



# BIG DATA

Christophe PARAGEAUD



LIVRE BLANC

Septembre 2016

**IPPON**

Digital . Technologies . Hosting

# TABLE DES MATIÈRES

1	L'auteur	6
2	A propos d'Ippon technologies	7
2.1.	Nos solutions	8
2.2.	Nous contacter	9
3	Licence	10
4	Présentation du livre blanc	11
5	A qui se destine ce livre blanc ?	12
6	Big Data : Présentation	13
6.1.	Chronologie du Big Data	14
6.2.	Données du Big Data	17
6.3.	Les 5 V du Big Data	18
6.4.	Les acteurs du Big Data	22
7	Freins et opportunités	23
7.1.	Opportunités du Big Data	25
8	Méthodes de mise en œuvre	28
8.1.	Cheminement analyse de données	28
8.2.	Impacts sur organisation	28
8.3.	RoadMap	30
8.4.	Approche Bottom-Up	30
8.5.	Approche Top-Down	31
8.6.	Conseil pour l'expérimentation	32
9	Mise en œuvre	33
9.1.	Principes d'une architecture Big Data	33
9.2.	Théorème CAP	40
9.3.	Garantie livraison/traitement des systèmes distribués	42
9.4.	Traitements Big Data	43
9.5.	Pré requis matériel	44
10	Les Solutions	47
10.1.	Évolution des solutions Big Data	47
10.2.	Catégories	48
10.3.	Ingestion/Collecte	49
10.4.	Messages	51

10.5. Stockage	55
10.6. Transformation/Enrichissement	68
10.7. Analyse/Requêtage/Visualisation	79
10.8. Indexation	84
10.9. Prédiction/Apprentissage	87
10.10. Acteurs	90
10.11. Interaction et Visualisation (notebook)	92
10.12. Offres Cloud	97
<b>11 Architectures</b>	<b>102</b>
11.1. Hadoop	102
11.2. Traitements de type Batch (incrémentaux)	104
11.3. Traitements temps réel	106
11.4. Architecture Lambda	108
11.5. Architecture Kappa	111
11.6. Architecture SMACK	112
11.7. Architecture acteurs	114
11.8. Architecture Microservices	116
11.9. Indexation (moteurs de recherche)	119
11.10. Architecture Data Lake	121
11.11. Critères de sélection d'une architecture	123
<b>12 Sécurité et Big Data</b>	<b>125</b>
12.1. Introduction	125
12.2. Challenges	125
12.3. Implémentation	126
12.4. Tests et Big Data	127
<b>13 Conclusion et annexes</b>	<b>131</b>
13.1. Matrice de recommandation	132
13.2. Lexique	133
13.3. Liens et références intéressantes	138
<b>14 Remerciements</b>	<b>141</b>

# 1 — L'AUTEUR



Christophe **Parageaud**

Découvrez tous ses articles sur **[blog.ippon.fr](http://blog.ippon.fr)**

---

***TamTam – Java 8 functional programming :  
don't neglect optimisations!***

***MapReduce et les grilles de données ou Hadoop sans Hadoop***

***Livre blanc Ippon : Grilles de données mémoire***

***MongoDB v3 : la révolution?***

***Java Ergonomics : Faut-il encore tuner la JVM?***

***Apache Flink et Spark: redondance?***

***Retour sur l'année 2014 : Big Data, Mobilité, Agilité, Cloud***

## 2 — A PROPOS D'IPPON TECHNOLOGIES

Ippon est un cabinet de conseil en technologies, créé en 2003 par Stéphane Nomis, sportif de Haut-Niveau et un polytechnicien, avec pour ambition de devenir leader sur les solutions Digitales, Cloud et Big Data.

Ippon accompagne les entreprises dans le développement et la transformation de leur système d'information avec des applications performantes et des solutions robustes.

Ippon propose une offre de services à 360° pour répondre à l'ensemble des besoins en innovation technologique : Conseil, Design, Développement, Hébergement et Formation.

Nos équipes tirent le meilleur de la technologie pour transformer rapidement les idées créatives de nos clients en services à haute valeur ajoutée. Elles accompagnent à la fois les grands groupes (Axa, EDF, Société Générale, LVMH,...) et les champions français de l'industrie numérique (CDiscount, LesFurets.com, Aldebaran....) dans leurs innovations.

Nous avons réalisé, en 2015, un chiffre d'affaires de 20 M€ en croissance organique de 27%. Nous sommes aujourd'hui un groupe international riche de plus de 200 consultants répartis en France, aux USA, en Australie et au Maroc.

### IPPON VOTRE PARTENAIRE END TO END



CONSEIL  
FORMATIONS



UX  
DESIGN



RÉALISATION  
AGILITE/DEVOPS



HÉBERGEMENT

## 2.1. Nos solutions

Nous accompagnons nos clients sur la mise en œuvre de projets Data ou NoSQL avec une offre structurée pour faciliter les 3 étapes d'un projet Data:

### UNE IDÉE, UN POC, UN PRODUIT...

---



idée



lean  
startup



conseil  
POC



product  
backlog



projet par  
itérations



équipe  
agile



projet  
run



conseil  
DevOps



infogérance  
cloud

---

**Vos besoins, nos solutions**

## 2.2. Nous contacter



---

PARIS  
BORDEAUX  
NANTES  
LYON

RICHMOND, VA  
WASHINGTON, DC  
NEW-YORK

MELBOURNE  
MARRAKECH

[www.ippon.fr/contact](http://www.ippon.fr/contact)  
[contact@ippon.fr](mailto:contact@ippon.fr)  
[blog.ippon.fr](http://blog.ippon.fr)

+33 1 46 12 48 48  
@ippontech

## 3 — LICENCE

Ce document vous est fourni sous licence Creative Commons Attribution Share Alike. Le texte ci-dessous est un résumé (et non pas un substitut) de la licence.

### Vous êtes autorisé à :

- Partager — copier, distribuer et communiquer le matériel par tous moyens et sous tous formats
- L'offrant ne peut retirer les autorisations concédées par la licence tant que vous appliquez les termes de cette licence.
- Selon les conditions suivantes :

### Attribution :

- Vous devez créditer l'œuvre, intégrer un lien vers la licence et indiquer si des modifications ont été effectuées à l'œuvre. Vous devez indiquer ces informations par tous les moyens raisonnables, sans toutefois suggérer que l'offrant vous soutient ou soutient la façon dont vous avez utilisé son œuvre.
- Pas d'utilisation commerciale — vous n'êtes pas autorisé à faire un usage commercial de cette œuvre, tout ou partie du matériel la composant.
- Pas de modifications — dans le cas où vous effectuez un remix, que vous transformez, ou créez à partir du matériel composant l'œuvre originale, vous n'êtes pas autorisé à distribuer ou mettre à disposition l'œuvre modifiée.
- Pas de restrictions complémentaires — vous n'êtes pas autorisé à appliquer des conditions légales ou des mesures techniques qui restreindraient légalement autrui à utiliser l'œuvre dans les conditions décrites par la licence.

### Notes :

- Vous n'êtes pas dans l'obligation de respecter la licence pour les éléments ou matériel appartenant au domaine public ou dans le cas où l'utilisation que vous souhaitez faire est couverte par une exception.
- Aucune garantie n'est donnée. Il se peut que la licence ne vous donne pas toutes les permissions nécessaires pour votre utilisation. Par exemple, certains droits comme les droits moraux, le droit des données personnelles et le droit à l'image sont susceptibles de limiter votre utilisation.





## 4 — PRÉSENTATION DU LIVRE BLANC

Dans le monde traditionnel de l'analyse des données, un seul système est mis en œuvre. Les données sont acheminées, entreposées et requêtées dans une solution unique (et souvent une seule machine).

Cela pose les problèmes suivants :

- que faire face à l'augmentation de la volumétrie ?
- que faire si la latence du système est trop importante pour les nouveaux besoins ?
- que faire pour répondre à un besoin non couvert par la solution ?

Les architectures Big Data ont évolué à partir de ces systèmes afin de proposer une réponse à ces problématiques.

En contrepartie ces solutions sont souvent très spécialisées, c'est la fin de la solution unique

En conséquence, le nombre de solutions dans le SI augmente, le rendant plus complexe, et se pose la question de l'architecture et des solutions d'administration à mettre en place.

De plus il n'existe pas de standard ou de normes Big Data : les solutions et les architectures se multiplient.

### **Ce livre blanc prend «un peu» de hauteur sur les technologies Big Data**

afin d'aider les décideurs à faire les bons choix techniques et à y voir plus clair sur les opportunités offertes par les différentes technologies.

## 5 — A QUI SE DESTINE CE LIVRE BLANC ?

Ce livre blanc se veut pluridisciplinaire et a été rédigé afin de tenter de répondre aux questions que se posent les acteurs de l'entreprise au moment du choix de leur solution Big Data. Pour les indécis, il permet de se faire une idée du potentiel du Big Data et des opportunités métiers et techniques qui en découlent.

Un constat général :

- croissance exponentielle du volume des données (réseaux sociaux, applications, objets connectés),
- leurs sources et leurs formats multiples :
  - > historiquement les solutions ne sont compatibles qu'avec des données structurées.
- identification difficile des bonnes solutions techniques :
  - > elles se multiplient et ont tendance à s'adresser à une problématique précise.

Les enjeux importants :

- **comment extraire de la valeur** des données brutes et **rester compétitif** ?
- **quelles solutions adaptées** aux volumes de données et usages envisagés ?
- **quelle intégration possible** dans un Système d'Information existant ?

Le public visé est donc très large :

- DSI,
- architecte,
- chef de projet,
- Data Scientist,
- AMOA , ...

## 6 — BIG DATA : PRÉSENTATION

Tout d'abord le Big Data n'est plus qu'une problématique de volume contrairement à ce que son nom laisse supposer.

Même s'il est vrai que le Big Data est capable de traiter des volumétries conséquentes, l'enjeu principal est la valorisation de ces données quelque soit leur volume.

Il y a des tentatives pour remplacer ce terme, soit par une dénomination technique : Fast Data, soit en mettant l'accent sur la valorisation des données : Smart Data.

### **Les cas d'utilisation les plus répandus des technologies Big Data :**

- analyse des logs des serveurs,
- recoupement entre différentes sources de données,
- analyse des informations produits, catalogue, ...
- sécurité des accès (analyse des requêtes du SI),
- analyse de risque ou panne,
- détection de fraude,
- détection de tendances (analyse sémantique / sentiment analysis),
- segmentation et ciblage (recommandations, Data Management Platform ...),
- recherche d'information dans des données non structurées
- analyse prédictive des comportements,
- prédiction de l'attrition pour mise en place de campagnes ciblées de fidélisation ou de reconquête,
- real-time marketing,
- contrôle de l'image et de la communication par analyse sémantique,
- optimisation opérationnelle de la chaîne d'approvisionnement,
- parcours client type,
- mesure de la performance des campagnes marketing,
- click and collect,
- click to chat,
- moteur de recommandation,
- contenu personnalisé,
- programme promotionnel personnalisé,
- consolidation des données clients (360°),
- ...

## 6.1. Chronologie du Big Data

Les enjeux et les challenges du Big Data sont loin d'être nouveaux. Si l'on s'intéresse à la chronologie des événements qui ont créé le Big Data tel que nous le connaissons aujourd'hui, on s'aperçoit que cette genèse remonte assez loin dans le temps et qu'il a fallu que les idées convainquent et que les solutions murissent afin d'en arriver là.

Date	Evénements
1997	Première apparition du terme Big Data (NASA pour désigner le challenge de travailler avec de large volume de données non structurées)
1998	Première base NoSQL (et utilisation du terme) par Carlo Strozzi
2000	Graph database Neo4j
2001	Première définition du Big Data (Volume / Variété /Vélocité) par Gartner
2005	Naissance Hadoop
2005	Naissance de CouchDB (base NoSQL orientée documents)

2006	Publication Google BigTable
2007	Naissance de Hbase (base orientée documents)
2007	Publication base orientée colonnes Amazon Dynamo
2007	Base de MongoDB (base orientée documents)
2007	Création société 10Gen (MongoDB)
2008	Naissance de Cassandra (base orientée documents)
2008	Création société Cloudera
2008	Hadoop bat le record « Terabyte sort Benchmark »
2009	Naissance de Flink
2009	Base clé/valeur Redis

2009	Création société MapR
2009	Naissance de Mesos (gestion des ressources)
2009	Naissance de Spark (analyse de données)
2010	Création société DataStax (Cassandra)
2011	Création société Hortonworks
2012	YARN en remplacement de Hadoop v1
2013	Création société DataBricks (Spark)
2014	Spark bat le record «Terabyte sort Benchmark»

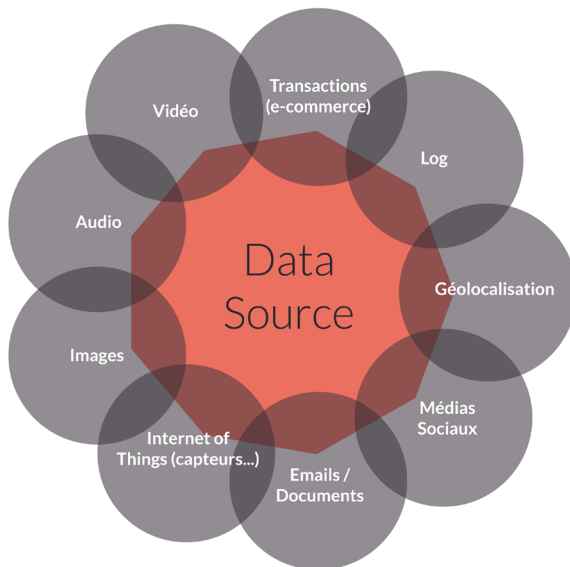
## 6.2. Données du Big Data

### 6.2.1. Sources des données

Voici une liste des sources de données les plus répandues dans le Big Data :

- données métier,
- historique d'utilisation de vos services par les utilisateurs,
- fichiers clients (Customer Relationship Management (CRM)),
- données fournies par les partenaires,
- réseaux sociaux,
- centres d'appels, ...

### LES SOURCES DES DONNÉES



## 6.2.2. Nature des données

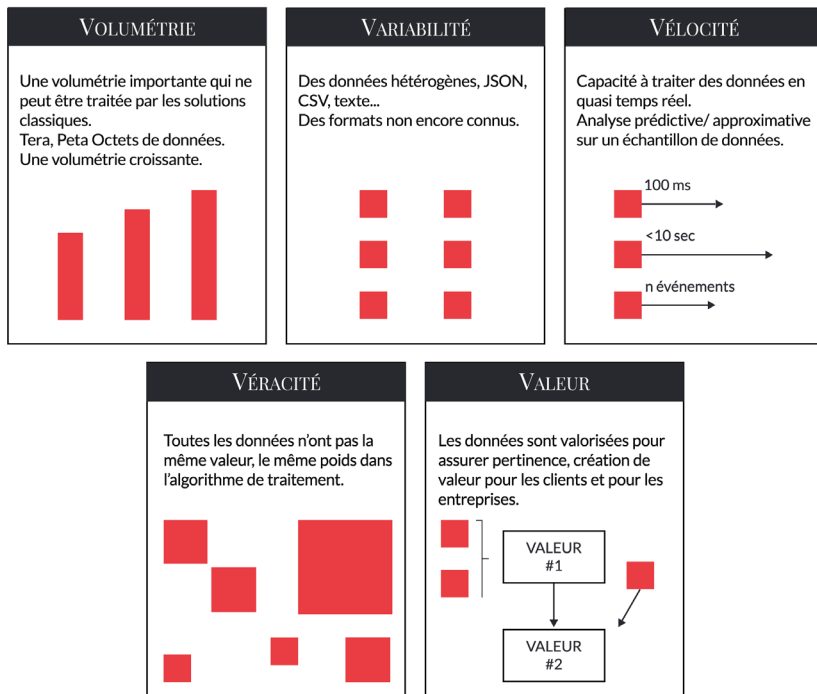
Voici une liste du type de données les plus répandues dans le Big Data :

- transactions (e-commerce),
- log,
- géolocalisation,
- social media,
- emails/documents,
- Internet of Things (capteurs, ...),
- images,
- audio,
- vidéo, ...

## 6.3. Les 5 V du Big Data



### DÉFINITION LA PLUS RÉPANDUE DU BIG DATA





### 6.3.1. Volume

Les entreprises font face à une augmentation exponentielle des données (jusqu'à plusieurs milliers de téra octets) :

- logs,
- réseaux sociaux,
- e-commerce,
- catalogue produit,
- analyse des données,
- monitoring, ...

