local et HDFS. En termes de fonctionnalités, on est donc au même niveau qu'un client FTP.

En ce qui concerne l'implémentation, HBase s'appuie sur HDFS pour stocker ses données brutes. Il bénéficie donc des avantages de ce dernier : réplication, distribution sur plusieurs nœuds de stockage, et scalabilité.

**Apache Storm** et **Spark Streaming** seront appropriés pour des traitements événementiels plus complexes, avec des calculs portant sur des flux de données. Storm est issu du monde du *Complex Event Processing*. Twitter, à l'origine de ce projet, l'a très tôt présenté comme le « MapReduce temps réel ». L'architecture de Storm s'appuie sur des agents<sup>10</sup> qui s'exécutent de manière distribuée en s'échangeant des messages. Au départ indépendant d'Hadoop, Storm a été récemment modifié par Yahoo! pour s'exécuter sur YARN et pouvoir déposer des données sur Hadoop. Pour les scénarios de type « Internet des objets », **Apache Nifi**, récemment acquis et mis en open source par Hortonworks, semble intéressant – mais il est trop tôt pour avoir un avis définitif dessus.

**Apache Sqoop** va se charger d'échanger avec Hadoop des données de bases relationnelles : soit pour importer des données sources que l'on souhaite traiter, soit pour envoyer vers un datawarehouse « classique » le résultat de tels traitements.

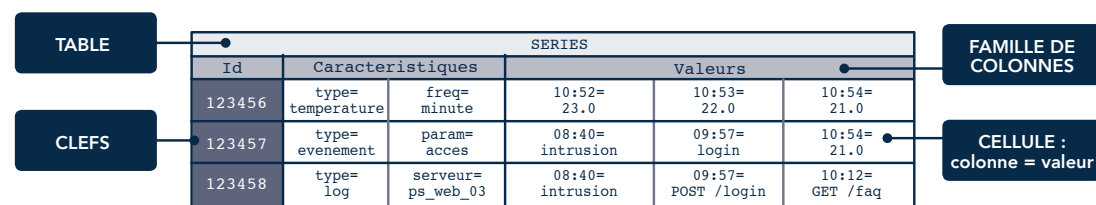
**Les ETL et ELT** offriront en plus des fonctions de transformation de données. Ce sont des outils bien connus des aficionados de la BI, et la plupart proposent maintenant des connecteurs Hadoop (par exemple Talend, Informatica, etc.). On bénéficie ainsi de la richesse fonctionnelle des ETL : connecteurs, transformation, ordonnancement, monitoring. Dans un contexte Big Data, et pour des raisons de performances, on préférera souvent l'approche ELT décrite plus bas<sup>9</sup>.

**Apache Flume** est l'outil Hadoop historique pour les données de type logs (logs applicatifs, serveurs web, infrastructure, monitoring, etc.).

On peut résumer les besoins d'intégration, et les solutions associées par le petit tableau suivant :

X	BATCH	FIL DE L'EAU
TRANSPORT SIMPLE	ligne de commandes sqoop	Flume, Kafka
TRANSFORMATIONS	ETL / ELT	Storm, Spark Streaming, Nifi

## Le modèle column family



Les enregistrements sont regroupés en tables, et repérés par des clefs primaires. Chaque enregistrement comprend un nombre arbitraire de colonnes regroupées en familles (column families). Chaque colonne de chaque enregistrement stocke finalement un paquet d'octets (équivalent de BLOBs). Grâce à la flexibilité du modèle en familles de colonnes, on peut stocker dans une même table des données hétérogènes, ici par exemple des données de capteurs et des logs.

⑨ ETL (Extract, Transform, Load), ou sa variante ELT (Extract, Load, Transform) est une approche visant à transporter de la donnée d'un système à un autre en lui appliquant des transformations. Par extension, on désigne ainsi les logiciels qui outillent cette approche.

<sup>(10)</sup> Pour une discussion très générale sur ce concept, voir [http://fr.wikipedia.org/wiki/Système\\_multi-agents](http://fr.wikipedia.org/wiki/Système_multi-agents).

N’oublions pas le séquençement des traitements : réagir à l’arrivée d’un nouveau fichier, le déposer dans le cluster, le transformer, appliquer des algorithmes, exporter les résultats... Toutes ces étapes répétitives doivent être enchaînées au moyen d’un ordonnanceur. Pour cela, le standard sur Hadoop est **Oozie**. Il ne remplacera pas un ordonnanceur de SI, du type Control-M ou \$U, mais piloté par ceux-ci il prendra en main l’exécution de workflows applicatifs.

Requêtage et transformation de données

La plateforme Hadoop présente un vaste choix d’outils qui répliquent les patterns d’accès habituels des bases de données.

Un premier besoin est la **transformation de la donnée** en mode batch. On est dans une approche de type ELT, dans laquelle les transformations s’exécutent sur Hadoop, pour profiter de sa scalabilité. Concrètement, de telles transformations font appel à toute la panoplie de l’algèbre relationnelle ensembliste : sélection, filtrage, jointure, regroupement, etc. Par rapport au monde des bases de données relationnelles, deux différences sont à noter. D’une part, le langage d’écriture de ces transformations n’est pas limité à SQL. D’autre part, la plupart des outils proposés interprètent la structure des données à la volée, ce qui ouvre la possibilité

de manipuler des structures de données plus riches que les simples lignes et colonnes des bases de données relationnelles. C’est l’approche *schema on read* que nous avons déjà évoquée.

Un deuxième besoin est le requêtage (interroger les données pour obtenir un résultat). Au départ, le requêtage sur Hadoop consistait en des batches « courts », de plusieurs minutes à une heure. Au gré des améliorations des outils, ces temps de requêtage se rapprochent de ce qui est acceptable pour une utilisation interactive : c’est-à-dire de quelques secondes à quelques minutes. Ce mode d’accès, qui était initialement

Spark, ou la pièce rapportée d’Hadoop

Spark n’est pas qu’un langage de transformation de données. C’est au départ une plateforme d’exécution distribuée, un peu comme Hadoop, et créée indépendamment de ce dernier. Depuis peu dans le giron d’Apache, il est en train de fusionner avec Hadoop pour devenir un moteur d’exécution alternatif à MapReduce, et son écosystème suit le même chemin. Son API écrite en Scala tire parti des techniques de programmation fonctionnelle, concises et particulièrement adaptées au calcul distribué<sup>11</sup>. Cela explique l’intérêt considérable que la communauté Big Data et les éditeurs Hadoop portent à Spark. Sans aucun doute, une partie du futur d’Hadoop se tient là. Leur descendance sera nombreuse et conquerra le monde !

<sup>11</sup> La programmation fonctionnelle stricte fait opérer des fonctions sur des données, sans recours à des variables globales. Variables globales et parallélisme ne font pas bon ménage car la synchronisation exige des acrobaties compliquées qui sont source de bugs et nuisent aux performances.

réservé à des cas d’usage de reporting, s’ouvre ainsi de plus en plus à de la consultation interactive de données telle qu’offerte par les bases de données relationnelles. La compatibilité maximale avec le langage SQL (et donc avec les outils de reporting comme Business Objects ou Cognos) reste un enjeu majeur, qui a été longtemps sous-estimé, et pour lequel des réponses se développent.

Voyons les outils les plus connus répondant à ce double besoin de transformation et de requêtage.

**Pig, Cascading** (ou sa variante **Scalding** pour Scala), et plus récemment **Spark, Crunch** ou **Flink**, proposent des API spécifiques centrées autour de la notion de *dataset* (jeu de données) : les instructions consomment et produisent des datasets, dans un enchaînement d’opérations.

**Hive, Impala, Drill, Kylin, Hawq, Presto** et d’autres, quant à eux, ont l’ambition d’exécuter en temps interactif des requêtes en langage SQL (syntaxe et expressivité), en y ajoutant la possibilité de traiter des données semi-structurées. Sans entrer dans les détails, précisons que les architectures de ces outils sont très différentes. La compatibilité avec les normes SQL est encore imparfaite, mais l’écart se réduit car les éditeurs ont compris le besoin crucial d’interopérabilité avec les outils de BI. Bien que tous open source, ces outils sont en général portés par des éditeurs en compétition sur le créneau du requêtage SQL interactif.

Enfin, un nombre toujours croissant d’ETL ajoutent à leur palette de connecteurs la capacité à s’interfacer avec Hadoop. On bénéficie alors

des designers graphiques de ces solutions pour définir des jobs de transformation de données.

En conclusion, tous les outils que nous avons mentionnés ont en commun de simplifier le développement, en ajoutant des fonctions de plus haut niveau que MapReduce. Ils prennent également en charge une partie des optimisations qu’un développeur devrait autrement faire sur du code technique MapReduce. Ils sont donc un accélérateur important pour développer des flux et des transformations.

Les échelles de temps sur Hadoop, pour fixer les idées



Calcul scientifique, data science et machine learning

On ne peut pas parler de Big Data sans parler de data science et de machine learning. Une plateforme Big Data se doit donc de supporter des calculs statistiques massifs. C’est le cas d’Hadoop, où cette fonction est assurée par plusieurs outils provenant de son écosystème. En ce moment, ces outils tournent beaucoup autour de Spark.

Nous l’avons déjà dit, Hadoop est bon comme ETL et pour faire tourner certains types d’algorithmes sur de gros volumes. En revanche, il n’est pas (encore) adapté au travail d’un data scientist qui, en phase d’exploration, a besoin de tester rapidement et de façon interactive des hypothèses variées. Dans ce type d’activité, des outils comme **R** ou **Python** sont beaucoup plus souples. Tout d’abord parce qu’ils travaillent en mémoire et sont donc rapides, ensuite parce qu’ils bénéficient de nombreuses bibliothèques de machine learning aptes à construire des modèles prototypes. Travaillant en mémoire, ils ne savent manipuler qu’une taille limitée de données. On peut donc être amené à extraire des échantillons pour mettre au point un modèle.

Lorsque l’on a défini un modèle pertinent, Hadoop revient dans la boucle, pour appliquer le modèle sur l’ensemble des données. On distingue donc bien la construction du modèle de son application en taille réelle. Cependant, si vous pouvez vous offrir un gros serveur pour tout calculer en mémoire, ne vous privez pas !

Data science et machine learning

Pour certains, data science et machine learning sont des synonymes, pour d’autres des marqueurs sociaux comme le bonnet du hipster et le chapeau de l’académicien. Quant à nous, nous pensons que ce sont des activités complémentaires.

