Tutoriel pour apprendre à faire le choix d'une architecture big data

Par Naoufal BEN BOUAZZA

Date de publication : 14 mars 2017

Avant de commencer il me semble judicieux de préciser que ce cours est une goutte d'eau dans l'océan du « big data ». L'objectif de ce cours est d'aider humblement à comprendre les opportunités et les défis du big data, ainsi que les critères de choix d'une architecture big data selon le cas d'utilisation.

Dans ce cours est proposé un modèle suite à de nombreuses lectures de différentes références que vous trouverez par la suite. Ce modèle aide à aborder les questions communes auxquelles nous sommes confrontés lorsque nous traitons la problématique des choix d'architecture big data.

Commentez

I - Résumé

Aujourd'hui, les entreprises ont des informations provenant de différents canaux pour tous leurs aspects métier. L'utilisation correcte de ces données permet de créer la valeur et d'avoir un avantage concurrentiel. Différentes entreprises ont compris la valeur des données et les bénéfices qu'ils peuvent en tirer, telles que Google, Facebook et Amazon, entre autres. Cependant, d'autres entreprises ont du mal à comprendre ce que signifie le big data pour eux.

Les entreprises sont habituées à utiliser les SGBDR pour stocker des données structurées sur une seule machine, qui prennent en charge des centaines d'utilisateurs simultanés. Les SGBDR assurent les opérations ACID et lorsque les entreprises font face à des problèmes de performance ils utilisent des solutions de scale verticale (voir l'annexe) en achetant du matériel coûteux comme la mémoire, de grands processeurs et cela devient très rapidement pesant financièrement.

Pendant des décennies les SGBDR ont permis de stocker, de servir et de traiter des données. Ils sont également adaptés aux enregistrements transactionnels (OLTP) et de l'analyse (OLAP) pour servir les exigences de la BI (Business Intelligence).

I-A - Comprendre les limites des SGBDR

L'approche SGBDR ne respecte pas les exigences des entreprises du Web 2.0 telles que Google, Amazon, Yahoo, Facebook et LinkedIn. Ces sociétés gèrent une grande quantité de données en raison de l'explosion des sites, des réseaux sociaux et IdO (Internet des objets). Ils travaillent avec différents types de données qui doivent être stockées et traitées. Alors, quand il s'agit de volumétrie importante de données, les propriétés ACID des SGBDR ne sont plus valides, et lors de l'utilisation des solutions de réplication basées sur des architectures maître/esclave, les données sont répliquées de manière asynchrone qui signifie que l'opération d'écriture des données prend du temps pour se propager du maître à l'esclave, causant ainsi la perte de la cohérence des données. Face aux problèmes de performance, dénormaliser le modèle de données semble la solution, mais nous risquons d'obtenir des données dupliquées.

I-B - Pourquoi passer de SGBDR à des solutions big data ?

La plupart des données sont générées par des utilisateurs ou des machines. Ces données générées sont non seulement volumineuses, ont une vélocité importante, mais aussi diversifiées, ce qui traduit le terme «big data». big data impose de nouveaux défis à la manière traditionnelle de traiter les données en utilisant les SGBDR, car ils ne sont pas conçus pour l'échelle de Web comme nous l'avons mentionné auparavant. Un nouveau mouvement de gestion des données est né collectivement appelé « NoSQL » qui est l'abréviation de Not Only SQL. C'est une approche différente des SGBDR qui offre une mise à l'échelle linéaire en suivant le théorème CDP (Cohérence, Disponibilité,tolérance au Partitionnement), et peut stocker du big data.

Les bases de données NoSQL peuvent traiter différents types de données, qui ne nécessitent pas un schéma strict. Le NoSQL a différente catégorisation (Value Key Store, entrepôt orienté colonne, entrepôt orienté document, base de données orientée graphe) pour répondre aux différents besoins de stockage du big data.

NoSQL a introduit de nouveaux défis lorsqu'ils traitent avec la modélisation des données. Chaque catégorie a ses techniques de modélisation, par exemple Apache Cassandra utilise une technique appelée « Modélisation par Query ».

I-B-1 - Pourquoi de nouveaux outils statistiques ?

Le besoin de rapidité dans le traitement des données a conduit à l'élaboration de solutions big data telles que Hadoop. Hadoop a deux composantes principales notamment : HDFS un système de fichiers Hadoop et MapReduce, une API de traitement. Cependant, il existe de nombreux composants qui répondent à des besoins différents, tels Sqoop, Kafka, Flume, HBase, ZooKepeer, Strom, etc.

L'un des sujets les plus importants dans la mouvance big data aujourd'hui est l'architecture big data, en partie parce qu'il y a de nombreux composants open source big data, et les entreprises commencent à s'intéresser à la façon de construire des plateformes big data, en combinant ces nombreux composants. De nouveaux modèles d'architecture ont émergé, comme l'architecture Lambda et l'architecture Kappa pour répondre aux différents besoins des entreprises en termes de traitements big data, et ont contribué à la définition de l'architecture big data.

Depuis 2009, l'architecture Lambda a été inventée pour répondre à la tolérance aux pannes humaines au détriment de la tolérance aux pannes logicielles et matérielles. Cette architecture a trois couches : la couche batch, la couche de vitesse (Speeding) et la couche de service (Serving). Les données entrantes sont à la fois présentes dans la couche batch et la couche de vitesse. Dans la couche batch le concept de base est que les données sont immuables dans le data set, les données ne sont jamais mises à jour dans cette couche, et les nouvelles données sont ajoutées à la fin du fichier. Ensuite, nous créons des vues de ces données qui répondent aux exigences métier. Dans la couche de vitesse, nous traitons avec des flux de données provenant des réseaux sociaux. Dans cette couche, nous pouvons utiliser des frameworks de traitement de flux tels que Stream Spark ou Storm. La nature de l'architecture Lambda est de traiter une petite quantité de données. Dans la couche de service nous fusionnons les données des vues batch et celles des vues de la couche de vitesse. L'architecture Lambda est indépendante des technologies à mettre en œuvre.

Une alternative de l'architecture Lambda est l'architecture Kappa, qui améliore le système de traitement de flux.

Au-delà des différentes architectures et solutions techniques, afin de fournir une valeur ajoutée, nous devons comprendre la gestion du cycle de vie des données. Ce terme est utilisé pour décrire les données de bout en bout (de la collecte de données brutes à la visualisation de données ou d'un produit de données).

La clé pour réussir le choix d'une architecture big data est dans un premier temps de répondre à la question « pourquoi a-t-on besoin d'une solution big data ?». Étudier la valeur métier des données contre les coûts du stockage, le transport et le traitement des données. Car chaque métier a des besoins et des attentes différentes et un contexte particulier, et n'y a pas une solution générique qui permette de traiter les différents besoins métier.

II - Introduction

Après chaque révolution technologique, la question de l'intérêt des nouvelles technologies se pose. Pourtant, quelle qu'en soit la réponse, le résultat est toujours le même : les avancées technologiques ne cesseront jamais d'attirer la société de consommation.

Annoncés comme le nouvel Eldorado depuis le début de cette décennie pour créer de la valeur et acquérir un avantage concurrentiel, les projets « big data » modifient la vie courante des entreprises et commencent à prendre de l'ampleur. D'abord perçu comme un luxe, le « big data » est aujourd'hui vu par la plupart des entreprises comme une nécessité, et se retrouve au cœur même de la stratégie des entreprises comme le confirme l'étude menée par le *Club décision DSI* à travers son dernier baromètre de la transformation digitale, bien qu'il ne soit pas encore totalement en production, afin de créer de la valeur dans différents domaines en valorisant l'information.

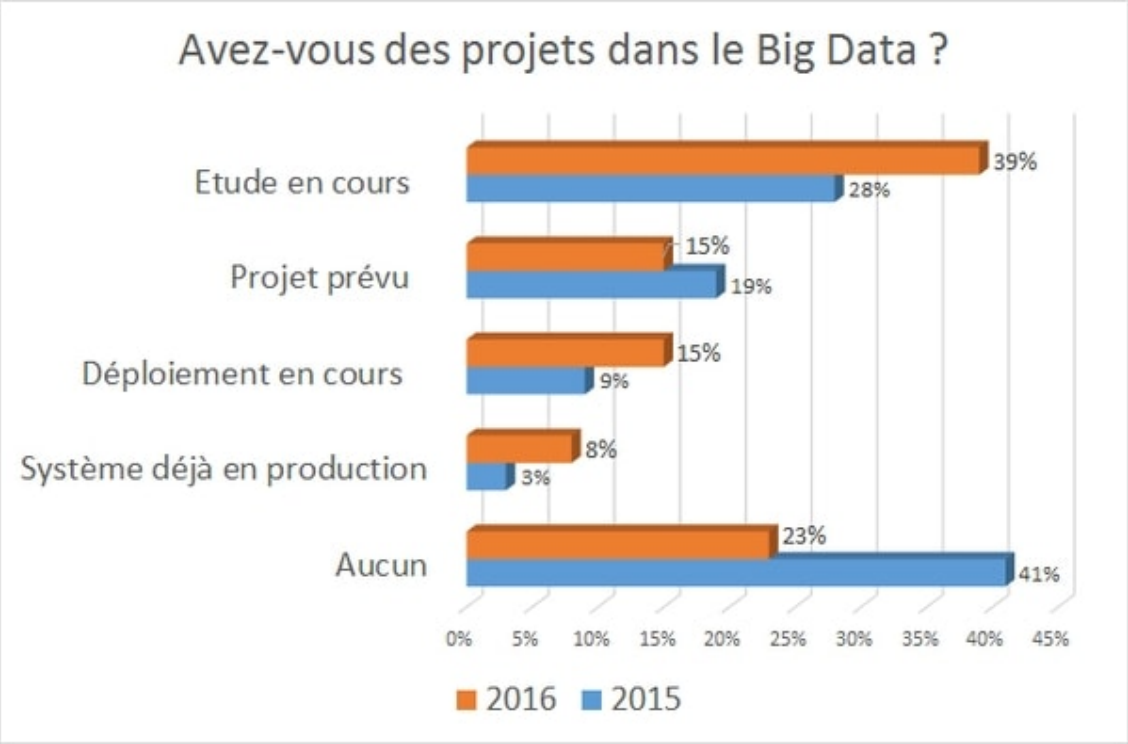


Figure 1 : Indicateur tiré du dernier baromètre de la transformation numérique JDN / Club Décision DSI / IT Research

Nous sommes passés des systèmes de gestion des bases de données relationnelles (SBGDR) qui suivent un modèle rigide pour assurer les besoins de l'informatique de gestion vers un modèle plus flexible celui du web, centré sur un contenu varié (structuré, semi-structuré et non structuré), volumineux et qui arrive à des vitesses différentes.

Avec l'arrivée du Web, de nouveaux besoins sont apparus, tels que la scalabilité à l'époque où on pouvait faire des calculs intensifs sur des grandes quantités de données à l'aide de super calculateur tel que IBM Roadrunner ou des grilles d'ordinateurs et en cas de besoin, il suffisait d'ajouter du disque de stockage, RAM et du CPU pour augmenter la puissance (Scale up). Les besoins d'aujourd'hui sont d'une part d'assurer une grande puissance de stockage, que les SGBDR ne peuvent pas atteindre au moyen du principe de scalabilité horizontale (Scale out) qui consiste à ajouter des machines à faible coût, afin de paralléliser les traitements. D'autre part, assurer une grande puissance de calcul de données très variées qu'on ne peut pas stocker et traiter dans des SGBDR classiques et qui sont de différentes sources telles que les flux d'évènements, les fichiers de logs et les réseaux sociaux.

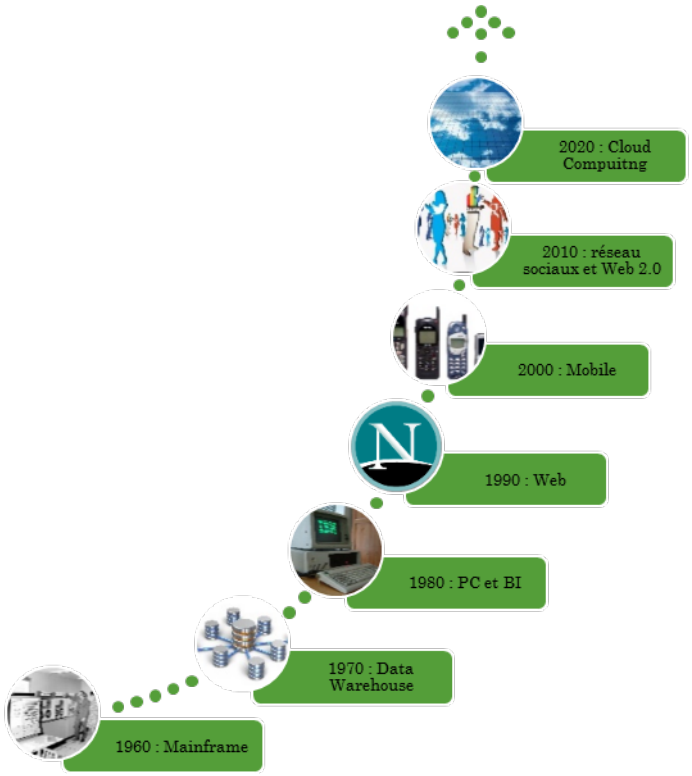


Figure 2 : Évolution de l'informatique - Du mainframe au cloud computing et big data

Ces besoins de scalabilité de stockage et de traitement de gros volumes de données par les grands acteurs du Web a donné naissance au terme « big data ».

Loin du buzz médiatique, le big data a pour perspective un usage maîtrisé qui regroupe les nouvelles technologies de stockage distribué et de traitement parallélisé de données massives (à l'échelle du pétaoctet) souvent non structurées. Tout en profitant des évolutions matérielles qui se traduisent par la baisse des coûts de stockage (disque dur, SSD, RAM, et CPU) pour ajouter des machines bon marché (Commodities hardware en anglais) pour la distribution du stockage et traitement sur plusieurs nœuds, afin d'assurer la scalabilité (montée à l'échelle) à moindre coût, ce qui peut générer un retour sur investissement (ROI) important.

Le big data peut être traité sous différents angles, mais le présent cours a pour but de traiter la problématique de « Comment choisir une architecture big data ? ».

Pour répondre à cette question, nous nous basons principalement sur

- la littérature ;
- les différents apports du web ;
- le retour d'expérience auprès d'architectes qui ont pu mettre en place des POC (Proof Of Concept) ou ayant travaillé avec des technologies big data.

Le cours sera décomposé de la façon suivante pour essayer de répondre humblement à notre problématique, sachant que beaucoup de choses sont susceptibles de changer en fonction des futurs retours d'expériences et en fonction des nouveaux besoins d'usage :

dans la première partie, nous allons voir l'émergence et les enjeux du big data. Nous allons revoir brièvement l'histoire de l'informatique, aborder les limites des bases de données relationnelles, ensuite définir le terme big data, les architectures distribuées, les solutions de stockage distribuées avec le paradigme NoSQL, établir le lien qui existe entre le big data et la Business Intelligence (B.I.) et conclure avec les enjeux du big data ;

dans la deuxième partie, nous aborderons les architectures et les solutions big data. Nous allons commencer par expliquer les notions conceptuelles du big data notamment le modèle de conception MapReduce, traitement temps réel et traitement par lot, puis nous allons voir les architectures big data les plus connues notamment l'architecture Lambda et l'architecture Kappa, présenter les solutions big data telles que Hadoop et Spark et finir avec une liste non exhaustive des différentes distributions de Hadoop ;

dans la troisième partie, nous allons aborder le workflow du big data avant de voir notre modeste contribution afin de faire le choix d'une architecture big data en suivant une méthodologie bien précise, puis enfin nous allons conclure sur le choix d'une architecture big data. Nous allons conclure ce cours par un bref rappel des différentes parties, une présentation des difficultés rencontrées lors de sa rédaction et ses perspectives.

III - Émergence et enjeux du big data

III-A - Comment est-on arrivé là ?

Très souvent les données de l'entreprise proviennent des **PGI** (Progiciels de données de gestion), **GRC** (gestion de relation client), commerce électronique (**E-Commerce**), **GCL** (gestion de la chaîne logistique), des entrepôts de données (Data Warehouse en anglais) et des bases de données issues d'un développement applicatif interne. La majorité de ces données sont souvent stockées dans un système de gestion de bases de données relationnelles (**SGBDR**).

