جامعة سيدي محمد بن عبد الله بفاس
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH DE FES
كلية العلوم ظهر المهراز فاس
FACULTE DES SCIENCES DHAR EL MAHRAZ FES

# Cours de Big Data

# 2021-2022

*Pr. El fazazy khalid*

***Pr. EL FAZAZY KHALID***

# Introduction aux Big Data

## 1. Définition «BigData»

Le terme «BigData» fait généralement référence à des ensembles de données qui dépassent la capacité de manipulation des outils traditionnels. En effet, la taille de données a un impact sur la capture, le mouvement, le stockage, le traitement, la présentation, l'analyse, les rapports et la latence des données (le temps qu'il faut pour accéder aux données).

Les entreprises utilisent Big Data pour résoudre une diversité importante de problèmes problème d'organisation et de management :

- Détection des fraudes

Les banques utilisent Hadoop pour détecter les fraudes transactionnelles en fournissant des analyses sur de grands clusters de matériel de base. Elles appliquent des modèles analytiques à un ensemble complet de transactions pour leurs clients et fournissent une détection des fraudes en temps quasi réel.

- Analyse marketing des médias sociaux

En surveillant, en collectant et en agrégeant les données de diverses sources Internet telles que les blogs, les forums, les fils d'actualité, les tweets, et les médias sociaux, les entreprises utilisent Hadoop pour extraire et agréger des informations sur leurs produits, services et concurrents, découvrir des modèles et les tendances importantes,…

- Analyse des modèles d'achat

Les entreprises du secteur de la vente utilisent Hadoop pour déterminer les produits les plus appropriés à vendre dans un magasin particulier en fonction de l'emplacement du magasin et des habitudes d'achat de la population qui l'entoure.

- Détermination des exigences en matière d'extension du réseau routier

En surveillant le trafic à différentes heures de la journée, les urbanistes peuvent déterminer les goulots d'étranglement de la circulation, ce qui leur permet de décider si des rues / voies supplémentaires sont nécessaires pour éviter les embouteillages aux heures de pointe.

- Analyse et médiation du réseau

L'analyse en temps réel d'une grande quantité de données générées sous forme de données (transaction d'utilisation, performance du réseau, informations sur les sites de cellule, back-office, …) permet aux entreprises de réduire les dépenses opérationnelles et améliorer l'expérience des utilisateurs sur les réseaux.

## 2. Problème du Stockage

Il n’est pas facile de mesurer le volume total de données stockées électroniquement, mais selon une estimation IDC (International Data Corporation), la taille de «l'univers numérique» est de plusieurs zettaoctets.

| | | | |
|---|---|---|---|
| 1 kibioctet | (kio) | = $2^{10}$ octets = 1 024 octets | |
| 1 mébioctet | (Mio) | = $2^{20}$ octets = 1 024 kio | = 1 048 576 octets |
| 1 gibioctet | (Gio) | = $2^{30}$ octets = 1 024 Mio | = 1 073 741 824 octets |
| 1 tébioctet | (Tio) | = $2^{40}$ octets = 1 024 Gio | = 1 099 511 627 776 octets |
| 1 pébioctet | (Pio) | = $2^{50}$ octets = 1 024 Tio | = 1 125 899 906 842 624 octets |
| 1 exbioctet | (Eio) | = $2^{60}$ octets = 1 024 Pio | = 1 152 921 504 606 846 976 octets |
| 1 zébioctet | (Zio) | = $2^{70}$ octets = 1 024 Eio | = 1 180 591 620 717 411 303 424 octets |
| 1 yobioctet | (Yio) | = $2^{80}$ octets = 1 024 Zio | = 1 208 925 819 614 629 174 706 176 octets |

Les organisations n'ont plus à gérer simplement leurs propres données le succès futur dépendra dans une large mesure de leur capacité à extraire de la valeur des données d’autres organisations.

Le problème est simple:

- Un lecteur typique à 1990 pouvait stocker 1370 Mo de données et avait une vitesse de transfert de 4,4 Mo / s, et environ cinq minutes pour que vous puissiez lire toutes les données d'un lecteur complet.
- 20 ans plus tard, les lecteurs de 1 téraoctet sont la norme, mais la vitesse de transfert est environ 100 Mo / s, il faut donc plus de deux heures et demie pour lire toutes les données du disque.
    - C'est long pour lire toutes les données sur un seul lecteur et l'écriture est encore plus lente.

Imaginez si nous avions 100 disques, en parallèle, chacun contenant un centième des données, nous avons pu lire les données en moins de deux minutes.

- **Le premier problème** à résoudre est la défaillance matérielle: dès que vous commencez à utiliser de nombreux composants matériels, le risque que l'un d'entre eux échoue est assez élevé.
    - Une manière courante d'éviter la perte de données est la réplication: des copies redondantes des données sont conservées par le système afin qu'en cas de panne, une autre copie soit disponible. C'est ainsi que fonctionne RAID

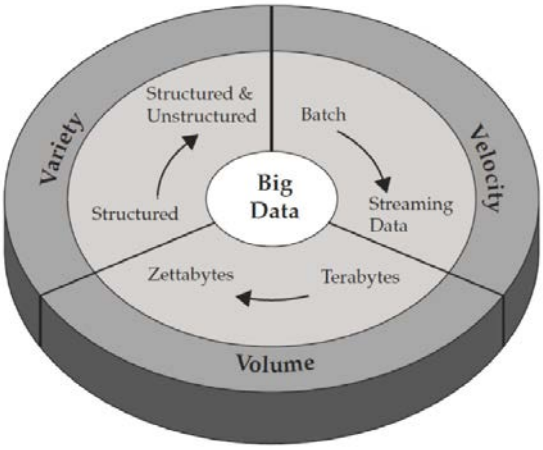
- Bien que le système de fichiers distribué Hadoop (HDFS), adopte une approche légèrement différente.