Les technologies traditionnelles (Business Intelligence, bases de données) n'ont pas été pensées pour de telles volumétries.



**DATA DÉLUGE**



### 6.3.2. Variabilité/Variété

Les données à traiter dans une entreprise sont de natures multiples.

Exemple de données structurées :

- flux,
- RSS,
- XML,
- JSON,
- bases de données.

Ce à quoi peuvent s'ajouter des données non structurées :

- mails,
- pages web,
- multimédia (son, image, vidéo, etc.).

Ces données non structurées peuvent faire l'objet d'une analyse sémantique permettant de mieux les structurer et les classer, entraînant une augmentation du volume de données à stocker. La solution doit être évolutive car les formats de données ne sont pas tous actuellement connus (voir par exemple comment le format JSON a supplanté XML très rapidement).

 **VARIABILITÉ/VARIÉTÉ**



### 6.3.3. Vitesse

Dans certains cas l'accès et le partage des données doivent se faire en temps réel (on verra par la suite que la notion de temps réel varie selon les entreprises).

Une vitesse de traitement élevée permet d'offrir des capacités temps réel d'analyse et de traitements des données.

 **VÉLOCITÉ**



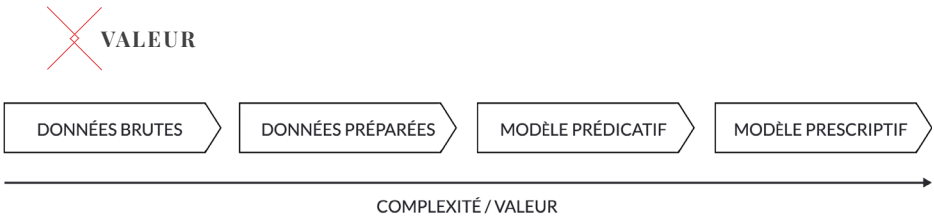
### 6.3.4. Valeur

C'est un point essentiel du Big Data car il va permettre de monétiser les données d'une entreprise.

Ce point n'est pas une notion technique mais économique.

On va mesurer le retour sur investissements de la mise en œuvre du Big Data et sa capacité à s'autofinancer par les gains attendus pour l'entreprise.

Plus on souhaite apporter de la valeur aux données, plus le coût et la complexité de la chaîne augmente :



### 6.3.5. Véracité

C'est la capacité à disposer de données fiables pour le traitement.

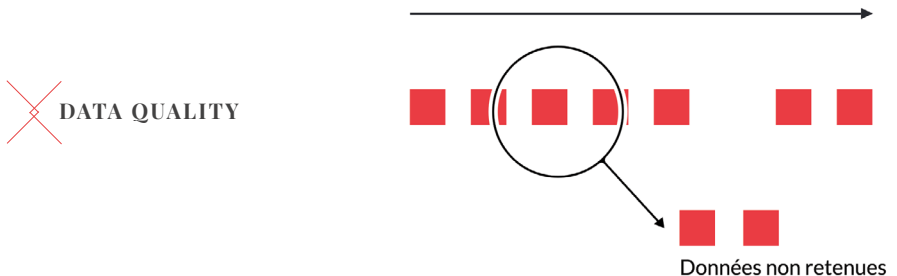
On va s'intéresser à la provenance des données afin de déterminer s'il s'agit de données de confiance.

En fonction du critère de confiance, on accordera plus ou moins d'importance à la donnée dans les chaînes de traitement.

Parmi les données dont il faut éventuellement se méfier on trouve les données des réseaux sociaux dont la provenance et l'objectivité est difficile à évaluer.

De plus même pour des données dont on connaît la provenance, la pondération n'est pas constante.

Par exemple il peut s'agir de données incomplètes, dont l'anonymisation a enlevé une partie de la valeur statistique ou encore de données trop anciennes.



## 6.4. Les acteurs du Big Data

Le Big Data a besoin de nouvelles compétences, il est donc normal de voir apparaître de nouveaux rôles :

**1- Data Engineer** : c'est l'informaticien, spécialiste du Big Data, qui va mettre en œuvre tous les outils et solutions à destination des utilisateurs (utilisateur final, data scientist, ...). Un Data Architect est un Data Engineer expérimenté apte à concevoir une architecture complète.

**2- Data Scientist** : poste à double compétence car il est capable d'utiliser les outils informatiques du Big Data (voire de coder en Python, R, ...) et de comprendre les enjeux business de ses analyses.

**3- Data Analyst (Statisticien)** : lorsque les analyses sur les données sont plus complexes il faut alors faire appel à des statisticiens qui sont capables d'implémenter de nouveaux algorithmes et définir de nouveaux modèles. Ses outils sont plus les modèles mathématiques que les outils de Data Mining.

**4- Chief Data Officer** : Dans les grandes entreprises on nomme parfois un directeur des données. Il est en charge des données de l'entreprise quelles soient internes ou externes :

- gouvernance des données.
- acquisition de nouvelles sources de données.

Mais son rôle principal est de monétiser les données de l'entreprise.

## 7 — FREINS ET OPPORTUNITÉS

Plusieurs sondages sur le Big Data permettent d'en savoir un peu plus sur les éléments qui empêchent certaines entreprises de franchir le pas.

Cf.

**<http://www.zdnet.fr/actualites/big-data-qu-est-ce-qui-bloque-encore-les-entreprises-39821966.htm>**

Parmi les points cités on trouve :

- coût,
- manque de compétences,
- manque de visibilité sur les opportunités,
- les entreprises n'ont pas cherché à quantifier le ROI des investissements Big Data (les investissements ne sont pas pondérés par les gains attendus),
- la collecte de la donnée est limitée aux canaux traditionnels,
- les données sont non structurées (et on ne sait pas la traiter).

A l'inverse voici les gains potentiels pour celles qui utilisent le Big Data.

Les entreprises les plus matures en matière d'exploitation des données clients se distinguent par les critères suivants :

- anticipation des enjeux stratégiques liés à une meilleure utilisation des données internes et externes,
- diversité des données collectées et des canaux de collecte,
- constitution d'équipes de Data Scientists et autres «experts data»,
- adoption de nouvelles technologies d'exploitation de la donnée,
- meilleure prise en compte des enjeux de protection de la vie privée et des données à caractère personnel dans l'exploitation des données clients.

L'étude suivante montre les gains constatés par les entreprises ayant mis en œuvre le Big Data :

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 %

Source : <http://barcresearch.com/research/bigdatausecases2015/>

## 7.1. Opportunités du Big Data

### 7.1.1. Les opportunités techniques

#### 7.1.1.1. Réduire les coûts

Réduire les coûts en réduisant les investissements matériels et logiciels (licence open source, matériel de type “commodity hardware”).

- Pour un même besoin de stockage il en coûtera 5 fois moins avec une solution NoSQL qu’avec une solution traditionnelle grâce à la diminution:
  - du coût des licences
  - du coût des machines
- Pour de l’analyse des données, le rapport est encore plus en faveur des technologies du Big Data par rapport aux systèmes traditionnels que sont les Data Warehouse et la BI.

#### 7.1.1.2. Améliorer les performances/scalabilité

La scalabilité est souvent “native” dans les solutions Big Data puisque cela faisait partie des challenges à résoudre.

La scalabilité horizontale permet à une architecture de répondre à une augmentation de la sollicitation par un ajout de machines. C’est la capacité de la solution à répartir la charge sur un ensemble de nœuds.

Concernant les performances, les solutions Big Data ont beaucoup d’avantages en utilisant certaines particularités des plateformes :

- co-localisation des données et traitements qui limitent les échanges réseau,
- utilisation de la mémoire plutôt que des accès disque,
- dénormalisation des données dans le stockage,
- ...

#### 7.1.1.3. Réduire le Time To Market (variabilité)

Avec plus de souplesse dans le support des données non structurées, des connecteurs existant et open sources, ...

Les plateformes Big Data vont permettre de faciliter les évolutions de type :

- ajout de nouvelles sources de données,
- interconnexion avec un silo existant,
- mise en œuvre d'un nouvel environnement (datalab, ...).

Ceci doit permettre de raccourcir le délai entre l'émergence d'une idée et sa mise en œuvre.

#### 7.1.1.4. Valorisation et collecte de données

Évidemment la mise en œuvre d'une architecture Big Data est tout d'abord une envie de collecter les données de l'entreprise, souvent dans un Data Lake (entrepôt universel des données) en vue de les analyser et les monétiser.

Ce n'est pas un nouveau besoin, mais les caractéristiques premières des architectures Big Data facilitent la mise en œuvre d'une telle plateforme:

- capacité à traiter des volumétries importantes,
- scalabilité des solutions,
- évolutivité des formats traités.

Un autre atout des solutions du Big Data est de proposer un écosystème complet et capable de traiter toutes les problématiques de la donnée (stockage, traitement, analyse, ...).



## 7.1.2. Les opportunités par secteurs d'activité

Secteur d'activité	Caractéristiques				Opportunités métier
	Volume	Vélocité	Variété	Véracité	
Secteur public	+++	++	+++	+++	Analyse fraude Analyse des risques
Service financier (Banque/ Assurance...)	+++	+++	++	+++	Fidélisation Analyse fraude Analyse réputation Scoring Analyse des risques
Santé	+	+++	++	+++	Analyse fraude Diagnostic Suivi épidémiologique
Télécoms	+++	+++	+++	++	Analyse fraude Ciblage marketing Analyse de conformité Optimisation des prix (dynamic pricing)

e-commerce	+++	+++	+	+++	<p>Fidélisation Churn (attrition)</p> <p>Réduire le TTM</p> <p>Analyse utilisateurs (vue client 360)</p> <p>Améliorer le taux de transformation en achat</p> <p>Optimisation des prix (dynamic pricing)</p>
Transport	++	+	++	+	<p>Optimisation trajets/ livraisons</p> <p>Maintenance prédictive</p> <p>Optimisation des prix (dynamic pricing)</p>
Énergie	+++	++	+	+	<p>Optimisation consommation énergétique</p> <p>Analyse utilisateurs (vue client 360)</p> <p>Ciblage marketing</p>

# 8 — MÉTHODES DE MISE EN ŒUVRE

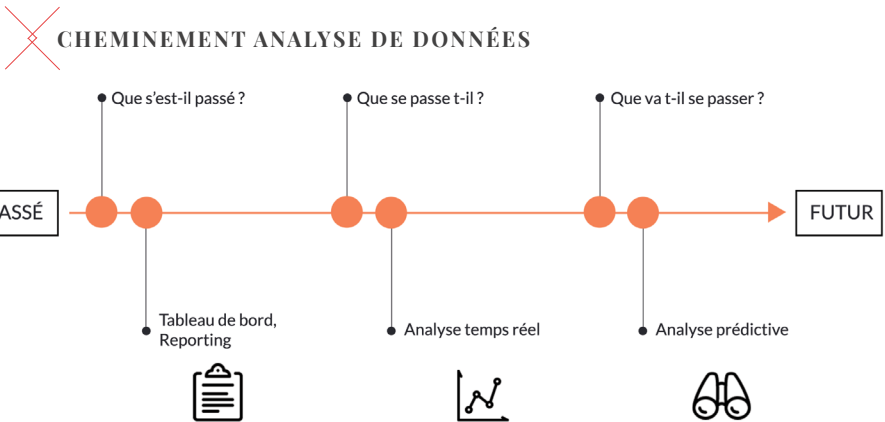
Voici quelques questions à se poser avant la mise en œuvre d'une plateforme Big Data :

- le coût de la collecte de la donnée,
- quelle est la plus value ?
- est-on prêt (conduite du changement, ...) ?

## 8.1. Cheminement analyse de données

Voici une représentation simplifiée et parlante de la mise en œuvre de l'analyse des données d'une entreprise qui définit trois niveaux de progression et de maturité

- **descriptive Analytics** : on tente de comprendre ce qui s'est passé
- **predictive Analytics** : on tente de prédire ce qui va se passer
- **prescriptive Analytics** : on est en mesure de simuler et de prédire des comportements afin de prendre des décisions métier.



## 8.2. Impacts sur organisation

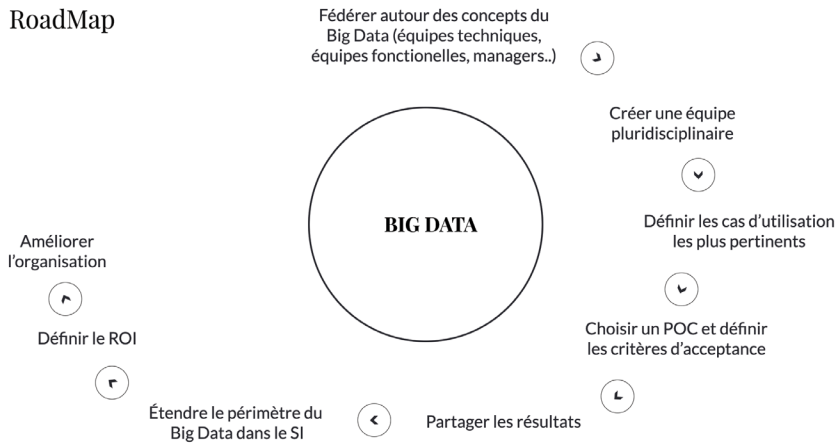
Tous les impacts listés ci dessous ne sont pas directement liés au Big Data mais d'un mouvement plus large. Toutefois il s'agit d'un mode de fonctionnement que l'on devrait retrouver dans les entreprises axées sur le Big Data (ou dans les départements spécialisés pour les grandes entreprises).

	Organisation classique	Organisation Big Data
Emergence, opportunités	Basées sur l'expression de besoin	Basées sur les opportunités métier
Organisation des développements	Approche top-down	Approche bottom-up
Cohérence du SI	Limitation de nombre de solutions voire un seule solution pour tous les besoins	La meilleure solution pour chaque besoin
Capitalisation	Réutilisation de composants	Composants et applications jetables
Organisation	Regroupement des compétences dans des pôles/ divisions	Approche pluridisciplinaire
Support et licence	Utilisation de solutions commerciales	Utilisation de solutions Open Sources
Montée en compétences	Formation classique	Hackathon, Coding Dojo...
Organisation du SI	Centrée sur les applications	Centrée sur les données
Préoccupations	Centrées sur l'entreprise	Centrées sur les clients

### 8.3. RoadMap



#### RoadMap



### 8.4. Approche Bottom-Up

Cette approche part du bas (la technique) pour aller vers le haut (l'organisation).

Avec cette approche on va tout d'abord valider les choix techniques au moyen d'un POC et d'un cas d'utilisation jugé pertinent.

On ne lance pas tout de suite les grands chantiers qui vont permettre de transformer l'entreprise et placer la donnée au cœur de toutes les problématiques.

Une fois la plateforme validée on peut enchaîner sur des domaines techniques annexes (analyse de la donnée, visualisation, ...) ou alors concrétiser rapidement un cas d'utilisation et ainsi apporter tout de suite de la valeur.

Cette organisation est hautement itérative tant techniquement que fonctionnellement.

La maîtrise technique des plateformes et l'analyse des données vont grandir conjointement au fur et à mesure de la complexification des cas

d'utilisation.

C'est évidemment la méthode qui apporte le plus rapidement des résultats et permet de gagner l'adhésion de tous au fur et à mesure.

A l'opposé cette approche peut donner l'impression de naviguer à vue et la visibilité est limitée.

## **8.5. Approche Top-Down**

Complètement à l'opposé de la méthode précédente l'approche top down va d'abord impacter l'organisation de l'entreprise, la transformer, afin de lui permettre de lancer des projets Big Data.

On va définir une stratégie Big Data pour l'entreprise complète, un planning de mise en œuvre des objectifs concrets qui se traduisent souvent par de nouvelles offres pour l'entreprise ou bien l'amélioration des offres existantes.

Cette approche va tout de suite mobiliser des équipes plus importantes au sein de l'entreprise.

Avec cette approche les résultats concrets sont plus longs à obtenir (l'inertie est plus forte et l'effet tunnel réel).

A l'opposé les objectifs, les responsabilités ainsi que les sponsors sont clairement identifiés dès le départ.