Les SGBDR sont toujours au cœur du système d'information des entreprises et les investissements lourds de ces dernières dans ces solutions de stockage rendent leurs remplacements problématiques, et toutes les tentatives ont échoué (à titre d'exemple les bases orientées objet), devant leurs puissances, parce qu'ils ont d'abord de très nombreux avantages que nous allons expliquer dans la partie « limitations des bases de données relationnelles », de plus ils sont devenus un référentiel et un mécanisme d'intégration pour la majorité des DSI.

Aujourd'hui, avec le Web 2.0, nous sommes confrontés à une croissance incomparable de la quantité de données que nous traitons, et qui augmente très rapidement comme on peut le voir sur des sites comme : **http://www.internetlivestats.com/**. Le volume des données atteindra certainement l'exabytes ou yottabytes, comme le montre la figure ci-dessous.

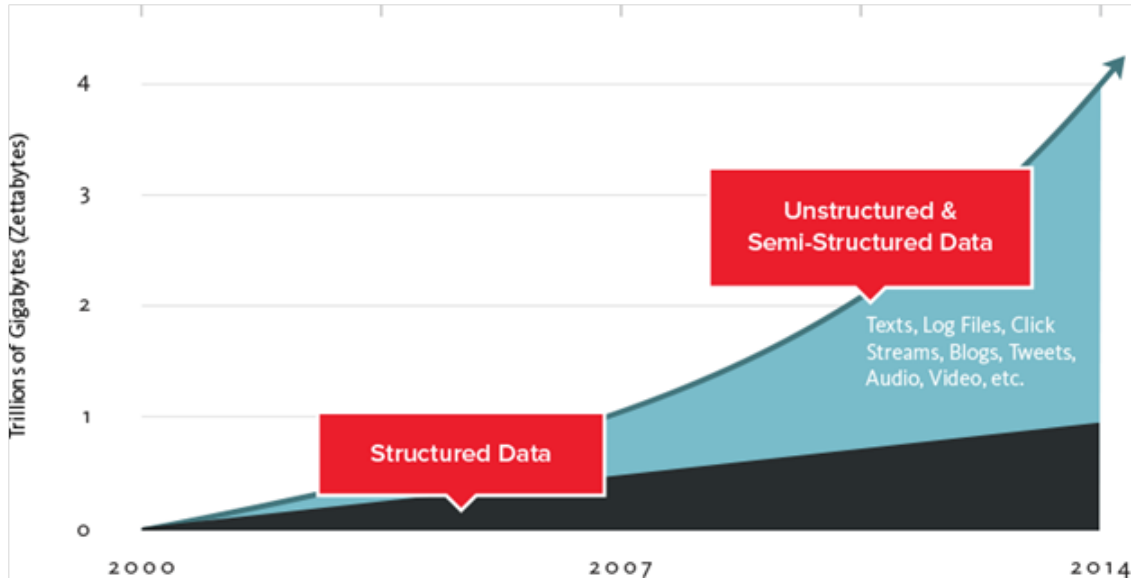


Figure 3 : Quantité et structure des donnéesen 2014

Source : **http://air.imag.fr/index.php/EA2014-NoSQL**

Les sources des données sont des systèmes d'information traditionnels comme on l'a vu, qui sont jusqu'à présent bien gérés par les entreprises. Les nouveaux arrivants sont les réseaux sociaux (Facebook, Twitter, Google +, blogs) où les utilisateurs génèrent d'énormes flux de données et d'évènements semi-structurés, qu'on ne peut pas stocker dans un SGBDR classique lorsqu'on a des besoins opérationnels avec une très faible latence, et enfin les données des objets connectés qui envoient de plus en plus d'informations en flux continu et qu'on doit pouvoir stocker et traiter.

Nous allons voir pourquoi les SGBDR ne sont pas adaptés à ces nouveaux besoins.

III-B - Limitation des bases de données relationnelles

On va rappeler brièvement les caractéristiques des SGBDR classiques, afin de voir les challenges qu'ils posent lorsqu'on a des données massives dans le contexte d'applications web à grande échelle.

Les SGBDR reposent sur des fondements mathématiques élaborés par Edgar Frank Codd (1923-2003), qui a publié un papier intitulé « *A relational Model of Data for Large Shared Data Banks* », ces théorèmes sont appelés l'**algèbre relationnelle,** qui sont principalement des opérations comme la sélection, la projection, l'agrégation, l'union, la jointure, etc., et définissent des règles de normalisation qui ont pour but d'obtenir un modèle performant et sûr.

Les SGBDR traditionnels assurent, les points suivants :

- la cohérence des données à travers le schéma des bases de données, qui est établie à la phase de la conception, avec le choix des meilleurs types de données possibles, ce qui évite le risque de changer le schéma relationnel ;
- les contraintes d'intégrité référentielle, un gage de cohérence du contenu de la base de données, car une fois les contraintes déclarées, aucune violation de ces règles ne sera permise pas le SGBDR ;
- un mode de fonctionnement transactionnel avec la gestion des transactions caractérisé par l'acronyme ACID (Atomicité, Cohérence, Isolation, Durabilité), qui garantit que soit toutes les opérations d'une transaction sont effectuées avec succès, soit aucune opération n'est effectuée.
- bien que le SGBDR soit basé sur l'algèbre relationnelle, l'utilisateur utilise un **langage déclaratif SQL,** qui exécute rapidement des requêtes SQL complexes avec une optimisation automatique des plans d'exécutions, sur des volumes de données raisonnables**.**

Lorsqu'on a de grands volumes de données, des solutions peuvent être envisagées :

- la montée en charge verticale (Scale-up) reste possible, pour augmenter l'espace de stockage et de traitement de ces volumes au sein de la même machine ;
- la montée en charge horizontale (Scale-out), qui consiste à distribuer les SGBDR ;
- les administrateurs de bases de données optimisent les performances à travers la dénormalisation, afin d'éviter les jointures, ou génèrent des tables agrégeant un certain nombre de données, afin d'augmenter les performances.

Malgré ces solutions les SGBDR rencontrent certains challenges techniques et/ou financiers qui sont :

- les coûts de la montée en charge verticale ;
- la montée en charge horizontale limitée à 10 nœuds, pour des contraintes techniques ;
- les coûts en ressources humaines pour mettre en place un tel matériel et assurer son bon fonctionnement, ce qui implique plus de coûts pour héberger plus de données ;
- les bases de données relationnelles ne sont pas adaptées pour les traitements temps réel ;
- les prix des licences de logiciels comme Oracle s'ajoutant au prix des machines spécifiques.

Pour résumer les contraintes des moteurs relationnels sont trop lourds, dans le contexte big data.

III-C - Définition du terme big data

Le big data fait référence à l'explosion du volume de données informatiques qui transitent dans le monde. Ces données deviennent tellement volumineuses qu'elles ne peuvent plus être stockées dans des bases de données de tailles normales, ce qui mène à l'utilisation d'outils informatiques de plus en plus performants pour les gérer. Le big data peut être défini par ses trois (voire quatre) caractéristiques majeures, toutes commençant par la lettre V : le Volume, la Vélocité, la Variété, auxquelles peut s'ajouter la Véracité des données.

III-C-1 - Volume

La première caractéristique majeure du big data est l'explosion du Volume de données.

Ce développement exponentiel oblige les entreprises à changer leurs outils classiques de stockage de données. Pour avoir une idée de l'ampleur du phénomène, le magazine *Fortune* affirme qu'en 2013, nous pouvions générer en dix minutes la même quantité de données qui a été générée depuis le début de l'ère informatique jusqu'en 2003 (5 Mds de GB).

Les avancées technologiques telles que l'avènement des objets connectés, des smartphones et la multiplication des infrastructures d'échanges de données permettent en partie d'expliquer ce phénomène. Certaines entreprises de télécommunications voient leurs coûts de stockage de données augmenter significativement au fil des années, et ce malgré un coût unitaire de stockage en baisse.

Aussi, aujourd'hui, la multiplication de l'accès à internet partout dans le monde, l'automatisation des systèmes (administratifs entre autres), ainsi que la tendance grandissante du recours aux outils informatiques dans la société sont des facteurs d'accélération du développement du big data.

III-C-2 - Vélocité

La deuxième caractéristique majeure du big data est le traitement dynamique de données. Avec l'avènement du big data, il est maintenant possible de traiter les données en temps réel. En effet, le progrès au niveau des infrastructures d'échange de données permet aujourd'hui d'éviter de passer par la méthode désuète consistant à stocker d'abord les données, avant de les afficher au moyen d'outils de reporting.

La vélocité du big data est une caractéristique que l'on retrouve notamment dans le phénomène d'enchère en temps réel, ou *Real-Time Bidding (RTB)*, technologie permettant de mettre aux enchères un emplacement publicitaire ciblé. Les critères sont principalement le profil de l'internaute qui va sur le site (qu'a-t-il vu avant, a-t-il cliqué, etc.), ainsi que les caractéristiques de la page (emplacement, thématique, dimensions, etc.). Voyons l'exemple de l'entreprise *Turn*, plus grande entreprise indépendante du secteur de la publicité sur les vidéos, les mobiles et la télévision.

Turn exécute, en seulement quelques millisecondes, un système affichant la publicité de l'annonceur qui a fait la meilleure enchère pour le segment où l'internaute s'étant connecté au site est classé. Cet utilisateur est au préalable classé dans ce segment suivant son historique de navigation et des informations tirées des réseaux sociaux.

III-C-3 - Variété

Si le big data est aussi répandu aujourd'hui, il le doit à sa troisième caractéristique fondamentale, la Variété. Cette variété, c'est celle des contenus et des sources des données. Les données étant le plus souvent reçues de façon hétérogène et non structurée, elles doivent être traitées et catégorisées avant d'être analysées et utilisées dans la prise de décision.

En analysant les données et en les catégorisant selon leur type, les entreprises peuvent mieux connaître leurs clients et leurs besoins. Cela peut concerner plusieurs types de données : données de localisation, données démographiques, données relatives aux réseaux sociaux, historiques de navigation, achats en ligne, et. Ainsi, chaque

utilisateur a un profil avec des caractéristiques qui lui sont propres et qui permettent aux entreprises de développer des stratégies commerciales bien ciblées.

III-C-4 - Véracité

Un quatrième aspect peut être ajouté aux trois caractéristiques précédentes, il s'agit de la Véracité des données. Il est nécessaire de vérifier l'exactitude des données reçues. Même si elles paraissent homogènes et cohérentes entre elles, ces données peuvent être incomplètes et/ou inexactes. La véracité répond à : « Est-ce qu'on peut faire confiance à la donnée qu'on a ? Contient-elle suffisamment d'information ? »

III-D - Les architectures distribuées

On a vu que la quantité des données dépasse ce qu'on peut stocker sur une seule machine. D'où le besoin de répartir le stockage sur différentes machines, pour cela il existe deux types d'architectures distribuées, qui sont au cœur du big data et du NoSQL, afin de distribuer et répartir les données et les traitements le plus efficacement possible. Ces solutions sont la distribution avec maître ou sans maître.

- Architecture maître/esclave

Une architecture distribuée avec maître repose sur l'existence d'une machine « maître » qui conserve la configuration du système et reçoit les requêtes des clients puis les distribue vers la machine contenant la donnée. Ce modèle peut souffrir de la présence « Point unique de Défaillance » lorsqu'un des éléments n'est pas redondant.

- Architecture sans maître

Dans une architecture distribuée sans maître, toutes les machines ont la même importance et ont les mêmes capacités dans un cluster. Elle repose sur une table de hachage distribuée DHT, qui permet de répartir le stockage sur tous les nœuds du réseau.

III-E - Les bases de données NoSQL

Avec l'apparition du web 2.0 et des réseaux sociaux, outre le problème de stockage de données et leur gestion par des SGBDR, les contraintes sont devenues de plus en plus importantes et les solutions évoquées de montée en charge en particulier le scale-out et le clustering ont permis la naissance d'une nouvelle famille de moteur de données dite : Not Only SQL ou « NoSQL » (terme apparu pour la première fois en 2009), pour contourner les limites des bases de données relationnelles.

À l'origine de cette nouvelle famille de moteur de données, ce sont les leaders web qui ont commencé à publier un certain nombre de papiers sur le sujet, pour expliquer comment ils ont résolu les problèmes de stockage de données volumineux, par exemple :

Google a expliqué comment elle stocke des données de Google Calender, YouTube, les e-mails, etc., en s'appuyant sur des technologies solides et puissantes qui permettent une rapide montée en charge, et de supporter des volumes importants de données, avec son système de fichiers distribués et ses algorithmes de traitements distribués, afin de réduire les temps de réponse de son moteur de recherche et le stockage des données structurées dans sa base de données distribuée BigTable ;

Amazon a publié des papiers sur Dynamo qui est un système de bases de données distribué, qui permet de gérer son panier d'achats.

Le mode de stockage NoSQL, permet de gérer de gros volumes de données qui peuvent être non structurées dans des serveurs de stockage distribués qui sont capables de monter en charge à moindre coût.

Ces nouvelles bases abandonnent la notion de durabilité, comme le cas des bases de données en mémoire vive, d'autres abandonnent la notion de transaction pour laisser place aux caractéristiques du **théorème CDP** (**C**ohérence, **D**isponibilité, tolérance au **P**artitionnement) établies par Éric Brewer en 2000, auxquelles répondent les moteurs NoSQL.

Voici les caractéristiques du théorème CDP :

- **Cohérence** : la même version des données sur tous les nœuds ;
- **Disponibilité** : les données sont accessibles à tout instant, et une requête reçoit une réponse ;
- **Tolérance au Partitionnement** (pannes) : le système doit pouvoir continuer à fonctionner et à répondre, même si un problème survient sur un nœud du cluster.

Impact du théorème **CDP** :

Dans un système à état partagé en réseau, seules deux des trois propriétés peuvent être satisfaites comme le montre le schéma, ce qui a un impact sur les choix du moteur NoSQL, pour répondre à des problèmes de montée en charge en fonction de la complexité des données. Toutefois on peut combiner différents moteurs NoSQL c'est ce qu'on appelle « persistance polyglotte » qu'on va détailler plus loin à travers un exemple.

Sachant que dans un contexte big data la « scalabilité » est le facteur le plus important, on ne parle plus d'un système consistant, mais « éventuellement consistant » lors du choix d'une base NoSQL, et qui fait partie du concept « **BASE** », qui veut dire Basically Available, Soft state, Eventual Consistant.

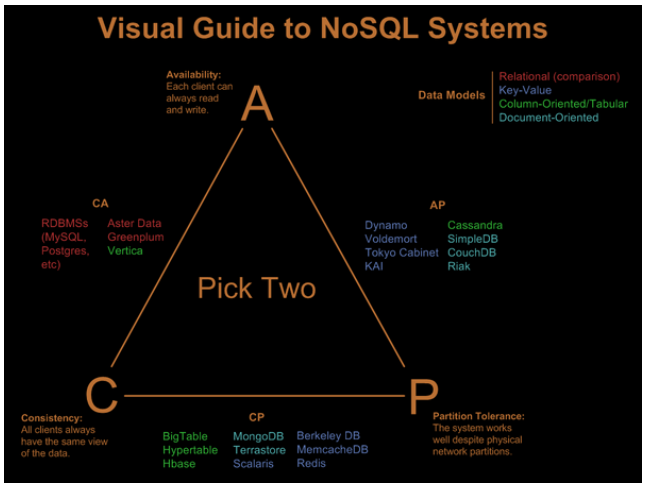


Figure 4 : Théorème CDP et classifications des moteurs NoSQL

Source : **http://air.imag.fr/index.php/EA2014-NoSQLC**.

Contrairement au SGBDR, les schémas des données ne sont pas obligatoires pour les moteurs NoSQL, ils utilisent une approche dite : « sans schéma » ou schéma « relaxé », et offrent plusieurs formats et modèles de stockage là où on travaille avec des tables contenant des enregistrements sous forme de lignes. Dans les moteurs NoSQL on peut stocker les données sous forme de base orientée agrégat : clé/valeur dans des entrepôts dites : entrepôts clé/valeur qui supportent la montée en charge, document (JSON, XML…) dans des bases dites : bases orientées documents, colonnes dans des bases dites : bases orientées colonnes qui supportent les données complexes, ou graphe dans des bases dites : bases orientées graphes qui fournissent un modèle de données très complexe.