- **Le deuxième problème** est que la plupart des tâches d'analyse doivent pouvoir combiner les données d'une manière ou d'une autre, et les données lues à partir d'un disque peuvent devoir être combinées avec les données de l'un des 99 autres disques.

Différents systèmes distribués permettent de combiner des données provenant de plusieurs sources. MapReduce fournit un modèle de programmation qui fait abstraction du problème des lectures et des écritures de disque, le transformant en un calcul sur des ensembles de clés et de valeurs. Il y a deux parties dans le calcul - la map et la réduction - et c'est l'interface entre les deux où se produit le «mixing». Comme HDFS, MapReduce a une fiabilité intégrée.

# 3. Exigences

Le volume de données n'est pas le seul moyen pour catégoriser le Big Data. Par exemple, l'analyste Doug Laney  a décrit le Big Data en termes de ce que l'on appelle maintenant les 3V:




- **Volume**: la taille globale de l'ensemble de données.
- **Vélocité** : la vitesse à laquelle les données arrivent et la vitesse à laquelle elles doivent être traitées.
- **Variété**: la diversité de formats de l'ensemble de données : journaux Web, l'audio, les images, les données de capteurs ou de périphériques et le texte non structuré, entre autres types.

En effet un système Big Data nécessite un ensemble d'outils riche en fonctionnalités et une plate-forme de stockage distribué capable de déplacer de très gros volumes de données dans le système sans perdre de données.

En résumé, pour manipuler le Big Data, un système nécessite les éléments suivants:

- Une méthode de collecte et de catégorisation des données
- Une méthode de transfert de données dans le système en toute sécurité et sans perte de données et la possibilité de surveiller les tendances des données en temps réel
- Un système de stockage distribué sur de nombreux serveurs, extensible à des milliers de serveurs et qui offre la redondance et la sauvegarde des données en cas de panne matérielle
- Outils de reporting et de planification pour déterminer quand les tâches seront exécutées et afficher l'état des tâches

# 4. Hadoop

Hadoop est un framework libre et open source écrit en Java destiné à faciliter la création d'applications distribuées permettant aux applications de travailler avec des milliers de nœuds. Ainsi chaque nœud est constitué de machines standard regroupées en grappe.

Hadoop a été inspiré par la publication de MapReduce, GoogleFS et BigTable de Google. Hadoop a été créé par Doug Cutting et fait partie des projets de la fondation logicielle Apache depuis 2009.

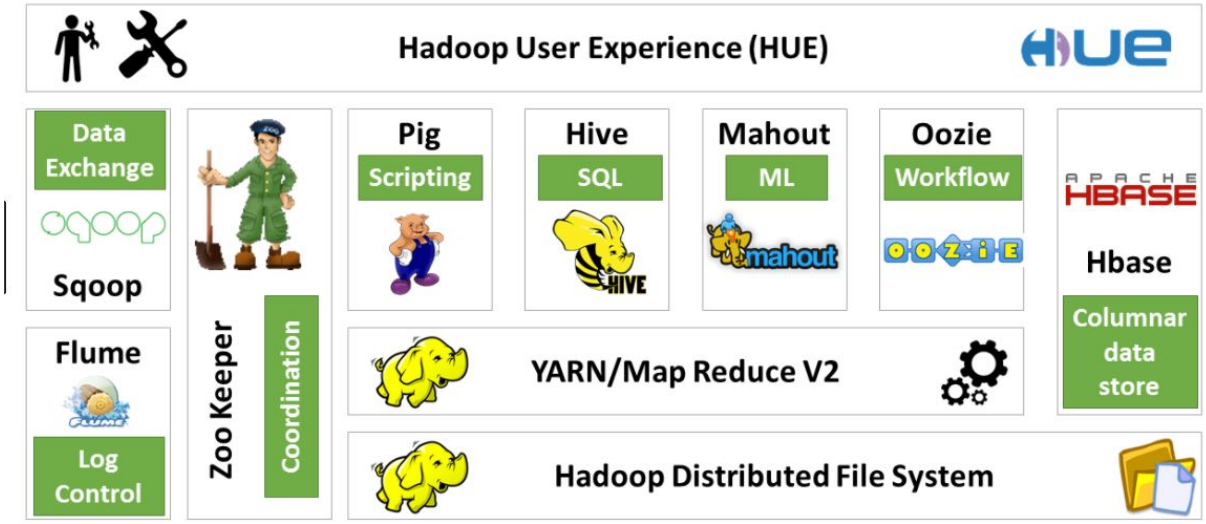
Hadoop relève les défis du Big Data en simplifiant la mise en œuvre d'applications distribuées. Il permet de diviser les tâches en fragments de travail et de les répartir sur des milliers d'ordinateurs en offrant un temps d'analyse rapide et un stockage distribué de quantités massives de données multi-structurées. Il fournit un mécanisme évolutif et fiable pour traiter de grandes quantités de données sur un cluster de matériel de base.