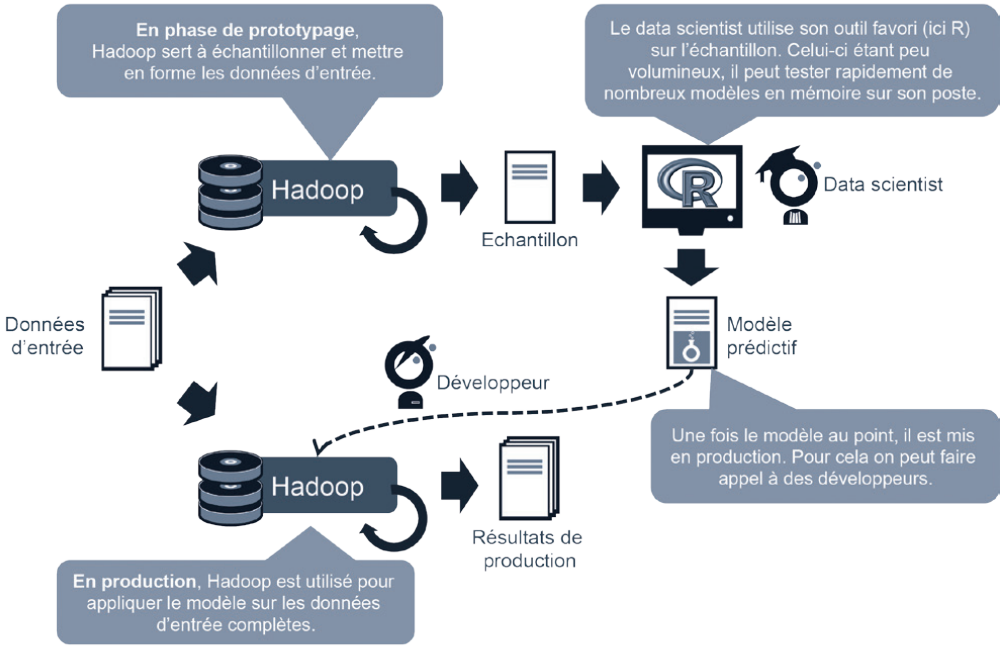
Le data scientist est l’expert technique et métier de la donnée. Il travaille avec des outils d’exploration, de data mining, de prototypage. Il cherche à découvrir, visualiser, expliquer et modéliser les corrélations entre les données.

L’expert en machine learning cherche à construire des systèmes pouvant apprendre à partir des données qui leur sont présentées, plutôt que reposant sur des règles pré-établies par un humain. Il utilise pour cela des algorithmes capables de construire des modèles statistiques.

Ces personnes sont amenées à collaborer pour répondre aux problèmes business qui leur sont présentés. Pour les deux profils, Big Data apporte son lot de défis : le volume des données et leur hétérogénéité (format, qualité).



Hadoop, un outil au service du data scientist



La transposition d’un modèle vers Hadoop n’est pas une tâche simple. On ne peut pas prendre un programme R ou Python et le déposer dans Hadoop en espérant qu’il se distribue tout seul. Il va falloir l’adapter voire changer l’algorithme<sup>12</sup>. Dans beaucoup de cas, heureusement, l’écosystème arrive à la rescousse. En voici les principaux membres.

**Mahout** est l’outil de machine learning historique d’Hadoop. Il est inclus dans toutes

les distributions et propose un catalogue d’algorithmes sous forme de framework Java. Java n’est pourtant pas un langage particulièrement adapté au machine learning : il n’offre pas le niveau d’abstraction qui convient à l’écriture simple de formules mathématiques. La prise en main de Mahout n’est donc pas aisée.

De fait, la lourdeur de Mahout a toujours été un frein à son adoption, laissant aujourd’hui

<sup>12</sup> Rien d’impossible mais voici un papier qui donne une idée de l’effort à fournir pour l’adaptation sur MapReduce : <http://www.cs.stanford.edu/people/ang/papers/nips06-mapreducemulticore.pdf>.

le champ libre à des frameworks plus simples comme **SparkML** (ou Spark MLlib), issu de l'écosystème de Spark.

**Le langage R** est aussi capable de s'interfacer avec un cluster Hadoop, via des packages. Ceux-ci sont d'assez bas niveau cependant, et les scripts qui tournent sur vos laptops ne se laisseront pas porter sans modifications. Les packages les plus connus sont :

- *RHive*, qui permet de manipuler de la donnée – charger des tables Hive en mémoire, lancer des requêtes SQL ou effectuer depuis R quelques opérations simples comme des agrégations.
- *RHadoop*, qui donne accès aux stockages HDFS et HBase depuis R, et permet de développer des fonctions `map()` et `reduce()` composant un traitement MapReduce.

Une édition commerciale et «Hadoop-aware» de R est aussi proposée par la société Revolution Analytics, avec quelques algorithmes de machine learning à la clef. Nous n'avons malheureusement pas eu l'occasion de la voir à l'œuvre.

**Python** offre aux programmeurs un large choix, qu'il s'agisse d'accéder à un cluster Hadoop, d'écrire du MapReduce ou de piloter des clusters sur des fournisseurs de cloud. Ce langage est d'ailleurs nativement présent sur la plupart des serveurs, ce qui en facilite l'utilisation. Il se distingue par la qualité de ses librairies scientifiques, dont les plus connues sont **Scikit-learn** (machine learning, leadé par l'INRIA), **Pandas** et **SciPy** (calcul matriciel), ou

encore **NLTK** (traitement du langage). Ces librairies sont regroupées sous la distribution **Anaconda** de Python.

**SAS**, éditeur de logiciels scientifiques, développe actuellement des moteurs d'exécution Hadoop. Ils ne sont pas encore finalisés à ce jour. Notons aussi que de plus en plus de data scientists se tournent vers R aux dépens de SAS.

### L'avenir de Mahout

*Nous craignons que Mahout ne soit en route vers le cimetière des éléphants. Les contributeurs de ce projet ont décidé que les nouveaux algorithmes du catalogue Mahout seraient développés avec Spark au lieu de MapReduce.*

*D'un point vue technique c'est une bonne idée, mais dès lors que SparkML arrive en fanfare, avec l'appui de la communauté, que faire d'un outil de plus, qui ne survira peut-être même pas à un chantier de refonte ?*

*Aujourd'hui, nous préconisons sans hésiter SparkML plutôt que Mahout.*



### Spark, Python ou R, quel sera le langage des data scientists demain ?

*R, le plus riche, est très utilisé dans le monde universitaire, avec une communauté qui contribue énormément au corpus de packages, pour nombre de disciplines scientifiques.*

*Python a pour lui le fait d'être un langage de programmation complet. Et c'est bien connu, les data scientists sont aussi des geeks. Il est aussi plus facile d'intégrer Python avec Hadoop en phase d'industrialisation.*

*Spark, prend du galon chez les geeks ; son langage de base, Scala, est peut-être un peu ardu pour les data scientists qui n'ont pas un passé de développeurs. Mais il s'intègre très bien avec le gestionnaire de ressources d'Hadoop, YARN.*

*Comme souvent, il n'y a pas de vainqueur absolu. Il s'agit d'utiliser la technologie appropriée au problème donné. Nous utilisons beaucoup R et Python en data science exploratoire ; Spark, par son intégration forte avec Hadoop, est plus intéressant dès lors que l'on veut mettre des traitements en production. Et nous gardons un œil sur Flink, qui se veut une alternative à Spark optimisée aux petits oignons...*

### Le «search»

Le search est un usage encore balbutiant sur Hadoop. Il s'agit ici d'offrir une interface de type moteur de recherche, permettant de retrouver instantanément des documents stockés sur un cluster, parmi plusieurs téra- ou péta-octets de données plus ou moins structurées. Par exemple, un entrepôt de données pourra indexer toutes les données relatives à des clients, quelle que soit la source : CRM, messageries des conseillers, archives documentaires, etc. La fouille de métadonnées (qui a accédé à quel contenu, quand, etc.) est un autre cas d'usage intéressant pour le search.

Les éditeurs de distribution Hadoop ont tous adopté la même stratégie : intégrer un moteur de recherche existant, **SolR**, à leur offre logicielle Hadoop. Cette intégration prend la forme de connecteurs, capables de lire ou d'écrire des fichiers d'index SolR dans les traitements.

On pourrait s'attendre à une intégration avec ElasticSearch, plus en vogue que SolR, mais faute d'accord entre les éditeurs respectifs, cela ne s'est pas passé ainsi. Des connecteurs existent toutefois, mais sans intégration avec YARN à ce jour : les données indexées avec Hadoop doivent être copiées vers le cluster ElasticSearch.



o L’exploitation

> L’administration

L’administration recouvre un large panel de tâches : installation des binaires, entretien de la topologie du cluster (via des nœuds et configuration des points d’entrée aux services et aux données), allocation des ressources de calcul ou de stockage, tuning fin des applications, mises à jour de versions, etc.

Les éditeurs Hadoop fournissent de tels outils d’administration, qui sont au cœur de la partie payante de certaines distributions (ex. **Cloudera Manager**), ou disponibles en open source (tel **Ambari**, intégré à l’initiative Open Data Platform). Mais ces outils ne suivent pas en temps réel les dernières évolutions d’Hadoop et la pléthore d’options avancées qui en résulte : il faut quand même connaître le détail des fichiers de configuration. Pour des clusters isolés, non critiques et pas trop gros (<10 nœuds), pas de problème, les outils font le gros du travail et les opérations supplémentaires ne pèsent pas lourd. À partir d’un certain nombre de nœuds, ou lorsqu’il faut répliquer une configuration sur plusieurs environnements synchronisés, cela devient ingérable.

Dans ces cas-là, point de salut sans automatisation des déploiements et des reconfigurations ! L’enjeu est important car les fichiers de configuration sont répartis sur tout le cluster. Il faut veiller à leur cohérence globale, pouvoir documenter les changements et savoir revenir en arrière en cas d’erreur de manipulation. On rentre alors dans une logique DevOps.

Deux stratégies d’automatisation sont possibles :

- **Par les API d’administration Hadoop** : outre une interface graphique, les outils d’administration ont le bon goût d’exposer des API qui peuvent être pilotées par script.
- **Par le système** : les logiciels Hadoop sont également disponibles sous forme de paquets RPM pouvant être installés avec des commandes système. Cela se fait au prix d’un plus gros effort à fournir pour construire soi-même la configuration à partir de zéro. Certaines organisations prennent cette voie pour rester indépendantes des outils d’administration, à leur goût trop fortement liés aux éditeurs de distribution. Ces organisations maîtrisent déjà les outils de déploiement du marché, comme Chef, Puppet ou Ansible, et s’orientent naturellement vers ceux-ci.

DevOps ?

DevOps est un mouvement qui promeut l’application des méthodes agiles au monde de la production. Entre autres, il prône un enrichissement mutuel des pratiques de développement et de production. L’exposition d’API d’administration, le scripting de déploiements avec Puppet, sont des moyens d’automatiser des tâches de production, de la même manière qu’un développeur automatise la construction des logiciels avec des outils de build.

Vous trouverez de nombreux articles traitant du sujet sur notre blog : <http://blog.octo.com/tag/devops>.

> La supervision

La supervision sur Hadoop est un sujet complexe. Il faut surveiller les processus serveur, s’assurer que les nœuds sont en bonne santé et pas trop saturés, suivre l’exécution des traitements, diagnostiquer les erreurs en consultant les logs épars sur le cluster, etc.

À cet égard les distributions d’Hadoop ne sont pas égales. Certaines proposent des interfaces unifiées où données, traitements, logs et métriques sont rassemblés dans un moteur de recherche, d’autres quelques tableaux de bord de santé du cluster et des pointeurs vers les IHM de bas niveau d’Hadoop. Que ces interfaces soient unifiées ou non, les développeurs en auront besoin pour la mise au point de leurs traitements.

Les exploitants seront plutôt intéressés par des API à connecter aux outils de supervision de la DSI. Là encore, même si certaines distributions proposent des API unifiées de façade, leur manque d’exhaustivité oblige à compléter par des appels directs aux API de bas niveau d’Hadoop : il faut faire son marché dans les API, et la popote derrière. Au niveau le plus bas, on trouve des outils de supervision d’infrastructure connus par ailleurs dans le monde de la supervision (Ganglia), ou développés expressément pour Hadoop (Ambari Metrics).