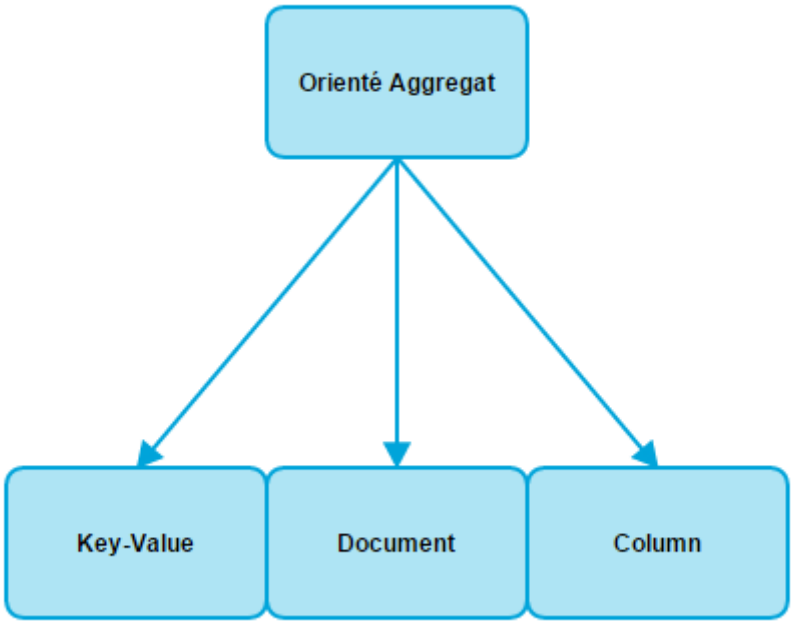
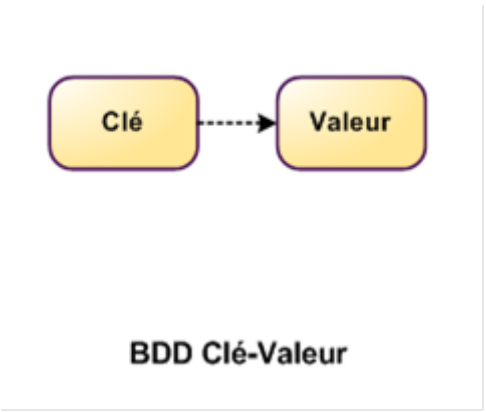


Figure 5 : Stockage orienté agrégats

Maintenant nous allons détailler les trois modes de stockage NoSQL orientés agrégats :

Les bases orientées graphes, ne seront pas détaillées ici, vu qu'elles ne sont pas adaptées pour une utilisation big data.

III-E-1 - Entrepôt clé/Valeur - ECV



C'est le type de stockage le plus simple qui est un ensemble formé de deux données liées entre elles comme le montre le schéma, par une clé unique qui représente une information atomique et la valeur qui contient les données qui peuvent être complexes et représentées sous forme de liste, tableau ou d'autres clés qui pointent vers d'autres valeurs, et contiennent aussi plusieurs métadonnées.

Ce type de stockage peut être en mémoire ou sur disque dur, et le seul moyen d'accès est par la clé de la ligne qui est indexée.

Ils représentent l'avantage d'être très facilement distribués au sein d'un cluster, ainsi que de meilleures performances en écriture et lecture à travers un dictionnaire. De plus, ces structures de données sont très utilisées par les développeurs.

III-E-1-a - Panorama des solutions ECV

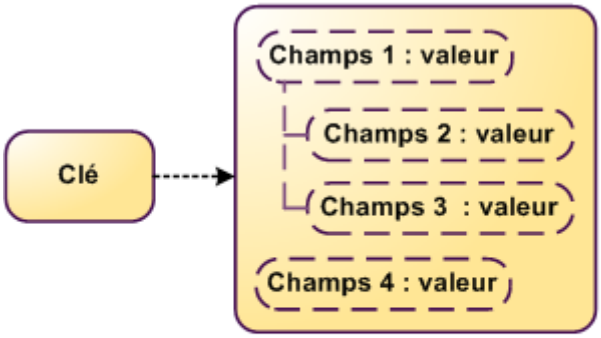
La plupart des solutions se basent sur le papier **d'Amazon Dynamo** :

- **Voldemort** : initié en 2008 au sein de la société Linkedin, peut être stocké en mémoire et sur disque, extensible à la fois par réplication horizontale et verticale, favorise la disponibilité et la tolérance au morcellement ;
- **Riak** : initié en 2008, par la société Basho, pas de point unique de défiance (SPOF), favorise la cohérence et la tolérance au morcellement ;
- **Amazon DynamoDB** : base de données en tant que service BDaaS fournie par Amazon AWS.

D'autres solutions se basent sur **memcached** conçue par Brad Fizpatrick :

- **Redis** initié en 2009 par Salavador Sanfilippo, peut être stocké en mémoire et sur disque ;
- **OrientDB** initié en 2010 par Luca Garulli, est une base multi-model qui peut être configurée pour supporter la cohérence et la tolérance au morcellement, ainsi que la disponibilité selon la configuration du cluster.

III-E-2 - Base orientée document



BDD Orientée document

Les bases de données orientées document sont semblables aux ECV, mais avec des valeurs qui sont semi-structurées (format JSON, XML), appelées documents comme le montre le schéma, et qui peuvent être hétérogènes.

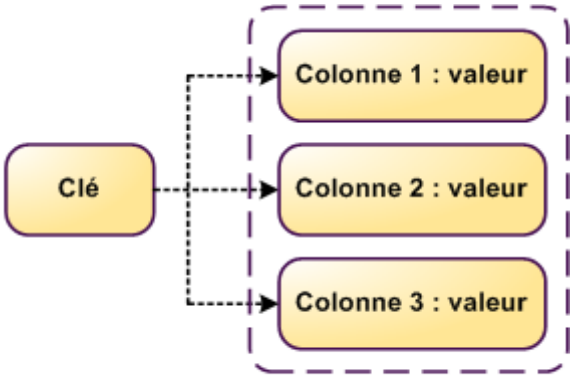
Les bases orientées document offrent des fonctionnalités avancées, comme la sélection et la modification de la valeur d'un champ à l'intérieur d'un document, le requêtage et l'indexation de certains champs, ce qui réduit les développements niveau client.

Ce modèle est très utilisé dans le développement des applications web, de par leur facilité d'accès et de versionning qui est très utile dans le cas de modification des structures des documents.

III-E-2-a - Panorama des bases orientées documents

- **MongoDB** : initié en 2007 par la société 10gen pour créer une Plateforme (PaaS), favorise la cohérence et la tolérance au morcellement.
- **CouchDB** : initié en 2005 par Damien Katz pour la mise en place d'un système de stockage dédié au site web, favorise la disponibilité et la tolérance au morcellement.
- **RavenDB** :
- **OrientDB** : initié en 2010 par Luca Garulli, est une base multi-model.
- **DocumentDB**

III-E-3 - Base orientée colonnes



BDD Orientée colonnes

Un autre modèle d'ECV, à l'origine conçu par Google pour BigTable qui est hautement distribué et qui gère des données tabulaires et stocke les données par colonnes et non par lignes. Les tables de données ne sont pas reliées (pas de jointures) entre elles comme dans les SGBDR, favorisant ainsi la dénormalisation au profit de la performance, très utile lorsqu'on a une grosse volumétrie de données avec beaucoup d'opérations d'écriture/lecture.

Les bases orientées colonnes ont des tables qui peuvent contenir des structures de données complexes. Ces tables sont constituées d'une clé unique qui référencie un enregistrement, et d'un groupe de colonnes, ou super-colonnes qui peuvent être dynamiques contrairement à des colonnes de SGBDR comme le montre le schéma.

Le choix de la clé est crucial pour avoir de bonne performance, puisqu'on n'a pas des index secondaires.

Grâce à leurs modèles de stockage par colonnes, elles favorisent l'enrichissement du schéma initial sans avoir un impact.

III-E-3-a - Panorama des bases orientées colonnes

- **Cassandra** : initiée et développée par Facebook en 2008, pour son système de mail puis devenu un projet de la fondation Apache, Cassandra est très utile pour le traitement temps réel et favorise la disponibilité et la tolérance au morcellement.
- **Cas d'utilisation d'Apache Cassandra**
- **HBase** : initiée en 2007 par la société Powerset, qui est un clone open source de BigTable, qui utilise le système de fichier **HDFS** de **Hadoop**, **ZooKeeper** pour la coordination et a un accès direct (aléatoire) à la donnée favorisant la cohérence et la tolérance au morcellement, avec une latence inférieure à la seconde ce qui le rend utile pour les traitements temps réel.

Le tableau comparatif suivant montre d'une manière synthétique les différents moteurs NoSQL et leurs caractéristiques sans être exhaustif.

	Format de stockage	Transactions	Relation	Haute disponibilité	Fonctions avancées
HBase	Colonnes	Non	Oui	Cluster	
Cassandra	Colonnes	Non	Oui	Cluster	Réplication avancée
Redis	Clé/Valeur	Non	Non	Cluster	Communication asynchrone
Riak	Clé/Valeur, Série temporelle			Cluster	Tolérance aux pannes
MongoDB	Document JSON	Non	Non	Cluster	Agrégation
OrientDB	Multi-Format	Oui	Oui	Cluster	Transactions et sécurité
Neo4j	Graphe	Oui	Oui	Cluster	Intégration avec d'autres solutions NoSQL

D'après le schéma suivant, on voit que les bases orientées colonnes, comme Apache Cassandra, Apache HBase, Apache Accumulo et OrientDB sont les plus utilisées grâce à leur flexibilité et leur rapprochement du modèle SGBDR, ainsi que leur capacité à stocker d'énormes quantités des données.

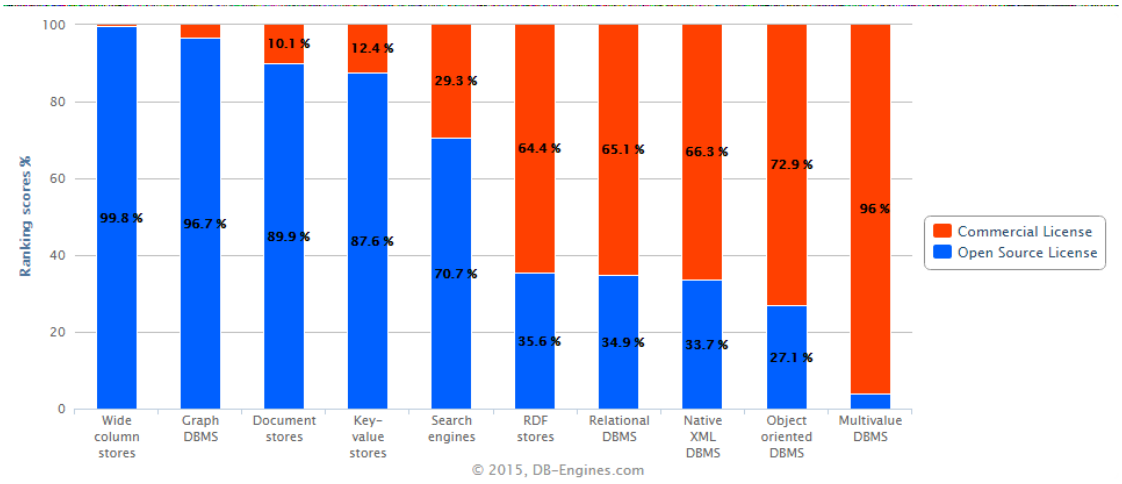


Figure 6 : Classement d'utilisation des différentes bases de données

Source : **http://air.imag.fr/index.php/EA2014-NoSQL**

À titre d'information aujourd'hui il n'existe pas moins de 255 moteurs NoSQL d'après le site : **http://nosql-database.org/** qui ont tous été conçus pour stocker de gros volumes de données, donc le choix dépendra forcément des besoins métier.

Toujours par rapport au besoin métier, on peut mettre en place une persistance polyglotte, comme le montre bien le schéma suivant. Une base orientée clé/valeur en mémoire pour la gestion des sessions et la gestion du panier d'achats, un SGBDR pour les données financières, et le reporting, une base orientée document pour la gestion du catalogue des produits, une base orientée colonnes pour l'analytique et la gestion des logs des utilisateurs et enfin une base orientée graphe pour la gestion du système de recommandation :

Figure 7 : Exemple d'utilisation de la persistance polyglotte

Source : **http://datadventures.ghost.io/2014/07/06/polyglot-processing/**

III-F - big data et Business Intelligence

Lorsqu'il s'agit des besoins statistiques, d'analyse d'historique, d'analyse d'activité ou de tableau de bord de suivi pour connaître les tendances de ventes par exemple selon différents paramètres, le modèle relationnel ne permet pas de répondre à ce besoin.

« *Le modèle relationnel est performant pour une utilisation purement transactionnelle, ce qu'on appelle OLTP (On Line Transactional Processing)* » (Bruchez, 2015)

OLTP est destinée à recevoir des opérations d'écriture et lecture très fréquentes, sur des volumes de données qui peuvent être importants, grâce à une bonne modélisation du schéma des données et l'utilisation des index, afin d'optimiser ces opérations.

Pour répondre à ces besoins les entreprises se basent sur des solutions très maîtrisées, en particulier les entrepôts de données (DW pour Data Warehouse) qui étaient une discipline très populaire, depuis les années 1970, qui a aidé les entreprises à transformer les données en informations, en utilisant OLAP (On Line Analytical Processing) qui est généralement en lecture seule, cette analyse est ce qu'on appelle « Business Intelligence ou BI », et qui est apparue début 1980.

« *DW est l'abréviation de Data Warehouse, ce qui signifie que la base de données est à fort volume, et a été spécialement préparée pour l'analyse, avec des tables de faits et des tables de dimensions.* » (GOUIGOUX, 2014)

L'objectif principal des entrepôts des données est d'intégrer les données à partir de différents systèmes et différentes applications, puis les restituer sous différentes formes, à travers « la chaîne de traitement » comme le montre la figure suivante, et très souvent, pour un métier particulier, afin de faciliter l'analyse pour la prise de décision stratégique pour l'entreprise, nous ne nous arrêtons pas au niveau des entrepôts des données (DW), mais on utilise des magasins de données (Data marts en anglais) qui sont alimentés par un sous-ensemble de ces derniers.

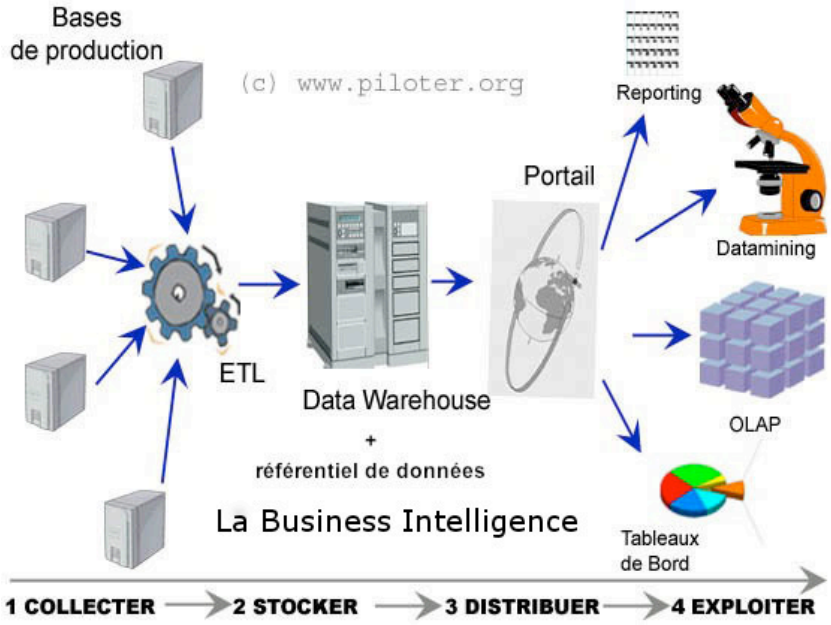


Figure 8 : Chaîne de traitement en Business Intelligence

Source : **http://www.piloter.org/business-intelligence/business-intelligence.htm**

Cette chaîne de traitement est un instrument **d'aide à la décision** pour des besoins transversaux, qui consiste à extraire des informations des bases de données de l'entreprise, puis construire des entrepôts de données afin de modéliser l'information à **des fins d'analyse**.

L'approche BI se base sur différentes approches OLAP :

- des bases de données relationnelles pour construire les entrepôts de données, c'est ce qu'on appelle ROLAP abréviation de « Relational OLAP » qui n'a pas de limite de quantité des données au détriment de la performance ;
- des cubes pour construire des entrepôts de données, c'est ce qu'on appelle MOLAP abréviation de « Multi OLAP », utilisé sur des tailles de données limitées et qui ne changent pas ;
- la dernière approche est une approche Hybride HOLAP, qui est un compromis entre les deux solutions précédentes ;
- DOLAP pour Desktop OLAP, consiste à récupérer en local une partie d'une base de données multidimensionnelle ;
- SOLAP pour Spatial OLAP.