Hadoop diffère des approches distribuées précédentes des manières suivantes:

- Les données sont diffusées à l'avance.
- Les données sont répliquées dans un cluster d'ordinateurs pour assurer la fiabilité et la disponibilité.
- Le traitement des données se produire là où les données sont stockées, éliminant ainsi les goulots d'étranglement de la bande passante.

La caractéristique la plus importante de Hadoop est la séparation nette entre la programmation et l'infrastructure : Il prend en charge et cache la complexité de l'infrastructure.

L’écosystème Hadoop se compose des éléments suivants:




➢ HDFS : Stockage et réplication

- Un composant fondamental de l'écosystème Hadoop
- HDFS est le mécanisme par lequel une grande quantité de données peut être distribuée sur un cluster d'ordinateurs, et les données sont écrites une fois, mais lues plusieurs fois à des fins d'analyse

➢ MapReduce : Traitement distribué

- Un modèle de programmation pour le traitement de données distribué et parallèle, divisant les Jobs en deux phases: Map et Reduce.
- Les développeurs écrivent les Jobs de MapReduce, en utilisant les données stockées dans HDFS pour un accès rapide aux données.
- En raison de la nature du fonctionnement de MapReduce, Hadoop apporte le traitement des données de manière parallèle, ce qui se traduit par une mise en œuvre rapide.

➢ HBase : Accès rapide en lecture / écriture

- HBase est utilisé pour un accès rapide en lecture / écriture à de grandes quantités de données.
- HBase utilise Zookeeper afin de s'assurer que tous ses composants sont opérationnels.
- Une base de données NoSQL orientée colonnes est construite sur HDFS.

➢ Zookeeper : Coordination

- Zookeeper est le service de coordination distribué de Hadoop.
- Il s'agit d'un service conçu pour fonctionner sur un cluster de machines et utilisé pour la gestion des opérations et de nombreux composants d'Hadoop.

➢ Oozie : Workflow et planification

- Un système de flux de travail évolutif utilisé pour coordonner l'exécution de plusieurs tâches MapReduce.
- Il est capable de gérer une quantité importante de complexité, en basant l'exécution sur des événements externes qui incluent le timing et la présence des données requises.

➢ Pig : Scripting

- La plateforme Pig comprend un environnement d'exécution et un langage de script (Pig Latin) utilisé pour analyser l'ensemble de données Hadoop. Son compilateur traduit Pig Latin en séquences de programmes MapReduce.

➢ Hive : SQL

- Un langage de haut niveau de type SQL utilisé pour exécuter des requêtes sur des données stockées dans Hadoop,
- Hive permet aux développeurs non familiarisés avec MapReduce d'écrire des requêtes de données qui sont traduites en Jobs de MapReduce dans Hadoop.
- Comme Pig, Hive a été développé en tant que couche d'abstraction, mais s'adressait davantage aux analystes de bases de données plus familiarisés avec SQL que la programmation Java.

**L'écosystème Hadoop contient également plusieurs frameworks pour l'intégration avec le reste de l'entreprise:**

➢ Sqoop

- Est un outil de connectivité permettant de déplacer des données entre des bases de données relationnelles et des entrepôts de données et Hadoop.
- Sqoop exploite la base de données pour décrire le schéma des données importées / exportées et MapReduce pour l'opération de parallélisation et la tolérance aux pannes.

➢ Flume

- Est un service distribué, fiable et hautement disponible pour collecter, regrouper et déplacer efficacement de grandes quantités de données de machines individuelles vers HDFS.
- Il repose sur une architecture simple et flexible, et permet un streaming des flux de données.
- Il exploite un modèle de données extensible simple, vous permettant de déplacer des données de plusieurs machines d'une entreprise vers Hadoop.➢Whirr
- Il s'agit d'un ensemble de bibliothèques qui permet aux utilisateurs de créer facilement des clusters Hadoop par-dessus Amazon EC2, Rackspace ou toute infrastructure virtuelle.

➢ Mahout

- Il s'agit d'une bibliothèque d'apprentissage automatique et d'exploration de données qui fournit des implémentations MapReduce pour les algorithmes populaires utilisés pour le clustering, les tests de régression et la modélisation statistique.

➢ BigTop

- Il s'agit d'un processus et d'un framework formel pour le packaging et les tests d'interopérabilité des sous-projets Hadoop et des composants associés.

➢ Ambari

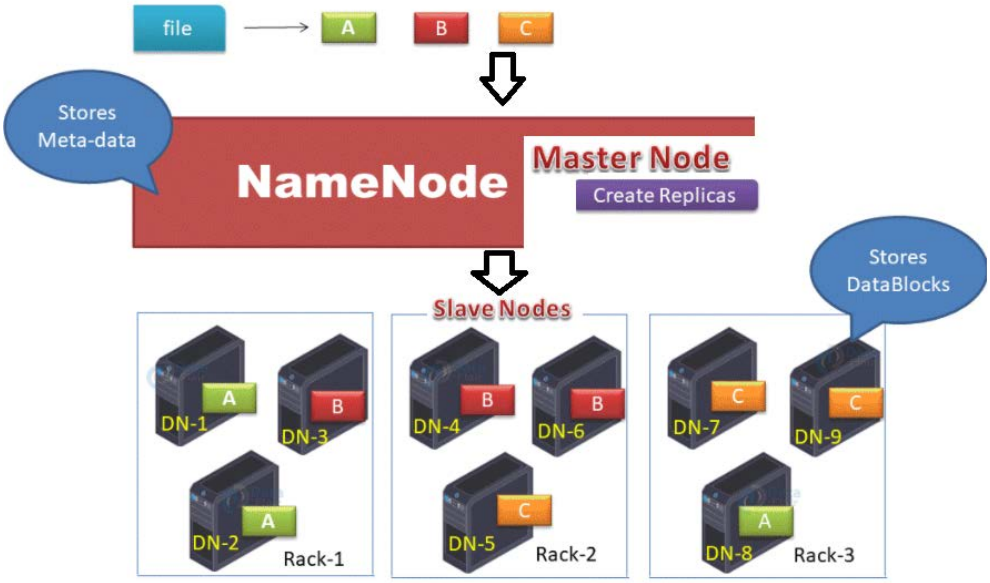
- Il s'agit d'un projet visant à simplifier la gestion Hadoop en fournissant un support pour l'approvisionnement, la gestion et la surveillance des clusters Hadoop.