On pourrait croire qu’à côté de ces initiatives les gros éditeurs de supervision se seraient lancés sur le créneau. En fait, si quelques tentatives existent (de la part d’HP ou de BMC

par exemple), elles couvrent plutôt l’intégration d’outils de supervision classiques avec Hadoop. On est encore loin de la fourniture de tableaux de bord spécialisés à destination des exploitants experts d’Hadoop. La tendance est plutôt de se servir du cluster Hadoop, en particulier de HBase, comme base de données de métriques de supervision – l’éléphant se mord la queue !

> La sécurité

La sécurité est un vaste sujet... Dans nos missions Big Data, où nous avons l’outrecuidance de recommander une ouverture des données de l’entreprise à tous ses acteurs (voire à l’extérieur), la sécurité tient une place importante – en général, pas celle du mort. Il ne faut pas se voiler la face, c’est un sujet de préoccupation majeure des organisations qui entament une transformation digitale.

Hadoop n’a commencé à s’y intéresser que tardivement, vers 2012 comme nous l’avons vu dans l’introduction. C’est l’époque des premiers « data lakes ». Autant le dire tout de suite donc, le sujet de la sécurité n’a pas encore atteint sa pleine maturité. La sécurisation globale d’un cluster demande un effort de configuration important. Quant à l’outillage, il commence seulement à arriver. Voyons cela.

**L’authentification** (« t’es qui toi ? ») est très bien couverte, Hadoop déléguant cette tâche à un annuaire respectant le protocole Kerberos<sup>13</sup>. Un exemple bien connu de tel annuaire est Active Directory, de Microsoft.

<sup>13</sup> Kerberos est un protocole d’authentification forte : en d’autres termes, il garantit à des utilisateurs accédants, à des comptes de service et à des machines, tous en interaction, que chaque protagoniste est bien celui qu’il prétend être.

**L’habilitation** (« qu’est-ce que tu fais là ? ») à base de groupes d’utilisateurs est bien couverte aussi, même s’il n’y a pas encore de mécanisme universel pour les fichiers, bases de données, services, etc. Des éditeurs sont en concurrence pour imposer « leur » framework de sécurité ; pour l’instant, match nul<sup>14</sup>.

**La traçabilité et l’audit** (« qui a fait ça ? Y a-t-il des trous dans la raquette ? ») sont maintenant bien établis. La stratégie retenue à chaque fois est d’intercepter les accès aux services et aux données, pour produire des logs de sécurité centralisés. Cette capacité technique ne doit pas masquer le gros enjeu, encore naissant au moment de la rédaction de cet ouvrage, celui de la gouvernance des accès, avec l’administration de politiques de sécurité complexes. Techniquement, l’interception est assurée par les frameworks d’habilitation évoqués ci-dessus, ou bien par des services assurant le rôle de « guichets d’entrée » par lesquels passent tous les accès (c’est le cas de la gateway Knox d’Hortonworks).

**Le chiffrement** (« qu’est-ce que tu dis ? ») est maintenant possible sur les flux réseaux internes et aux frontières du cluster, et sur disque (chaque utilisateur ou entité chiffre ses fichiers HDFS avec ses propres clefs ; on parle de *security at rest*). Les éditeurs ont procédé à des rachats stratégiques dans le domaine, pour se positionner sur un créneau où l’intégration d’Hadoop dans l’infrastructure de sécurité du SI est un sujet complexe.

Où placer les comptes et les groupes de sécurité Hadoop ?

*Dans un annuaire bien sûr ! Mais lequel ? Votre entreprise dispose probablement d’un annuaire où se trouve le compte qui vous sert à ouvrir votre session Windows le matin. Active Directory, l’annuaire de Windows, respectant le protocole Kerberos, Hadoop saurait lui déléguer l’authentification de ses utilisateurs.*

*Pour autant, il ne faut pas « polluer » l’annuaire d’entreprise avec des comptes techniques ou des groupes de rôles Hadoop qui ne relèvent pas de la gestion du personnel. Comment faire ?*

*La meilleure pratique, dans ces conditions, est d’adjoindre au cluster Hadoop un petit annuaire d’appoint. Il contiendra les comptes techniques, les groupes de sécurité et plus généralement tout ce qui est nécessaire au fonctionnement propre de la plateforme.*

*Les groupes de sécurité, dans l’annuaire d’appoint, pourront mélanger des comptes et des sous-groupes des deux annuaires : comptes techniques et humains, groupes de rôles et groupes d’organisation.*



<sup>14</sup> Ainsi Cloudera promeut Apache Sentry et Navigator, tandis qu’Hortonworks pousse Apache Ranger, ex-Argus, et le très embryonnaire Apache Atlas. Sentinelle des mers ou géant garde forestier, choisis ton camp !

> Survivre à un désastre

Les Plans de Reprise et de Continuité d’Activité (PRA et PCA) classiques prévoient de sauvegarder et de répliquer des données sensibles. Avec un cluster de plusieurs péta-octets, ce n’est pas instantané (ni très économique...). Les bandes magnétiques, ou les répliquions inter-sites, sont tout de suite compliquées. Il faut donc faire des choix !

**Pour les métadonnées**, les recommandations sont simples : les configurations du cluster et des outils, les schémas d’interprétation des données, les catalogues, doivent être sauvegardés et, si nécessaire, répliqués sur un cluster de secours. Ce sont des métadonnées de faible volume et qui changent assez rarement pour que les techniques habituelles de sauvegarde soient appliquées. Il en va de même pour les utilisateurs et groupes de sécurité.

**Pour les données** elles-mêmes, il faut définir des stratégies. Peut-être les données brutes, volumineuses, sont-elles déjà conservées par les producteurs de données, et peut-être est-il acceptable de perdre quelques jours de production le temps de les récupérer ? Peut-être les résultats de calcul intermédiaires sont-ils inutiles à la production et donc, peuvent être sacrifiés – ou peut-être veut-on en conserver certains pour ne pas avoir à relancer des calculs trop longs ? Quant aux résultats finaux des traitements, s’ils sont fortement agrégés on pourra facilement les sauvegarder ou les dupliquer. Les données propres aux utilisateurs et stockées dans leurs bacs à sable sont une

autre catégorie pour laquelle il faut se poser la question. Bref, il faut le prendre avec bon sens et estimer les coûts pour faire les bons compromis.

Idéalement, il faudrait mesurer l’usage réel des données pour choisir les plus pertinentes ; c’est une utilisation possible des outils d’audit vus plus haut. Enfin un débouché utile de l’obsession pour la sécurité : offrir une qualité de service au plus juste, et éviter que le cluster ne devienne un immense dépotoir de données hétéroclites.

Certaines organisations font le choix de mettre un petit cluster de secours sur le cloud, auprès d’un fournisseur d’infrastructure à la demande. Par exemple, les données peuvent être répliquées telles quelles sur Amazon S3 au fil de l’eau, et un cluster Elastic Map Reduce démarré uniquement lorsque nécessaire. Les indisponibilités étant rares, cela peut s’avérer plus économique que le maintien d’un cluster de secours. Mais il faut que les applications et traitements ne soient pas trop dépendants de la version d’Hadoop sur laquelle ils s’exécutent car on ne maîtrise pas ce paramètre chez le fournisseur de cloud.



# Hadoop dans le SI

*Si vous êtes convaincu de l'intérêt  
d'Hadoop dans votre organisation,  
il n'y a qu'à se lancer ! Mais comment,  
et par quoi commencer ?*

# Quelle organisation pour opérer des clusters Hadoop ?

Hadoop est à la fois une technologie et une plateforme multi-usages, et suscite lui-même de nouveaux usages – ceux de Big Data. Un changement se produira à tous les niveaux :

• Les **études informatiques** et les **équipes décisionnelles** (BI) devront se former à la panoplie d'outils que nous avons vus dans la partie précédente.

- Les **architectes** devront maîtriser cette plateforme complexe. Cette compétence leur permettra de choisir les outils adéquats dans un écosystème mouvant, d'optimiser les traitements et d'aider au débogage. Ils joueront aussi un rôle crucial dans les projets d'intégration de la plateforme avec le reste du SI.

- La **production** devra être prête à construire des clusters d'un type nouveau, à superviser de nouveaux logiciels.

- Les **métiers**, les **data miners** et les **data**

**scientists** bénéficieront eux aussi de nouveaux outils à disposition et susciteront des use cases innovants. Ils croiseront des données qui n'ont pas l'habitude de l'être, parce qu'elles résident dans des silos aujourd'hui cloisonnés ; pour cela ils devront faire bouger les frontières d'organisation internes à l'entreprise.

Dès lors, on se pose la question de l'organisation à mettre en place pour que tout ce petit monde collabore dans ce contexte compliqué et mouvant. C'est que la bête ne se laisse pas facilement mener par la trompe...

Il va déjà falloir faire face à une maturité inégale des composants. Certains sont instables, certains à peine émergents, notamment dans les couches hautes de l'architecture où sont les attentes des utilisateurs finaux. Le tout évolue allègrement à un rythme soutenu. Un travail de veille permanente est donc indispensable, il ne faut surtout pas négliger son ampleur ! Les équipes auront besoin de temps pour cela.

Allons plus loin encore : un cluster Hadoop est un service de traitement de données partagé entre plusieurs utilisateurs, ce que l'on appelle multi-tenant. Vous allez vite vous rendre compte qu'il est un composant critique de vos processus métier. Or c'est une plateforme en évolution, que vous êtes en train d'approprier.

**Comment résoudre cette équation ?** Sûrement pas avec une organisation classique silotée en services. Là encore, ce sont les modèles d'organisation agiles qui donneront la direction.

Par exemple, un embryon d'équipe transverse pourra être constituée d'un architecte, d'un développeur expert Hadoop, d'un data scientist et d'un exploitant. Ils collaboreront pour maintenir un cluster et aider les premiers projets désireux de goûter aux charmes de Big Data.

Hadoop va vite se trouver sous les feux des projecteurs de l'entreprise. Quoi de plus normal ! Des usages innovants à la pelle, des promesses d'économies fabuleuses, et surtout un bon gros effet de mode – les projets vont se bousculer au portillon. Évitez le piège de l'ouverture d'un cluster universel à tous les vents : il est impossible de trouver du premier coup l'organisation parfaite qui soit capable d'absorber autant de projets et la charge d'exploitation qui va avec. Il vaut mieux y aller doucement en choisissant bien ses premiers combats, puis apprendre au fur et à mesure.

## Hadoop et l'agilité à l'échelle de l'entreprise

*L'agilité à grande échelle cherche à adapter l'organisation pour faciliter l'emploi de méthodes agiles. C'est une pratique récente mais qui a déjà quelques incarnations, par exemple dans le framework SAFe (Scaled Agile Framework (sic)).*

*Une feature team est une équipe pluridisciplinaire (métier, développeurs, production, architectes, etc.) travaillant sur un même produit final – application ou service.*

**Une component team** maintient un composant utilisé par plusieurs feature teams – par exemple (au hasard) un cluster Hadoop multi-tenant.