Historiquement la **B.I.** est concentrée sur **:**

- la consolidation financière ;
- la planification budgétaire.

Au fil du temps elle s'est étendue à différents autres métiers :

- la gestion de la relation client-GRC ;
- la gestion de la chaîne logistique-GCL ;
- les ressources humaines-RH ;
- la fabrication.

Le besoin de l'entreprise aujourd'hui n'est pas uniquement de faire le diagnostic de ses propres données comme le montre la figure suivante, mais également de croiser des données provenant de différentes sources (réseaux sociaux, open data...), afin d'anticiper des évènements par l'analyse prédictive.

Un exemple de l'analyse prédictive serait : l'entreprise aimerait savoir quels facteurs influencent les ventes. Si nous réduisons le prix, combien allons-nous vendre de plus ? Si nous faisons de la publicité, quel serait notre profit ? Est-ce que la saison est importante pour ce que nous vendons ? Est-ce que nous vendons bien les jours fériés ? La réponse à ces questions peut aider l'entreprise à faire du profit.

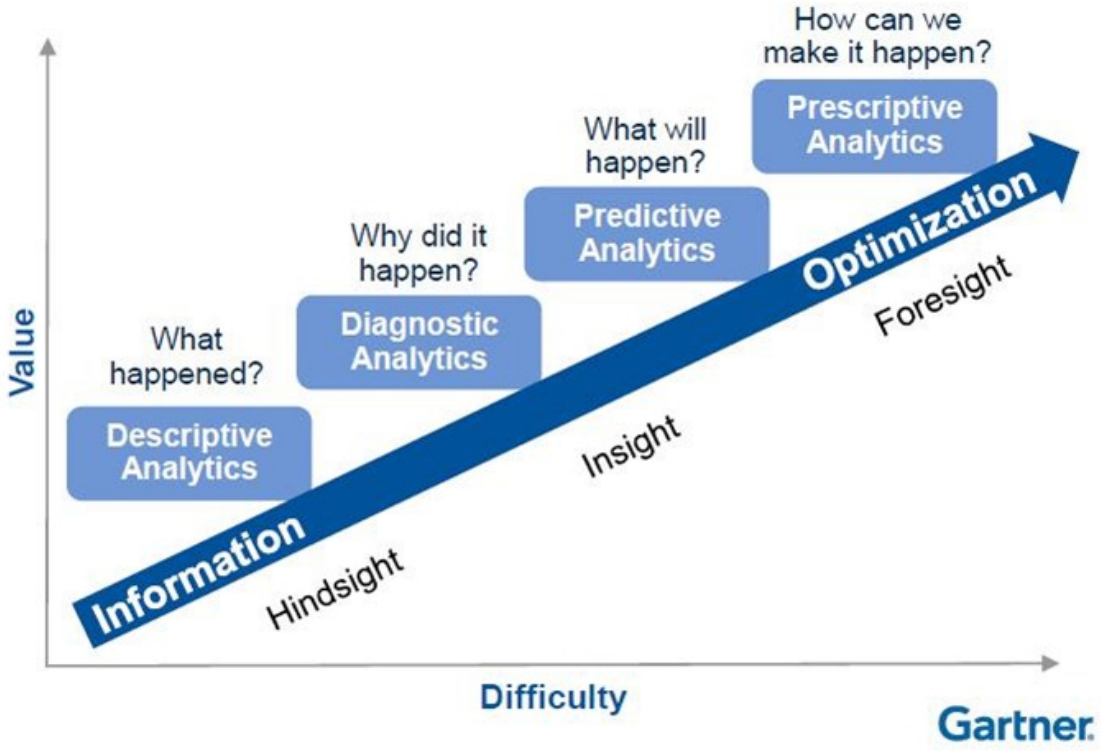


Figure 9 : De l'information à l'optimisation

Source : **http://www.gartner.com/it-glossary/predictive-analytics/**

Ce genre de besoin pose des problèmes de « scalabilité » et de performance précédemment évoqués et comme le témoigne ce retour d'expérience.

« *Après deux ans passés à construire des applications BI pour leboncoin à l'aide d'ETL et de bases de données relationnelles, le constat est criant : les limites frustrent le développeur, l'analyste, et in fine, l'utilisateur.* » (Baltus, 2016)

Ce témoignage montre les limites des bases de données relationnelles dans le contexte des applications BI.

Les architectures traditionnelles de BI, sont dans l'incapacité de se projeter sur des téraoctets voir des pétaoctets de données, malgré la baisse des prix de la RAM, CPU, et disque dur, comme le montre le schéma :

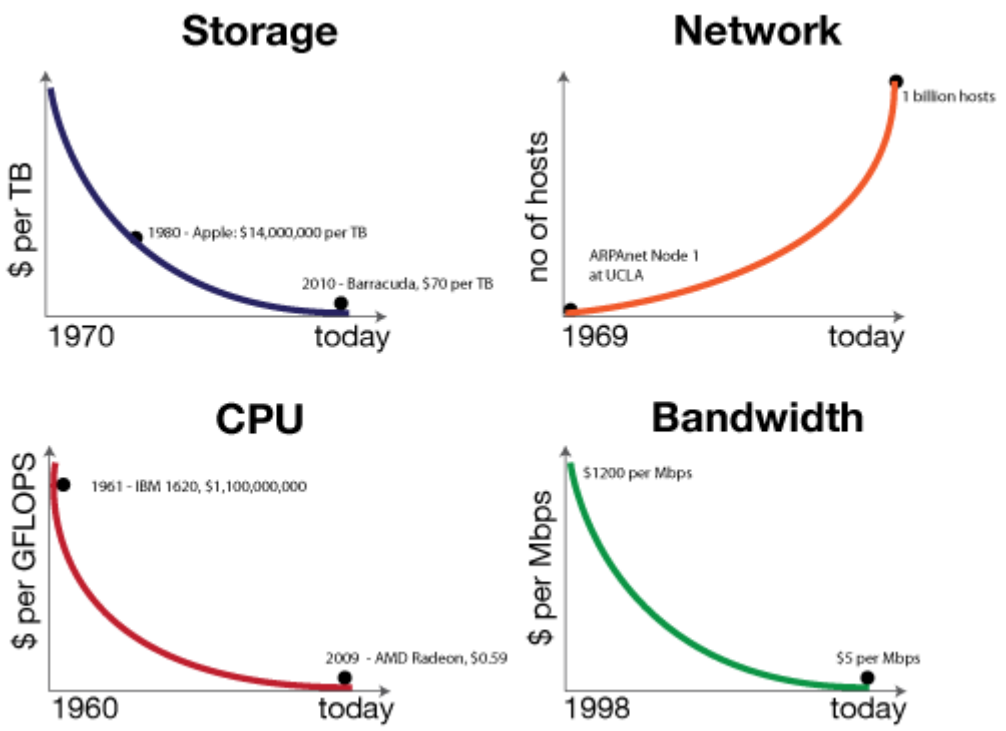


Figure 10 : évolution des composants d'un serveur

Source : **http://radar.oreilly.com/2011/08/building-data-startups.html**

La capacité du débit du disque dur est un problème qui n'a pas de solution technique aujourd'hui, « la capacité de stockage des disques a augmenté de 100 000, le débit lui n'a augmenté que de 100 ». Ce problème est le goulot d'étranglement « *bottleneck* » de l'architecture.

Pour contourner ce problème, il faut soit minimiser l'utilisation du disque dur (utilisation de la mémoire) soit paralléliser le débit (architecture distribuée), afin qu'il soit acceptable, en utilisant le big data.

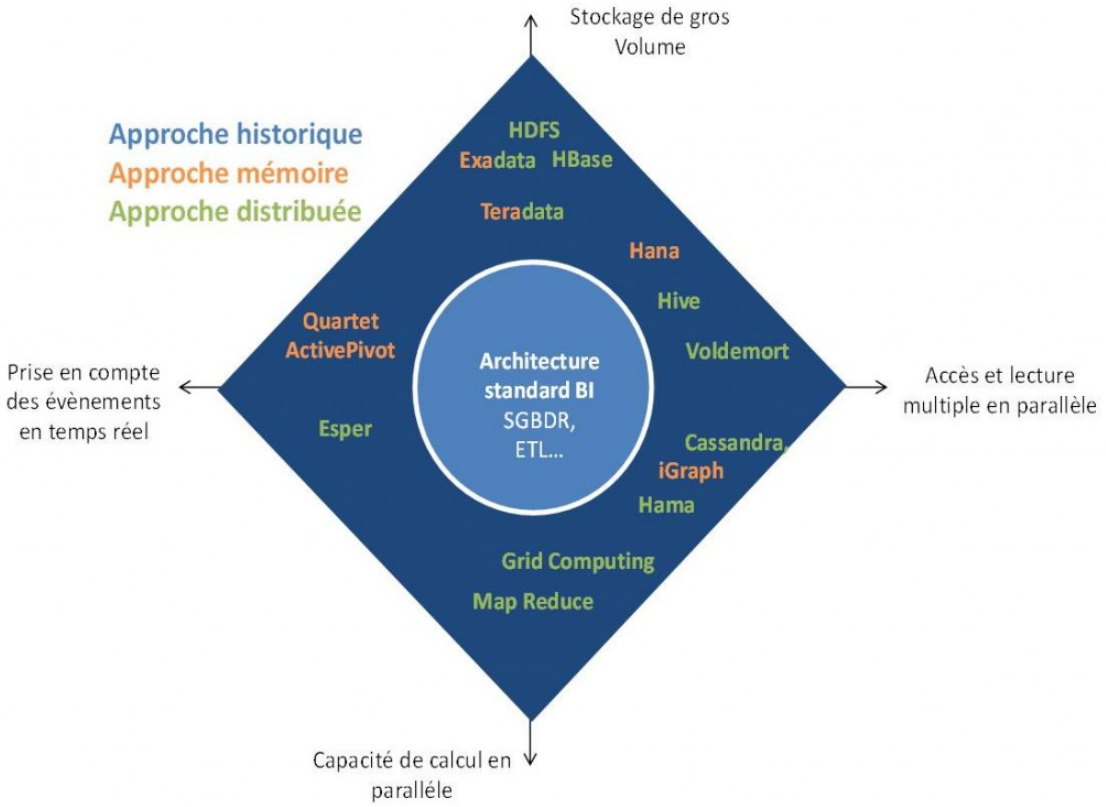


Figure 11 : Cartographie des solutions pour construire une architecture décisionnelle

Source : **http://blog.octo.com/levolution-des-architectures-decisionnelles-avec-big-dataures-decisionnelles-avec-big-data**

« La collaboration la plus naturelle entre le big data et la BI consiste à alimenter les datawarehouses (entrepôts de données) et les cubes OLAP avec des éléments de prédictions issus des traitements de data-sciences effectués sur des données gérées par une architecture big data. » (LAUDE, 2016)

III-G - Les enjeux du big data et les cas d'usage

Désormais dans la société de la connaissance (Knowledge society), les entreprises doivent être proactives et agiles dans un monde instable au lieu d'être réactives au détriment des enjeux économiques et concurrentiels ardus. Or les lourdeurs des outils classiques (SGBDR, BI) ne permettent pas de suivre les nouveaux besoins du marché.

On pense modestement que l'enjeu majeur pour les entreprises est un enjeu économique, car les entreprises stockent plusieurs données de leurs clients qui ne sont pas *exploitées.* Ces données engorgent de l'argent, et si les entreprises arrivent à proposer des services innovants et personnalisés pouvant répondre aux besoins de clients, ils peuvent représenter une source de valeur. On va illustrer cela à travers quelques cas d'usage du big data, pour créer de la valeur. Sachant que toutefois le big data est utilisé dans plusieurs contextes et couvre plusieurs cas d'usage.

- Exemple de la banque privée

Les activités marketing

Le développement de la proximité avec le client : grâce à la dénormalisation des données des clients, et les données des transactions, les banques utilisent les techniques d'apprentissage machine (Machine Learning), afin d'exploiter les données pour déterminer de nouvelles possibilités. Cette connaissance permet la personnalisation du conseil.

Le profilage des clients permet d'améliorer les conseils en investissement : les profils peuvent être analysés sous les différents angles possibles en combinant plusieurs caractéristiques du client.

La réduction des coûts de calcul : l'utilisation des technologies big data est peu onéreuse pour le stockage et pour le traitement soit par lots ou temps réel sur des quantités considérables d'informations, ce qui représente des gains économiques.

- La grande distribution

Fait usage du big data pour faire « le suivi des émissions des tickets de caisse, le flux de marchandises entre ses fournisseurs, entrepôts et magasins, et le parcours utilisateur sur son site e-commerce ».

- Open data : le gouvernement met en avant des banques de données, qui permettent à des entreprises de proposer des services innovants, à titre d'exemple l'application « citymapper », qui utilise des données fournies par l'agglomération lyonnaise, pour proposer des services dans le domaine du « transport », et qui donne des résultats très satisfaisants.
  Le tableau suivant dresse d'autres utilisations du Big Databig data, ainsi que les gains.

Gains	Entreprises ayant constaté un gain
Meilleure habilité à prendre des décisions stratégiques	69 %
Meilleure gouvernance opérationnelle	54 %
Meilleure connaissance et amélioration de l'expérience utilisateur	52 %
Réduction des coûts	47 %
Accélération des décisions	44 %
Développement d'un nouveau produit/service	43 %
Meilleure connaissance du marché et des concurrents	41 %
Développement d'un nouveau business model	38 %
Augmentation des revenus	35 %
Automatisation des décisions	24 %

Tableau 2 : Gains constatés par la mise en place des solutions big data

IV - Les architectures et solutions techniques du big data

IV-A - Les notions conceptuelles du Big Databig data

On a vu dans la définition des caractéristiques du big data, l'importance de la volumétrie, cette dernière a conduit à l'émergence du mouvement NoSQL, afin de stocker les données et répondre à la problématique de la scalabilité. Le traitement d'une telle quantité de données a engendré un nouveau paradigme appelé « MapReduce ».

Qu'est-ce que le paradigme MapReduce ?

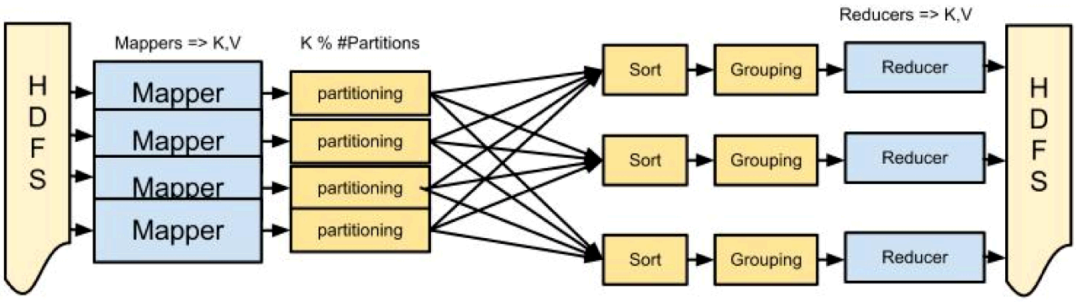
IV-A-1 - MapReduce

À l'origine de ce paradigme Google, dans les années 2004 avait effectué des travaux de recherche dans l'automatisation des traitements parallèles pour son moteur de recherche, et s'est inspiré de la programmation fonctionnelle.

MapReduce est un paradigme de programmation qui permet de distribuer des traitements parallèles sur des volumétries de données dépassant typiquement 1To, dans un cluster composé de centaines, voire de milliers de nœuds - machines - peu coûteux, avec une architecture de type maître/esclave, grâce à la séparation des données et des traitements.

Le traitement parallèle des données avec MapReduce est simple et s'effectue avec deux opérations qu'on appelle map et reduce. Les données d'entrée doivent être structurées sous forme d'une liste de clé/valeur (une Map) qui peut être très volumineuse (téraoctets voire pétaoctets). Ce format unique facilite les entrées\sorties.