## 8.6. Conseil pour l'expérimentation

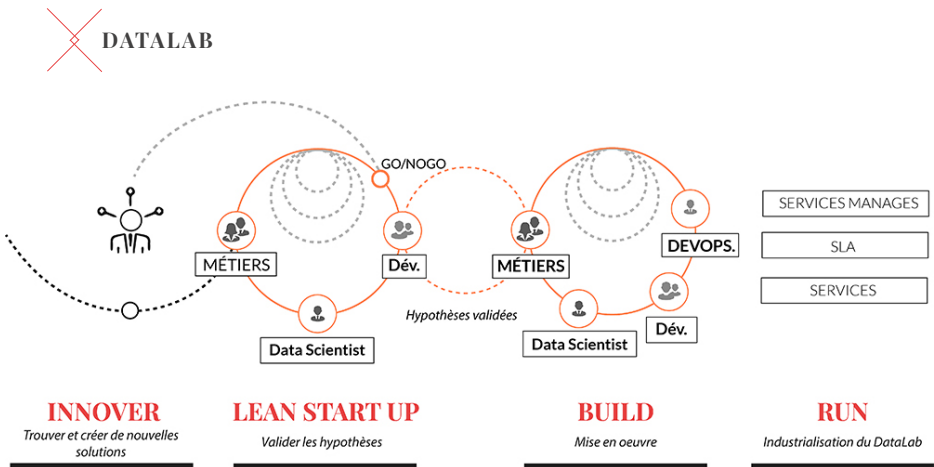
### Cas d'utilisation :

Choix du premier cas à implémenter :  
sélectionner le plus critique et le plus intéressant  
en termes de plus value.

### Approche technique et généralisation :

- commencer petit et itérer,
- lors de cette phase d'expérimentation, n'hésitez pas à ne retenir que les solutions/concepts qui fonctionnent.

Voici par exemple la démarche proposée par Ippon Technologies lors de la mise en œuvre d'un DataLab.



## 9 — MISE EN ŒUVRE

Le Big Data a vocation à traiter des problématiques métiers complexes. Le Big Data déplace le centre d'intérêt des entreprises vers les données et la valeur qu'elles peuvent apporter à l'entreprise.

Pour les entreprises les moins matures, l'exploitation de la donnée sera tout d'abord une dette :

- coût de l'acquisition et du stockage de données,
- coût matériel, logiciel,
- coût humain (recrutement, montée en compétences).

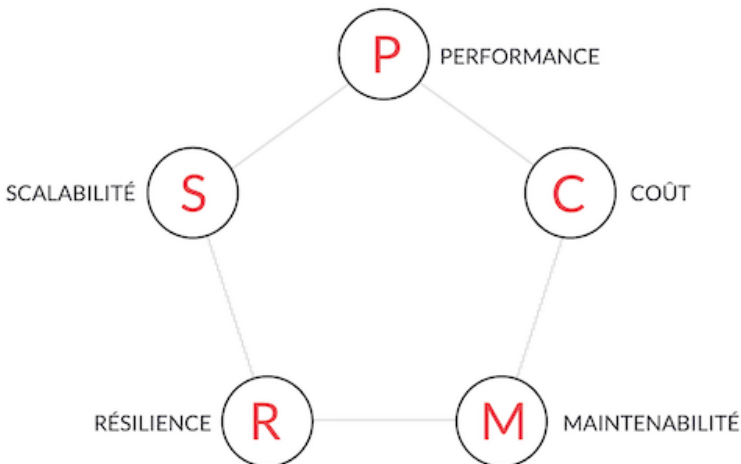
Ce n'est que par la suite que le retour sur investissement pourra être possible.

Il faut passer d'une entreprise pilotée par les projets à une entreprise pilotée par les données.

### 9.1. Principes d'une architecture Big Data

Il n'existe pas un modèle d'architecture Big Data idéal adapté à tous les usages mais des grands principes.

#### ~~X~~ PRINCIPES D'UNE ARCHITECTURE BIG DATA





## 9.1.1. Stockage

### 9.1.1.1. Données immutables (P/R)

Avoir des données immutables, c'est à dire permettre l'insertion de nouvelles données mais pas leur modification dans un datastore (fichier, table, ...).

Ce principe peut paraître contre productif et très contraignant mais il simplifie la distribution des données et les performances en écritures. C'est le modèle adopté par Cassandra, par HDFS.

Un autre avantage concerne la résilience du système aux fausses manipulations, aux bugs, ...

Avec un système immutable les données précédentes sont toujours présentes et seules les nouvelles données sont impactées.

En cas de problème on peut donc toujours restaurer la version n-1 des données.

Soit ce principe est supporté par les solutions de stockage, soit il devra être implémenté (staging des données, CQRS, ...).

### 9.1.1.2. Stockage distribués des données (R)

Ce principe va assurer la disponibilité des données et la capacité du système à gérer les défaillances.

Une même donnée va être répliquée plusieurs fois dans le système sur des nœuds différents.

En général le respect de ce principe va intervenir au moment du choix de la solution de stockage et rarement au moment de l'implémentation.

### 9.1.1.3. Dénormalisation des données (P)

Dans le Big Data, la modélisation des données est très importante pour les raisons suivantes :

- performances (redondance des données),
- adaptation aux solutions utilisées.

Un système de stockage plus souple ne signifie pas pour autant une phase de modélisation raccourcie ou simplifiée.

Il est courant de construire une vue spécifique à un besoin afin de garantir les performances.

#### 9.1.1.4. Modèle non relationnel (P)

La plupart des solutions NoSQL ne supportent pas les relations pour des raisons de performances.

Cela a évidemment un impact sur la phase de modélisation ainsi que sur la manière d'utiliser les systèmes de stockage.

Cela dit toutes les architectures n'utilisent pas les systèmes NoSQL et l'utilisation de solutions relationnelles telles que les SGBD-R peut s'avérer plus pertinente dans certains cas.

#### 9.1.1.5. Modèle de données dynamique (M)

L'évolution du modèle de données est un challenge difficile qui peut être traité avec les systèmes traditionnels.

Les nouveaux systèmes de stockage simplifient ce besoin et favorisent l'évolutivité du modèle de données.

#### 9.1.1.6. Durée de vie des données automatique (M)

Les nouveaux systèmes de stockage (NoSQL, grilles de données mémoire) permettent de gérer le cycle de vie des données de manière automatisé (en proposant un Time To Live).

Les avantages sont nombreux :

- réduction des tâches de maintenance (plus de script métier de purge des données),
- réduction de la volumétrie de stockage,
- le flot de suppression des données est continu et non plus discret (passage de batchs),
- favorise le renouvellement et donc la pertinence des données.

## 9.1.2. Traitements

### 9.1.2.1. Parallélisation des traitements (P/S)

Lorsque les traitements sont indépendants entre eux (du moins en partie) il est alors possible de les paralléliser (différents thread, différents nœuds).

C'est le principe de nombreux algorithmes tels que MapReduce, Fork/Join, ...

Les avantages :

- Performances, les données sont traitées plus rapidement,
- Évite la congestion d'un système en répartissant les efforts.

### 9.1.2.2. Prendre en compte la topologie réseau (P/S)

Cette capacité d'un système à prendre en compte la topologie réseau (Data Locality) lui permet de colocaliser données et traitements.

Les données sont manipulées au plus près de leur stockage ce qui permet :

- de répartir les traitements sur un cluster,
- de minimiser les échanges réseaux.

### 9.1.2.3. Traitements Asynchrones (P/S)

La capacité d'un système à gérer l'asynchronisme (systèmes non bloquants) permet d'augmenter les performances d'un point de vue client en augmentant le nombre de requêtes.

### 9.1.2.4. Failover/reprise automatique (S/R)

Un système réparti, gérant nativement le failover, limite les interventions, et favorise grandement sa résilience à la panne.

Il en résulte un système qui offre une garantie de service (SLA) plus importante.

#### 9.1.2.5. Pas de transactions (P/S)

Les nouveaux systèmes de stockage (NoSQL, grilles de données mémoire) ne permettent pas toujours de gérer les transactions.

Ce qui peut être vu comme un frein est un énorme avantage en termes de performance et de scalabilité.

Toutefois cette particularité peut impacter la cohérence des données et suppose un effort supplémentaire dans la conception du modèle ainsi que sa prise en compte dans le système (transaction de compensation, ...)

### 9.1.3. Architecture

#### 9.1.3.1. Solutions majoritairement open sources (C)

Le retour sur investissement d'une architecture Big Data est un moteur important de sa mise en œuvre.

L'utilisation de solutions open sources va permettre de réduire les coûts de mise en œuvre mais aussi favoriser l'évolutivité de la plateforme en évitant les solutions propriétaires.

#### 9.1.3.2. Scalabilité horizontale (commodity hardware) (C/S)

C'est à dire la capacité d'un cluster à augmenter sa puissance avec l'ajout de machines.

Cela traduit la capacité d'un système à distribuer automatiquement les traitements et les données sur les ressources du cluster.

#### 9.1.3.3. Pas de SPOF (masterless) (P/S)

Dans une architecture un Single Point Of Failure met à défaut plusieurs points importants :

- résilience à la panne du système,
- performances et scalabilité.

Dans l'idéal, aucune défaillance d'un nœud ou d'un service ne met en défaut le bon fonctionnement du cluster.

Cela suppose soit une architecture complètement décentralisée ou chaque nœud peut tenir n'importe quel rôle.

Soit cela passe par la mise en œuvre de nœuds secondaires qui en cas de défaillance prendront le relais.

#### 9.1.3.4. Extensible (M)

Bien évidemment, personne ne peut prévoir quelles seront les normes, les cas d'usage, les solutions de demain.

Il faut donc au maximum respecter les standards et normes actuelles (quelles soient réelles ou de facto) et éviter les formats et les normes propriétaires.

Ce principe est important pour garantir l'évolutivité de la plateforme.

#### 9.1.3.5. Répétable/déployable sur tous les environnements (M/R)

On verra par la suite l'importance des tests dans les environnements Big Data. Cela passe donc par des environnements représentatifs du comportement de la solution en production.

Il faut donc s'assurer :

- de disposer du même OS dans tous les environnements,
- de disposer d'une architecture similaire (cluster notamment),
- de disposer d'une configuration identique,
- de disposer d'une volumétrie identique,
- de ne pas être contraints par des problèmes de licences (un environnement d'intégration est-il soumis à une licence commerciale ?).

L'idéal est de s'appuyer sur des systèmes de containerisation comme Docker afin de s'assurer de la reproductibilité sur tous les environnements.

#### 9.1.3.6. Share nothing/Stateless (P/S)

En termes de scalabilité et de résistance à la charge les architectures stateless ont démontré de meilleures performances. Ce qui est vrai pour les architectures web l'est aussi pour les architectures Big Data.

Le meilleur exemple est Kafka qui, dans certains cas, va déléguer la gestion des messages consommés coté client et non coté serveur.

### 9.1.3.7. Push vers les référentiels secondaires (M)

L'alimentation des référentiels secondaires tels que les moteurs d'indexation doit se faire par mode push.

La source de référence (base NoSQL, Data Lake Hadoop, ...) est en charge de l'alimentation des référentiels secondaires.

Cela peut être simplement géré par les possibilités de synchronisation des solutions utilisées.

Avantages :

- fraîcheur de la donnée,
- contrôle de la chaîne d'alimentation.

## 9.1.4. Récapitulatif



### PRINCIPES D'UNE ARCHITECTURE BIG DATA

ARCHITECTURE	
Solutions majoritairement Open Sources	C
Scalabilité horizontale (commodity hardware)	C-S
Pas de SPOF (Masterless)	P-S
Extensible	M
Répétable/ déployable sur tous les environnements	M-R
Share nothing/ stateless	P-S
Push vers les référentiels secondaires	M
TRAITEMENTS	
Parallélisation des traitements	P-S
Prend en compte la topologie réseau	P-S
Traitement asynchrone	P-S
Failover/ reprise automatique	S-R
Pas de transactions	P-S
STOCKAGE	
Données immutables	P-R
Stockage distribué des données	R
Dénormalisation des données	P
Modèle non relationnel	P
Modèle de données dynamique	M
Durée de vie des données automatique	M

## 9.2. Théorème CAP

Le théorème CAP part d'une conjecture énoncée par le chercheur en informatique Eric Brewer (université de Berkeley) en 2000.

En 2002, Seth Gilbert et Nancy Lynch du MIT publient une preuve formelle de la vérifiabilité de la conjecture de Brewer.

Le théorème CAP dit qu'il est impossible sur un système informatique de calcul distribué de garantir en même temps les trois contraintes suivantes :

- **consistency** : Tous les nœuds du système voient exactement les mêmes données au même moment;
  - En cas d'écriture sur le nœud A, une lecture sur le nœud B renvoie la nouvelle valeur instantanément (ce qui exclut les architectures décentralisées).
- **availability** : Garantie que toutes les requêtes reçoivent une réponse (succès ou échec);
  - Tous les nœuds du cluster peuvent adresser les lectures et les écritures (ce qui exclut les architectures centralisées).
- **partition** : Aucune défaillance de tout ou partie des nœuds du cluster ne doit empêcher le système de répondre correctement. En cas de morcellement en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome.

D'après ce théorème, un système de calcul distribué ne peut garantir à un instant  $t$  que deux de ces contraintes mais pas les trois ( $t$  est important et ne supporte aucun délai même infinitésimal).

On va donc retrouver trois catégories de solutions, avec chacune leur points forts et leur faiblesses qui doivent orienter votre choix en fonction du cas d'utilisation :

- systèmes "CA",
- systèmes "CP",
- systèmes "AP".

NB : La tolérance à la partition est une composante essentielle des systèmes distribués, elle est donc de fait dans tous les systèmes

distribués. Les systèmes de type “CA” sont tout de même évoqués car toutes les composantes d’une application n’ont pas vocation à être distribuées.

### 9.2.1. Systèmes “CP”

Ces systèmes vont privilégier la consistance plutôt que la disponibilité. Ils vont éviter à tout prix de retourner une donnée périmée (i.e. pas la dernière modification), ils vont préférer retourner une erreur.

Si la donnée est présente sur n nœuds alors les n nœuds doivent être opérationnels.

Les cas d’utilisation privilégiés de ces solutions sont les systèmes où la valeur de la donnée est préférable à une haute disponibilité. Ils ne sont donc pas conseillés dans des systèmes de e-commerce par exemple mais conseillés pour un site bancaire ou de santé par exemple.

Ex : MongoDB, HBase, Hazelcast

### 9.2.2. Systèmes “AP”

Ces systèmes vont plutôt privilégier la disponibilité plutôt que la consistance même si le choix est souvent laissé au moment de la configuration du système (consistance variable). Ces systèmes ne vont donc pas obligatoirement retourner la dernière valeur d’une donnée (en cause le temps de réplication de la valeur dans le cluster). Ce délai (appelé entropie) est souvent très faible (de l’ordre de la ms) mais il est bien réel. Ces systèmes sont donc à privilégier quand la disponibilité est la problématique principale (site marchand par exemple).

Ex : Cassandra, Riak, CouchDB, Redis

### 9.2.3. Systèmes “CA”

On va trouver dans cette catégorie tous les systèmes de type maître/esclave ou non distribués. Les données ne sont pas répliqués ou le sont obligatoirement de manière synchrone.

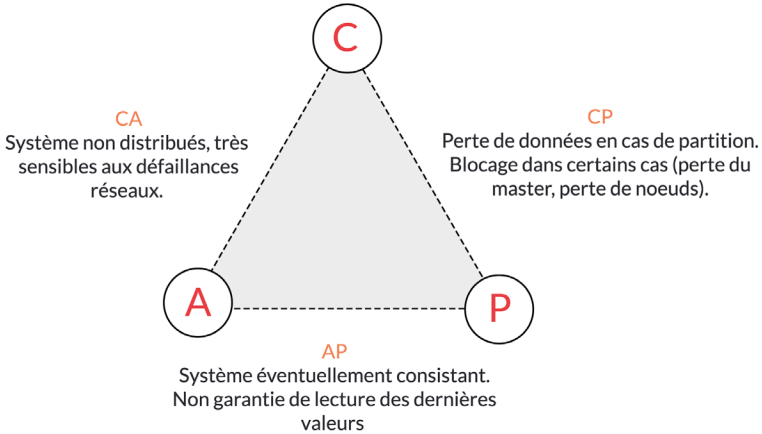
Les performances sont moins importantes que la fraîcheur et la disponibilité des données.

Ces systèmes sont donc à éviter pour les systèmes temps réels. En cas



de crash, l'indisponibilité peut être conséquente.  
Ex : Base de données, LDAP.

## THÉORÈME CAP



### 9.3. Garantie livraison/traitement des systèmes distribués

Les systèmes de traitements distribués comme Spark ou Flink sont souvent catalogués selon les garanties de livraison/traitement des messages :

Type de garantie	Détails
Exactly once delivery/processing	Chaque message est délivré/traité une et une seule fois
At least once delivery/processing	Chaque message peut être délivré/traité plusieurs fois
At most once delivery/processing	Chaque message est délivré/traité au maximum une fois (pertes possibles)

Idéalement nous souhaitons un système de type “Exactly once delivery/processing”.