*L'intérêt des organisations agiles, c'est que tout ce petit monde partage de la connaissance par le biais de **communautés de pratiques** (par exemple, la communauté des utilisateurs de Hive, ou celle des experts des données RH), bien placées pour faire avancer leurs sujets de prédilection. Tout l'inverse d'une cellule d'architecture perchée dans sa tour d'ivoire, en somme.*

*Avec Hadoop, les obstacles sont petits mais quotidiens, il faut une grande autonomie pour les surmonter et avancer rapidement. Une organisation agile permet cela, en favorisant l'échange et la discussion. Ce sont des idées conceptuellement simples mais difficiles à mettre en marche quand la culture d'entreprise résiste !*



# Le processus d'adoption

## Premières étapes

Hadoop est à l'œuvre depuis près de 10 ans chez les Géants du Web et les entreprises américaines. En Europe et notamment en France, on observe un décalage. À cette date (fin 2015), de nombreuses organisations ont mené des expérimentations, parfois poussées. Les start-ups et certaines entreprises à la pointe (dont de grandes organisations) ont déjà Hadoop en production, parfois sur des processus critiques.

On peut faire tant de choses avec cette plateforme si complexe, qu'il est difficile de savoir par où commencer. L'astuce est de l'introduire par étapes, ce qui permet de se familiariser avec la technologie, et de laisser la DSI adapter ses processus à un nouveau type d'environnement. C'est une route jalonnée d'étapes, de victoires même. Bien sûr, il y a quelques embûches mais notre expérience nous montre que la solution mais aussi les entreprises sont entrées dans une phase de

*L'approche métier bouscule les processus IT, pour assurer la mise à disposition rapide des données, des outils et des infrastructures.*

maturité suffisante pour voir apparaître les premiers cas innovants. **L'avenir réserve encore de belles surprises !**

Il y a deux approches possibles. L'une et l'autre sont valables, et nous accompagnons des clients sur les deux.

Le premier chemin passe par un POC (Proofs of Concept) d'innovation métier. Une nouvelle idée à tester, un nouvel emploi des données du SI, le rapprochement avec des données externes susceptibles d'apporter un nouvel éclairage à votre activité ? Profitez-en, qui dit

nouveau besoin dit nouvelle solution, et c'est peut-être l'occasion de faire monter Hadoop sur scène. Du fait de la double nouveauté métier et technique, la mise en œuvre de ce POC nécessitera une collaboration étroite entre les équipes métier et IT.

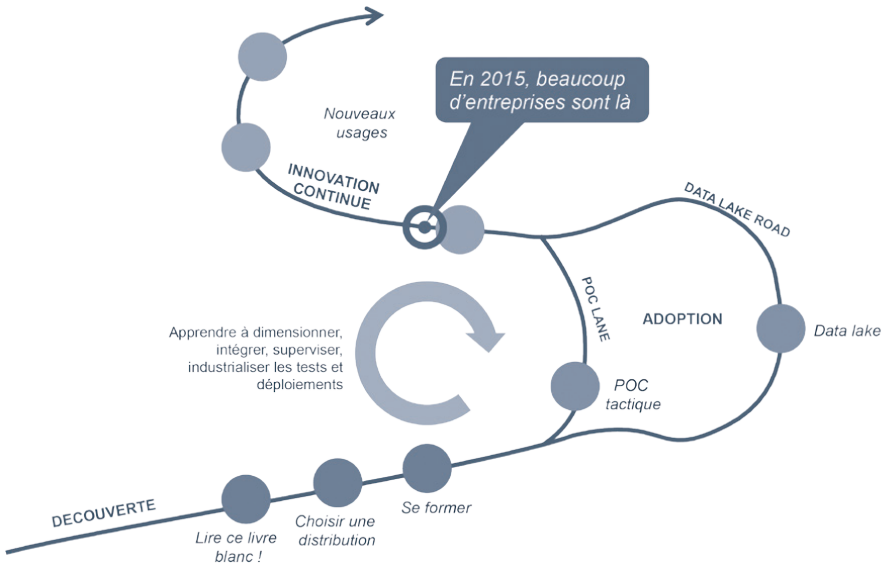
Contrairement au premier, le deuxième chemin peut et doit être guidé par l'IT. Il s'agit ici d'offrir à l'entreprise un entrepôt de données riche, outillé pour susciter des POC, ce que

l'on appelle souvent un *data lake*. Pour éveiller l'intérêt, le *data lake* doit satisfaire quelques critères qui le différencient du premier datawarehouse venu. Citons-en quelques-uns :

- Mettre ensemble des données que l'on n'a pas l'habitude de rapprocher : des données provenant de plusieurs métiers, des données externes, des données « oubliées » comme les logs ou les emails, etc.
- Offrir un vaste choix de connecteurs ou d'outils, pour permettre l'exploration, l'analyse, la modélisation, la visualisation, etc. en intéressant des acteurs différents.

- Garder une profondeur d'historique inédite en évitant les compromis sur le stockage.
- Démontrer que la plateforme apporte autant, à moindre coût, que les outils habituels du SI – ou plus pour le même coût. Dans beaucoup d'organisations, créer un entrepôt de données relationnel demande de longs mois, en particulier à cause de l'étape centrale de modélisation. Avec Hadoop, vous pouvez mettre très vite des données brutes à disposition de vos utilisateurs, et reporter à plus tard l'étape de modélisation en la lissant dans le temps.

## L'adoption d'Hadoop, un chemin jalonné d'embûches mais aussi de succès !



### Le data lake, ou l’art de la métaphore

Dans un entrepôt terrestre, un *warehouse* donc, c’est bien rangé. Les boîtes sont organisées en rayons, et la plupart du temps faciles à empiler. Évidemment, tout ne rentre pas forcément dans un parallélépipède, on perd parfois de l’espace ou on doit répartir les pièces détachées dans des boîtes séparées. Mais c’est simple, bien étiqueté, et tout a été bien planifié. Le contenu des boîtes, ici, c’est la donnée structurée du *datawarehouse*.

Dans un lac, en revanche, le liquide déversé occupe tout le volume, sans en gaspiller : on peut en mettre plus. On peut y verser des seaux d’eau, ou des glaçons bien cubiques qui finiront par fondre. C’est sûr, une fois

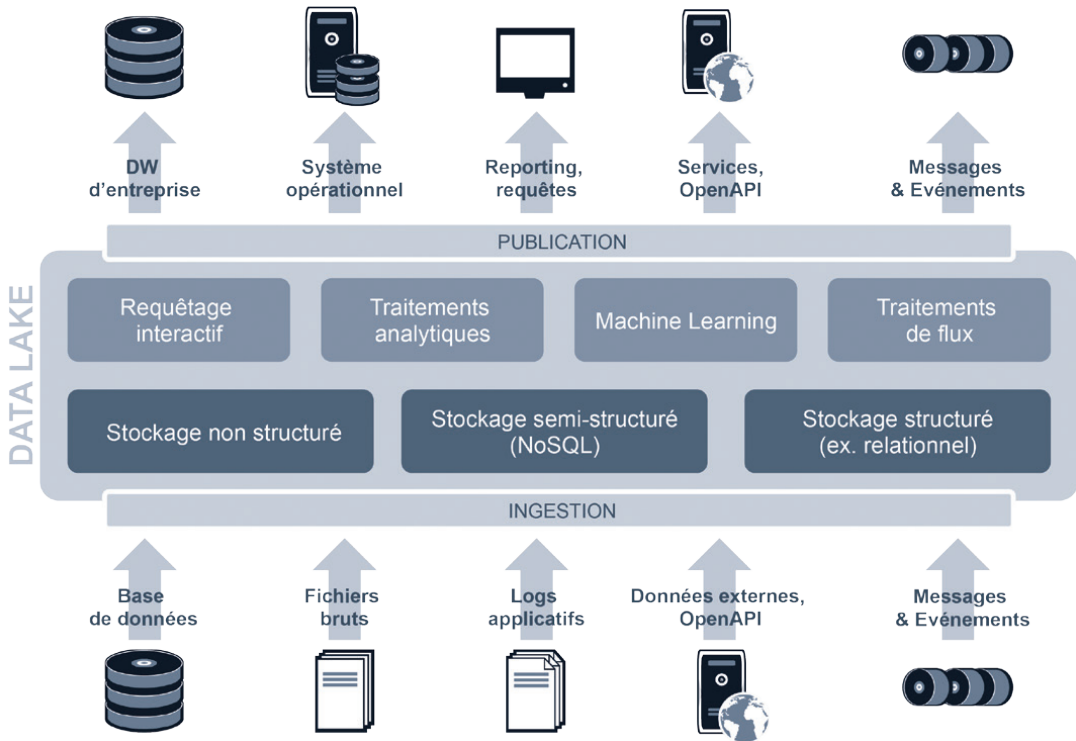
mélangé, c’est noyé dans la masse. Mais ce n’est pas très grave, car l’eau n’a d’intérêt que comme contribution au volume global, qu’on peut toujours pomper au besoin. Dans cette analogie approximative, l’eau c’est la donnée « Big data », structurée ou non. Quelle que soit l’approche adoptée, technique ou métier, les projets construits sur le cluster Hadoop seront autant de ruisseaux contributeurs au volume du grand lac.

### Les échelles hydrographiques

L’analogie venait au départ du marketing Big Data, puis elle est restée. Les sceptiques ironisent en parlant de *data pond* (mare de données), les mégalos s’emportent avec des images de *data ocean*, les pessimistes parlent de *data swamp* (marécage) pour souligner l’enlissement de l’utilisateur dans une masse de données déversées sans ordre. Finalement, le lac est un juste milieu ! Du reste, on peut compartimenter un lac pour gérer un minimum la circulation des eaux. Bizarrement, rien n’a été proposé comme analogues des bestioles plus ou moins sympathiques qui grouillent au fond des lacs...

### L’architecture d’un data lake

Les données versées ne sont pas destinées à croupir dans le lac : transformées ou non, elles ont ensuite vocation à être utilisées.



# Choisir sa distribution

La multiplicité des distributions et des acteurs Hadoop est un fait. Le choix d’une distribution est évidemment compliqué, d’autant qu’on n’en change pas facilement après coup. Et c’est aussi le choix d’un éditeur, dont vous allez plus ou moins dépendre.

Il est illusoire d’énumérer et d’analyser toutes les distributions du marché, elles sont trop nombreuses. Cependant, après quelques années de maturation d’Hadoop, des éditeurs et distributions incontournables se détachent, souvent généralistes. Nous en citons par ailleurs d’autres qui, par leurs particularités, se destinent à des usages spécifiques.

Pour départager les généralistes, nous voyons trois critères à ce jour :

- La **culture** plus ou moins **open source** de chaque éditeur.
- Les **partenariats technologiques** ou **d’intégrateurs** de chacun, qui faciliteront l’accueil de sa solution dans votre SI.

- Les **outils spécifiques** apportés par chaque éditeur à sa distribution, à mettre en regard de vos besoins. Il faut savoir que ces outils ne sont pas facilement portables d’une distribution à l’autre et ne seront pas supportés par les autres éditeurs<sup>15</sup>.

*Choisir une distribution Hadoop, c’est comme choisir une distribution Linux : le critère principal est rarement technique.*

Par conséquent, si vous n’êtes pas encore affilié à une chapelle, nous vous recommandons de déterminer la solution qui vous convient le mieux grâce à un POC.

À notre connaissance, aujourd’hui seul Hortonworks propose un support en français, soit directement soit par l’intermédiaire de ses partenaires (OCTO étant l’un d’eux).