Cette simplicité est composée d'une couche logicielle ou d'un Framework. Le Framework open source le plus connu est Apache Hadoop, (que nous allons détailler dans la section suivante) qu'on peut découper en quatre phases, dont deux à la charge du développeur (map/reduce) et le reste à la charge du Framework :



The MapReduce Pipeline

A mapper receives (Key, Value) & outputs (Key, Value)
A reducer receives (Key, Iterable[Value]) and outputs (Key, Value)
Partitioning / Sorting / Grouping provides the Iterable[Value] & Scaling

Figure 12 : Framework MapReduce

Source : **http://blog.matthewrathbone.com/2013/04/17/what-is-hadoop.html**

- Phase de « ***split*** » : dans cette phase c'est le Framework qui divise la liste en entrée en plusieurs splits et chaque split sera attribué à une machine qu'on appelle *mapper*, afin de passer à grande échelle dans le traitement, en se basant sur une architecture de type « share nothing », ce qui rend les traitements indépendants et aucune donnée ne sera traitée par deux nœuds différents. Toutefois le choix de la clé à utiliser et la structure du code est à la charge du développeur.
- Phase de « ***map*** » : les mappers vont exécuter l'opération « map », sur les splits qui leur ont été attribués puis produire une liste intermédiaire de clés/valeurs.
- Phase de « ***Shuffle and Sort*** », où le Framework fait le tri des fichiers par rapport à la clé, puis échange les splits entre les nœuds, et chaque split contient les valeurs associées à une même clé, qui sera ensuite attribuée au reducer.
- Phase de « ***Reduce*** » : dans cette phase chaque reducer appelle la fonction reduce et applique des fonctions d'agrégations, afin de rassembler et produire le résultat souhaité.

Bien que le concept de MapReduce semble simple, sa conception est complexe et souvent confiée à des développeurs confirmés dans la programmation parallèle.

Pour résumer MapReduce propose une architecture logicielle qui simplifie la complexité de l'informatique distribuée, toutefois il faut avoir l'architecture technique qui permet de mettre en place les phases de split et Shuffle and Sort qu'on a vus, c'est ce que nous allons voir dans la section suivante avec l'implémentation open source Apache Hadoop.

IV-A-2 - Qu'est-ce que le traitement par lots ?

Le traitement par lots (Batch) est un traitement automatique et sans intervention humaine, avec un temps d'exécution élevé. ((Lemberger, Batty, Médéric, Raffaelli, & Delattre, 2015)

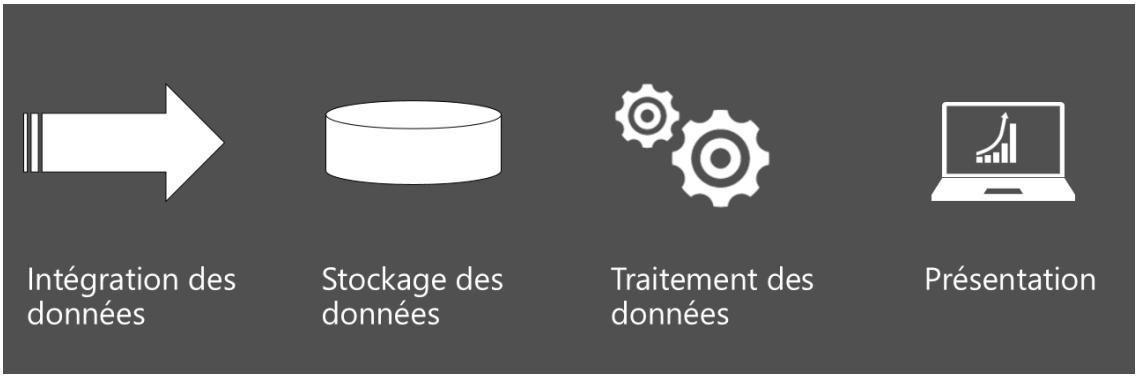


Figure 13 : Traitement par lot

IV-A-3 - Qu'est-ce que le temps réel ?

Le traitement temps réel (streaming) est capable de prendre en compte les contraintes temporelles, pour délivrer des résultats exacts avec le respect d'une latence très faible. Ce qu'il faut noter c'est que la signification de la latence change d'un contexte à l'autre. Dans le contexte du big data la latence peut être de l'ordre de la seconde (opération de bourse, système de réservation, détection de fraude), voire plusieurs secondes - presque temps réel - (système de traking, monétisation des tweets, analyse des données de géolocalisation dans un centre commercial pour envoyer des offres promotionnelles).

Exemple : Apache Spark Streaming, Apache Storm

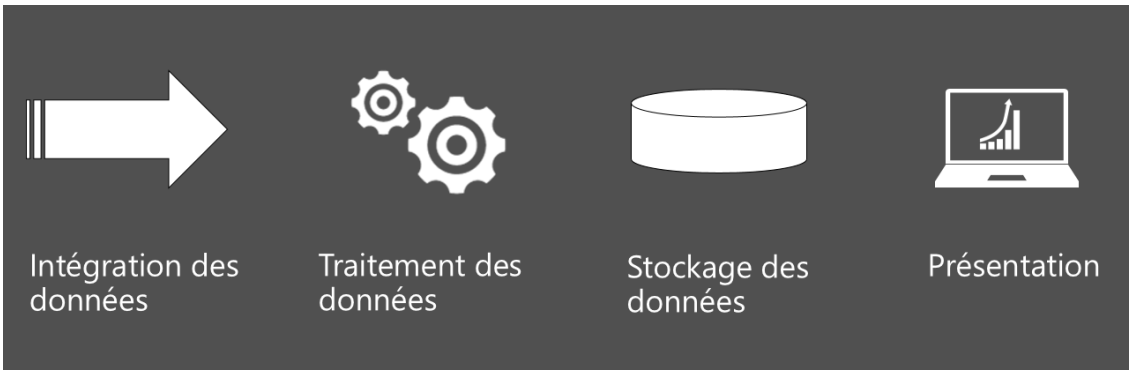


Figure 14 : Traitement temps réel

IV-B - Les architectures big data

Il existe aujourd'hui un nombre important d'architectures big data, l'architecture Lambda, l'architecture Kappa ou l'architecture Zeta, regroupées sous le nom de traitement polyglotte (Polyglot Processing). Dans ce qui suit, nous allons nous intéresser à l'architecture Lambda qui est la plus répandue en ce moment.

IV-B-1 - L'architecture Lambda

Créée par Nathan Marz en se basant sur ses expériences chez Twitter et Backtype, le but de l'architecture Lambda est de fournir un modèle de traitement presque temps réel sur des volumes importants de données, en proposant un nouveau modèle de calcul. Ce modèle essaie de trouver l'équilibre entre la tolérance aux pannes, les contraintes de latence (latence très faible pour les lectures/écritures) et le débit des disques durs en se basant à la fois sur les traitements batch qui fournissent des vues batch et les traitements temps réel qui fournissent des vues, puis les joint avant leur présentation.

Hadoop n'est pas capable de traiter un grand volume de données qui doit satisfaire une faible latence, même en ajoutant d'autres serveurs de calcul, d'où la naissance de cette architecture qui ne remet pas en question le paradigme MapReduce, mais propose une amélioration, afin de contourner les contraintes de latence de Hadoop.

L'architecture Lambda est indépendante de la technologie, et se base sur le précalcul des résultats, puis à les récupérer dans une base et les envoyant au demandeur. Elle est composée de trois couches.

* **Couche Batch** : mode de fonctionnement classique des applications big data type Hadoop, cette couche est responsable de deux choses : récupérer les données et les stocker en format brut (AS-IS : pour pouvoir répondre à de nouveaux besoins métier sans impacter les données initiales) dans des puits de données (Data Lake en anglais), et lancer périodiquement des traitements sur les données, pour précalculer les résultats sous forme de vue logique. Le résultat est ensuite stocké typiquement dans des bases en lecture seule et les mises à jour remplacent les vues logiques précalculées.

Cette couche peut être implémentée à l'aide de Apache Hadoop, MapReduce, Spark que nous allons voir par la suite.

* **Couche de vitesse ou temps réel (Speeding)** : traiter les nouveaux flux de données en temps réel, sans aucun prétraitement (correction des jeux des données). Cette couche minimise la latence et fournit en temps réel des vues avec les données les plus récentes. C'est un fonctionnement en mode continu unitairement pour chaque nouvelle donnée. Les résultats (les vues temps réel) fournis par cette couche ne sont pas aussi fiables, que ceux de la couche batch.

Cette couche peut être implémentée à l'aide de Apache Storm ou Spark Streaming.

* **Couche de service (Serving)** : rend exploitables les résultats précalculés par la couche batch et la couche temps réel, pour effectuer des requêtes à la volée (ad hoc).

Cette couche peut être implémentée à l'aide des technologies NoSQL Apache HBase, Cassandra, et ElasticSearch qui permettent de merger les vues batch et les vues temps réel.

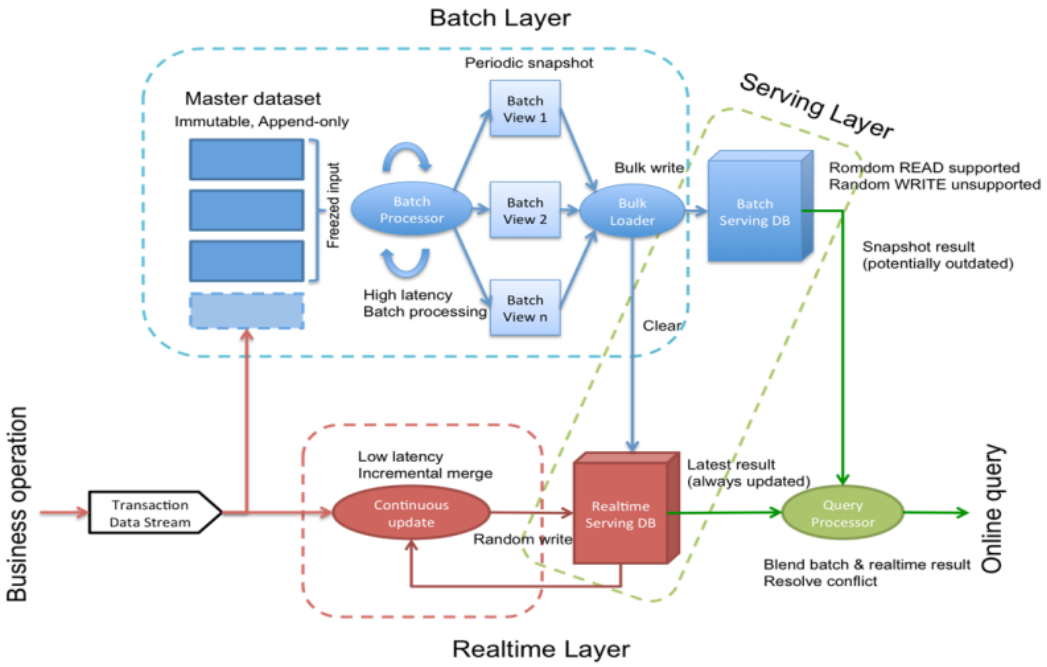


Figure 15 : Architecture Lambda

Source : **http://fr.slideshare.net/MohanBavirisetty/polyglot-processing-an-introduction-10**

Par rapport au stockage des données l'architecture Lambda recommande de ne plus toucher à la donnée après leurs insertions dans la couche batch.

Pour plus d'informations, vous pouvez voir le site **http://lambda-architecture.net/**

IV-B-2 - L'architecture Kappa

Créée par Jay Kreps en se basant sur ses expériences chez LinkedIn et son retour d'expérience de l'architecture Lambda, dans un article : ‘‘Questioning the Lambda Architecture".

Partant d'un constat que la plupart des solutions sont capables de faire à la fois des traitements temps réel (streaming) et traitements batch, l'architecture Kappa, permet de simplifier l'architecture Lambda, en fusionnant la couche batch et la couche Speeding.

Elle apporte aussi une modification sur les SGBD qui doivent être un système de fichiers de log immuable.

L'architecture Kappa n'est pas destinée au stockage des données, mais uniquement à leur traitement, comme le montre le schéma suivant :

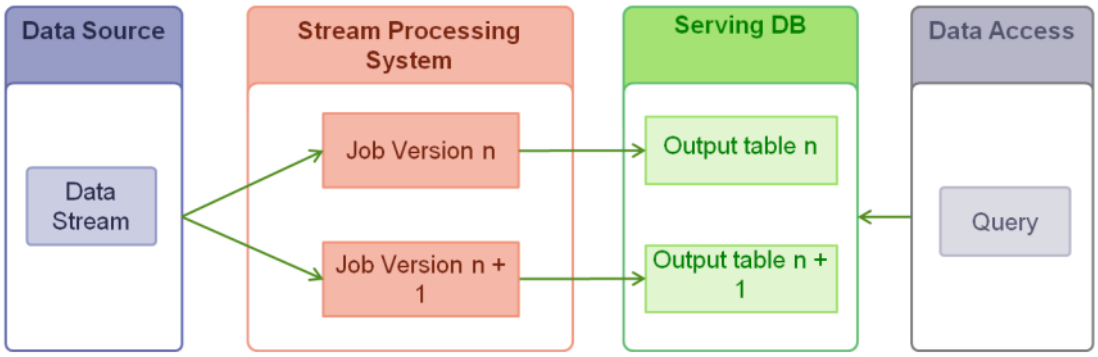


Figure 16 : Architecture Kappa

Pour plus d'informations, voir : **http://milinda.pathirage.org/kappa-architecture.com/**

IV-C - Les solutions big data

Dans cette partie nous allons présenter les différentes solutions techniques pour le big data.

La solution Hadoop a été développée par Doug Cutting bénévole à la fondation Apache qui avait des besoins similaires en termes de stockage et traitement de grosses volumétries de données en s'inspirant des publications de Google sur le sujet et les entreprises comme Facebook, Yahoo, Amazon, Twitter et Netflix ont contribué à son développement.

IV-C-1 - Le noyau d'Hadoop

Hadoop est la plateforme logicielle open source de la fondation Apache permettant de répondre aux besoins du big data, à savoir la volumétrie, la véracité et vélocité des données, ainsi qu'au paradigme de traitement MapReduce pour avoir une meilleure vitesse d'exécution d'algorithme. Elle permet de stocker et d'effectuer des traitements sophistiqués sur des données de différents types et différentes provenances à grande échelle et à moindre coût.

Grâce aux différentes contributions, Hadoop est aujourd'hui un écosystème complexe. En effet si le cœur (Kernel) de Hadoop est basé sur le système de fichier HDFS (Hadoop Distributed File System) et le paradigme MapReduce, pour les traitements que nous allons voir en détail par la suite, on a également une multitude d'outils autour de ce noyau. On peut citer entre autres :

- HBase, basée sur BigTable le système de stockage de Google, une base de données NoSQL de type colonne ;
- ZooKeeper, un système de gestion de service distribué ;
- des outils d'interrogation et de traitement des données, notamment Hive et Pig ;
- des outils de planification et coordination des traitements MapReduce Oozie ;
- des outils de transfert de données entre hadoop et les SGBDR classiques, Sqoop.

IV-C-2 - Le système de stockage de fichier HDFS

Dans cette partie nous allons aborder le système de stockage HDFS, ses objectifs, ses fonctionnalités, son accès et son architecture.

HDFS (Hadoop Distributed File System) est le système de stockage de fichiers d'Hadoop, basé sur le système de fichiers développé par Google, GDF (Google File System) et dont la théorie a été publiée dans des papiers de recherche de Google.

HDFS a pour objectifs d'être :

- **tolérant aux pannes** d'une manière native (Fault tolerant) : c'est-à-dire qu'il peut résister à l'erreur, en partant du principe que la défaillance est la règle générale plutôt que l'exception, en premier lieu la panne matérielle et notamment le disque dur ;
- **scalable** : il peut supporter une montée en charge horizontale, avec très peu de contraintes et de pertes de performance ;
- **modèle d'accès immuable** : écriture unique (il n'est plus modifiable) et multiples lectures pour les fichiers, ce qui favorise la cohérence des données.
- **Déplacer les calculs vers les données** : parce que le déplacement des fichiers volumineux peut conduire à un goulot d'étranglement réseau et impacter les performances globales du système.
- **simple à mettre en place** en assurant la portabilité sur différentes plateformes.

Il propose de nombreuses fonctionnalités :

- **gestion des fichiers par blocs** : chaque fichier sera découpé en blocs de 64 Mo ;
- **réplication et distribution** : tous les blocs seront à la fois répliqués et distribués pour assurer un stockage fiable des fichiers de très grand volume. Répliqués sur différents endroits sur l'HDFS, par défaut trois endroits, le premier est le serveur pour positionner le fichier, le second est un serveur voisin sur le même rack et le troisième sur autre rack ou un autre Datacenter, pour permettre d'avoir la meilleure distribution possible ;
- **gestion des droits** : les permissions en lecture et écriture et l'accès au répertoire à la fois pour le propriétaire et pour le groupe. L'authentification peut être soit absente, soit gérée par login et mot de passe ou bien par Kerberos ;
- **accès aux données en continu** (Streaming) : il a un accès continu aux jeux de données, avec un haut débit d'accès au détriment de la faible latence, parce qu'il est adapté au traitement par lots ;
- **stockage des grands jeux de données**, les fichiers stockés dans HDFS sont typiquement de plusieurs gigaoctets ou téraoctets, avec une bande passante de haut débit pour les différents nœuds du cluster.