Sans rentrer trop dans les détails, sachez que pour ce genre de systèmes :

- exactly-once delivery est impossible en conditions dégradées,
- exactly-once processing of messages est possible en conditions dégradées.

Mais le plus important est le traitement unique d'un message soit le respect de la règle "Exactly-Once Processing".

En général les systèmes distribués vont garantir :

- "exactly once" en conditions normales,
- "at least once" en conditions dégradées.

Pour garantir un niveau plus élevé les systèmes utilisent un système externe de persistance des messages comme Apache Kafka.

## 9.4. Traitements Big Data

Il y a trois grandes familles de traitement dans le Big Data :

- batch,
- micro-batch,
- temps réel (streaming).

### 9.4.1. Batches

Les traitements vont analyser l'ensemble des données disponibles à un instant t.

- données en entrée : fichiers, résultat d'une requête (HDFS, Sqoop, ...),
- résultats : les résultats ne seront disponibles qu'à la fin des traitements,
- latence : souvent de l'ordre de la minute.

Exemple d'implémentation : MapReduce

## 9.4.2. Micro-batches

Les traitements vont analyser l'ensemble des données disponibles toutes les n secondes.

- données en entrée : petits fichiers, API Web, ...
- résultats : les résultats ne seront disponibles qu'à la fin des traitements d'un micro-batch,
- latence : souvent de l'ordre de la seconde.

Exemple d'implémentation : Spark streaming

## 9.4.3. Temps réel

Les traitements vont analyser les données au fur et à mesure de leur disponibilité.

- données en entrée : petits fichiers, API Web, ...
- résultats : les résultats sont disponibles au fur et à mesure,
- latence : parfois inférieur à la seconde.

Exemple d'implémentation : Flink, Tez, Storm

## 9.5. Pré requis matériel

### 9.5.1. Dimensionnement des serveurs

Big Data ne veut pas dire Big Hardware, en général on parle de "Commodity hardware", c'est à dire de serveurs moyenne gamme car l'atout de cette technologie est la scalabilité horizontale.

Évidemment ce point est à modérer car si l'on suit les recommandations des éditeurs, les machines doivent comporter un nombre élevé de cœurs et une capacité en RAM importante.

## Disques :

Les disques locaux sont à privilégier (plutôt que SAN) afin de faciliter la colocalisation des traitements et des données.

De même il vaut mieux multiplier les disques durs que d'augmenter leur capacité et les technologies de mirroring (RAID) sont rarement nécessaires pour la redondance (elle est gérée par le système de stockage).

La technologie SSD est bien sûr recommandée pour les fichiers accédés fréquemment (journal des modifications pour les systèmes NoSQL).

### 9.5.2. Utilisation des puces graphiques (GPU)

Une tendance qui commence à se dessiner est l'utilisation des GPU en complément des CPU pour les traitements.

De part leur architecture les GPU sont particulièrement adaptés aux traitements Big Data, ils sont composés d'un maximum de puces dédiées aux traitements parallèles, de milliers de cœurs, ... Ils sont donc idéaux pour les calculs répétitifs et mathématiques.

A l'opposé les CPU ne gèrent pas uniquement les traitements dans un serveur et leur nombre de cœurs est limité.

Les GPU ne remplacent pas les CPU mais permettent une architecture hybride pour le Big Data.

Il existe des expérimentations Hadoop sur GPU menées par Yahoo pour du Deep Learning.

Ils ont constaté un gain important (x10) pour ce type de traitement par

rapport à des architectures sur CPU.

Il existe déjà des initiatives sur le cloud :

- Amazon propose des clusters GPU pour son offre cloud EC2 (l'utilisateur choisit entre CPU et GPU)

Deep-learning sur Spark à l'aide de GPU : <http://fr.slideshare.net/SparkSummit/a-scaleable-implementation-of-deep-learning-on-spark-alexander-ulanov>

### **9.5.3. Virtualisation**

La virtualisation n'est pas mise en avant par la majorité des solutions Big Data qui recommandent souvent une approche de type « bare metal ».

La virtualisation est toutefois supportée par la plupart des solutions. Il est à noter qu'en termes de coût de licence elle n'est pas toujours avantageuse (système de facturation qui ne tient pas compte des ressources allouées aux machines virtuelles mais des ressources physiques).

# 10 — LES SOLUTIONS

## 10.1. Évolution des solutions Big Data

### 10.1.1. Traitement

Hadoop a posé les bases du Big Data (surtout en termes de traitements). La plateforme est capable de traiter des volumes important de données, mais avec une latence importante.

C'est pourquoi sont apparus des systèmes temps réels tels que Spark, Storm, Flink, Druid, ou encore Tez.

L'enjeu étant de conserver les points forts de la plateforme Hadoop :

- capacité à traiter des quantités énormes de données,
- sécurité,
- distribution,
- tolérance à la panne.

Tout en augmentant les possibilités d'interactions grâce à des traitements temps réels.

Les solutions temps réel améliorent les performances grâce à une meilleure utilisation de la mémoire mais aussi (surtout) en offrant la possibilité de définir des fenêtres de traitements (micro batch, windowing) ainsi que des traitements itératifs.

### 10.1.2. Stockage

Côté stockage, il y a émergence des solutions NoSQL.

Même si certaines solutions se détachent en termes de part de marché (MongoDB, Cassandra, CouchDB, ...) il n'y a pas de solutions universelles capables de répondre à tous les besoins.

Une erreur classique héritée de pratiques répandues dans le monde des SGBD-R ou des serveurs d'applications consiste à ne retenir pour l'ensemble d'un SI qu'une seule solution.

Cette approche était déjà ambiguë et souvent contre productive avant, elle l'est encore plus dans le monde du Big Data.

**Il n'y a rien de pire que l'utilisation d'une solution en dehors de son cadre d'utilisation, c'est typiquement le genre de pratique qui peut faire perdre beaucoup de temps et aboutir à une solution instable.**

Évidemment cela n'est pas vrai pour l'ensemble des entreprises qui n'ont pas des besoins nécessitant plusieurs implémentations mais c'est un écueil fréquent.

### 10.1.3. Interrogations

Le (pseudo)SQL tente de s'imposer comme standard pour l'interrogation des données stockées dans le Big Data.

C'est le cas des solutions comme Hive, Drill, ou Spark SQL, ...

Même s'il manque une normalisation entre les différentes implémentations du pseudo-SQL, l'apprentissage est rapide pour quiconque possède une base dans la syntaxe SQL.

Au final le but est de permettre une exploration interactive des données, et d'ouvrir la plateforme à une population plus large que celle des programmeurs.

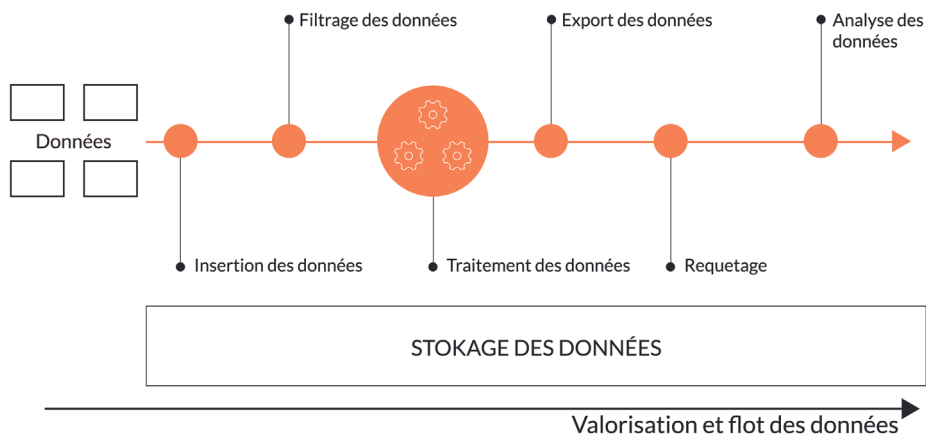
## 10.2. Catégories

Catégories des solutions Big Data :

- ingestion/Extraction de données,
- traitement de données,
- analyse/Apprentissage ,
- data visualisation,
- requête,
- workflow,
- stockage,

- ordonnancement,
- sécurité,
- gouvernance,
- messages,
- frameworks.

## ~~TRAITEMENTS~~



## 10.3. Ingestion/Collecte

### 10.3.1. Talend

Site web	<a href="http://www.talend.com">http://www.talend.com</a>
Licence	GPL et commerciale
Technologie	Java
Année création	2005

Talend est un ETL open source, développé par la société française Talend. Il permet de créer graphiquement des processus de manipulation et de transformation de données puis de générer l'exécutable correspondant sous forme de programme Java.



Talend permet :

- la synchronisation entre deux systèmes de données,
- la transformation des données.

Connecteurs :

- SGBD,
- fichiers,
- Base NoSQL,
- Hadoop,
- Spark
- ...

### 10.3.2. Sqoop

Site web	<a href="http://sqoop.apache.org"><b>http://sqoop.apache.org</b></a>
Licence	Apache
Technologie	Java
Année création	2009

Sqoop est un projet de la fondation Apache qui a pour objectif de permettre une meilleure cohabitation des systèmes traditionnels de type SGBDs avec la plateforme Hadoop.

Il est ainsi possible d'exporter des données depuis la base de données et de procéder aux traitements coûteux en exploitant le cluster Hadoop. Les bases de données traditionnelles sont très répandues et Sqoop permet à des applications traditionnelles de profiter d'un cluster Hadoop.

Inversement, il est possible d'exporter le résultat d'un traitement vers une base de données tierce afin qu'il soit exploité par une application (à des fins de restitution par exemple).

Sqoop a été conçu avec comme objectif principal d'assurer des performances élevées pour ces opérations d'import ou d'export massifs

grâce à la parallélisation des traitements.

Un job Sqoop c'est :

- une requête de sélection des données dans la cible,
- un mapping entre source et cible.

Datastax fournit une version de Sqoop capable de s'interfacer avec Cassandra

## 10.4. Messages

### 10.4.1. Kafka

Site web	<a href="http://kafka.apache.org"><b>http://kafka.apache.org</b></a>
Licence	Apache
Technologie	Java
Année création	2009

Kafka est un broker de messages crée par LinkedIn à la fois très performant et scalable.

Kafka est un système de messagerie distribué, initialement développé par l'équipe de Jay Kreps chez LinkedIn, et publié en 2011, en tant que projet open source Apache.

Les créateurs du projet ont ensuite fondé la société Confluent en 2014 qui propose du support mais aussi sa propre plateforme de streaming. Kafka s'appuie sur le système de fichiers pour gérer les événements reçus.

La file de messages est donc persistée (et les données distribuées) et reconsultable à tout moment.

Contrairement à ce qu'on retrouve dans les systèmes de messagerie traditionnels, un message stocké dans un cluster Kafka ne possède pas d'identifiant. En fait, chaque message d'une partition se voit assigné un offset (ou position) unique permettant de le localiser au sein d'un log. C'est notamment de cette manière qu'un cluster Kafka peut ainsi

supporter un nombre très large de consommateurs tout en retenant un très gros volume de messages.

#### 10.4.1.1. Concepts

- Broker : un service Kafka,
- Topic : une file de messages,
- Log : stockage des messages,
- Zookeeper : gestion du clustering.

#### 10.4.1.2. Consommation des messages

Kafka propose la gestion de groupe de consommateur, qui sont des clients abonnés à un même topic. Ceci qui permet de ne traiter qu'une seule fois un message d'un topic lu par plusieurs consommateurs.

Kafka propose trois API pour les consommateurs :

- Low Level,
- High Level,
- New Consumer (en bêta depuis la 0.9).

Ces API permettent aux clients de mieux contrôler la façon de consommer les messages et notamment la gestion du message courant (offset). Si la perte de message n'est pas primordiale alors on peut opter pour une gestion automatisée de l'offset : quand un message est lu par un consommateur alors la position courante évolue automatiquement (au bout d'un certain délai le plus souvent). Si jamais le consommateur crash avant d'avoir pu traiter le message alors ce message ne pourra plus être traité (en tout cas sans code particulier).

Au contraire on peut opter pour une gestion coté client de l'avancement dans la file de message, le client ne faisant évoluer l'offset qu'après avoir traité le message. C'est notamment le cas avec le connecteur Spark/Kafka (direct stream) qui va même plus loin puisqu'il ne modifie pas du tout l'offset. En cas de crash tous les messages sont relus et donc traités. Toutefois Kafka ne conserve pas indéfiniment les messages, ceux-ci ont une durée de vie au sein des topics (168 heures soit 7 jours par défaut).

#### 10.4.1.3. Partitions

Kafka étant un système distribué, les topics sont partitionnés à travers plusieurs nœuds du cluster.

Kafka diffère des systèmes de messagerie traditionnels par sa capacité à traiter chaque topic comme un log, c'est-à-dire un ensemble ordonné de messages.

En raison de sa scalabilité et de ses performances, Kafka est le système privilégié par de nombreux systèmes (Apache Spark, Flink, ...) pour assurer la reprise sur erreurs.

## 10.4.2. Redis

Site web	<a href="http://redis.io">http://redis.io</a>
Licence	BSD
Technologie	C/C++
Année création	2009

Redis est un dépôt de données clé/valeur issue de la mouvance NoSQL et créé en 2009 par Salvatore Sanfilippo et Pieter Noordhuis.

Le projet est sponsorisé par VMware et Pivotal.

Une des principales caractéristiques de Redis est de conserver l'intégralité des données en mémoire. Cela permet d'obtenir d'excellentes performances en évitant les accès disques, particulièrement coûteux.

Lorsque la taille des données est trop importante pour tenir en mémoire, Redis peut également utiliser de la mémoire virtuelle (swap).

### 10.4.2.1. Tolérance à la panne

Redis s'appuie sur deux mécanismes afin d'assurer la continuité de service ainsi que la non perte de données malgré son stockage uniquement en mémoire.

- sauvegarde :

Il est possible de créer régulièrement une image de l'état des données dans un fichier (principe du snapshot).

- journal des modifications :

Il est également possible de conserver une trace de toutes ces manipulations. En cas d'incident, la base peut être restaurée en rejouant dans l'ordre les événements.

#### 10.4.2.2. Réplication des données

Redis supporte la réplication via un modèle maître/esclave à des fins de résistance aux pannes et de répartition de la charge. Toutes les écritures doivent se faire via l'instance maîtresse, mais il est possible de faire des lectures sur les instances esclaves.

#### 10.4.2.3. Partitionnement (sharding)

Il est possible, afin d'améliorer les performances en écriture, de partitionner les données au sein des différentes instances Redis du cluster. Deux modes sont proposés pour la répartition des données en fonction de la valeur de la clé : intervalle ou hash.

Les deux modes partitionnement et réplication ne sont pas compatibles.

#### 10.4.2.4. Clients natifs :

- C#,
- C++,
- Clojure,
- Java,
- Javascript,
- Objective-C,
- Python,
- R,
- Scala, ...

En termes de commandes et de manipulation des données Redis va bien au delà d'un système simple de type CRUD :

- richesse de manipulation des chaînes de caractères,
- richesse de manipulation des listes :
  - récupérer le premier/dernier élément d'une liste

- extraire et insérer un élément d'une liste à l'autre
- requêtes plus complexes (géo-spatiales, agrégations, mathématiques, ...)
- hyperLogLog, Compteurs, ...

Redis propose aussi une interface de type REST pour la manipulation des données.

#### 10.4.2.5. Fonctionnalités avancées

- transactions,
- enchaîner des commandes (pipeline).

#### 10.4.2.6. Sécurité

Il est possible de sécuriser Redis avec un mot de passe obligatoire. Ce mot de passe est commun à l'ensemble des clients et il n'est pas possible de définir de rôles.

## 10.5. Stockage

### 10.5.1. Cassandra

Site web	<b><a href="http://cassandra.apache.org">http://cassandra.apache.org</a></b>
Licence	Apache
Technologie	Java
Année création	2008

Cassandra est une autre base de données de la mouvance NoSQL. Initialement développée par Facebook en 2008, elle a été par la suite libérée et son développement est aujourd'hui assuré par la fondation Apache. La société Datastax fournit du support ainsi qu'une version Entreprise avec quelques fonctionnalités supplémentaires.

Cassandra est une base de données orientée colonne. Étudiée pour des déploiements massivement distribués (éventuellement sur plusieurs datacenters).

Cassandra permet de stocker un élément qui est un ensemble de valeurs reliées entre elles par un identifiant unique.

Une valeur dans Cassandra est caractérisée par :

- rowKey (identifiant unique),
- nom colonne,
- valeur colonne,
- + Timestamp automatiquement géré par Cassandra,
- + Date d'expiration de la donnée (option).