Le paysage aura sûrement évolué d’ici un an ou deux, à l’occasion des partenariats voire de rachats possibles des généralistes. Des rumeurs circulent régulièrement, pour l’instant elles ne se sont pas vérifiées...

<sup>15</sup> Cette situation pourrait évoluer prochainement, avec le lancement tout récent de Open Data Platform (<http://opendataplatform.org/>), un effort de standardisation des couches basses d’Hadoop. Plusieurs gros éditeurs adhèrent à l’initiative, à l’exception notable de Cloudera. L’avenir nous dira si cela suffira à rendre les outils de haut niveau portables d’une distribution à l’autre.

## Les grands éditeurs Hadoop avec quelques critères de choix

DISTRIBUTION	POUR QUELS USAGES ?	REMARQUES
GÉNÉRALISTES		
Apache open source	• POC, développement	• Choix fin des versions de chaque outil mais risques d’instabilités et de régressions lors des mises à jour. • Moins de choix pour un outil d’administration, facteur important de choix d’une distribution. • Pour le développement, il vaut mieux leur préférer une VM « bac à sable » d’un éditeur généraliste comme Cloudera ou Hortonworks.
Cloudera Distribution including Apache Hadoop (CDH)	• Projet généraliste • Intégration avec les produits des partenaires SAS, Oracle ou IBM.	• Gratuit et en grande partie open source (les exceptions concernent les produits « entreprise » : administration, PRA, etc.). • L’éditeur propose du support et des services en option. • Projets phares : Impala, Sentry, Cloudera Manager (propriétaire).
Hortonworks Data Platform (HDP)	• Projet généraliste • Intégration avec les produits des partenaires SAS, Teradata.	• Gratuit et open source. • L’éditeur propose du support et des services en option (support de niveau 1 en français possible). • Projets phares : Stinger, Ranger, Knox, Ambari.
GoPivotal	• Projet généraliste • Intégration avec le catalogue d’EMC (Greenplum) et VMWare (GemFire, SQLFire).	• Au départ issu du rapprochement d’EMC et de VMWare, GoPivotal a été cédé à Hortonworks. • Projets phares : Hawq
IBM Infosphere BigInsights	• Projet vertical « full IBM »	• Distribution payante et supportée par l’éditeur, basée sur la souche Apache. • Elle embarque des outils propriétaires IBM de data mining.
SPÉCIALISÉES		
MapR	• Projet avec stockage multi-sites. • Accès mixtes batch/aléatoire de type NFS.	• Distribution payante et supportée par l’éditeur. • MapRFS est une réimplémentation complète d’HDFS. • Amazon Web Services propose MapR dans son offre Elastic MapReduce, comme alternative à la distribution Apache. • Projets phares : Drill.
APPLIANCES & CLOUD		
Microsoft Azure HD Insight	• Projet sur le cloud pour une entreprise « Microsoft ». • Traitement de données stockées sur le cloud Azure.	• Cette distribution a été implémentée sur Azure en partenariat avec Hortonworks. Une compatibilité complète avec la souche open source est annoncée (à éprouver sur cas réel).
Teradata	• Projet nécessitant une architecture hybride (Hadoop, Teradata) avec des transferts à haut débit. • Intégration avec les autres outils de la suite Teradata (Aster, Intelligent Query).	• La distribution Hadoop embarquée est celle d’Hortonworks (HDP).
Oracle Big Data Appliance	• Projet nécessitant une architecture hybride (Hadoop, Oracle Database ou Exadata) avec des transferts à haut débit.	• La distribution Hadoop embarquée est celle de Cloudera (CDH).



# L'infrastructure physique

## Des serveurs pour exécuter des services

Nous parlons beaucoup de **services** (name nodes, data nodes, resource manager, etc.), de **serveurs** et de **nœuds**, en utilisant les trois termes à tour de rôle. Ils ne sont pas toujours utilisés à bon escient dans la littérature Hadoop, mais nous avons choisi de respecter les usages, quitte à entretenir une certaine confusion. Explicitons-les.

- **Un serveur** est une machine (physique ou virtuelle) faisant partie du cluster. Facile.
- **Un service** est un processus qui tourne sur un ou plusieurs serveurs, et participant à l'architecture logicielle d'Hadoop. Pas d'ambiguïté.
- **Un nœud** est un serveur. Easy, sauf que... on parle de data nodes et de name nodes depuis le début. Ces noms sont trompeurs car ce sont bien des services ; ils ne sont jamais tous seuls sur un nœud. Dans la suite, un master node, slave node ou edge node est bien un serveur. Sur le web et dans la littérature, vous trouverez tout de même les confusions name node / master node et data node / slave node. Ce n'est pas gênant car le contexte empêche toute méprise.

Tudidu, nous vous prions de nous excuser pour la gêne occasionnée (en même temps, on n'y peut rien !).

Cette mise au point étant faite, on classe les serveurs en 3 groupes :

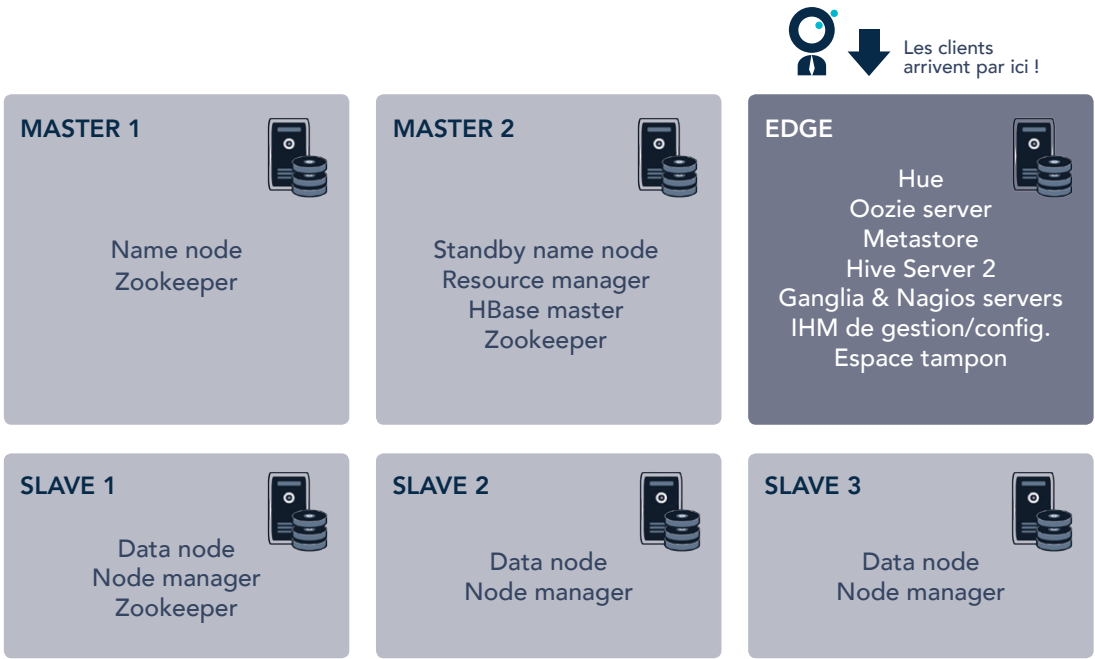
- Les *master nodes* exécutent les services centraux d'Hadoop, ceux qui se retrouvent en un exemplaire, voire deux quand ils sont redondés : name node et son backup, resource manager, HBase Master, etc.
- Les *slave nodes* exécutent les services nécessaires au stockage et au calcul : data node, node manager, régions HBase et containers éphémères créés par les traitements.
- Les *edge nodes* enfin portent tous les services périphériques qui servent à interagir avec le cluster, sans en constituer le cœur. On y retrouve les serveurs Oozie, Hive, les outils d'administration... ainsi que des espaces de stockage tampon (fichiers et SGBD) et des canaux d'accès comme FTP, SSH, etc.

Par simplicité, nous avons omis les composants plus techniques qui servent au fonctionnement interne du cluster (par exemple la gestion de la haute disponibilité, ou les agents de monitoring).

Les edge nodes ont un super pouvoir : on peut les multiplier sans toucher au cœur du cluster. Mais pourquoi le ferait-on ? Eh bien si plusieurs équipes ou services utilisent le cluster, il peut être intéressant que chacun ait « son » edge en

propre. Chacun peut ainsi choisir ses versions ou patches d'outils périphériques, ajouter des outils tiers dont il possède la licence, ou disposer de son propre SLA vis-à-vis de l'exploitant. Tout en souplesse...

Un exemple de cluster complet, avec 2 masters, 3 slaves et 1 edge node





◉ Serveurs physiques ou virtuels ?

C’est une question qui revient souvent. Parfois même, c’est la première question qu’on nous pose : pour beaucoup d’organisations, il est plus facile d’obtenir des machines virtuelles, mais **Hadoop les supporte-t-il ?**

Notre réponse est en général la suivante : oui mais il faut prendre des **précautions**. Des master et slave nodes virtuels font très bien l’affaire si on accepte le risque d’avoir des performances moindres en cas de contentions sur les châssis ou les disques partagés : ce risque est souvent acceptable pour un POC. Attention quand même, s’il y a d’autres applications sur le même châssis, elles peuvent pâtir de la gourmandise d’Hadoop en I/O disques et réseau – pensez à limiter les bandes passantes du cluster.

Les edge nodes, eux, sont moins sollicités que les masters et slave nodes : ils peuvent être virtuels. Dans le cas d’un cluster mixte virtuel et physique, cela complexifie un peu la configuration réseau, serveurs physiques et virtuels devant appartenir à un même réseau virtuel (VLAN) mais d’autres avantages équilibrent la balance. Ainsi, l’hyperviseur peut offrir des services de redondance avec basculement automatique, sur des serveurs qui sont importants car accédés directement par les utilisateurs.

*Pour beaucoup d’organisations, il est plus facile d’obtenir des machines virtuelles, mais Hadoop les supporte-t-il ? Cela fonctionne mais il faut prendre des précautions.*

◉ Le dimensionnement

Quand il s’agit de dimensionner des infrastructures Hadoop, on peut jouer sur deux paramètres : la taille et le nombre des slave nodes. Ce nombre est au minimum de trois (le plus petit cluster de production viable est donc de 5 nœuds : 2 masters en haute disponibilité, et 3 slaves pour respecter le facteur de réplication d’HDFS).

Nous avons évoqué au début de ce document la notion de commodity hardware. Pour rappel, il s’agit des serveurs faciles à obtenir et de coût unitaire raisonnable, en général ceux qui sont au catalogue des standards de l’entreprise. L’idée est donc de choisir dans un tel catalogue, dans la mesure du possible en limitant les « personnalisations » qui rajoutent des coûts. L’éditeur de la distribution que vous aurez choisie aura certainement des abaques qui vous permettront de monter un cluster généraliste compatible avec nos hypothèses de départ. De ce point de vue, les préconisations des grands éditeurs diffèrent quelque peu mais heureusement se rejoignent sur l’essentiel.

Demain, les progrès d’Hadoop remettront ces chiffres en question. Un paramètre en particulier devra être revu : **le dimensionnement de la mémoire**. En effet, HDFS est en train de se doter de fonctionnalités de cache, afin d’accélérer l’accès aux fichiers fréquemment lus dans les traitements. Processus, traitements et caches vont ainsi se trouver en compétition pour la mémoire des slave nodes.