On peut utiliser HDFS grâce :

- aux commandes bash ;
- aux bibliothèques Java ou autres langages ;
- aux interfaces REST avec HttpFS ou WebHDFS.

L'architecture HDFS repose sur une architecture maître/esclave comme le montre le schéma. Elle est composée de :

- un serveur maître nommé « NameNode », qui gère l'espace de noms des fichiers et répertoires ainsi que les droits d'accès ;
- un certain nombre de serveurs esclaves nommés « DataNodes », qui gèrent le stockage des fichiers divisés en plusieurs blocs répliqués et distribués.

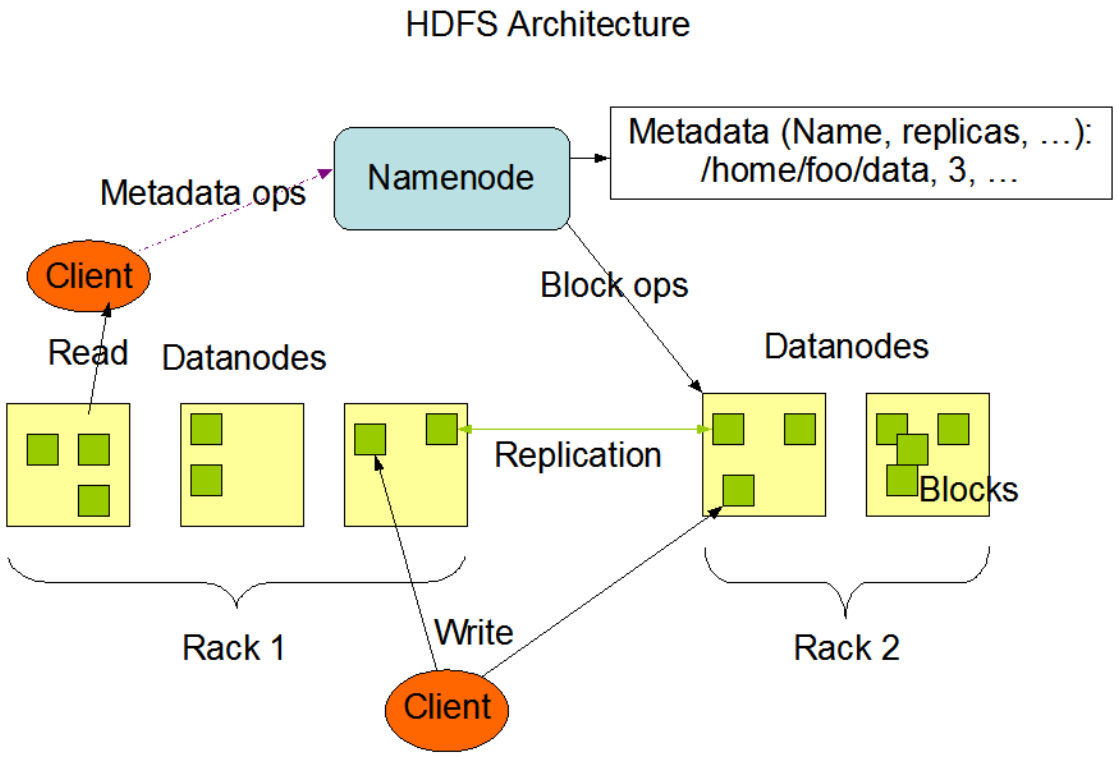


Figure 17 : Architecture HDFS

Source :**https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/images/hdfsarchitecture.png**

IV-D - L'écosystème d'Hadoop

Nous allons voir les différents outils de l'écosystème d'Hadoop.

IV-D-1 - HBase

HBase est la base NoSQL de Hadoop, inspirée de BigTable, qui a pour objectif de stocker une grosse volumétrie de données avec des accès rapides en lecture/écriture, et qui est utilisée en entrée et en sortie de MapReduce.

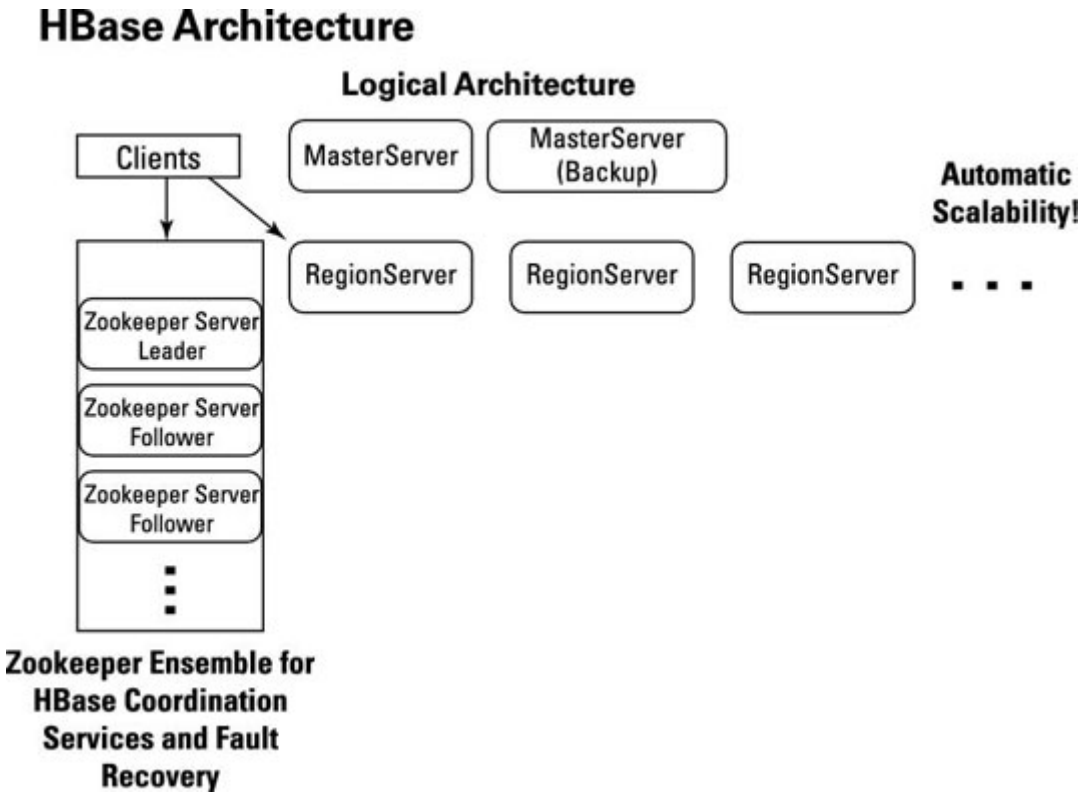


Figure 18 : Architecture de HBase

Source : **http://www.dummies.com/how-to/content/the-hbase-masterserver.html**

HBase a plusieurs propriétés qu'on a déjà vues dans la partie consacrée à NoSQL et que nous allons rappeler ici :

- **base orientée colonnes**, c'est-à-dire qu'on peut ajouter ou supprimer des colonnes et modifier la structure de la base sans impact sur les performances, les données sont stockées dans des fichiers distincts dans des colonnes ;
- **schéma flexible** ou « schema-less », qui ne veut pas dire qu'elle n'a pas de schéma, mais elle contient des métadonnées qui sont définies à l'avance, citons les informations sur la table, avec les familles de colonnes (permet de définir des groupements de colonnes en fonction par exemple de spécificités métier), sans définir forcement chaque colonne à l'avance, mais on peut les insérer dynamiquement aux familles de colonnes ;
- **découpage en régions** : toutes les données sont réparties sur des régions de façon alphabétique en fonction de leur clé de stockage. Ce découpage est fait avec des espaces de noms (namespace), sans chevauchement entre les régions ;
- **scalable horizontalement** : grâce au découpage en régions qui sont montées en serveur, toutefois si le découpage est bien fait ;
- **non transactionnelle** : toutes les opérations sont unitaires et ne peuvent pas être annulées ;
- **clé unique** : cette clé est obligatoire et il faut lui accorder une grande réflexion, car elle permet entre autres de faire le découpage en régions, en plus de l'indexation, et du tri.

IV-D-2 - Hive

Développé par Facebook puis donné à la fondation Apache. Hive est un entrepôt de données structurées tables avec des partitions pour pouvoir mettre les tables sur plusieurs nœuds et qui seront décomposés en Bucket. Hive est également composé d'un langage de requêtage nommé *HiveQL* pouvant s'exécuter sur Hadoop, qui est fortement inspiré de SQL, et qui a pour objectif de faire des requêtes sur les données pour avoir des résultats. Les requêtes SQL sont transformées vers leur équivalent en Map/Reduce à l'aide de HCatalog, et par conséquent les résultats ont une forte latence.

Hive est déployé sur les principales distributions d'Hadoop (CDH, HDP et MapR), que nous allons voir dans la section dédiée aux distributions d'Hadoop.

IV-D-3 - Pig

Développé par Yahoo et le créateur d'Apache Hadoop, Pig est considéré comme le langage de traitement des données d'Hadoop, contrairement à Hive, Pig définit son propre langage nommé *Pig Latin* , qui est un langage procédural pour les traitements parallèles qui agit sur des flux de données. Pig peut fonctionner sur des données structurées ou non structurées sous différents formats. Le grand avantage de Pig par rapport à Hive, c'est qu'on peut laisser les données à l'état brut, et demander à Pig de les lire. Donc l'avantage c'est qu'on n'a pas besoin de transformer les données pour les rendre accessibles à Pig. Ainsi pour analyser les données, on peut réaliser des scripts Pig qui s'appliqueront directement sur nos données. Le langage Pig Latin est transformé vers son équivalent en Map/Reduce, par son moteur d'exécution.

Pig est déployé sur les principales distributions d'Hadoop (CDH, HDP et MapR), que nous allons voir dans la section dédiée aux distributions d'Hadoop.

IV-D-4 - Zookeeper

Zookeeper est une base de données hiérarchisée pour la coordination distribuée pour les applications distribuées. Zookeeper est composé de nœud nommé *ZNode,* qui a comme caractéristique, une latence très faible avec des temps de réponse inférieur à la milliseconde, de ce fait un ZNode a pour but de stocker les données de configuration, de les sérialiser, de synchroniser les différents services Zookeeper, et doit avoir au maximum une taille de quelques kilooctets en données. Zookeeper est hautement disponible.

Zookeeper peut être couplé à un *Watcher,* il permet de faire des notifications lors d'un changement dans un ZNode,

Zookeeper est au cœur de Hadoop, pour les NameNode, YARN, HBase, Kafka, Storm.

IV-D-5 - Sqoop

Sqoop est créé par Cloudera, Sqoop est un outil Hadoop en ligne de commande permettant d'échanger les données entre Hadoop et les SGBDR.

Sqoop gère les données stockées dans HDFS, Hive et HBase, et fonctionne dans les deux sens de Hadoop vers les SGBDR et vice versa. Il permet :

- d'importer des tables individuellement ou des schémas entiers vers le HDFS ;
- de générer des classes Java qui permettent d'interagir avec les données importées ;
- d'exporter les données de HDFS vers les SGBDR.

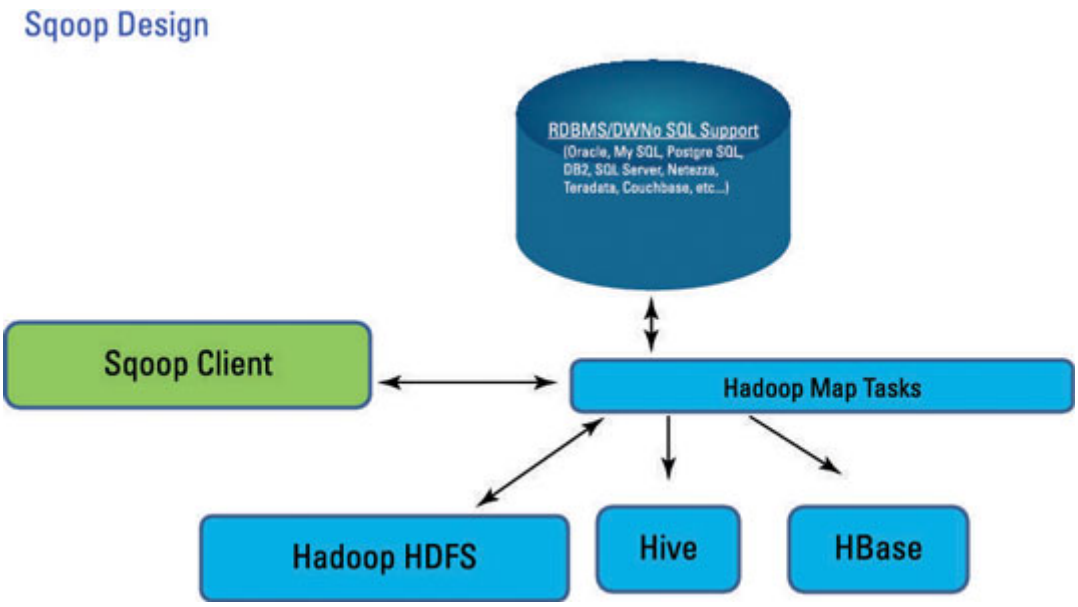


Figure 19 : Sqoop

Source : **http://www.dummies.com/how-to/content/the-principles-of-sqoop-design.html**

IV-D-6 - Oozie

Oozie a été créé par la fondation Apache.

La plupart des algorithmes ne font pas un seul Map/Reduce, mais un enchainement de Map/Reduce, par conséquent Oozie a été créé, afin de coordonner et de créer de workflow de Map/Reduce. Il permet de faire appel à Hive, aux scripts Pig, commandes Batch et des appels Java, comme le montre la figure suivante :

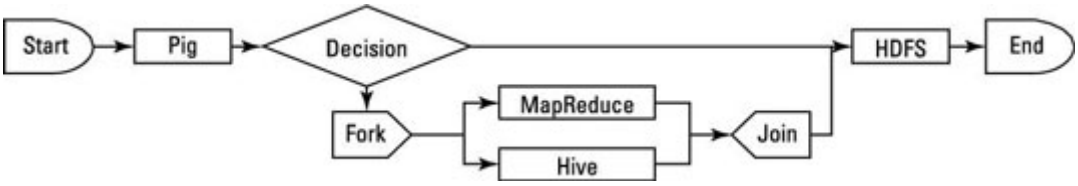


Figure 20 : Exemple d'un worflow Oozie

Source : **http://www.dummies.com/how-to/content/developing-oozie-workflows-in-hadoop.html**

IV-D-7 - Flume

« *Flume est une solution de collecte, agrégation et transfert de gros volumes de logs. Il a été pensé pour gérer des débits importants avec une fonctionnalité native d'écriture dans HDFS au fil de l'eau* ».

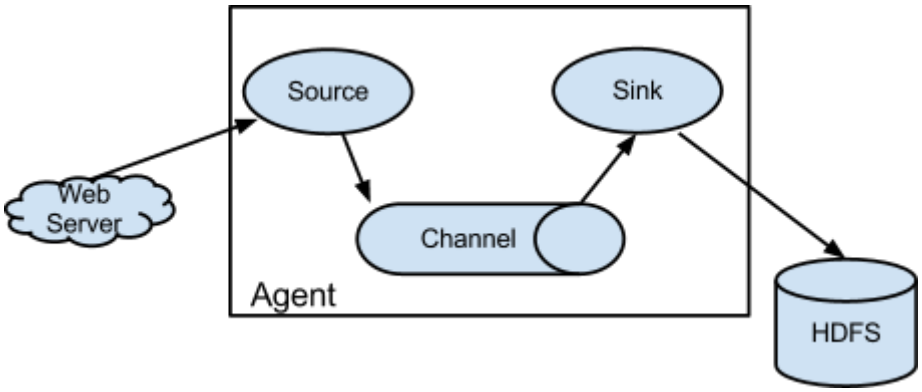


Figure 21 : Apache Flume

Source : **https://flume.apache.org/**

Flume permet d'apporter des données de serveurs et de les importer vers Hadoop HDFS. Conçu à la base pour traiter les journaux serveur, il se base sur trois notions :

- La source : c'est la donnée qu'on veut importer ;
- Channel : c'est la manière dont il stocke l'information lorsqu'elle est en traitement (RAM, sur disque, SGBDR, etc.) ;
- Sink : l'endroit vers lequel il exporte la donnée, qui peut être HDFS, une table HBase, Kafka, Avro et Thrift (voir l'annexe pour plus d'explications).