Son architecture complètement décentralisée lui assure une résistance à la panne très importante.

Elle est particulièrement adaptée aux problématiques "timeseries" (suite de données chronologiques).

#### 10.5.1.1. Les différentes distributions

- Apache Cassandra
  - licence : Apache License, Version 2.0
  - comprend la base de données et les utilitaires
- DataStax
  - DSC : Version communautaire (Apache Cassandra + exemples + Ops Center light)
  - DSE : Version entreprise

#### 10.5.1.2. Cycle de stockage d'une écriture

- Écriture dans le commit log,
- Écriture en mémoire : Memtable,
- Écriture sur disque : SSTable.

#### 10.5.1.3. Tolérance à la panne

Cassandra s'appuie sur deux mécanismes afin d'assurer la continuité de service ainsi que la non perte de données :

- Sauvegarde

Il est possible de créer une image de l'état des données dans un fichier.

- Journal des modifications

Le commit log sert à stocker les modifications, avant qu’elles ne soient stockées dans les SSTables lors de la compaction. Il est ensuite purgé.

#### 10.5.1.4. Réplication des données

Cassandra supporte la réplication via un modèle “masterless” à des fins de résistance aux pannes et de répartition de la charge. Les nœuds se partagent les données qui sont répliquées un certain nombre de fois. Tous les Replicas ont le même rôle, on peut lire/écrire sur n’importe quel nœud, le nœud contacté sert de coordinateur et est en charge de solliciter les nœuds qui hébergent (lecture), hébergeront (écriture) la donnée.

Toutefois pour des raisons évidentes de performances, le driver contacte directement les nœuds concernés (grâce au token).

#### 10.5.1.5. Partitionnement des données

Les données sont réparties uniformément sur les réplicas (facteur de réplication).

Chaque partition se voit affecter un lot de données à héberger en fonction du hash de la clé primaire.

#### 10.5.1.6. Sécurité

L’accès au cluster peut être protégé par un mot de passe. Pour des fonctionnalités plus avancées (LDAP, chiffrement des données) il faut recourir aux versions commerciales.

### 10.5.2. HBase

Site web	<b><a href="http://hbase.apache.org">http://hbase.apache.org</a></b>
Licence	Apache
Technologie	Java
Année création	2006

HBase est un sous projet d’Hadoop, c’est un système de gestion de base



de données non relationnelles distribué, écrit en Java, disposant d'un stockage structuré pour les grandes tables.

HBase est né afin de proposer un accès temps réel à un cluster Hadoop. Contrairement à HDFS l'accès aux données est aléatoire et donc plus performant malgré les très fortes volumétries.

Facebook pourtant à l'origine de Cassandra a décidé de l'abandonner en 2010 au profit de HBase.

HBase est inspiré des publications de Google sur BigTable, c'est donc une base de données orientée colonnes et sans schéma (en théorie).

HBase est très lié à Hadoop et en mode cluster HDFS est obligatoire. Chaque valeur d'une cellule (champ dans le monde relationnel) est horodaté (tout comme Cassandra) ce qui permet de récupérer une ancienne version du contenu.

Concernant la purge de ces anciennes valeurs, HBase propose deux options :

- définir le nombre de versions à conserver,
- définir une durée de vie pour les données (en secondes).

Il est possible de combiner ces deux paramètres.

Une table est segmentée en plusieurs partitions que l'on nomme région, ces partitions sont réparties entre les différents serveurs.

Ces partitions sont horizontales, c'est à dire découpées par intervalle de clés contiguës.

La gestion des régions est dynamique :

- la région est divisée quand elle dépasse une certaine taille,
- deux régions peuvent être fusionnées pour des raisons d'optimisation.

Lors d'une modification les opérations sont consolidées en mémoire (tampon) avant d'être persistées sur disque.

Contrairement à Cassandra l'architecture est de type maître/esclave, un maître (HMaster) est chargé de la coordination des opérations et du cluster.

Parmi les opérations gérées directement par le HMaster, on trouve les manipulations sur les tables (création, modification et suppression).

Par contre les lectures, écritures se font directement sur les nœuds abritant les données.

Zookeeper est utilisé pour la gestion des ressources du cluster.

### **Cas d'utilisation :**

HBase est souvent utilisé conjointement au système de fichiers HDFS, ce dernier facilitant la distribution des données de HBase sur plusieurs nœuds.

HBase est alors un moyen supplémentaire d'interroger les données construites avec Hadoop.

Contrairement à HDFS, HBase permet de gérer les accès aléatoires read/write pour des applications de type temps réel.

### **10.5.3. HDFS**

Site web	<b><a href="http://hadoop.apache.org/">http://hadoop.apache.org/</a></b>
Licence	Apache
Technologie	Java
Année création	2005

HDFS est un système de fichiers Java utilisé pour stocker des données structurées ou non sur un ensemble de serveurs distribués.

HDFS est un système de fichiers distribués, extensible et portable développé par Hadoop à partir du système développé par Google (GoogleFS).

Écrit en Java, il a été conçu pour stocker de très gros volumes de données sur un grand nombre de machines équipées de disques durs standards.

HDFS s'appuie sur le système de fichier natif du système d'exploitation

pour présenter un système de stockage unifié reposant sur un ensemble de disques et de systèmes de fichiers hétérogènes.

Un cluster HDFS comporte deux types de composants majeurs :

- NameNode :

Ce composant gère les fichiers et les répertoires du cluster de manière centralisée.

Il est unique (pour un namespace) mais dispose d'une instance secondaire afin d'assurer la continuité du fonctionnement du cluster Hadoop en cas de panne.

- DataNode (nœud de données) :

Ce composant stocke et restitue les blocs de données (données primaires) et abrite des copies des autres instances.

Par défaut, les données sont stockées sur trois nœuds différents : dans deux nœuds proches (même machine ou rack) et l'autre sur un nœud plus distant. Le RAID est par conséquent inutile sur un cluster HDFS.

La consistance des données est basée sur la redondance. Une donnée est stockée sur au moins  $n$  volumes différents (souvent 3).

Un principe important d'HDFS est que les fichiers sont de type «write-once/immutable» car dans des opérations analytiques, on lit la donnée beaucoup plus qu'on ne l'écrit. C'est donc sur la lecture que les efforts ont été portés.

Ce qui signifie que l'on ne modifie pas les données déjà présentes, on les remplace entièrement.

Un principe lié est qu'à partir du moment où un fichier HDFS est ouvert en écriture, il est verrouillé pendant toute la durée du traitement.

Il est donc impossible d'accéder à des données ou à un résultat tant que le job n'est pas terminé et n'a pas fermé le fichier (et un fichier peut être très volumineux avec Hadoop).

Résumé de Hadoop Distributed File System :

- Open Source,

- développé en Java,
- supporte de très gros volumes,
- tolérant à la faute,
- un ensemble de disques de taille modeste,
- possibilité d'OS hétérogènes (c'est une surcouche),
- blocs de 64 Mo à 1 Go (un fichier est donc découpé en n blocs, le prendre en compte dans les traitements),
- de type write-once (impossible de modifier a posteriori un enregistrement),
- le verrou est posé sur tout le bloc pendant la phase d'écriture (impossible de lire),
- plus lent que l'OS :  $\pm 25\%$  en lecture,  $\pm 50\%$  en écriture.
- haut débit (volume),
- scalable,
- localisation des disques dans une machine/un rack.

Politique de réplication :

3 replicas dont 1 replica sur la machine locale et 2 replicas sur un rack distant.

#### 10.5.4. MongoDB

Site web	<a href="http://www.mongodb.org">http://www.mongodb.org</a>
Licence	Affero GPL
Technologie	C/C++
Année création	2009

MongoDB est une base de données orientée documents de la mouvance NoSQL permettant le stockage de documents au format BSON (une forme binaire de JSON).

Son langage d'interrogation est aussi au format JSON.

Elle dispose de mécanismes de réplication et de sharding afin d'assurer la résilience et la scalabilité même si son architecture est de type maître. Un mécanisme d'élection permet d'assurer la continuité de service en cas de défaillance du maître.

MongoDB est schemaless, ce qui signifie que les documents d'une même collection n'ont pas tous les mêmes champs. On peut ajouter / retirer des champs à un document sans migration.

MongoDB se positionne comme une base NoSQL généraliste avec des cas d'utilisation proches des SGBD relationnels (à l'exception du transactionnel) : 80% des applications actuelles qui utilisent un SGBD sont visées.

Chaque opération (lecture, écriture, mise à jour) demande un lock avant de commencer à accéder aux données.

Il y a deux types de locks dans MongoDB :

- lock de lecture : peut être partagé entre plusieurs opérations de lecture,
- lock d'écriture : exclusif, ce qui veut dire qu'aucune autre opération ne peut avoir lieu (écriture ou même lecture).

Depuis la version 3.0 le lock d'écriture peut être positionné au niveau documents (uniquement en utilisant le moteur de stockage WiredTiger). Plus globalement, la version 3 apporte une ouverture des API internes pour permettre à un éditeur de plugger son propre moteur de stockage.

Actuellement sont disponibles : MMAPv1, WiredTiger, TokuMX, RocksDB.

Les frameworks d'agrégation permettent d'aller au delà des opérations simples sur les données afin de proposer des capacités d'analyse poussées.

L'un des points forts de MongoDB est la multitude de clients disponibles:

- JavaScript,
- Python,
- Ruby,
- PHP,
- Perl,
- Java,
- Scala,
- C#,
- C,
- C++,
- Haskell,

- Erlang.

La solution est supportée par la société MongoDB qui propose une version commerciale et du support.

La version commerciale apporte :

1. Sécurité étendue
2. Utilisation de la solution de monitoring en local et non sur le cloud
3. Sauvegardes incrémentales
4. MongoDB Compass (analyse des modèles)
5. Base mémoire
6. ...

### 10.5.5. Neo4j

Site web	<b><a href="http://neo4j.org">http://neo4j.org</a></b>
Licence	GPLV3
Technologie	Java
Année création	2000

Neo4j est une des plus anciennes base NoSQL (première version en 2000). C'est une base orientée graphes écrite en Java.

Les données sont stockées sur disque sous la forme d'une structure de données optimisée pour les réseaux de graphes.

Les bases de données orientées graphes, sont particulièrement adaptées dans des contextes où les données sont fortement connectées et organisées selon des modèles complexes. La structure de l'entité (nœud ou relation) y est définie au moment du stockage de la donnée (structure schemaless), ce qui lui confère une très grande flexibilité.

Étant spécialisée dans ce domaine le parcours de graphes est particulièrement performant.

En plus de la structuration des données, Neo4j diffère aussi techniquement des autres bases NoSQL par les points suivants :

- support des transactions ACID (commit à 2 phases, compatibilité XA),
- utilisable en mode embarqué ou en serveur,
- langage d'interrogation CYPHER.

Du fait du support de pseudo transactions l'architecture de Neo4j (et celle des autres bases orientée graphe) est obligatoirement de type maître/esclave (les mises à jours doivent être centralisées).

Cas d'utilisations :

- web Sémantique et RDF,
- web des données (LinkedData),
- données cartographiques (GIS),
- réseaux sociaux,
- configurations/catégories de produits,...

Liste des connecteurs disponibles :

- Java,
- .NET,
- Ruby,
- Python,
- Go,
- etc

Fonctionnalités de la version entreprise :

- cache répliqué,
- haute disponibilité, réplication
- sauvegarde à chaud,
- etc

## 10.5.6. CouchBase

Site web	<a href="http://www.couchbase.com/">http://www.couchbase.com/</a>
Licence	Apache + commerciale
Technologie	C/C++
Année création	2011

CouchBase est une base orientée document née en 2011, elle est issue du rapprochement de deux projets Open Source : Membase (basé sur memcached) et Apache CouchDB avec lesquels il reste compatible.

CouchBase stocke les documents au format JSON et propose les moyens suivant pour accéder aux données :

- en utilisant la clé unique,
- à travers les vues (de deux types MapReduce/spatial construites à partir de fonctions JavaScript),
- au moyen d'un langage pseudo SQL (N1QL),
- grâce à la recherche plein texte (Full Text Search).

Tout comme MongoDB, le format JSON permet à CouchBase d'être shemaless, il est aussi possible de définir une durée de vie automatique aux données (TTL).

CouchBase supporte la gestion multi-datacenter en mode bidirectionnel. Ceci permet d'assurer la continuité de service en cas d'indisponibilité d'un datacenter mais aussi à l'utilisateur d'interroger le datacenter le plus proche en fonctionnement normal.

CouchBase propose les filtres de réplication qui permettent de sélectionner les données à répliquer entre deux datacenter (version entreprise uniquement).

La version 4.0 apporte un nouveau langage SQL pour requêter les données et capitaliser sur les compétences SQL des entreprises et aussi simplifier la montée en compétence.

N1QL étend la syntaxe de base de SQL afin de faciliter l'interprétation des documents JSON et de les traiter comme des données tabulaires en provenance d'une base SQL classique (fonctions Nest et UnNest).



Contrairement à d'autres bases NoSQL, N1QL supporte les jointures entre les tables.

Un autre avantage est la présence de pilotes ODBC/JDBC qui vont permettre de créer des jointures entre des bases relationnelles (mais aussi Excel et Tableau) et des bases CouchBase.

CouchBase propose de nombreux connecteurs qui vont permettre l'intégration avec des solutions de traitement, de gestion d'événements ou d'indexation :

- Talend,
- ElasticSearch,
- Hadoop,
- Kafka,
- Spark.

Depuis la version 4, il est possible d'attribuer des rôles aux nœuds CouchBase parmi les possibilités suivantes :

- requêtage (Query Service),
- indexation (Indexing Service),
- stockage des données (Data Service).

Cette répartition des fonctionnalités (Multi Dimensional Scaling) n'est disponible que dans la version entreprise. Il existe aussi un autre rôle, Cluster Manager : tous les nœuds ont ce rôle mais un seul est choisi pour une opération donnée, c'est l'orchestrator.

Une donnée est répliquée sur les nœuds du cluster afin d'assurer la continuité de service en cas de crash d'un nœud. CouchBase peut partitionner les données grâce au concept de Bucket.

Un Bucket permet de distribuer les données d'une table sur différents nœuds (c'est le principe du sharding). Le choix d'un Bucket est important car il va permettre de distribuer la charge sur différents nœuds.

Pour implémenter le principe de Bucket CouchBase utilise les vBuckets. Chaque Bucket est divisé en 1024 vBuckets actifs et 1024 vBuckets répliqués qu'il va distribuer équitablement sur tous les nœuds qui exécutent le service de stockage de données (Data Service).

Un service permet de faire le lien entre un vBucket et sa localisation (cluster map). CouchBase bénéficie d'une déclinaison mobile native qui peut être déployée sur un terminal. Ceci permet de synchroniser automatiquement la base mobile à partir du serveur CouchBase, et à l'inverse de conserver les données du terminal déconnecté pour une synchronisation ultérieure.

Architecture :

- Couchbase Lite : déclinaison mobile de la base de données (iOS, Android et Java),
- Sync Gateway : gestion de la synchronisation (authentification des utilisateurs, contrôle des accès, filtrage et validation des données) entre les terminaux mobiles et le serveur,
- Couchbase Server : base de données NoSQL.

Les plus de la version entreprise :

- sécurité avancée,
- filtres de réplication,
- sauvegardes incrémentales,
- localisation des nœuds (racks, ...),
- Multi-dimensional scaling.

En temps normal, CouchBase va combiner les deux solutions : memcached pour le cache et couchDB pour les fonctionnalités NoSQL et la persistance.

Mais il est possible de choisir uniquement Memcached :

API	Memcached	CouchDB
Limite d'un document	1 Mo	20 Mo
Réplication	Non	Oui

Réplication DataCenter	Non	Oui
Chiffrement des données	Non	Oui
Sauvegarde	Non	Oui
Stockage	Mémoire	Persistent

Une API REST permet d'accéder à l'administration du cluster.  
 Une interface web (Couchbase Web Console) permet d'accéder aux statistiques ainsi qu'à l'administration du cluster.  
 Enfin un outil en ligne de commande permet d'accéder aux métriques du cluster (cbstats).

## 10.6. Transformation/Enrichissement

### 10.6.1. Spark

Site web	<a href="http://spark.apache.org/">http://spark.apache.org/</a>
Licence	Affero GPL
Technologie	Scala
Année création	2009

Spark est né en 2009 dans le laboratoire AMPLab de l'université de Berkeley en partant du principe que :

- d'une part, la mémoire coûte de moins en moins cher et les serveurs en ont donc de plus en plus à disposition,
- d'autre part, beaucoup de jeux de données dits "Big Data" ont une taille de l'ordre de 10 Go et ils tiennent donc en mémoire.