Des configurations matérielles moyennes pour démarrer sur Hadoop

TYPE DE NŒUD	CPU	RAM	DISQUE	RÉSEAU
Master node	4 cœurs	96 Go	4 x 1 To : <ul style="list-style-type: none"><li>• 1 disque pour le système d’exploitation</li><li>• les autres montés en RAID<sup>16</sup></li></ul>	2 cartes : <ul style="list-style-type: none"><li>• 1 quelconque pour les accès administratifs</li><li>• 1 à 1 Gb/s pour le trafic Hadoop</li></ul>
Slave node	6 cœurs	96 Go	6 x 1 To : <ul style="list-style-type: none"><li>• 1 disque pour le système d’exploitation</li><li>• les autres montés en JBOD<sup>17</sup></li></ul>	2 cartes : <ul style="list-style-type: none"><li>• 1 quelconque pour les accès administratifs</li><li>• 1 à 1 Gb/s pour le trafic Hadoop</li></ul>

<sup>16</sup> Redundant Array of Inexpensive Disks : plusieurs disques sont montés en redondance pour garantir de ne pas perdre de données en cas de défaillance d’un disque. Le name node, sur un master, maintient le plan du système de fichiers HDFS, plan qu’il ne faut surtout pas perdre.

<sup>17</sup> Just a Bunch Of Disks : plusieurs disques sont montés indépendamment pour totaliser un grand volume de stockage. On ne cherche pas à fiabiliser les disques comme avec du RAID, car HDFS réplique déjà naturellement les blocs de fichiers.

### ⦿ Pour des workloads plus compliqués

Les hypothèses généralistes, qui servent de base aux abaqes ci-dessus, ne sont pas toujours suffisantes pour dimensionner correctement. Lorsque d'autres applications sont en jeu, il faut affiner. Comme il est impossible d'être précis en dehors de tout contexte, contentons-nous de donner quelques règles.

Selon le type de traitement envisagé, les besoins en ressources ne seront pas les mêmes. Ainsi, si HBase a besoin de mémoire pour être performant (mise en cache), Storm a surtout besoin de CPU.

L'affaire se complique encore lorsqu'un cluster « multi-usages » est mis en place, pour satisfaire des utilisateurs humains divers et des besoins applicatifs. Dans ces conditions, les sollicitations sont imprévisibles et les traitements variés, rendant difficile un dimensionnement pragmatique. Il faut analyser les besoins précis, leurs proportions dans le mix des usages, et faire des hypothèses de passage à l'échelle en appliquant des règles de 3. Quand les besoins s'opposent trop fortement, il n'est pas économique de dimensionner au pire, et on peut être amené à dédier des nœuds spécifiques du cluster à certains types d'applications. Les labels de YARN, récemment ajoutés, servent précisément à cela. Sinon, il reste toujours possible d'entretenir plusieurs clusters !

En tout cas une règle reste valable : que ce soit pour un cluster global ou des « bulles »

### Les nouvelles architectures physiques d'Hadoop

*Des mécréants ont l'outrecuidance de mettre en question le principe fondateur de l'architecture d'Hadoop : la colocalisation des traitements et des données. Faut-il envoyer les hérétiques au bûcher ?*

*Et bien non. Le principe de colocalisation est tout sauf un dogme. Il est vrai... pour du commodity hardware. Après tout, qu'est-ce qui nous empêche d'investir dans un matériel différent ? En particulier, un réseau performant entre les nœuds d'un cluster rend compétitif l'accès distant aux données. Les technologies (RDMA sur Infiniband, RoCE, etc.) combinent hauts débits (jusqu'à 40 Gbit/s) et une latence faible, de l'ordre de la µs.*

*Libérés du joug de la colocalisation, nous pouvons spécialiser les nœuds : des nœuds de stockage dédiés à certains média (disque dur, SSD, mémoire, etc.), des nœuds de calcul optimisés par usage (intensif en I/O, en calcul ou en mémoire, GPU, basse consommation, etc.), voilà de quoi construire des clusters optimisés et surtout très évolutifs au gré des besoins changeants. À un coût... qui reste bien plus important que celui du commodity classique ! Mais on n'a rien sans rien.*



de nœuds dédiés via les labels, la principale variable d'ajustement reste le nombre de slave nodes. Il faut bien sûr valider les hypothèses en suivant la consommation des ressources au jour le jour, et faire des projections pour s'assurer que le cluster est bien dimensionné pour les besoins qu'il sert. S'il s'avère trop juste, il est toujours possible d'ajouter des nœuds a posteriori – on n'est pas contraint d'investir tout de suite dans une configuration musclée en priant pour qu'elle suffise du premier coup.

## EXAMPLE

# Dimensionnement

Supposons que le besoin, tel qu'il est connu aujourd'hui, vous conduise à estimer une taille de données sur HDFS à 15 To, soit  $3 \times 15 = 45$  To réels en tenant compte du facteur de réplication habituel. Après examen du catalogue, vous avez retenu deux modèles candidats :

- Le Modèle A, qui dispose d'un CPU quadricœur à 2,25 GHz, de 128 Go de RAM et pouvant accueillir 6 disques de 2 To chacun. Il vous coûterait 6000 € à l'unité.

- Le Modèle B, qui dispose d'un CPU quadricœur à 2,5 GHz, de 256 Go de RAM, pouvant accueillir 10 disques de 1,5 To chacun, pour la modique somme de 8 500 € par serveur.

Pour stocker vos 45 To, il vous faudrait donc soit :

- 12 Modèle A pour un coût total de 72000 €, avec 3 To réels de marge.
- 10 Modèle B pour un coût total de 85000 €, avec 5 To réels de marge.

Vous décidez qu'une marge de 3 To est suffisante pour démarrer, il semble donc plus intéressant de choisir les Modèles A malgré les 2 serveurs de plus.

Après quelques semaines d'utilisation, les chiffres montrent que les 128 Go de RAM par nœud sont un petit peu justes compte tenu des traitements exécutés sur le cluster. On vous propose d'équiper les serveurs avec 192 Go de RAM à la place, pour un coût unitaire de 500 €. Cette quantité vous convient, et la facture

révisée s'élève à  $12 \times 6\,500 \text{ €} = 78\,000 \text{ €}$ . C'est toujours moins cher que les 10 Modèles B : adjugé !

Et si un jour le cluster doit grandir, et que les Modèles A ne soient plus au catalogue pour cause d'obsolescence ? Aucune importance, Hadoop se satisfait très bien de serveurs hétérogènes.

## Un peu plus de détails

Les indications ci-dessus sont volontairement très générales. Nous faisons un aparté pour expliquer grossièrement comment sont construites de telles abaques.

Il faut voir un cluster Hadoop comme un ensemble de ressources. Les CPU, mémoire et disques de chaque slave node vont coopérer pour mener à bien les traitements demandés. Pour que ces traitements soient efficaces, il faut limiter les goulets d'étranglement. La parallélisation sur plusieurs nœuds, par exemple avec un algorithme MapReduce, est un premier niveau d'optimisation ; encore faut-il que les serveurs eux-mêmes soient efficaces.

Un slave node va exécuter plusieurs containers en même temps, typiquement autant de containers que de cœurs de CPU présents sur la machine (moins 1 si on tient compte du système d'exploitation qui a besoin d'un cœur pour ses affaires). Au-delà, les containers se trouveraient bloqués par la pénurie de puissance de calcul.

Prenons l'exemple d'un traitement MapReduce. Chaque mapper va lire des données d'HDFS par blocs entiers. Sur un serveur donné, il faut éviter que les mappers ne sollicitent plus d'accès disque que ce que la machine est capable de fournir, pour éviter un goulet au niveau des contrôleurs disques. On doit donc veiller à brancher 1 à 2 disques par cœur de calcul (en tenant compte de l'hyperthreading) sur chaque serveur – d'où le montage en JBOD pour les data nodes. Chaque disque disposant de son contrôleur indépendant, il pourra en principe servir un mapper de manière exclusive.

En ce qui concerne la RAM, il faut raisonner là aussi par container. Un container est essentiellement une JVM, dont les besoins en mémoire seront la somme de deux paramètres :

- La « **PermGen** » : mémoire occupée par les classes Java du JDK et des frameworks Hadoop utilisés par le traitement (MapReduce ou Tez par exemple). Cette taille est incompressible, invariable pour un traitement donné, et peu élevée : inférieure à 1 Go.
- La « **heap** » : mémoire obtenue à la demande par le code des frameworks et du traitement, au fur et à mesure de son exécution. Elle dépend fortement de l'environnement : traitement mis en œuvre et données sur lesquelles il opère. Il n'est pas toujours facile de l'estimer à l'avance.

Avec MapReduce, on configure séparément la heap des mappers et des reducers. On considère qu'une valeur de départ raisonnable est 4 Go pour les mappers et 6 Go pour les reducers, quitte à augmenter ces valeurs si des traitements échouent par manque de mémoire. En ajoutant les besoins de PermGen et un peu

de marge, on arrive à un ordre de grandeur de 8 Go par container. Multipliez par le nombre de cœurs, ajoutez 1 Go pour le système d'exploitation, et vous obtenez la mémoire totale d'un slave node. La configuration fait intervenir quantité d'autres paramètres bien plus fins, comme les buffers de tri pour le shuffle & sort, ou les rapports entre mémoire physique (RAM) et virtuelle (débordement sur disque quand la RAM est pleine, au détriment des performances). Il s'agit de tuning élaboré à des fins de performances, spécifiques à une application, mais nous ne nous y aventurerons pas ici !

Pour le réseau, l'équation est plus simple et les choix plus réduits. Les slave nodes échangent des données en permanence, pour la réplication HDFS, lors du shuffle & sort... Pour cette raison, on part souvent sur des liens à 1 Gb/s, offrant un bon rapport performance / coût. Là encore, des applications spécifiques peuvent apporter des contraintes différentes. N'oubliez pas de dimensionner correctement les liens réseaux qui relient les racks entre eux : dans le pire des cas, ils devront supporter des échanges simultanés entre tous les serveurs du cluster. Pour un rack de 10 serveurs, on prévoit ainsi des switches top-of-rack de  $10 \times 1 = 10$  Gb/s.

Pour aller plus loin, nous vous suggérons de visionner des présentations très poussées sur le dimensionnement des clusters Hadoop, par exemple [http://fr.slideshare.net/Hadoop\\_Summit/costing-your-bug-data-operations](http://fr.slideshare.net/Hadoop_Summit/costing-your-bug-data-operations), par des ingénieurs de chez Yahoo!.. Cette présentation combine les aspects techniques et financiers pour vous aider à trouver les configurations optimales.

# Hadoop aujourd'hui e+ demain

*On assiste à un réel mouvement de transformation d'Hadoop, démarré depuis maintenant 2 ans. L'écosystème évolue dans toutes ses couches : socle technique, applications et outils de manipulation des données. C'est parfois difficile à suivre, cela méritait bien un travail de synthèse. Si vous voulez savoir à quoi ressemblera Hadoop dans le futur, montez dans la DeLorean !*

# Dès aujourd’hui :

## des modèles de programmation plus variés que MapReduce

L’introduction de YARN a libéré le développeur des contingences de MapReduce. YARN est à votre disposition pour écrire l’algorithme aux petits oignons qui vous fera gagner la partie face à vos concurrents : il suffit de se conformer à ses API Java et de construire un JAR contenant l’application, pour qu’elle s’exécute sur le cluster, en mode distribué.