# HDFS **(Hadoop Distributed File System).**

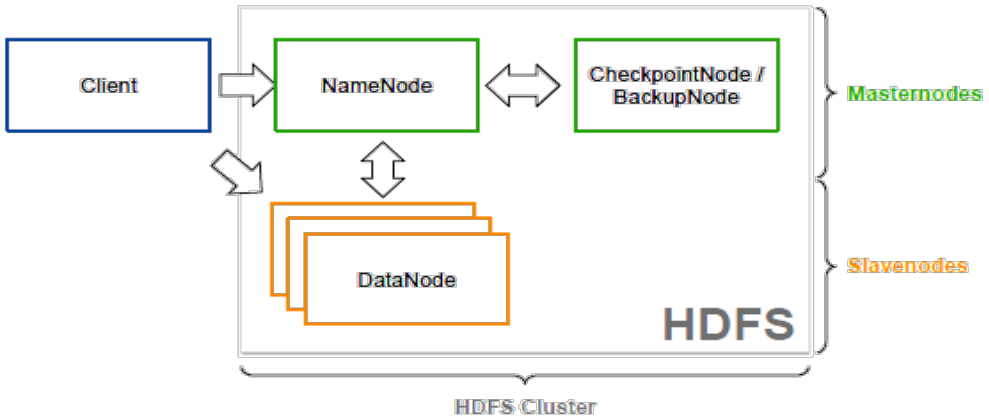
- HDFS est un système de fichiers distribué qui s'exécute sur de grands clusters et fournit un accès à haut débit aux données.
  - Les données sont traitées sur les nœuds sur lesquels elles sont stockées.
- HDFS est un système hautement tolérant aux pannes et conçu pour fonctionner avec du matériel standard.
- L'indépendance des nœuds de cluster les uns aux autres garantit l'évolutivité.
- Caractéristiques de HDFS:
  - ***Stockage évolutif pour les fichiers volumineux***: HDFS a été conçu pour stocker des fichiers volumineux avec la possibilité d'évoluer vers des clusters comprenant des milliers de nœuds.
  - ***Réplication***: HDFS réplique les blocs de données sur plusieurs machines dans un cluster, ce qui rend le système fiable et tolérant aux pannes. La taille de bloc par défaut utilisée est de 64 Mo et le facteur de réplication par défaut est de 3.
  - ***Accès aux flux de données en continu « Streaming Data Access »***: HDFS a été conçu pour diffuser des patterns d'accès aux données et fournit des lectures et des écritures en continu à haut débit.
  - ***Ajout à un fichier « Appends »***: HDFS a été conçu à l'origine pour avoir des fichiers immuables. Les fichiers une fois écrits sur HDFS ne pouvaient pas être modifiés par l'écriture ou par l'ajout au fichier. Les versions récentes de HDFS ont introduit la fonctionnalité d'ajout à un fichier.

# Architecture d'HDFS

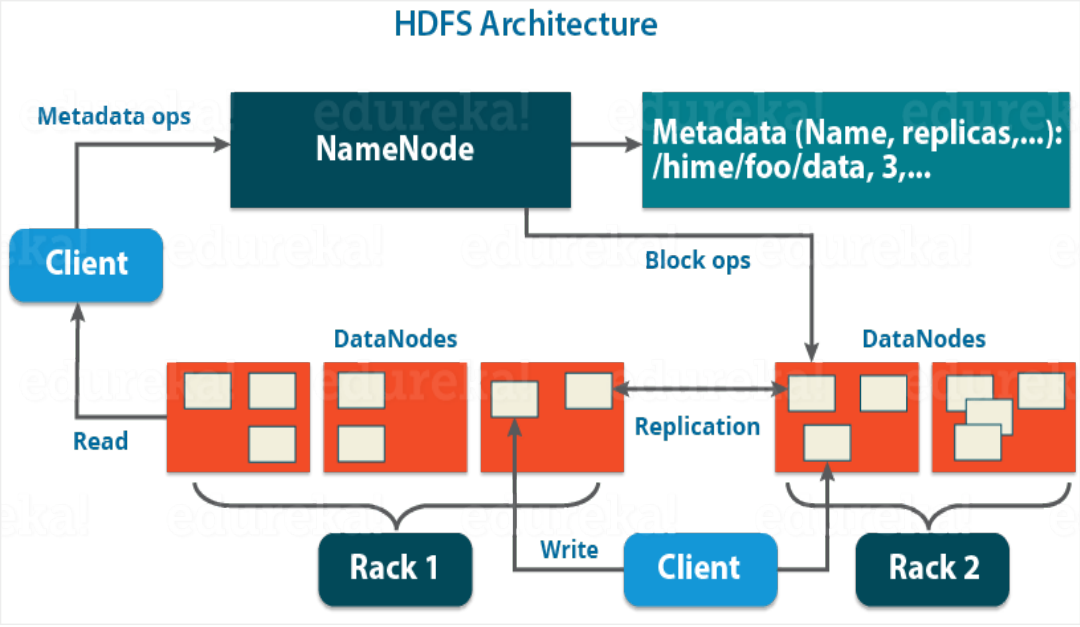
- HDFS est implémenté comme un système de fichiers structuré par **blocs** de taille fixe dans un cluster Hadoop.