Il fallait donc un système de traitement qui privilégie le traitement en

mémoire des données.

Le projet a intégré l'incubateur Apache en juin 2013 et est devenu un "Top-Level Project" en février 2014. Le projet s'est enrichi progressivement afin de proposer aujourd'hui un écosystème complet.

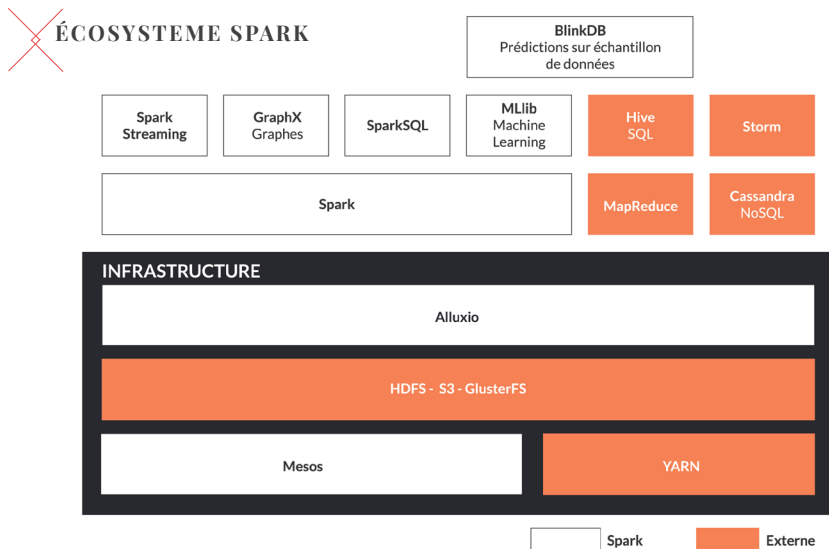
L'écosystème Spark comporte ainsi aujourd'hui plusieurs outils :

- Spark pour les traitements "en batch",
- Spark Streaming pour le traitement en continu de flux de données,
- MLlib pour le "machine learning", SparkML depuis 2012,
- GraphX pour les calculs de graphes,
- Spark SQL, une implémentation SQL-like d'interrogation de données.

Enfin, le framework est écrit en Scala et propose un binding Java qui permet de l'utiliser sans problème en Java. Java 8 est toutefois recommandé pour exploiter les expressions lambdas qui permettront d'écrire un code plus fonctionnel.

Il existe des versions de Spark spécifiques pour Python et R afin de capitaliser sur les langages habituellement maîtrisés par les Data Scientists.

Pour rappel, voici l'écosystème Spark :



### 10.6.1.1. Gestion des clusters

Options disponibles avec Spark :

- Standalone,
- Mesos,
- Yarn,
- Cloud (Amazon EMR, Google DataFlow, ...),
- Cassandra (en version entreprise et basé sur Mesos).

### 10.6.1.2. Batch

Un RDD (Resilient Distributed Dataset) est une abstraction de collection sur laquelle les opérations sont effectuées de manière distribuée tout en étant tolérante aux pannes matérielles.

En utilisant les types proposées par Spark et en respectant certaines contraintes, le traitement sera automatiquement découpé pour s'exécuter sur plusieurs nœuds. En cas de perte d'un nœud, le sous traitement sera automatiquement relancé sur un autre nœud par le framework, sans que cela impacte le résultat.

L'API exposée par le RDD permet d'effectuer des transformations sur les données (map, filter, reduce, etc).

Ces transformations sont "lazy" : elles ne s'exécuteront que si une opération finale est réalisée en bout de chaîne.

### 10.6.1.3. Streaming (micro batch)

Une API (Spark Streaming) permet à Spark de traiter les événements en temps réel et non plus en batch.

En réalité Spark utilise le principe des micro-batch, c'est-à-dire que l'on peut obliger Spark à fournir un résultat au bout d'un certain temps et ce même si les événements en entrée continuent d'arriver.

### 10.6.1.4. Structure spécifiques Spark

En plus des RDD, Spark propose deux API plus riches : Les DataFrames et les DataSet.

L'API DataFrame a fait son apparition en version 1.3, le but étant d'offrir une API plus fonctionnelle (en utilisant Spark SQL) et faire la jonction entre Spark et les DataAnalyst.

Pour cela l'API DataFrame exige de définir un schéma qui va permettre de décrire et de manipuler les données contenues.

Ce schéma peut être automatiquement créé lorsque la source de données le permet (SGBD, Hive, fichiers CSV, ...) ou devra être définie si le DataFrame est construit à partir d'un RDD par exemple.

L'API DataSet a fait son apparition en version 1.6, le but étant d'offrir une API profitant de la puissance des RDD et du moteur d'exécution Spark SQL. L'API DataSet permet de manipuler avec Spark SQL des objets métiers écrit en Java/Scala.

L'API DataSet est une extension de l'API DataFrame, c'est pourquoi Spark 2.0 unifie ces deux API.

#### 10.6.1.5. Gestion des Graphes

Pour la gestion des graphes, Spark s'appuie sur la solution GraphX.

GraphX est un framework distribué de graphes construit au dessus de Spark (depuis 2013).

Il fournit à la fois une API de modélisation et d'exécution et est basé sur le projet Google Pregel.

GraphX privilégie la simplicité d'utilisation et non la vitesse de traitement. C'est sans doute dû à une population ciblée différente (Big Data Analyst).

#### 10.6.1.6. Machine learning

Spark utilise la librairie MLlib dont la notoriété est grandissante. Une évolution nommée (Spark ML) est à la fois une refonte de certains traitements ainsi qu'un ajout de fonctionnalités.

Concrètement MLlib va utiliser l'API RDD, alors que Spark ML utilise les DataFrames.

#### 10.6.1.7. Tolérance à la panne

“exactly once” en conditions normales et “at least once” en conditions dégradées.

La haute tolérance doit être activée par l'utilisateur (Write Ahead Log).

#### 10.6.1.8. Performances

Afin d'améliorer les performances, Spark a lancé le projet Tungsten en 2015 qui a pour objectif une meilleure gestion de la mémoire, les opérations sur les données binaires, etc.

### 10.6.2. MapReduce

Site web	<a href="https://hadoop.apache.org/">https://hadoop.apache.org/</a>
Licence	Apache
Technologie	Java
Année création	2005

MapReduce est un framework de traitements parallélisés, créé par Google pour indexer les contenus de son moteur de recherche web.

Disponible sous forme de document de recherche, sa première implémentation est apparue au moment du développement d'Hadoop en 2005.

C'est un framework qui permet la décomposition d'une requête importante en un ensemble de requêtes plus petites qui vont produire chacune un sous ensemble du résultat final : c'est la fonction Map.

L'ensemble des résultats est traité (agrégation, filtre) : c'est la fonction Reduce.

Dans un traitement MapReduce, différents acteurs vont intervenir :

- **workers** : Liste de nœuds Hadoop capables de traiter des tâches MapReduce,
- **master** : Un worker dédié à la gestion des tâches,
- **client** : Lance le traitement MapReduce (souvent nommé driver)

### 10.6.2.1. Les différentes phases (Hadoop)

- **initialisation** : Le client/driver charge un/des fichiers dans HDFS et soumet un traitement MapReduce à la grille,
- **split** : Les données en entrée sont éventuellement divisées en blocs (16-64 Mo),
- **affectation** : Le master affecte les tâches (Map et Reduce) aux workers; la configuration définit le nombre de tâches de type Map et Reduce supportées par chacun des nœuds,
- **map** : Lecture des splits qui sont transmis à la fonction Map. Les ensembles clé/valeur produits par la fonction sont d'abord stockés en mémoire avant d'être périodiquement écrits localement et non sur HDFS,
- **shuffle** : Les résultats des fonctions Map sont agrégés par la valeur de la clé pour produire une liste de valeurs traitée par le Reducer,
- **reduce** : Le master distribue au Reducer la liste des données à traiter. Les résultats sont envoyés au flux de sortie (HDFS, web services, ...),
- **combiner** : Optimisation, utilise les résultats intermédiaires du Map en entrée pour un traitement qui est généralement équivalent au Reducer (pas de garantie de passage),
- **fin** : Le master redonne la main au programme client.

### 10.6.2.2. Implémentation

En plus d'Hadoop, MapReduce est disponible dans une multitude de frameworks :

- NoSQL : Cassandra, MongoDB,
- traitements distribués : Spark, Flink,
- cloud.



### 10.6.3. Flink

Site web	<a href="https://flink.apache.org/">https://flink.apache.org/</a>
Licence	Apache
Technologie	Java
Année création	2009

Apache Flink est un Top Level Project Apache depuis décembre 2014. Anciennement nommé Stratosphere et projet de recherche par Data Artisans il a été créé en 2009 (tout comme Spark).

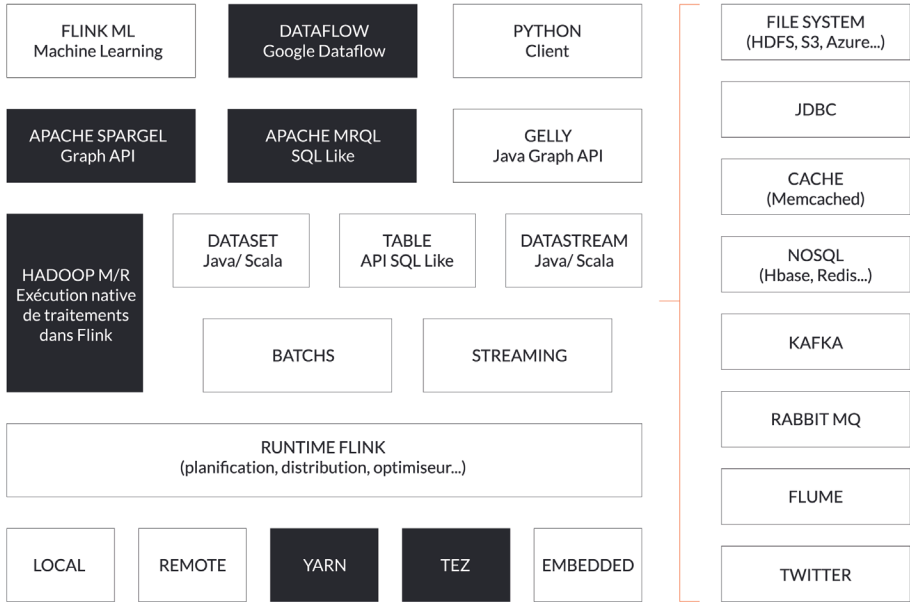
Son but est de fournir un framework de traitements distribués en mémoire adapté au traitement temps réel.

#### 10.6.3.1. Différences avec Spark

- Flink a été conçu dès le départ pour le temps réel,
- Flink a été écrit initialement en Java et supporte Scala grâce à des wrappers,
- Flink peut exécuter des traitements Hadoop directement (idéal pour une transition en douceur).

#### 10.6.3.2. Apache Flink comprend

- des APIs en Java/Scala pour le traitement par batch et en temps réel,
- un moteur de transformation des programmes en flots de données parallèles,
- un moteur d'exécution pour la distribution des traitements sur un cluster.



### 10.6.3.3. Gestion des clusters

Options disponibles avec Flink pour la gestion des clusters :

- Standalone,
- Yarn,
- Cloud (EC2, Google DataFlow, ...),
- Apache TEZ.

Flink propose une véritable API de streaming mais aussi de micro batch.

Liste des connecteurs streaming :

- File System (HDFS, S3, Azure,...),
- JDBC,
- cache (Memcached),
- NoSQL (Hbase, Redis, ...),
- Kafka,
- Rabbit MQ,
- Flume,
- Twitter, ...

#### 10.6.3.4. API Fonctionnelle

L'API Table est aussi très récente et permet de formaliser les traitements dans une forme proche de la syntaxe SQL.

Cette API est disponible pour le batch et le temps réel et offre une API de haut niveau qui apporte concision et clarté.

Jusqu'à récemment la gestion des graphes avec Flink était déléguée à **Apache Spargel** (projet générique)

#### 10.6.3.5. Gestion des Graphes

Un projet nommé **Gelly** a été lancé afin d'offrir à Flink, la gestion des graphes tout en tirant profit des spécificités de Flink.

Gelly offre :

- utilitaires d'analyse de graphes,
- traitements itératifs sur les graphes,
- algorithmes dédiés aux graphes.

Gelly n'est compatible qu'avec des objets héritant des DataSet (Vertex et Edges) et n'est donc compatible qu'avec les batchs et non les flux temps réel.

Parmi les algorithmes :

- PageRank,
- Weakly Connected Components,
- Single Source Shortest Paths,
- Label propagation.

Gelly est actuellement en bêta et le but à terme étant de remplacer Spargel.

#### 10.6.3.6. Machine learning

Flink Machine Learning Library (**Flink-ML**), orienté pipeline inspiré de **scikit-learn** (framework de ML écrit en python).

Implémentation des principaux algorithmes de Machine Learning optimisés spécifiquement pour Flink.

Flink-ML dispose des algorithmes suivants :

- classification,
- regression (Logistic regression),
- clustering (k-Means),
- recommandation (Alternating least squares).

NB : il existe d'autres algorithmes mais uniquement en Scala pour le moment (est-ce une tentative de séduction des "data analyst" ?).

Une intégration avec Mahout DSL comme moteur d'exécution est en cours.

#### 10.6.3.7. Tolérance à la panne

"exactly once" en conditions normales et conditions dégradées (grâce au mécanisme de checkpoints) qui est de plus automatique.

#### 10.6.3.8. Performances

L'optimisation des traitements est un point important de Flink, par analogie on peut voir l'optimiseur comme celui d'une base de données. Le meilleur « chemin » est choisi au moment de l'exécution en fonction de paramètres et en privilégiant le traitement le plus rapide.

Pour faire ce choix l'optimiseur analyse principalement :

- les types de données,
- les fonctions utilisées.

Il intervient à la fois pour les traitements batch et les traitements en temps réel.

Ainsi pour une méthode de type join utilisée dans le programme, Flink peut décider d'utiliser :

- un partitionnement ou non des DataSet impliqués,
- un hash join ou un merge join avec tri, ...

Une grosse partie a aussi lieu au moment des itérations :

- mise en cache des données constantes,
- les opérations les plus coûteuses sont déplacées en dehors de la boucle, ...

La plupart du temps, l'optimiseur va faire les bons choix sans que vous ayez à vous en préoccuper, toutefois il est possible de forcer la stratégie d'exécution.

#### 10.6.3.9. Itérations

Flink propose deux types d'itérations :

- itérations simples,
- delta-itérations.

La première facilite la gestion de flux successifs et l'agrégation des résultats.

La deuxième vise avant tout les performances.

Les itérations sont de moins en moins coûteuses au fur et à mesure des traitements.

L'inconvénient étant qu'il faut un certain nombre d'itérations avant d'obtenir le résultat final, mais ensuite si les entrées évoluent seules les nouvelles seront traitées (delta-itérations).

#### 10.6.3.10. Déploiement

Flink est déployable sur HadoopDataPlatform d'HortonWorks et des évolutions sont en cours pour :

- Cloudera (fichier de configuration Hadoop non compatible).
- Cloud Google : Disponibilité de Flink comme runtime pour Google Cloud Dataflow

Clients expérimentant Flink : Amadeus, Spotify , ...

## 10.7. Analyse/Requêtage/Visualisation

### 10.7.1. Hive

Site web	<b><a href="http://hive.apache.org">http://hive.apache.org</a></b>
Licence	Apache
Technologie	Java
Année création	2008

Hive est un projet initié par Facebook en 2008.

A l'image de Pig, Hive permet l'écriture de tâches de traitement/requêtage de données aux développeurs ne maîtrisant pas Java.

Là où Pig définit un langage procédural permettant d'exploiter le cluster, Hive permet de définir des tables structurées de type SQL et de les alimenter avec des données provenant soit du cluster, soit de sources externes.

Hive repose sur les notions suivantes :

- Tables : une table est une succession de colonnes et de lignes tout comme pour le monde des SGBD. La différence provient du système de stockage qui est distribué (HDFS ou autre). Une autre différence importante est l'absence de notion de clé primaire.
- Partitions : une table Hive peut être découpée en partitions afin de répartir les données sur le cluster (en fonction de la valeur de plusieurs champs)
- Buckets : une fois partitionnées, les données peuvent être ensuite découpées selon une seule colonne de la table afin de diviser le stockage des données. Le bucketing améliore les performances des jointures et permet de ne travailler que sur des échantillons.
- Metastore : c'est l'endroit où Hive stocke les métadonnées des tables (schéma, droits d'accès, statistiques, ...)