Cependant, on se lance rarement dans le développement de bas niveau sur YARN. Le véritable apport de ce framework est dans l’explosion du nombre d’outils qu’il a permis d’amener sur Hadoop. MapReduce et ses dérivés (Hive, Pig) bien sûr, mais aussi HBase, Tez, Impala, Drill, Storm, Spark, SolR, etc. En batch, en interactif ou en temps réel ; par programmation, par script ou par l’abstraction du machine learning, la couverture fonctionnelle d’Hadoop s’est considérablement étendue et continue de s’étendre<sup>18</sup>.

Des éditeurs commerciaux proposent aussi une implémentation Hadoop de leur moteur de calcul, permettant à leurs clients de faire vivre leur patrimoine applicatif sur Hadoop. C’est le cas de SAS, ou d’Actian par exemple.

Au final, le nombre de manières d’accéder à une même donnée est bien vaste. Et bien sûr, tous les traitements et requêtes exécutés par ce petit monde se partagent respectueusement les ressources du cluster ; c’est d’ailleurs la mission première de YARN.

<sup>18</sup> Attention, certains outils ne proposent pas encore de mécanisme de sécurité sur les données qu’ils manipulent ; c’est notamment le cas de Storm, Spark et SolR. Bien sûr, il est prévu à terme qu’ils le proposent, mais en attendant, vous êtes prévenus...

# Demain matin :

## le multi-tenant

Hadoop est un très bon candidat pour une plateforme multi-tenant d’entreprise. Par multi-tenant, nous entendons un système hébergeant les données de plusieurs entités, et servant plusieurs clients au moyen d’une offre de services logicielle adaptée, dans des scénarios de restitution, de transformation ou d’enrichissement des données.

Cette définition simple en apparence présuppose un certain nombre de choses :

- Un stockage élastique distribué, capable de s’accomoder de données structurées et non structurées avec des grandes profondeurs d’historique.
- Une capacité à exécuter plusieurs workloads : transactionnel, temps réel (streaming), interactif, batch analytique ou de machine learning.
- Un partage harmonieux des ressources de calcul et de stockage entre les clients, avec des règles de sécurité adéquates.
- Une gouvernance des données stockées, avec des fonctions de traçabilité, de gestion du cycle de vie (purge des historiques avec rotation, par exemple), et des mécanismes de sécurité robustes.
- Une interopérabilité logicielle grâce à une

offre d’outils et de connecteurs suffisante pour laisser de l’autonomie aux utilisateurs. Le support de SQL et des connecteurs xDBC sont le minimum.

- Une bonne exploitabilité, par le biais d’outils, d’APIs et de reportings suffisants, et bien sûr d’une architecture robuste.

La bonne nouvelle, c’est que le soldat Hadoop mérite déjà ses galons. S’il fallait juger de la maturité de la plateforme, voilà ce que cela donnerait :

- Stockage élastique distribué  
SUPER MATURE !
- Support de plusieurs workloads  
AU POINT
- Partage des ressources  
ÇA FONCTIONNE
- Gouvernance et sécurité  
PEUT MIEUX FAIRE
- Interopérabilité logicielle  
ON Y EST PRESQUE
- Exploitabilité  
ENCORE UN PEU TECHNIQUE

En particulier, YARN et HDFS bougent beaucoup. Les labels YARN annoncent la spécialisation des nœuds en fonction de l'usage, calcul ou stockage par exemple. HDFS est maintenant capable d'appliquer des politiques de stockage en fonction du medium sous-jacent (disque dur, SSD voire mémoire pour les données « chaudes »). En particulier, l'apparition de caches mémoire au niveau des data nodes devrait apporter des gains de performances significatifs en diminuant les accès disques. Cela va de pair avec l'augmentation des capacités RAM des serveurs.

Un domaine sur lequel il convient d'être vigilant est celui de la gouvernance des données : être capable d'explorer simplement les données présentes dans le cluster (un Toad pour Hadoop, en quelque sorte), pouvoir déterminer les opérations qui ont conduit à la construction d'un dataset particulier, ou encore être en mesure de purger la plateforme des données périmées. Les outils de la communauté open source se font encore attendre ; Apache Atlas est une belle promesse.

L'exploitabilité mériterait aussi de remonter dans les couches pour fournir une vision agrégée de ce que qui se passe sur la plateforme, vision à laquelle on accède aujourd'hui en jonglant avec des fichiers de logs joyeusement répartis sur toutes les machines.

# Dès demain et pour les mois à venir : fiabilisation et sécurité

Difficile de le cacher : Hadoop est encore dans sa prime jeunesse. Et il n'est pas tout à fait fini. Nous avons hésité à vous faire cette confidence de peur que vous ne détourniez votre chemin, ce qui serait dommage. Car on peut déjà faire beaucoup de choses avec Hadoop de façon fiable, dès que l'on reste sur les sentiers battus.

En revanche, dès que l'on s'en éloigne, il faut être prêt à relever ses manches et à mettre les mains dans le cambouis. Et dans ce cas, on aura plus besoin du bricoleur de génie que de l'architecte TOGAF. Enfin, quand on dit « du » bricoleur, il faudrait plutôt parler « des » bricoleurs.

Par exemple, sur un gros projet industriel basé sur une plateforme Hadoop multi-tenant dans lequel nous avons trempé, ce n'est pas moins de 5 à 7 personnes qu'il fallait en support (moitié en développement, moitié en exploitation) pour arriver à faire fonctionner la « bête » au quotidien. Par comparaison, si tout avait été sur un SGBD Oracle classique, on s'en serait sorti avantageusement avec 1 seul DBA. Oui, mais quel potentiel en comparaison...

Le gros point de complexité est aujourd'hui la sécurité. Non qu'elle ne soit pas bien gérée, les outils sont là et fonctionnent raisonnablement bien ! Cependant, c'est un sujet compliqué qui demande une expertise forte, de l'infrastructure à la gouvernance en passant par l'automatisation des scripts de provisioning. Sur des infrastructures distribuées, donc complexes, avec un outillage de supervision encore très bas niveau, le diagnostic des bugs et problèmes rencontrés relève beaucoup de l'acrobatie de haute voltige.

Il faut être cependant conscient que la situation s'améliore de jour en jour (même si on peut regretter que les éditeurs de distribution soient plus concentrés sur la course à la fonctionnalité que la course à la fiabilité), et qu'il est dangereux d'attendre des lendemains meilleurs si vous avez des use cases pertinents. Les outils de sécurité d'entreprise – chiffrement des données HDFS, intégration à une PKI externe – sont là et ne demandent qu'un peu de temps pour être éprouvés. Dès à présent, rien n'est impossible : avec de l'huile de coude et de la persévérance, on arrive à surmonter (ou au pire contourner) tous les obstacles.



# Dans un futur proche : vers des clusters à la demande

Le multi-tenant, c'est bien, mais c'est comme tout, il ne faut pas en abuser. En particulier, il est une chose qu'il faut éviter, c'est d'avoir un cluster unique qui hébergerait la production industrielle, des projets d'innovation hétéroclites, des environnements de développements et d'intégration. Même si l'architecture d'Hadoop le permet techniquement, ces espaces ont des cycles de vie trop différents pour cohabiter et vous aurez besoin de proposer des clusters plus ou moins éphémères voire à la demande.

En particulier, la gestion des environnements projets est friande de ce type d'offre. D'autant plus avec Hadoop, où il est essentiel que les développements et les tests soient faits sur des clusters représentatifs, en termes de configuration (sinon de performances) : le moindre écart peut provoquer des différences de comportement très difficiles à traquer. Et ça, vous pouvez nous croire, c'est du vécu !

Tout ceci rejoint le discours déjà tenu sur DevOps et sur la nécessité d'automatiser les opérations. La communauté ne s'est pas trompée, et les initiatives arrivent !

Ce que l'on voudrait, c'est un service à qui l'on demande simplement « un cluster avec 10 data nodes pour commencer, élastique, la

haute disponibilité activée, HBase en plus, un café et l'addition » et pouf ! les machines sont réservées, lancées et pré-installées.

Les outils de déploiement de bas niveau existent depuis longtemps, mais ils ne permettent pas cela. On pense aux outils Hadoop bien sûr (Cloudera Manager ou Ambari), à Chef et Puppet (des automates de déploiement de configurations), ou à Ansible (un SSH distribué scriptable). Il existe encore peu de livres de recettes (cookbooks) pour Hadoop, et surtout ces outils ne savent pas réserver des machines sur un cloud public ou privé. Il faut disposer des machines d'abord.

Pour répondre à notre besoin, il faut se tourner du côté des solutions de cloud. L'une d'entre elles en particulier fait beaucoup parler d'elle en ce moment : il s'agit d'OpenStack. Le projet Sahara (ex-Savanna), sous sa houlette, a pour but de proposer des distributions Hadoop sur des clouds OpenStack<sup>19</sup>.

Comme approche alternative, YARN est en passe de supporter des traitements soumis sous forme de conteneurs Docker. Cela ouvre la porte à des applications complètement autonomes dans leur packaging (penser aux WAR et EAR du monde JEE).

# À moyen terme : vers une plateforme universelle ?

Un certain nombre d'améliorations d'Hadoop ont été mises en œuvre pour transposer les fonctions des outils classiques dans le domaine du Big Data. Par exemple, concernant le machine learning, la communauté a œuvré pour doter Hadoop d'interpréteurs SQL suffisamment performants pour permettre le requêtage interactif – ce que les bases relationnelles font depuis longtemps.

Dans la continuité de cet effort, il sera bientôt possible de faire des mises à jour « transactionnelles » des données gérées (voir par exemple, les projets Hive Streaming<sup>20</sup> et Kudu<sup>21</sup>, encore à l'état d'ébauches). Nous mettons « transactionnelles » entre guillemets, car on reste loin des principes ACID des bases de données relationnelles.

Et ces améliorations se font en parallèle des vraies innovations que sont le traitement des données non structurées, le machine learning, l'absorption des très hauts débits du futur monde de l'internet des objets...

Plus généralement, le choix grandissant des frameworks et des workloads peut laisser penser qu'Hadoop répondra à tous les besoins de l'entreprise. C'est un peu exagéré : il reste

une quantité d'usages dits « classiques » pour lesquelles les architectures actuelles (SGBD, ETL, ESB, serveurs d'application, etc.) font très bien l'affaire, à un coût bien moindre. Ces architectures sont aussi bien mieux maîtrisées par les DSI ; Hadoop n'est pas prêt de les remplacer. Mais il est vrai que, dès lors que ces architectures atteignent leurs limites de scalabilité, Hadoop est une alternative économiquement crédible.

### Les architectures lambda, ou comment concilier temps réel et Big Data

C'est une promesse a priori irréalisable : vouloir traiter des données très rapidement en faisant des calculs des historiques très volumineux. Une architecture lambda<sup>22</sup> résout ce dilemme en précalculant périodiquement des résultats partiels, qui sont ensuite complétés avec les flux de données temps réel. Hadoop, avec le support des workloads multiples, est un bon candidat pour une plateforme unique et fiable hébergeant les deux couches.

Attention quand même, les architectures lambda sont avant tout un concept. Le diable est caché dans les détails, et un réel travail d'architecture doit être mené pour formaliser les enjeux de latence, de tolérance à la panne, de rejet de messages, etc.