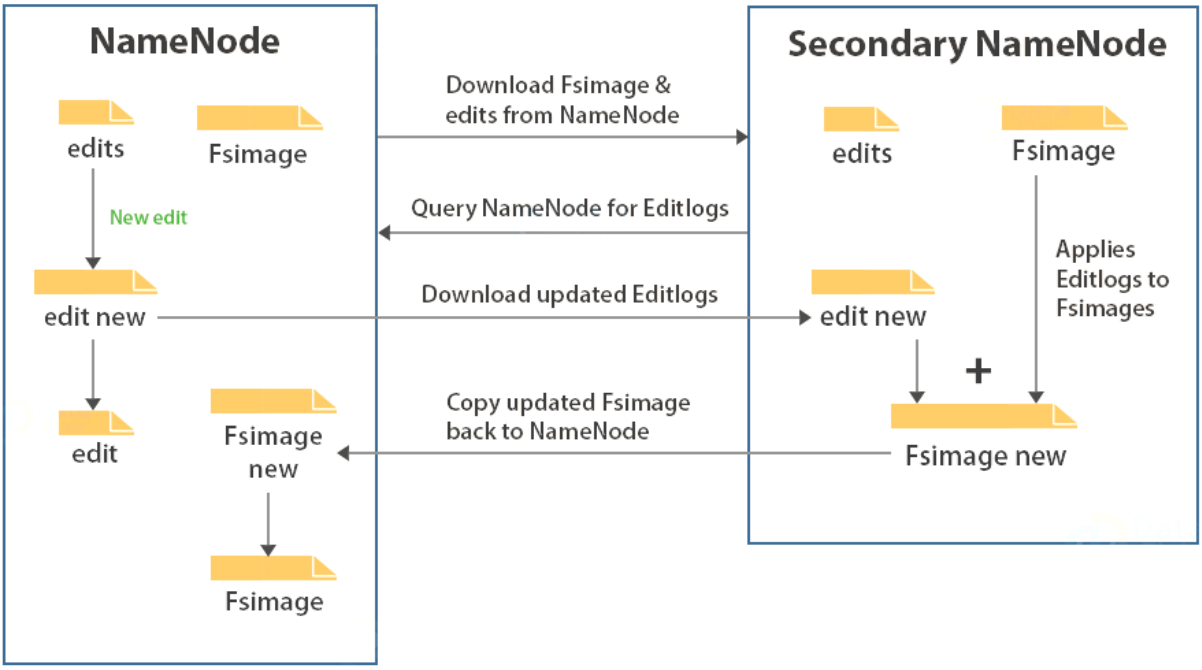

- Les DataNodes sont des démons ou processus esclaves qui s'exécutent sur chaque machine esclave.
- Un fichier peut être composé de plusieurs blocs, qui sont stockés sur différents **DataNodes** (DN). Par conséquent, l'accès à un fichier nécessite l'accès à plusieurs DataNodes.
    - Le DataNode stocke chaque bloc dans un fichier séparé sur son système de fichiers local.
    - Le DataNode ne crée pas tous les fichiers dans le même répertoire. Il utilise une heuristique pour déterminer le nombre optimal de fichiers par répertoire et sous-répertoires de manière appropriée.
- Les DataNodes exécutent les requêtes de lecture et d'écriture de bas niveau à partir des clients du système de fichiers.
- Ils envoient périodiquement des pulsations au NameNode pour signaler la santé globale de HDFS, par défaut, cette fréquence est définie sur 3 secondes.




- L'une des exigences de ce système de fichiers est la gestion et l'accès rapide aux métadonnées de fichier (informations sur les fichiers et les blocs : noms, autorisations, emplacements, etc.).
- Contrairement aux fichiers d'HDFS (qui sont accessibles dans un modèle à écriture unique et à lecture multiple), les métadonnées d'autres fichiers peuvent être modifiées par un grand nombre de clients simultanément. Cependant, il est important que ces informations ne soient jamais désynchronisées.
    - HDFS résout ce problème en introduisant une composante spéciale dédiée, appelée **NameNode**, qui stocke toutes les métadonnées du système de fichiers à travers le cluster. Cela signifie que HDFS implémente une architecture **maître / esclave**. Un seul NameNode (qui est un serveur maître) gère l'espace de noms du système de fichiers et régule l'accès aux fichiers par les clients.