Une table Hive peut être soit :

- interne, les données sont dupliquées dans un répertoire HDFS

dédié à Hive (warehouse).

- externe, les données sont entreposées dans un répertoire HDFS externe et seules les métadonnées sont conservées dans Hive.

### 10.7.1.1. Requêtes

Une fois le schéma des tables défini et les données insérées, il est possible d'utiliser le langage HiveQL pour requêter ces tables. HiveQL a une syntaxe proche de SQL et permet de réaliser l'essentiel des opérations de lecture permettant de produire des analyses classiques (sélection de champs, somme, agrégat, tri, jointure, ...).

### 10.7.1.2. Traitements

Les scripts Hive sont transformés afin d'être exécutés dans un cluster Hadoop :

- en traitement MapReduce,
- en traitement Spark,
- en traitement TEZ.

Le plus gros avantage de Hive est sa capacité à utiliser une compétence très répandue qu'est la connaissance de SQL rendant les développeurs très rapidement opérationnels pour extraire les données.

## 10.7.2. Pig

Site web	<a href="http://pig.apache.org">http://pig.apache.org</a>
Licence	Apache
Technologie	Java
Année création	2008

Pig est un outil de traitement de données qui fait partie de la suite Hadoop et qui permet l'écriture de scripts qui sont exécutés sur l'infrastructure Hadoop sans être obligé de passer par l'écriture de tâche en Java via le framework MapReduce.

Il dispose en outre de fonctionnalités permettant le chargement de données depuis une source externe vers le cluster HDFS ou de fonctionnalités permettant l'export de données pour utilisation par des applications tierces. Pig s'appuie sur son propre langage nommé Pig Latin. Il permet en outre d'accéder à la couche applicative Java. Ce langage est assez simple ce qui permet au développeur venant d'un autre monde que Java de produire des scripts de traitement s'exécutant sur Hadoop beaucoup plus rapidement.

Pig propose deux modes d'exécution :

- Local : sur une seule machine sans HDFS ni MapReduce,
- Hadoop : les traitements Pig sont convertis en code MapReduce et exécutés sur un serveur Hadoop.

Opérations possibles sur les données :

- chargement et stockage,
- streaming,
- filtre,
- agrégation,
- tri, ...

Dans la pratique, Pig est surtout utilisé pour charger des données externes vers HDFS et transformer des fichiers afin de faciliter leur analyse surtout dans des cas où plusieurs étapes sont nécessaires (du fait de la nature procédurale du langage et de sa capacité à stocker des résultats temporaires).

Un projet nommé DataFu permet d'enrichir les fonctions disponibles nativement dans Pig (notamment des fonctions statistiques).

### 10.7.3. Drill

Site web	<b><a href="http://drill.apache.org/">http://drill.apache.org/</a></b>
Licence	Apache
Technologie	Java
Année création	2012



Le projet Drill a été placé dans l'incubateur de la fondation Apache en août 2012, c'est un moteur de requêtes SQL pour Hadoop, développé initialement chez MapR.

Inspiré du projet Dremel de Google, Drill, quant à lui, permet d'effectuer des requêtes interactives SQL au dessus d'Hadoop, avec pour objectif premier de donner accès aux utilisateurs non développeurs de réaliser des analyses à partir de données stockées dans HDFS et Hadoop.

Même si la comparaison n'est pas totalement juste (Drill se présente plutôt comme un Spark facile d'accès) on retrouve donc les mêmes cas d'utilisation que Hive.

#### 10.7.3.1. Requêtage

Drill permet le requêtage de données :

- dans des fichiers,
- Hive,
- HBase,
- HDFS,
- Solr,
- JDBC, ...

Le langage utilisé est le SQL et permet de manipuler les données de sources différentes, les requêtes sont exécutées in-situ et permettent à Drill d'être shemaless, ou plutôt de découvrir le schéma à la volée.

Le parseur SQL est Apache Calcite (utilisé par les projets Apache : Hive, Phoenix et Kylin).

Drill offre aussi des extensions au langage SQL afin d'offrir des types spécifiques (tableau, map, ...) et fournit des drivers ODBC et JDBC afin de s'interconnecter avec le SI et des outils comme Tableau.

#### 10.7.3.2. Déploiement

Apache Drill peut être déployé en cluster afin d'augmenter sa puissance de traitement et garantir le failover.

### 10.7.3.3. Performances

Apache Drill privilégie la mémoire afin de favoriser les performances.

### 10.7.3.4. Stockage des résultats

Une fois requêtées, les données peuvent :

- être stockées dans un système compatible :
  - > File System : HDFS, MapR-FS, ...
  - > Hive,
  - > NoSQL : MongoDB, Cassandra, Hbase, MapR-DB, ...
  
- être disponibles sous forme de vues qui sont des fichiers distribués dans le file système sous-jacent :
  - > l'emplacement du stockage des vues est un paramètre de la requête

### 10.7.3.5. Sécurité

En termes de Sécurité, Drill offre deux possibilités :

- le système de permissions système de fichiers ou sont stockées les vues,
- un système interne d'authentification qui permet de contrôler l'accès aux vues.

Drill est un projet jeune avec des livraisons très fréquentes afin d'améliorer le support du format SQL, d'améliorer les performances.

### 10.7.3.6. Monitoring

Le monitoring des requêtes Drill ainsi que leur annulation peut se faire au moyen d'une application Web.

## 10.8. Indexation

### 10.8.1. Elastic (ElasticSearch)

Site web	<b><a href="http://elastic.org">http://elastic.org</a></b>
Licence	Licence Apache 2.0
Technologie	Java
Année création	2010

ElasticSearch est un moteur de recherche open-source, distribué et basé sur Lucene.

ElasticSearch a été créé par Shay Banon, initiateur du projet Compass, en remplacement de ce dernier.

Contrairement à Lucene, c'est un cluster d'indexation, qui s'appuie sur une base documentaire NoSQL interne utilisant le format JSON pour le stockage des documents.

De plus tout comme Solr et contrairement à Lucene son mode d'interrogation est de type client/serveur (il n'a pas besoin d'être déployé dans la même JVM que le client). Les documents sont répliqués sur différents nœuds de stockage, afin d'améliorer la performance des recherches mais également de gérer la haute disponibilité.

La décomposition des rôles est la suivante :

- Lucene gère les opérations « bas niveau » comme l'indexation et le stockage des données,
- ElasticSearch apporte plusieurs couches d'abstraction pour accepter du JSON, offrir une API REST sur HTTP et faciliter la constitution de clusters.

Contrairement à certaines bases NoSQL, ElasticSearch n'est pas schema-less. Il est possible d'indexer un document sans aucune information sur son format (uniquement la donnée), mais ElasticSearch va en créer un automatiquement à partir du format des données.

ElasticSearch évolue constamment et dans ses évolutions récentes il faut noter :

- abandon du fameux système de river au profit d'une approche de type push par les clients,
- abandon des facettes au profit des agrégations.

La langage d'interrogation est propriétaire (Query DSL) et utilise une approche de type DSL et se base sur le format JSON.

#### 10.8.1.1. Clients

Clients ElasticSearch disponibles :

- Java,
- JavaScript,
- Groovy,
- .NET,
- PHP,
- Perl,
- Python,
- Ruby.

#### 10.8.1.2. Écosystème

Elastic, l'éditeur d'ElasticSearch, propose un écosystème de plus en plus riche autour de son moteur de recherche.

#### 10.8.1.3. Logstash

Logstash, permet de collecter les log, d'appliquer quelques traitements simples sur les messages reçus pour ensuite les transmettre vers une solution tierce, en particulier ElasticSearch pour pouvoir effectuer des recherches parmi ces messages.

#### 10.8.1.4. Kibana

Kibana, autre solution très connue, propose une interface graphique pour explorer et visualiser les données poussées vers ElasticSearch ou celles stockées dans d'autres solutions que celles de l'éditeur.

Ces trois solutions forment le trio le plus répandu pour l'analyse des logs : ELK .

Malheureusement ces produits avaient jusqu'à peu un cycle de vie indépendant qui pouvait parfois entraîner des problèmes de compatibilité.

Conscient de ces problèmes et afin d'accroître encore l'attrait de ces trois produits, la société Elastic a récemment réorganisé son offre afin de :

- fournir un packaging cohérent de ces trois solutions,
- permettre l'intégration de nouvelles solutions et fonctionnalités,

Le nouveau packaging a été renommé Elastic Stack .

Elastic Stack intègre une nouvelle solution dédiée au monitoring, Beats, qui est en réalité composé des produits suivants :

- PacketBeat : supervision réseau,
- TopBeat : supervision Mémoire et CPU,
- FileBeat : supervision des fichiers,
- WinlogBeat : supervision des événements OS (Windows uniquement),
- LibBeat : utilitaires.

#### 10.8.1.5. Sécurité

En termes de sécurisation, un plugin Elastic, Shield, permet de sécuriser les accès aux documents indexés mais uniquement réservé à la version commerciale.

#### 10.8.1.6. Version commerciale

Marvel qui est une application web de monitoring du moteur de recherche en temps réel ainsi que Watcher (plugin de notifications) sont aussi réservés à la version commerciale.

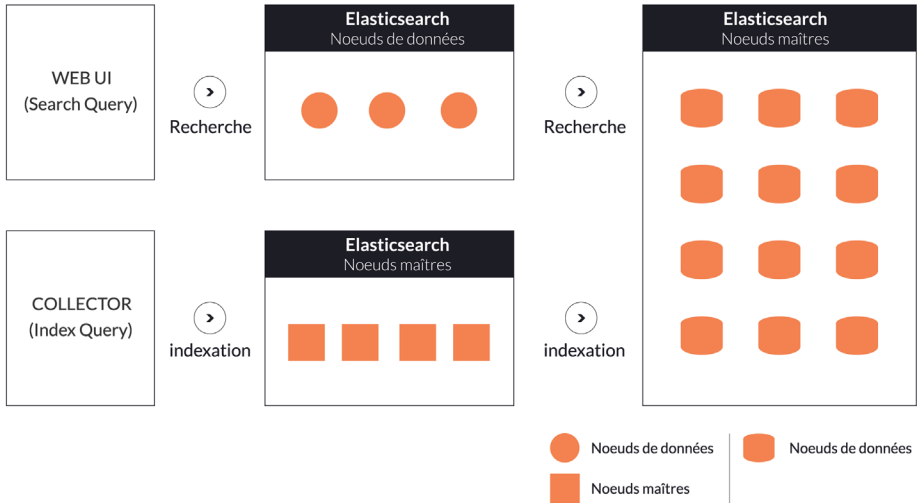
#### 10.8.1.7. Orientation

Parmi les orientations récentes :

- l'API Graph (d'abord présentée comme extension à l'outil de visualisation Kibana) permet à Elasticsearch d'explorer et d'afficher les relations qui existent entre des données (comme une base

orientée graphe),  
 - rapprochement de plus en plus visible avec le domaine du machine learning.

## ~~ELASTICSEARCH~~



## 10.9. Prédiction/Apprentissage

### 10.9.1. Spark ML et MLlib

Site web	<a href="http://spark.apache.org/mllib/">http://spark.apache.org/mllib/</a>
Licence	Licence Apache 2.0
Technologie	Scala
Année création	2003

#### 10.9.1.1. MLlib

MLlib est un sous projet Spark depuis la version 0.8. Tout comme Spark elle trouve son origine dans le laboratoire de Berkeley (AMPLab). MLlib est la librairie de Machine Learning de Spark. Les algorithmes sont conçus de manière à tirer profit du calcul en parallèle sur un cluster

Spark.

Elle est donc particulièrement adaptée aux fortes volumétries

Mllib, tout comme le reste du framework Spark est développé en Scala. Les opérations d'algèbre linéaire se basent sur les bibliothèques Breeze et jblas.

Algorithmes disponibles :

- classification: logistic regression, linear support vector machine (SVM), naive Bayes,
- régression: generalized linear regression (GLM),
- filtrage collaboratif : alternating least squares (ALS),
- clustering: k-means,
- décomposition: singular value decomposition (SVD), principal component analysis (PCA),
- données clairsemées. (Sparse data),
- arbres de Décision (CART), Régressions Logistiques,
- méthode de Broyden-Fletcher-Goldfarb-Shanno (L-BFGS),
- évaluation modèle,
- discrétisation.

En plus de tirer profit du noyau de Spark Mllib s'intègre avec :

- Spark SQL (DataFrames en entrée),
- Streaming (flux temps réel pour la prédiction),
- GraphX
  - > PageRank,
  - > Composants connectés,
  - > Propagation d'étiquette (Label propagation), ...

#### 10.9.1.2. Spark ML

Ce sous projet (disponible sous forme de package dans Mllib depuis la version 1.5) est une évolution visant à introduire de nouvelles fonctionnalités (DataFrame, Pipeline) et d'uniformiser les API.

## 10.9.2. Mahout

Site web	<a href="http://mahout.apache.org">http://mahout.apache.org</a>
Licence	Licence Apache 2.0
Technologie	Scala
Année création	2010

Apache Mahout est un projet de la fondation Apache depuis 2011 visant à créer des implémentations d'algorithmes d'apprentissage automatique et de Data Mining.

Même si les principaux algorithmes d'apprentissage se basent sur MapReduce, il n'y a pas d'obligation à utiliser Hadoop.

Apache Mahout ayant été conçu pour pouvoir fonctionner sans cette dépendance.

Précurseur historique, Mahout fait face à de nouvelles bibliothèques soit plus adaptées aux algorithmes itératifs, telles que MLlib de Spark (complétée par Spark ML), soit provenant du monde des Data Scientist telles que Scikit-learn ou bien le langage R.

C'est pourquoi depuis la version 0.10.0 (Avril 2015) Mahout a utilisé Apache Spark ainsi que H2O (en fonction des algorithmes).

Une intégration avec Flink est en cours.

Ce nouvel environnement mathématique est appelé Samsara.

En fonction des algorithmes Mahout va choisir le moteur d'exécution le plus adapté (et parfois laisser le choix) entre

- Spark,
- Hadoop MapReduce,
- H2O.

Mahout-Samsara est une refonte de Mahout afin de favoriser la scalabilité et les performances mais aussi fournir un canevas afin de créer ses propres algorithmes (développement en Scala).

Samsara propose aussi un shell interactif afin de lancer les traitements sur un cluster Spark

Mahout est très riche en algorithmes d'apprentissage, clustering,



classification, filtrage collaboratif, analyse d'items fréquents, etc.

Clustering :

- K-means (parallélisable),
- Fuzzy K-means (parallélisable),
- Spectral K-means (parallélisable).

Classification:

- Logistic regression (non parallélisable),
- Naive Bayes (parallélisable),
- Random Forest (parallélisable),
- Multilayer perceptron (non parallélisable).

Réduction de la dimensionnalité :

- Singular Value Decomposition (parallélisable),
- PCA (parallélisable),
- Lanczos decomposition (parallélisable),
- QR decomposition (parallélisable).

Texte :

- TF-IDF (parallélisable)

## 10.10. Acteurs

### 10.10.1. Akka

Site web	<a href="http://akka.io/">http://akka.io/</a>
Licence	Licence Apache 2.0
Technologie	Scala
Année création	2009

Akka est un framework pour construire des applications concurrentes, distribués, résilientes avec la JVM.

Akka est une implémentation basée sur les acteurs qui a démarré en

2009 par Jonas Bonér et qui a été tout d'abord intégré à Scala. La source d'inspiration est le langage Erlang qui propose une architecture hautement concurrente et basé sur les événements.

Akka est maintenant un framework OpenSource soutenu par TypeSafe (tout comme Play et Scala), disponible à la fois en Scala et en Java.

La programmation par acteurs propose les postulats suivants :

Un acteur est une entité qui sur réception d'un événement (message) peut :

- envoyer des messages à d'autres acteurs,
- créer de nouveaux acteurs,
- spécifier le comportement à avoir lors de la prochaine réception de messages.

L'exécution de ces tâches ci-dessus n'est pas ordonnée, elles peuvent donc être parallélisées.

Le paradigme Acteur découple l'émetteur du message du message lui-même, permettant donc l'asynchronisme des communications et l'introduction de structures dédiées à la gestion des messages. Un acteur doit connaître l'adresse de l'acteur à qui il veut envoyer un message. Les adresses peuvent être échangées par message.

En résumé un acteur :

- est un système avec état,
- réagit sur réception de messages,
- envoie des messages aux acteurs :
  - > par leur nom ou adresse,
  - > en retour de réception d'un message,
  - > à un acteur parent, ...