IV-D-8 - Kafka

Apache Kafka est un broker de message, qui gère des évènements en se basant sur un modèle publish-subscribe. Donc on a des producteurs qui vont envoyer des évènements vers le cluster Kafka, ensuite le cluster Kafka va réceptionner ces évènements et va les mettre en queue, puis va permettre au consommateur d'évènements de souscrire des abonnements sur ces queues d'évènements pour être prévenu s'il y a un évènement qui les intéresse.

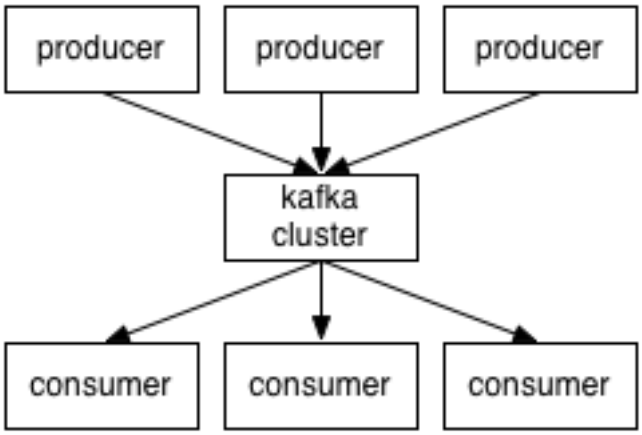


Figure 22 : Apache Kafka

Source : **http://kafka.apache.org/documentation.html**

Kafka se base sur ZooKeeper, pour assurer la scalabilité.

IV-D-9 - Spark

La plupart des traitements sous Hadoop se font avec le paradigme Map/Reduce, il faut lancer au minimum un Mapper et un Reducer, dans des containers différents et des JVM différentes puisqu'il y a une architecture shared-nothing, ce qui empêche de faire du temps réel.

Apache Spark sert à faire des traitements temps réel, avec un modèle (transform/action) plus flexible que le paradigme Map/Reduce, en utilisant un modèle plus général des données nommé RDD (Resilient Distributed Dataset), sachant qu'il est « Storage agnostic », c'est-à-dire qu'il fait abstraction du mode de stockage, contrairement à Map/Reduce qui utilise HDFS.

Les traitements sont réalisés en mémoire et non dans HDFS, ce qui rend leur exécution plus rapide.

IV-E - Les distributions Hadoop

Hadoop est un système complet, il comporte plusieurs briques logicielles, plusieurs contributeurs et plusieurs mainteneurs, ce qui entraine plusieurs cycles de version pour les outils. En conséquence une maitrise des versions compatibles entre elles est très complexe, cela nécessite également une configuration entre les composants, également très complexe et présente de grandes difficultés à l'installation.

De ce fait, on va avoir plusieurs distributions qui se positionnent, afin de faciliter le déploiement de Hadoop. Une distribution est un ensemble de briques d'Hadoop prépackagées, mises ensemble, et configurées pour fonctionner ensemble. L'intérêt d'un package complet c'est qu'il permet de démarrer plus facilement, plus rapidement et d'avoir un support.

Aujourd'hui il existe plusieurs distributions sur le marché, mais nous allons présenter une liste non exhaustive des solutions les plus utilisées.

IV-E-1 - Hortonworks

- **Hortonworks** ou **HDP** : a été fondée en 2011 par une équipe de Yahoo, et se base sur une philosophie **100 % open source** et libre sous la licence Apache. Par ailleurs, ils font très peu de développements spécifiques, très peu de corrections et ils essayent de rester le plus proches de la plateforme Apache d'origine. Elle est également utilisée dans l'offre Cloud de **Microsoft Azure**.

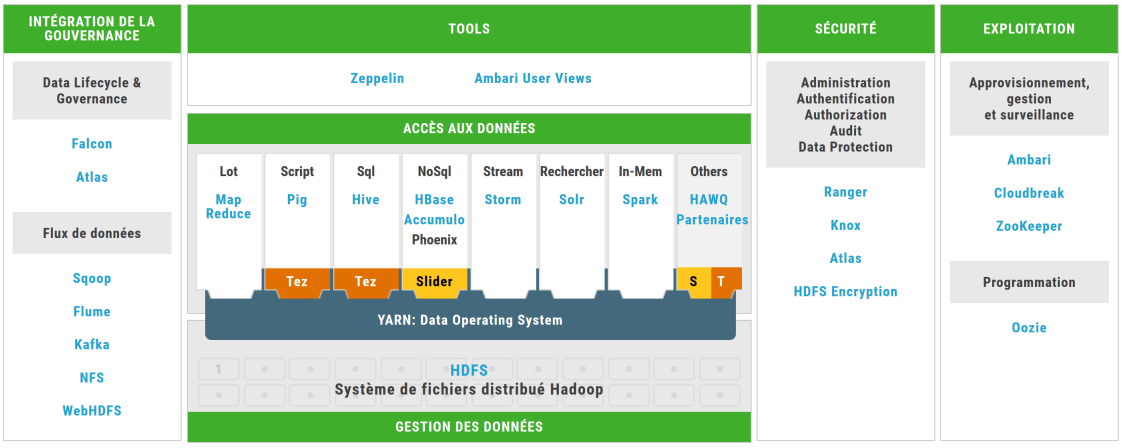


Figure 23 : Distribution MapR

Source : **http://fr.hortonworks.com/products/data-center/hdp/**

IV-E-2 - MapR

- **MapR** : a été créée en 2009 par une équipe de Google, MapR mentionne trois versions, une version open source et deux versions payantes. MapR à la spécificité d'avoir redéveloppé plusieurs composants du noyau d'Hadoop, notamment HDFS qui a été remplacé par MapR FS, et YARN qui a été remplacé par MapR MR et de la même façon, ils ont apporté beaucoup de modifications à HBase. MapR est utilisée dans les offres cloud de Amazon EMR (Elastic Map and Reduce) et de Google GCE (Google Compute Engine).

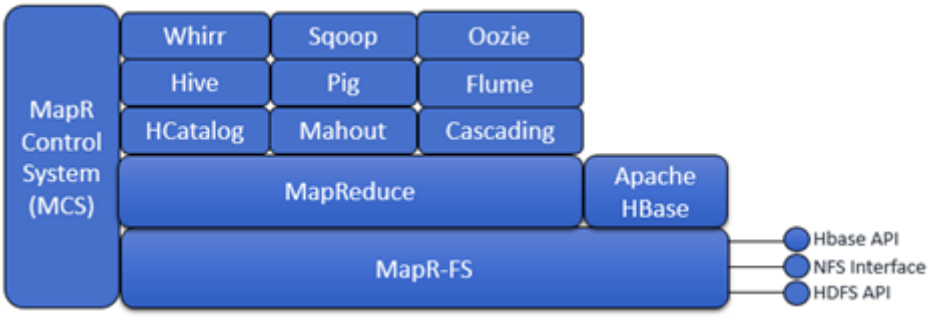


Figure 24 : Dsitribution HortOnWorks

Source : **http://doc.mapr.com/display/MapR/MapR+Overview**

IV-E-3 - Cloudera

Cloudera ou **CDH** : a été créée par divers experts de différents acteurs du web (Oracle, Facebook, Google, Yahoo), elle a une seule version qui est libre et ils ont développé quelques composants spécifiques qui ont été donnés à la fondation Apache notamment Cloudera Search, Hue et Impala. Cloudera est spécialisée dans les offres commerciales de support, formations et certifications.

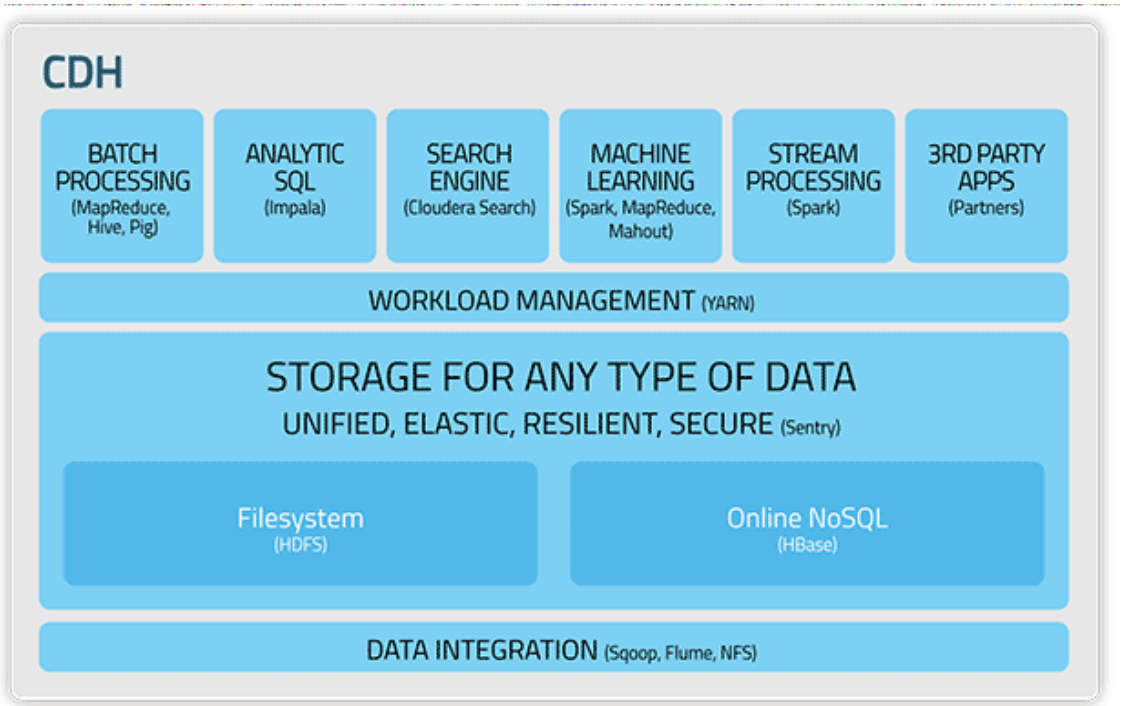


Figure 25 : Distribution Cloudera

Source : **https://www.cloudera.com/documentation/enterprise/5-4-x/topics/cdh_intro.html**

Malgré les solutions qu'on vient de citer, le big data est un domaine très vaste et les choix techniques ne cessent d'évoluer dans ce domaine, même s'il existe déjà d'innombrables choix disponibles. Souvent les entreprises supposent que la mise en place d'une solution Hadopp avec une analyse personnalisée va permettre de tirer parti des capacités du big data, mais ce n'est pas le cas.

Dans la partie suivante, nous allons voir qu'il faut réunir plusieurs critères afin de répondre aux besoins métier et tirer parti des technologies big data.

V - Proposition des critères de choix d'une architecture big data

La valeur du big data n'est pas dans la quantité des données, mais dans son utilisation efficace et sa capacité à fournir de la valeur ajoutée, en choisissant l'architecture et les technologies qui vont permettre de l'adapter à un contexte métier particulier et l'aligner avec la stratégie de l'entreprise.

Les parties 1 et 2 ont un double intérêt : voir les limites d'une architecture traditionnelle et les solutions pour répondre aux nouveaux besoins techniques et fonctionnels, que nous allons rappeler brièvement ici.

D'une part sur les points techniques le modèle SGBDR a montré son incapacité de répondre à des besoins de haute disponibilité, qui est un parmi les critères d'une architecture big data, d'où la naissance du modèle NoSQL qui permet d'offrir un modèle de stockage scalable et à moindre coût.

D'autre part, pour les besoins fonctionnels et métier, nous avons vu la naissance dans la deuxième partie de différentes architectures qui permettent de répondre à différents contextes métier, d'avoir des modèles de traitement temps réel, dans le cadre particulier du big data. Puis nous avons vu qu'il y a un grand nombre de choix technologiques pour le big data, parmi lesquels Hadoop qui est une grande évolution.

Auparavant les DW étaient accessibles uniquement aux entreprises qui étaient en mesure de se payer des systèmes sophistiqués pour consolider les données qui ont une grande valeur ajoutée.

Nous avons vu que Hadoop, permet

- *de déplacer le code (calcul) vers les données*, grâce à son système interne qui permet de localiser les données afin de déplacer le calcul vers elles, ce qui permet d'améliorer les performances ;
- *d'avoir une architecture Share-Nothing,* ce qui permet à chaque job de calcul de s'exécuter dans son nœud, sans partage de ressources avec les autres nœuds ;
- de supporter les défaillances système, grâce à l'architecture Share-Nothing, chaque job peut être redémarré indépendamment en cas de défaillance sans impacter le reste du système.

Hadoop peut être vu comme une infrastructure pour les données, car il permet :

- *la collecte et le stockage des données massives*, à partir de différentes et multiples sources de données qu'elles soient structurées telles que les SGBDR ou non structurées.

Exemple d'outils permettant la collecte et le stockage des données : Sqoop, Flume, HDFS, HBase.

- *le prétraitement des données*, il permet le nettoyage des données massives, qui peuvent être peu consistantes, corrompues, ou dupliquées qu'on nomme « données douteuses » (dirty data en anglais) et de les transformer en « données propres » (*clean data* en anglais).

Exemple d'outils permettant le prétraitement des données : Hive, Pig.

- *la création de nouvelles données* dans des vues, en combinant différentes sources de données.

Exemple d'outils permettant la création de nouvelles données : MapReduce, Sqoop.

- *l'analyse des données* à l'aide de son système distribué de traitement.

Exemple d'outils de traitement des données, MapReduce.

- *l'archivage des données* à l'aide de son système distribué de stockage.

Exemple d'outils permettant l'archivage des données : HBase.

Dans cette partie nous allons définir l'architecture d'une manière générale , puis nous allons voir les différents critères du choix d'une architecture big data à prendre en considération.

« L'architecture se réfère à la représentation abstraite du comportement d'un système et de ses composants. Il y a deux principes fondamentaux qui imposent la décomposition d'un système en sous-parties plus petites. D'une part, les systèmes modernes sont devenus trop complexes pour qu'on puisse les appréhender dans leur totalité. D'autre part, il y a autant de niveaux de lecture que de catégories de lecteurs d'un modèle. » (BAILET, 2016)

Le big data permet de collecter, stocker un volume important de données, traiter et analyser différentes sources de données (véracité) de nombreuses façons, afin de les utiliser par d'autres processus métier ou de les faire visualiser par les utilisateurs finals. La figure suivante résume les différentes étapes de traitement du big data.

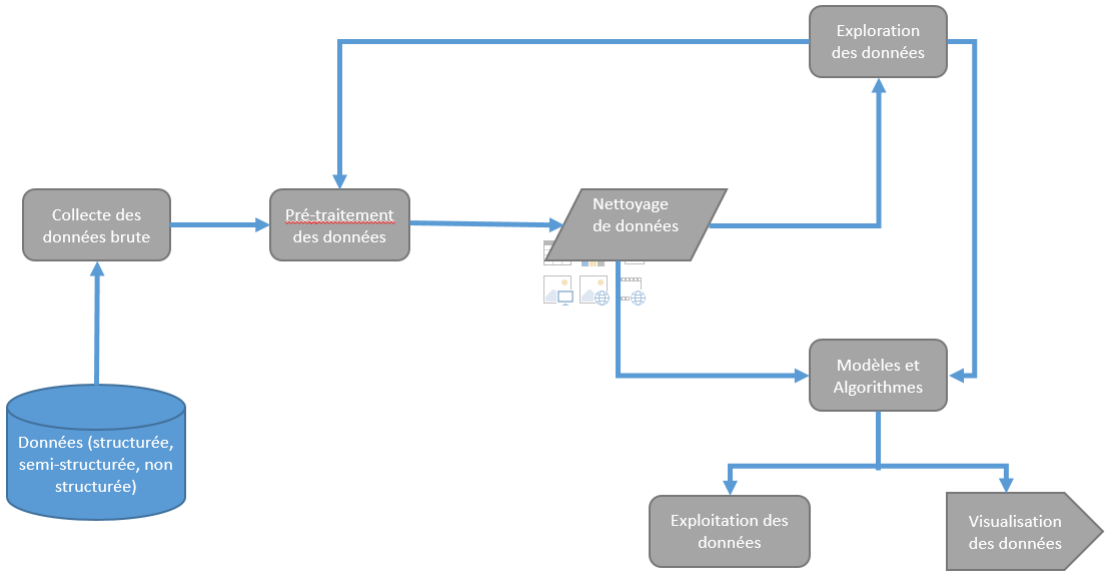


Figure 26 : Gestion du cycle de vie des données

La gestion du cycle de vie des données (Data Lifecycle Management en anglais), résume celle d'un SI, depuis leur création jusqu'à leur effacement ou archivage (End to End).