- **NameNode** enregistre les métadonnées de tous les fichiers stockés dans le cluster, par exemple
  - l'emplacement des blocs stockés,
  - la taille des fichiers,
  - les autorisations,
  - la hiérarchie, etc.
- Deux fichiers sont associés aux métadonnées:
  - FsImage: il contient l'état complet de l'espace de noms du système de fichiers depuis le début du NameNode.
  - EditLogs: il contient toutes les modifications récentes apportées au système de fichiers, donc l'état la plus récente de FsImage.
- Il enregistre chaque modification apportée aux métadonnées du système de fichiers. Par exemple,
  - si un fichier est supprimé dans HDFS, le NameNode l'enregistrera immédiatement dans EditLog.
- Il reçoit régulièrement un Heartbeat et un rapport de blocage de tous les DataNodes du cluster pour s'assurer que les DataNodes sont actifs.
- Il garde un enregistrement de tous les blocs dans HDFS et dans quels nœuds ces blocs sont situés.
- Le NameNode est également chargé de prendre en charge le facteur de réplication de tous les blocs. En cas de défaillance du DataNode, le NameNode choisit de nouveaux DataNodes pour les nouvelles répliques, équilibre l'utilisation du disque et gère le trafic de communication vers les DataNodes.
- L'existence d'un seul maître NameNode dans un cluster simplifie grandement l'architecture du système.
- L'architecture maître « NameNode » / esclave « DataNode » de HDFS crée un point de défaillance : perdre le NameNode signifie effectivement perdre HDFS. Pour atténuer quelque peu ce problème, Hadoop implémente un **NameNode secondaire**.

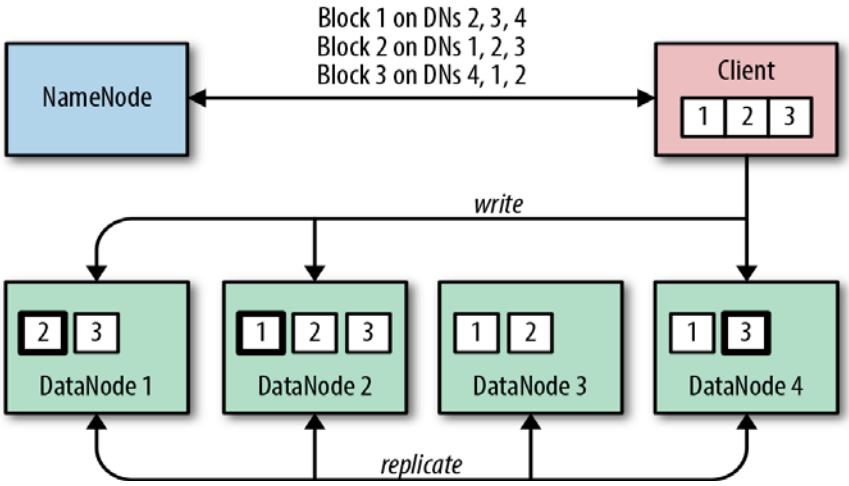



- Le NameNode secondaire lit constamment tous les systèmes de fichiers et métadonnées de la RAM du NameNode et les écrits sur le disque dur ou le système de fichiers.
- Il est responsable de la combinaison des EditLogs avec FsImage du NameNode.
- Il télécharge les EditLogs à partir du NameNode à intervalles réguliers
- La nouvelle FsImage est recopiée dans le NameNode, qui est utilisé chaque fois que le NameNode est (re)démarré ou pour la restauration en cas de panne
- Le NameNode effectue des points de contrôle réguliers dans HDFS. Par conséquent, il est également appelé **CheckpointNode**.
- Pour éviter ce problème, HDFS réplique chaque bloc sur un certain nombre de machines (trois, par défaut).

## Configuration du pipeline

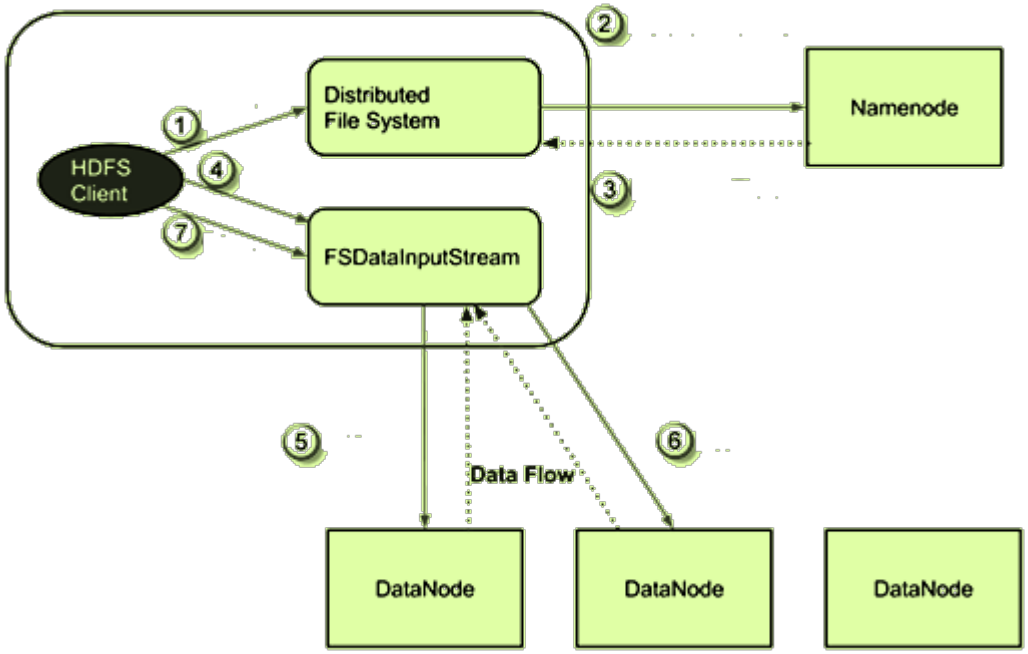
- Avant l''écrire sur les blocs, le client confirme si les DataNodes, présents dans chacune des listes d'adresses IP, sont prêts à recevoir les données ou non. Ce faisant, le client crée un pipeline pour chacun des blocs en connectant les DataNodes individuels dans la liste respective de ce bloc.
- Un pipeline de données est un concept faisant référence aux étapes de transport des données d'une source vers une cible