<sup>19</sup> Le site d'OpenStack est accessible à cette adresse : <http://www.openstack.org/>, et celui de Sahara à celle-ci : <https://wiki.openstack.org/wiki/Sahara>

<sup>20</sup> <https://cwiki.apache.org/confluence/display/Hive/Streaming+Data+Ingest>


<sup>21</sup> <http://getkudu.io>

<sup>22</sup> Le site de référence de ce pattern se trouve ici : <http://lambda-architecture.net/>



# Conclusion

*Si Hadoop est dès aujourd'hui prêt pour de la production, il n'a pas encore atteint son plein potentiel. Sa gouvernance particulière, son écosystème d'éditeurs-partenaires en font un outil en perpétuelle évolution, sans cesse en train de s'adapter aux nouveaux défis techniques de Big Data.*



# Il est urgent de ne pas attendre

En tant que plateforme de stockage et de parallélisation de traitements, Hadoop est aujourd’hui mature. Les Géants du Web ont construit un écosystème extrêmement riche, d’une stabilité suffisante pour des usages en production. Ils n’avaient pas le choix : eux-mêmes ont besoin d’Hadoop au quotidien, ils sont donc les mieux placés pour améliorer la plateforme. Eat your own dogfood, dit-on là-bas.

Cependant, ces contributeurs-utilisateurs représentent un type d’entreprise particulier, dont la culture est encore assez peu répandue en Europe. Ils ont su mettre en place des compétences et une organisation interne favorables à l’accueil d’une telle plateforme. Pour des entreprises plus « classiques », si on peut dire, la technologie est encore mal connue (nous espérons quand même que la diffusion de cette brochure aura permis de mitiger ce constat !).

*Le futur d’Hadoop, c’est un service de calcul à la demande, polyvalent, capable de traiter aussi bien du calcul scientifique que du fichier ou du flux à très haut débit, en s’appuyant sur des algorithmes de plus en plus distribués.*

Elle suscite aussi de la méfiance, parfois. Faire reposer sa stratégie digitale sur un data lake qui exécute un logiciel open source peut faire tiquer, alors que notre culture nous pousse à prendre le moins de risques possibles. Quant aux fonctionnalités « entreprise », telles la sécurité, la gouvernance de la donnée et les backups, elles ne sont pas encore au niveau de ce qu’une organisation attendrait d’un éditeur bien installé. Et il est vrai que la politique de versions et la multiplicité des éditeurs apportent une certaine confusion ! Qui peut dire quel sera l’écosystème de référence de demain ? Pas nous, en tout cas pas au moment où nous écrivons ces lignes...

Tout cela va mettre un certain temps à se décanter. Et pourtant, que d’opportunités à saisir d’ici là... Il est tout à fait possible, et souhaitable même, d’adopter Hadoop petit à petit, pour glisser vers de l’innovation digitale

où Big Data et les technologies afférentes (machine learning notamment) montrent tout leur pouvoir. Les cas d’usage sont infinis et s’enrichissent des nouvelles idées suscitées par les nouvelles capacités de calcul. La plateforme et l’écosystème continueront d’évoluer pour permettre leur mise en œuvre.

Mais le plus excitant, selon nous, ce sont les applications concrètes qui vont fleurir demain, rendues possibles par la richesse de cet écosystème et par la part grandissante du traitement in-memory. C’est vraiment à ce moment-là qu’Hadoop prendra son envol, quittera son passé d’exécuteur de batch spécialisé pour devenir un socle d’applications.

Pour autant, il ne faut pas voir en Hadoop un super marteau capable de transformer tous les problèmes de calcul en clous. Un framework de calcul distribué, un « système d’exploitation Big Data », c’est pourtant tentant ! Certains problèmes, notamment ceux où le calcul pur domine, ou bien ceux ayant à faire à de la donnée très structurée, pourraient être résolus plus efficacement avec d’autres technologies : bases de données parallèles, grille de calcul sur GPU, etc. Le critère final reste le coût de

la plateforme à maintenir (ou du service cloud à utiliser) pour traiter le problème en temps voulu. Cela dépend à la fois de vos enjeux métier, de la nature du problème et de sa nature ponctuelle ou récurrente.

Quoi qu’il en soit Hadoop est promis à un bel avenir. C’est une plateforme de référence, elle s’améliore sans cesse et... peut-être cette brochure sera-t-elle déjà obsolète au moment où vous la lirez ?

# Le mot de la fin : quand l'éléphant fait l'ours

Ce livre blanc est le fruit de 2 ans de travail et l'aboutissement de 5 ans de veille sur Hadoop et Big Data. Pendant tout ce temps, nous avons décortiqué Hadoop, nous l'avons installé chez nos clients, nous les avons formés et leur avons mis le pied à l'étrier, nous avons tissé des liens avec les éditeurs, nous avons donné des conférences... et nous avons réalisé cet ouvrage !

Selon l'adage, un éléphant se dévore bouchée par bouchée. Tout cela n'aurait pu se faire sans la mastication d'un grand nombre d'Octos dont voici les trigrammes : AHU, AMB, BJC, BLA, CAB, CAM, CBR, DZO, EBI, FHI, FRD, ILE, JCA, JGL, JGO, JOB, JPI, LCI, MAN, MBE, MBO, MDE, MDO, MGU, MLU, NGR, NSC, OMA, RCA, RGH, RKO, RMA, RSA, SDL, SIM, SWI, TVI.

Le livre à peine imprimé, Hadoop a déjà changé... L'aventure continue !

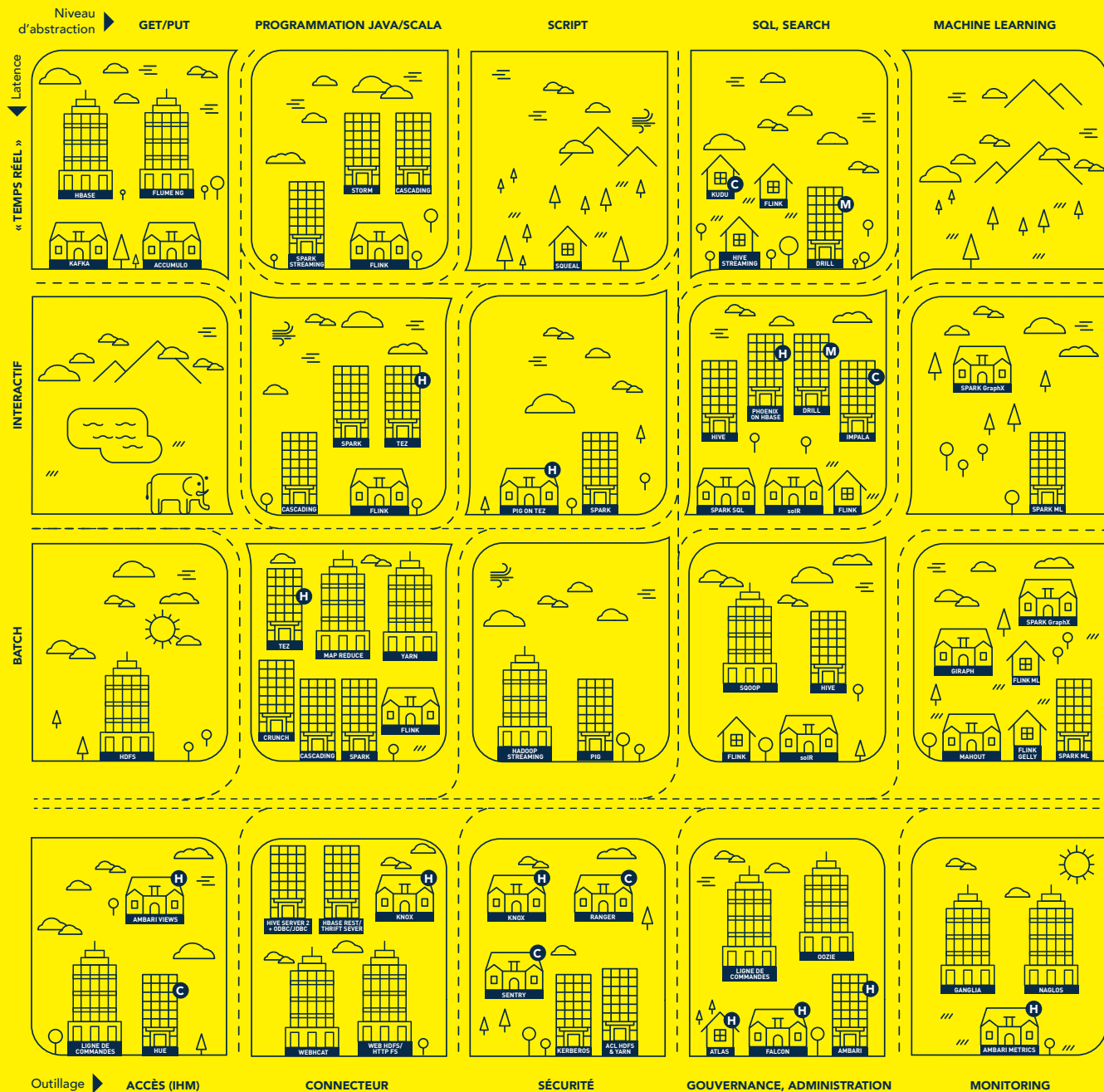
# OCTO Technology

*« Nous croyons que l'informatique transforme nos sociétés. Nous savons que les réalisations marquantes sont le fruit du partage des savoirs et du plaisir à travailler ensemble. Nous recherchons en permanence de meilleures façons de faire. THERE IS A BETTER WAY ! »*

– Manifeste OCTO Technology

OCTO Technology est un cabinet de conseil et de réalisation IT fondé en 1998 aujourd'hui présent dans cinq pays : La France, le Maroc, la Suisse, le Brésil et l'Australie. Les consultants OCTO accompagnent leurs clients dans la transformation digitale de leurs entreprises en intervenant à la fois sur la technologie, la méthodologie et la compréhension de leurs enjeux métier. Les équipes OCTO utilisent la technologie et la créativité pour les accompagner dans la construction de nouveaux business models. Pour la 4<sup>e</sup> année, OCTO s'est positionné sur le podium du palmarès Great Place to Work® des entreprises de moins de 500 salariés où il fait bon travailler. Le cabinet organise en parallèle USI qui s'est imposée depuis 2008 comme une référence parmi les plus grandes conférences internationales sur la transformation digitale.

# CARTOGRAPHIE HADOOP



## ÉDITEURS PORTEURS

(seul l'écosystème open source est représenté)



Hortonworks



MapR



Cloudera

## MATURITÉ DES OUTILS



Projet au stade alpha



Pas mature ou partiellement intégré à Hadoop



Mature et intégré, mais pas totalement éprouvé



Mature, éprouvé et robuste



Dépot légal : décembre 2015

Conçu, réalisé et édité par OCTO Technology,  
50 avenue des Champs-Élysées - 75008 Paris.

© OCTO Technology 2016

Les informations contenues dans ce document présentent le point de vue actuel d'OCTO Technology sur les sujets évoqués, à la date de publication. Tout extrait ou diffusion partielle est interdit sans l'autorisation préalable d'OCTO Technology.

Les noms de produits ou de sociétés cités dans ce document peuvent être les marques déposées par leurs propriétaires respectifs.



**Octo**  
Technology  
There is a better way

[www.octo.com](http://www.octo.com) - [blog.octo.com](http://blog.octo.com)

PARIS | RABAT | LAUSANNE | SÃO POLO | SYDNEY