La figure précédente nous donne une vision sur les différents flux possibles dans le cycle de vie des données, ainsi que les différents points auxquels nous devons nous intéresser pour gérer les données et faire le choix d'une architecture big data.

Les couches et composants qu'il faut identifier pour mettre en place une architecture big data, dans chaque couche on trouve différentes technologies possibles.

- Source des données.
- Collecte et stockage des données.
- Prétraitement des données.
- Nettoyage des données.
- Exploration des données.

- Modèles et algorithmes à appliquer sur les données.
- Exploitation des données.
- Visualisation des données.

Le changement le plus significatif dans la gestion du cycle de vie des données est l'avènement de Hadoop, ainsi que d'autres solutions équivalentes.

Chaque source big data a des caractéristiques différentes, y compris la fréquence, le volume, la vitesse, le type et la véracité des données. Lorsque les données big data sont traitées et stockées, d'autres dimensions entrent en jeu, telles que la gouvernance des données et la sécurité. Le choix d'une architecture big data est complexe et très difficile parce qu'il y a différents facteurs à prendre en considération.

Nous allons présenter une approche structurée, pour simplifier la complexité du choix de l'architecture big data à mettre en place.

La clé pour réussir le choix d'une architecture big data est dans un premier temps de répondre à la question « pourquoi une solution big data ?», voir la valeur métier des données, contre les coûts du stockage, le transport et le traitement des données. Car chaque métier a des besoins et des attentes différents et un contexte particulier, et n'y a pas une solution générique qui permet de traiter les différents besoins métier.

Une fois qu'on a répondu à cette question, les considérations techniques telles que les performances et l'administration de l'infrastructure viennent dans un deuxième temps.

V-A - Le type de traitement

On a vu précédemment les différents types d'architectures Lambda, Kappa et l'impact de la latence sur choix du type de traitement.

Le type de traitement va déterminer les caractéristiques suivantes.

Est-ce que les données sont traitées en temps réel (streaming), presque temps réel ou par lots pour une analyse ultérieure ? Est-ce qu'on va mélanger différents types de traitements ? Le choix du type d'analyse doit être examiné attentivement.

Pour aider à répondre à cette question, nous proposons de voir la nature du besoin, et du résultat attendu.

Est-ce que c'est un traitement opérationnel avec de petits ensembles ou sous-ensembles de données avec une faible latence qui ne dépasse pas la seconde et avec de faibles exigences sur l'exactitude des résultats ?

Dans ce cas, il faut choisir une architecture presque temps réel, qui intègre les données provenant de différentes sources, et fournir rapidement les résultats qui ne sont pas forcément cohérents, sans attendre la fin des données en entrée pour retourner des résultats.

Est-ce que c'est un traitement analytique avec d'énormes quantités de données, avec de fortes exigences sur l'exactitude des résultats ?

Dans ce cas, il faut choisir une architecture batch pour le traitement d'un ensemble de données, on a un résultat qui est cohérent et disponible uniquement à la fin des traitements. Il faut garder à l'esprit que les données qui parviennent en cours de traitement ne seront pas traitées.

V-B - L'utilisateur final des données

Il faut lister tous les utilisateurs possibles des données traitées :

- êtres humains lorsque l'utilisateur final est l'être humain, il faut accorder une grande importance à la visualisation des données, car l'être humain comprend mieux le graphique que le texte. Il faut donc garder ça à l'esprit constamment lorsqu'on modélise l'interface pour donner du sens à l'information, à son utilisation, ainsi qu'à son interprétation, pour permettre une prise de décision rapide et efficace ;
- processus métier : il faut penser au format des données et à leur intégration ;
- applications d'entreprise ;
- référentiel de données.

V-C - La source des données (où les données sont générées)

La source des données peut être problématique lorsqu'on a différentes sources de données. Contrairement à un modèle traditionnel où on a la plupart du temps une seule et unique source de données, le big data peut traiter différentes sources de données, et cela a un impact sur la qualité des résultats. Comme source de données, on trouve :

- Web et réseaux sociaux, dans ce cas il faut prendre en considération la qualité des données, le nettoyage et l'enrichissement des données en définissant des règles, et en cas de rejet des données, il faut voir pourquoi elles sont irrécupérables ;
- données saisies par des utilisateurs, générées par des machines ;
- données internes à l'entreprise.

La variété des données, rend la qualité des données difficile à atteindre, en fonction de la nature du besoin final, donc si l'application à des exigences fortes en termes de qualité de données, il faudra adapter des processus robustes. De même le nettoyage des données consiste à l'identification et l'élimination des données inexactes pour les futurs traitements. Et dans le cadre du big data l'inexactitude des données a un impact sur les qualités des analyses

La variété du big data, offre des possibilités d'explorations des données en se basant sur des processus de classification et clustering.

V-D - Format du contenu

Le format du contenu impacte directement la qualité de la recherche et l'analyse des données, car il repose sur un schéma de données qui prend en compte le format des données afin de répondre à un besoin métier. Le modèle de données doit être dynamique lorsqu'on a un volume important de données. Les formats big data supportés sont :

- structuré (SGBDR) ;
- non structuré (audio, vidéo et images) ;
- semi-structuré : log.

Le format des données détermine la façon dont les données entrantes doivent être traitées, et le besoin de faire des *transformations* d'un contenu non structuré vers un contenu semi-structuré.

V-E - Types des données à traiter

Métadonnées, transactionnelles, données historiques, connaître le type permet de séparer les données dans le stockage.

V-F - Fréquence et taille des données

Quelle quantité des données est attendue et à quelle fréquence va-t-elle arriver ? Connaître la fréquence et la taille permet de déterminer le format de stockage, le mécanisme de stockage que nous allons utiliser et traiter la complexité autour de la réplication, la compression, la disponibilité, l'accessibilité, la mutabilité, la capacité, la latence, le débit, et enfin la sécurité des données.

Parmi les différents points évoqués, il faut en particulier s'intéresser à la compression des données pour avoir une solution optimale en termes de réduction d'espace de stockage, ainsi que la réplication des données, car plus on a de données redondantes plus on aura des traitements rapides.

La fréquence et la taille dépendent des sources des données qu'on a vues :

- série temporelle : en fonction du temps ;
- temps réel, flux continu : données transactionnelles, données météorologiques ;
- flux à la demande : réseaux sociaux.

V-G - Méthodologie de traitement des données

Puisqu'on traite un volume important de données, il est important de choisir une méthodologie qui réponde aux besoins et au contexte métier, sans impacter les données brutes, car il se peut qu'un nouveau besoin métier se manifeste sur les mêmes données qu'on avait.

Il existe différents types d'analyses :

- analyse descriptive ;
- analyse prédictive ;
- génération de requête ;
- génération de rapport.

Les exigences opérationnelles déterminent la méthode de traitement appropriée.

V-H - Le choix du matériel

Ce type d'architecture ne requiert pas un type de matériel particulier, et les machines bon marché sont bien adaptées. Cependant, comprendre les limites du matériel contribue aussi au choix dans le cadre d'une architecture big data, et permet de renforcer les points de défaillance possible d'un tel système.

Vous devez choisir le matériel et le système d'exploitation en fonction des préférences de l'entreprise et des coûts.

Le schéma suivant résume les différentes étapes à suivre lors du choix d'une architecture big data.

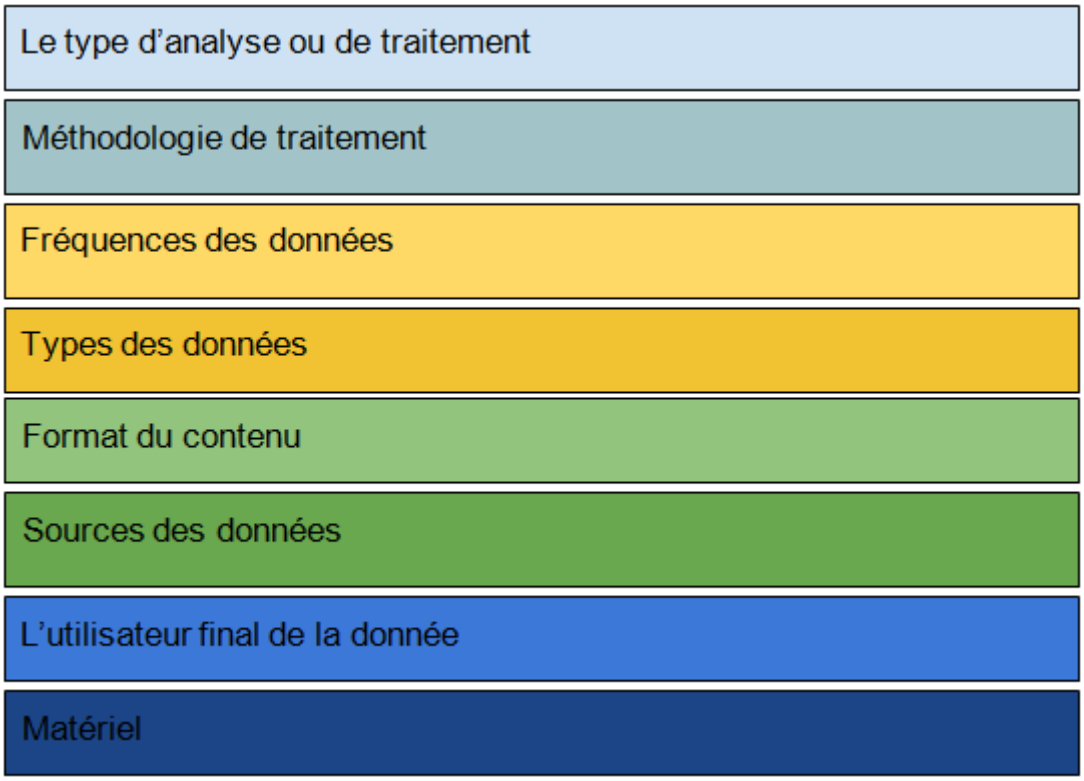


Figure 27 : Critères du choix d'une architecture big data

V-I - Conclusion sur le choix d'une architecture

Les différents critères qu'on a vus vont permettre de faire le choix technologique le mieux adapté pour chaque couche big data, afin de choisir l'architecture big data qui répond le mieux aux besoins métier.

Toutefois, lors du passage à un projet piloté par les données, il faut garder à l'esprit que c'est une architecture distribuée qui doit prendre en considération les problèmes de scalabilité, de performance et de synchronisation entre les différents couches et modèles. Au final il n'y a pas une solution, mais des solutions à chaque problème, et il faudra trancher en fonction des besoins et du contexte métier.

VI - Conclusion

Ce cours a été l'occasion pour moi, d'une part de découvrir le monde du big data à travers la partie 1 et de mieux comprendre les problématiques auxquelles il répond, d'autre part l'occasion aussi de découvrir la difficulté d'un tel sujet dans la partie 2 et à travers mes discussions avec les personnes impliquées de près ou de loin dans les sujets de l'architecture big data, afin d'avoir leurs retours d'expérience. Pour ensuite les analyser et pouvoir proposer la méthodologie qui a été présentée dans la partie 3 et qui peut aider à faire le choix d'une architecture big data. Les sujets autour du big data, ne cessent d'évoluer et demandent une très grande curiosité et veille technologique.

VII - Remerciements

Nous tenons à remercier **Winjerome** pour la mise au gabarit, **Laethy** pour la relecture technique et **Claude Leloup** pour la relecture orthographique. Nous remercions également toutes les personnes ayant contribué à l'aboutissement de ce cours.

VIII - Glossaire

ERP : en anglais pour Entreprise ressource Planning.

CRM : en anglais pour Customer Relationship Management.

SCM : en anglais pour Supply Chain Management.

DW : en anglais pour Data Warehouse.

BI: Business Intelligence.

ETL: Extract Transform Load.

OLTP: On Line Transactional Processing.

OLAP (On Line Analytical Processing).

Datamart : magasin de données.

Batch : traitement par lots.

Cluster : ensemble de machines indépendantes situées dans un réseau dans le même espace géographique.

DHT : Distributed Hash Table.

Appliance : matériel spécifique.

MPP : Massive Parallel Processing, Traitement Massivement Paralléle, chacun avec son CPU, mais partage la mémoire.

SSD : Solid State Disk.

CPU : Central Processing Unit.

RAM : Random Access Memory pour mémoire vive.

DCC : Distributed Computing Cluster.

Grid Computing (GRID) : un DCC s'exécutant sur Internet.

Stream processing : traitement de flux.

IX - Bibliographie

H LAUDE. « *Data Scientist et langage R - Guide d'autoformation à l'exploitation des big data* ». ENI (2016).

Le livre explique le lien entre le big data et le Data Science, et présente des solutions se basant sur le langage R, ainsi que les algorithmes du Machine Learning tels que la logique floue, série temporelle et manipulation des images. Ce livre nous a été utile, particulièrement le chapitre « Cadre méthodologique du data scientist » pour comprendre la méthodologie de travail, le cycle des projets big data, ainsi que le lien entre big data et Business Intelligence.

J.-P GOUIGOUX, « *Business Intelligence simple et efficace avec Excel et PowerPivot* ». (ENI, Éd.) ENI. (2014).

P Lemberger, M Batty, M Médéric, J.-L Raffaelli, & M.Delattre, « *big data et Machine Learning - Mauel du data scientist* » Paris: Dunod. (2015).

Le livre présente les notions du big data et Machine Learning, comme l'indique le titre et il n'est pas du tout technique. Il est composé de trois grandes parties :

la première et la dernière partie font partie de la problématique qu'on a traitée dans le cadre de ce cours et porte sur l'émergence du big data, et définit les principes de base tels que MapReduce et la plateforme Hadoop, ainsi que l'architecture Lambda.

La deuxième partie porte sur Machine Learning, mais n'a pas servi dans le cadre de ce cours.

R Bruchez, « *Les bases de données NoSQL et le big data* ». Eyrolles(2015).

Ce livre et très complet et nous a été très utile tout au long de la préparation du cours, car rédiger par un spécialiste des SGBRD ce qui permet déjà de faire une comparaison entre l'approche des SGBDR et celle de NoSQL. Puis une explication des différents types de bases NoSQL à travers différents exemples et cas d'utilisation. Enfin on y trouve aussi les problématiques liées au big data en termes de stockage et de traitement.

S Baltus, (2016, Mai). « Comment Spark et Redshift ont changé notre quotidien » *Programmez !*, p. 58.

T BAILET, (2016). « *Architecture logicielle - Approche organisationnelle, fonctionnelle, technique* » (2e édit°). ENI.

Le livre présente les différentes notions d'architecture logicielle comme l'indique le titre. Mais nous nous sommes documenté sur les parties big data et architecture Lambda, dans les chapitres suivants : Stockage distribué et traitements et architectures distribuées.

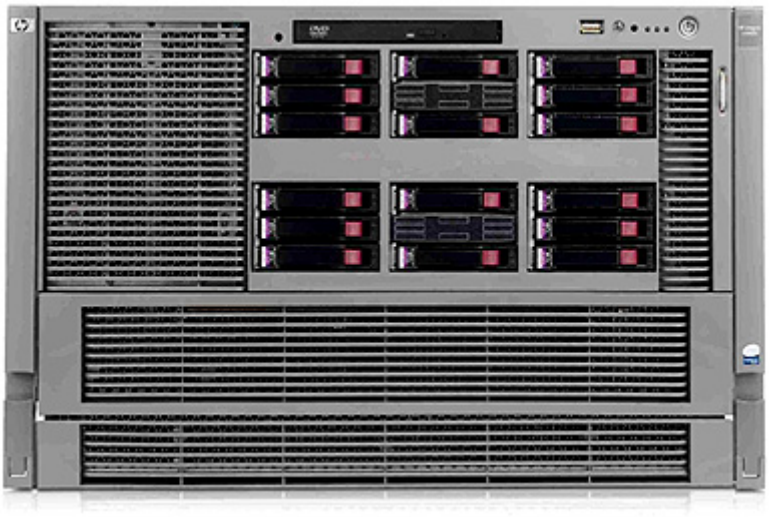
X - Annexes

La montée en charge horizontale VS la montée en charge verticale.

Avant mise en place du « scale up » (croissance interne) :

Après mise en place du « scale up » :

… et encore plus longtemps après :



- Les coûts de la montée en charge horizontale (Scale-out), voir l'impossibilité dans certains cas.

À partir d'un certain niveau de volume de données, les bases de données relationnelles ne peuvent pas supporter la montée en charge.

Avant la mise en place du « Scale-out »

Après la mise ne place du « Scale-out »