- La réplication des données dans HDFS est implémentée dans le cadre d'une opération d'écriture sous la forme d'un pipeline de données.
  - Lorsqu'un client écrit des données dans un fichier HDFS, ces données sont d'abord écrites dans un fichier local.
  - Le fichier local accumule un bloc complet de données, le client consulte le NameNode pour obtenir une liste des DataNodes affectés aux réplicas de ce bloc.
  - Le client écrit ensuite le bloc de données dans le premier DataNode.
  - Le DataNode stocke les blocs reçus dans un système de fichiers local et transmet ces données au DataNode suivant dans la liste jusqu'à ce que le dernier DataNode de réplicas reçoive les données.
    - Si l'un des DataNodes échoue pendant que le bloc est en cours d'écriture, il sera supprimé du pipeline. Quand l'opération d'écriture sur le bloc actuel se termine, le NameNode le réplique à nouveau pour compenser le réplica manquant (causé par l'échec du DataNode).
  - Lorsqu'un fichier est fermé, les données restantes du fichier local temporaire sont mises en pipeline vers les DataNodes.
    - Le client informe alors le NameNode que le fichier est fermé.
    - À ce stade, le NameNode valide l'opération de création de fichier.
    - Si le NameNode meurt avant la fermeture du fichier, le fichier est perdu.
  - Chaque DataNode envoie périodiquement un message de pulsation (heartbeat) au NameNode, qui est utilisé par le NameNode pour découvrir les échecs de DataNode (en fonction des pulsations manquantes).
  - Le NameNode marque les DataNodes sans pulsation récents comme morts et ne leur envoie aucune nouvelle demande d'E/S.
  - La mort de DataNode peut faire chuter le facteur de réplication de certains blocs en dessous de leurs valeurs spécifiées.
    - Par conséquent, le NameNode suit en permanence les blocs et lance la réplication chaque fois que nécessaire.
- Exemple de configuration du pipeline



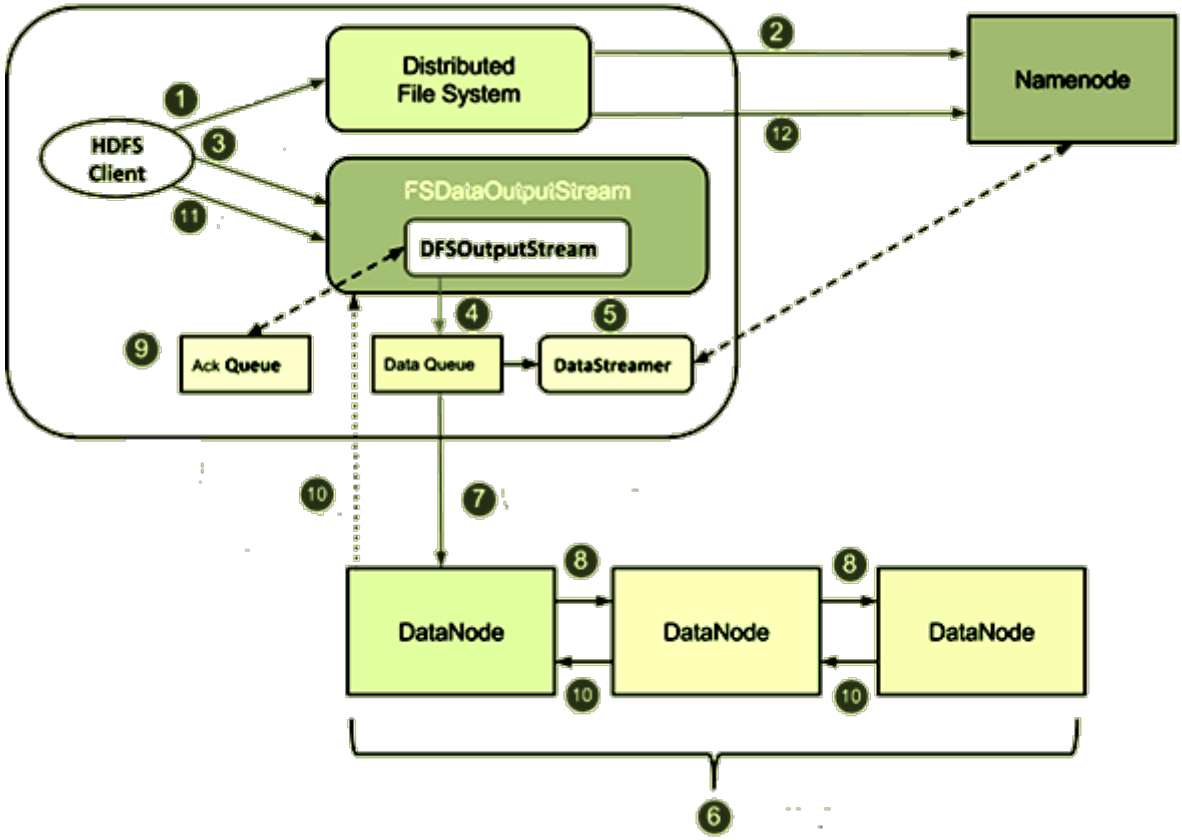

- Le client choisira le premier DataNode dans la liste (IP DataNode pour le bloc 1) qui est DataNode 1 et établira une connexion TCP / IP.
- Le client informera DataNode 1 d'être prêt à recevoir le bloc. Il fournira également les adresses IP des deux prochains DataNodes (2 et 4) au DataNode 1 où le bloc est censé être répliqué.
- Le DataNode 1 se connectera au DataNode 2.
- Le DataNode 1 informera DataNode 2 d'être prêt à recevoir le bloc et lui donnera l'adresse IP du DataNode 4.
- Ensuite, DataNode 2 indiquera au DataNode 4 d'être prêt à recevoir les données.
- Ensuite, la confirmation de disponibilité suivra la séquence inverse, c'est-à-dire du DataNode 4 à 2 puis à 1.
- Enfin, DataNode 1 informera le client que tous les DataNodes sont prêts et qu'un pipeline sera formé entre le client, DataNode 1, 2 et 4.
- La configuration du pipeline est maintenant terminée et le client va enfin commencer le processus de diffusion des données « Data Streaming »

# Opération de lecture dans HDFS




1. Le client lance la demande de lecture en appelant la méthode «open ()» de l'objet FileSystem. C'est un objet de type DistributedFileSystem.
2. Cet objet se connecte à NameNode à l'aide de RPC et obtient des informations de métadonnées telles que les emplacements des blocs du fichier. Veuillez noter que ces adresses sont du premier bloc de fichier.
3. Le Namenode répond avec les emplacements des blocs de données triés par la distance au client. Cela permet de minimiser le trafic entre les nœuds car le client peut lire les blocs à partir du nœud le plus proche. Par exemple, si le client se trouve sur le même nœud qu'un bloc de données, il peut lire le bloc de données localement.
4. Une fois les adresses des DataNodes reçues, un objet de type FSDataInputStream est renvoyé au client. FSDataInputStream contient DFSInputStream qui prend en charge les interactions avec DataNode et NameNode.
5. Les données sont lues sous la forme de flux dans lesquels le client appelle à plusieurs reprises la méthode «read ()». Ce processus d'opération read () continue jusqu'à ce qu'il atteigne la fin du bloc.
6. Une fois la fin du bloc atteinte, DFSInputStream ferme la connexion et passe à localiser le prochain DataNode pour le bloc suivant. Pendant le processus de lecture, si une réplique devient indisponible, le client peut lire une autre réplique sur un autre DataNode
7. Une fois que le client a terminé la lecture, il appelle la méthode close ().

# Opération d'écriture dans HDFS




1. Le processus d'écriture commence lorsque le client envoie une demande au NameNode en appelant la méthode « create () » de l'objet DistributedFileSystem pour créer un nouveau fichier dans l'espace de noms du système de fichiers.
2. Le NameNode vérifie si l'utilisateur dispose des autorisations suffisantes pour créer le fichier et si le fichier existe déjà. Si le fichier existe déjà ou si le client n'a pas l'autorisation suffisante, alors IOException est renvoyée au client. Sinon, l'opération réussit et un nouvel enregistrement pour le fichier est créé par le NameNode.
3. Le client écrit des données dans l'objet FSDataOutputStream qui divise les données en paquets et les met dans une file d'attente de données.
4. FSDataOutputStream contient un objet DFSOutputStream qui s'occupe de la communication avec DataNodes et NameNode. Pendant que le client écrire des données, DFSOutputStream créé des paquets. Ces paquets sont mis en file d'attente dans une file d'attente appelée DataQueue.
5. Les paquets sont consommés à partir de la file d'attente par DataStreamer, qui demande au NameNode d'allouer de nouveaux blocs sur les DataNodes dans lesquels les données doivent être écrites.

6. Désormais, le processus de réplication commence par la création d'un pipeline à l'aide de DataNodes. Dans notre cas, nous avons choisi le niveau de réplication de 3 et il y a donc 3 DataNodes dans le pipeline.
7. Le DataStreamer verse des paquets dans le premier DataNode du pipeline.
8. Chaque DataNode d'un pipeline stocke le paquet qu'il reçoit et le transmet au deuxième DataNode du pipeline.
9. Une autre file d'attente, «Ack Queue», est maintenue par DFSOutputStream pour stocker les paquets qui attendent un accusé de réception de DataNodes.
10. Une fois que l’accusé de réception d’un paquet en file d’attente est reçu de tous les DataNodes du pipeline, il est supprimé de la «Ack Queue». En cas d'échec de DataNode, les paquets de cette file d'attente sont utilisés pour relancer l'opération.
11. Une fois que le client a terminé l'écriture des données, il appelle la méthode close (). Ce qui entraîne le vidage des paquets de données restants vers le pipeline, suivi de l'attente d'un accusé de réception.
12. Après l'accusé de réception final reçu, NameNode est contacté pour lui indiquer que l'opération d'écriture de fichier est terminée.