Akka n'est pas qu'une implémentation du modèle d'acteurs :

- pseudo transactions (Transactors) qui permet en cas d'erreur d'annuler les modifications et de relancer automatiquement le traitement,

- messages ordonnés pour un couple émetteur/récepteur.

Liste des frameworks construit avec Akka :

- Apache Spark,
- Gatling,
- Apache Flink.

Akka possède toutefois les défauts suivants :

- traite les messages séquentiellement (mais on peut multiplier le nombre d'acteurs),
- pas de garantie de livraison/traitement des messages (at-most-once delivery).

## **10.11. Interaction et Visualisation (notebook)**

Les notebooks permettent d'analyser simplement de gros volumes de données.

Le tout avec une interface web ergonomique et un langage d'interrogation le plus accessible. Ils sont principalement destinés à des Data Scientist.

### **10.11.1. SlamData**

SlamData se positionne comme l'un des notebooks les plus simple à mettre en œuvre et à utiliser. Il élimine la phase de recopie des données (dans un bac à sable ou un Data Lake) en déployant ses capacités d'analyse directement au cœur de la donnée, sur les mêmes serveurs que le stockage.

C'est le principe "Zero Relocation".

Avantages :

- coût de stockage réduit,
- scalabilité identique à celle du stockage,
- données réelles.

Inconvénients :

- impacts sur stockage,
- pas de séparation claire entre environnement de production et d'expérimentation/analyse.

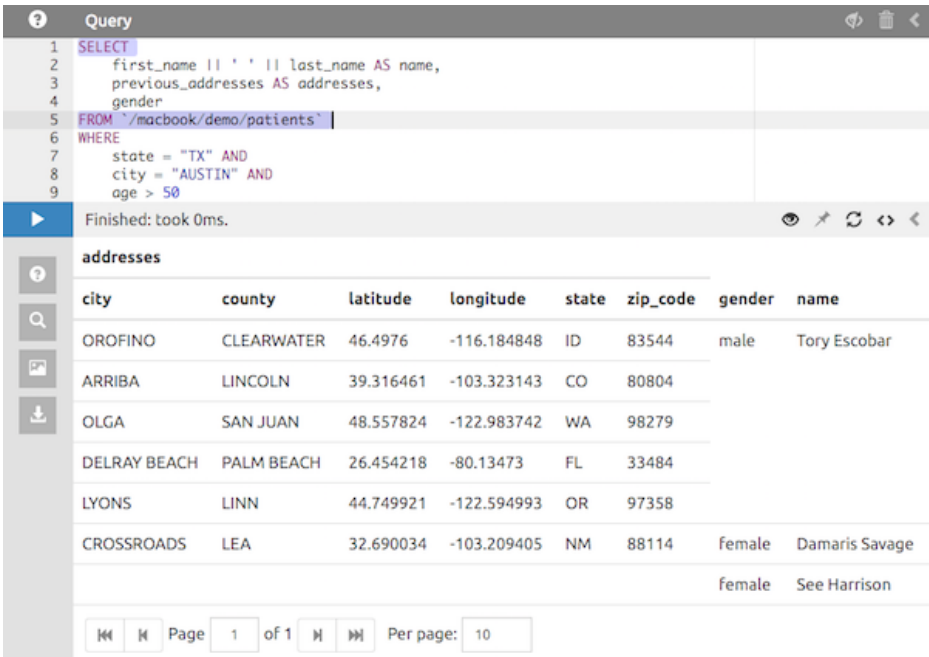
L'interface peut être déployée dans le cloud ou dans votre SI.

Langage d'interrogation :

- pseudo SQL avec une mise en forme JSON.

Liste des connecteurs :

- MongoDB,
- HBase,
- Cassandra,
- Spark.



The screenshot shows a query editor with the following SQL query:

```
1 SELECT
2   first_name || ' ' || last_name AS name,
3   previous_addresses AS addresses,
4   gender
5 FROM '/macbook/demo/patients'
6 WHERE
7   state = "TX" AND
8   city = "AUSTIN" AND
9   age > 50
```

Below the query, it indicates "Finished: took 0ms." and displays a table of results:

city	county	latitude	longitude	state	zip_code	gender	name
OROFINO	CLEARWATER	46.4976	-116.184848	ID	83544	male	Tory Escobar
ARRIBA	LINCOLN	39.316461	-103.323143	CO	80804		
OLGA	SAN JUAN	48.557824	-122.983742	WA	98279		
DELRAY BEACH	PALM BEACH	26.454218	-80.13473	FL	33484		
LYONS	LINN	44.749921	-122.594993	OR	97358		
CROSSROADS	LEA	32.690034	-103.209405	NM	88114	female	Damaris Savage
						female	See Harrison

At the bottom, there is a pagination control showing "Page 1 of 1" and "Per page: 10".

Source image : <http://docs.slamdata.com/en/v2.5/users-guide.html>

## 10.11.2. Zeppelin

Apache Zeppelin est un notebook inspiré par Jupyter et créé par **Lee Moon Soo**.

Au départ solution payante, Zeppelin a été offert à la fondation Apache en 2014.

Après presque deux ans d'incubation, Zeppelin est maintenant un projet majeur chez Apache (depuis juin 2016).

Techniquement Apache Zeppelin est une application web écrite en Java pour la partie back et en angularJS pour le front qui va permettre à des analystes de visualiser et requêter la donnée.

Comme les autres notebooks, Zeppelin vise à étendre la population des utilisateurs du Big Data en offrant de manière ergonomique et performante les fonctionnalités suivantes :

- ingestion de données,
- découverte et analyse des données,
- visualisation des données.

Zeppelin s'appuie fortement sur Spark pour l'analyse et l'exploration des données même si des alternatives sont possibles (Hive, Flink).

Zeppelin repose sur un système d'interpréteurs (Interpreters) qui permettent l'utilisation d'une technologie dans Zeppelin.

La liste bien que non exhaustive ci dessous permet de mesurer leur richesse :

- Hive,
- Markdown,
- Apache Spark (Scala, Python et R),
- Spark SQL,
- Apache Flink,
- Postgres,
- Pivotal HAWQ,
- Shell,
- Apache Cassandra,
- Apache Ignite.

Ajouté a cela il y a des interpréteurs très intéressants :

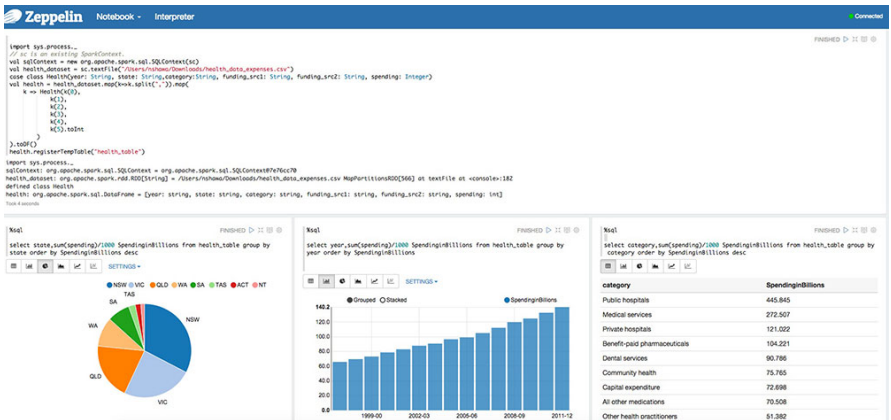
- AngularJS permet de créer des interfaces utilisateurs personnalisées,
- Apache Kylin (à l'origine un projet Ebay) est un projet open source qui vise à fournir des fonctionnalités de type OLAP et SQL à Hadoop,
- Apache Lens permet de fournir une vue unique à différentes sources de données (Hadoop, Data warehouse, ...) et de les requêter avec Spark.

L'interface est plus riche que Jupyter bien que le projet soit plus jeune. Le projet Hub a pour but de faciliter le travail collaboratif dans Zeppelin : différentes personnes d'une même société peuvent travailler sur les mêmes données sans toutefois interférer entre elles. Par contre il est possible de partager les résultats.

Le projet Helium est un changement de paradigme important puisque plutôt que de tout gérer dans un notebook Zeppelin (écriture du code et affichage des résultats), il permet d'exécuter un traitements puis d'intégrer les résultats à un notebook Zeppelin. Les possibilités de traitements sont donc plus importantes et seul l'envoi des résultats est intimement lié à Zeppelin. Devant le succès de Zeppelin on commence à le trouver dans différentes distributions Hadoop (Hortonworks).

Zeppelin est un projet jeune, toujours en incubation, ce qui se ressent sur certains points :

- pour l'installation il faudra le plus souvent construire le projet à partir des sources,
- instabilité.



Source image : [http://hortonworks.com/wp-content/uploads/2015/10/zeppelin\\_medical.jpg](http://hortonworks.com/wp-content/uploads/2015/10/zeppelin_medical.jpg)

### 10.11.3. Jupyter

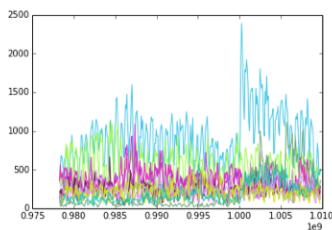
Jupyter (connu autrefois sous le nom de IPython Notebook) est un des précurseurs en termes d'accès unifié à la donnée. Le projet a démarré en 2001 sous l'impulsion de **Fernando Perez** mais la première version suffisamment mature et complète date de 2011. En 2015, IPython Notebook devient le projet Jupyter. La disparition de Python dans le nom du projet n'est pas innocente puisque Jupyter s'ouvre vers d'autres langages. On peut citer :

- Ruby,
- Javascript,
- Scala,
- Go,
- R,
- Java (à partir de la v9), ...

Dans Jupyter un notebook est une liste de cellules qui peuvent contenir du code, du texte, des formules mathématique ou encore des graphiques. Plus concrètement encore un notebook Jupyter est un document JSON. Rackspace, au travers de sa plateforme JupyterHub propose l'hébergement de notebooks et en héberge actuellement plus de 20 000. Le projet est assez actif est soutenu par une forte communauté (Rackspace, Microsoft, Continuum Analytics, Google, Github, ...) L'intégration avec Spark se fait au travers de PySpark.

Now we're ready to plot. Each object in `country_series` has the information we need to plot a single line.

```
In [27]: random_color = lambda: '#%02x%02x%02x' % tuple(np.random.randint(0,256,3))
fig = plt.figure()
ax = fig.add_subplot(111)
for (country, times, events) in country_series.takeOrdered(10, lambda x: -sum(x[2])):
    t = ax.plot(times, events, lw=1, c=random_color())
```



What's the big spike for the blue line above?

```
In [28]: country_day_counts.reduce(lambda x, y: max(x, y, key=lambda z: z[1]))
```

```
Out[28]: ((u'USA', u'20010912'), 2387)
```

Looks like it was the day after September 11th.

Source image : <http://blog.cloudera.com/wpcontent/uploads/2014/08/ipythonf21.png>

## 10.12. Offres Cloud

La multiplication des offres cloud a permis de démocratiser le Big Data, sur certains points elles sont même à l'origine du mouvement (beaucoup de solutions BigData proviennent des acteurs du cloud).

**Selon une étude IDC, les solutions Big Data & Analytics du cloud connaîtront une croissance 4,5 fois plus rapide que celle des plateformes physiques jusqu'en 2020.**

Le cloud sera donc la plateforme privilégiée pour les prochaines années.

Cf. <http://minely.com/blog/top10bigdataandanalyticspredictions2016/>



Les offres cloud facilitent l'adoption du Big Data en permettant de se focaliser uniquement sur la valeur métier.

Elles permettent une adoption plus rapide et une réduction des coûts grâce à une facturation sur mesure et extensible.

A l'opposé parmi les freins possibles il y a :

- la question de la sécurité des données,
- la maîtrise de l'évolution des versions des logiciels.

Voici une comparaison de quatre offres relativement complètes et matures.

### 10.12.1. Google

Nom offre	Google Cloud Engine Google Dataflow Google DataProc
NoSQL	BigTable
Traitements	Flink Spark (Google Cloud Dataflow)
Query	BigQuery
Hadoop	Google Cloud Dataproc (Spark et Hadoop) <ul style="list-style-type: none"><li>- Cloudera</li><li>- Hortonworks</li><li>- MapR</li></ul>

Stockage	Google Cloud Storage
Type cloud (public/ privé/Hybride)	Public
Machine Learning	Prediction API Mahout/MLlib

### 10.12.2. Rackspace

Nom offre	Managed Big Data ManagedNoSQL (ObjectRocket)
NoSQL	MongoDB Redis Cassandra HBase
Traitements	Spark Hadoop MapReduce TEZ
Query	Pig Hive Spark SQL
Hadoop	Hortonworks Data Platform

Stockage	HDFS
Type cloud (public/privé/Hybride)	Public/Privé
Machine Learning	Mahout/MLlib

### 10.12.3. Amazon

Nom offre	Amazon Elastic Compute Cloud (Amazon EC2)
NoSQL	<p>Solutions natives :</p> <ul style="list-style-type: none"> <li>DynamoDB</li> <li>SimpleDB</li> </ul> <p>Amazon EC2 :</p> <ul style="list-style-type: none"> <li>Cassandra</li> <li>Couchbase</li> <li>MarkLogic</li> <li>MongoDB</li> </ul> <p>Amazon AWS :</p> <ul style="list-style-type: none"> <li>Titan</li> <li>Neo4j</li> <li>OrientDB</li> <li>GraphDB</li> </ul> <p>Amazon ElastiCache :</p> <ul style="list-style-type: none"> <li>Redis</li> </ul>

Traitements	Spark (EC2) + Elastic Apache Mesos (EC2) Spark on EMR
Query	Spark SQL (EC2) Spark SQL (EMR)
Hadoop	Amazon Elastic MapReduce (EMR)
Stockage	HDFS Amazon Glacier Amazon Simple Storage Service (S3) Elastic Block Storage (EBS) Elastic File System (EFS)
Type cloud (public/ privé/Hybride)	Public
Machine Learning	Mahout/MLlib

## 10.12.4. Microsoft

Nom offre	Azure
NoSQL	MongoDB Neo4J DocumentDB Redis Table
Traitements	Azure Compute Stream Analytics Storm/Spark
Query	Stream Analytics
Hadoop	HDInsight (Hortonworks Data Platform)
Stockage	Azure Storage StorSimple (hybride)
Type cloud (public/ privé/Hybride)	Public/Hybride
Machine Learning	Mahout Cortana Analytics Suite Mahout/MLlib

## 11 — ARCHITECTURES

Aucune technologie ne permet de résoudre tous les types de problèmes posés.

Certaines architectures et solutions permettent de résoudre des problèmes complexes mais vont se retrouver sur dimensionnées pour des problématiques plus simples.

### 11.1. Hadoop

#### 11.1.1. Définition

Hadoop est un framework qui va permettre le traitement de données massives sur un cluster allant de une à plusieurs centaines de machines. Hadoop est écrit en Java et a été créé par Doug Cutting et Michael Cafarella en 2005 (après avoir créé le moteur de recherche Lucene, Doug travaillait alors pour Yahoo sur son projet de crawler web Nutch). Hadoop va gérer la distribution des données au cœur des machines du cluster, leurs éventuelles défaillances mais aussi l'agrégation du traitement final.

L'architecture est de type « Share nothing » : aucune donnée n'est traitée par deux nœuds différents même si les données sont réparties sur plusieurs nœuds (principe d'un nœud primaire et de nœuds secondaires).

Hadoop est composé de quatre éléments :

- Hadoop Common : ensemble d'utilitaires utilisés par les autres éléments,
- Hadoop Distributed File System (HDFS) : un système de fichiers distribué pour le stockage persistant des données,
- Hadoop YARN : un framework de gestion des ressources et de planification des traitements,
- Hadoop MapReduce v2 : un framework de traitements distribués basé sur YARN.

HDFS est un système de fichiers Java utilisé pour stocker des données structurées ou non sur un ensemble de serveurs.

C'est un système distribué, extensible et portable développé par le créateur d'Hadoop inspiré du système développé par Google (GoogleFS).

Écrit en Java, il a été conçu pour stocker de très gros volumes de données sur un grand nombre de machines équipées de disques durs locaux. HDFS s'appuie sur le système de fichier natif de l'OS pour présenter un système de stockage unifié reposant sur un ensemble de disques et de systèmes de fichiers hétérogènes.

MapReduce est un framework de traitements parallélisés, créé par Google pour son moteur de recherche web.

C'est un framework qui permet la décomposition d'une requête importante en un ensemble de requêtes plus petites qui vont produire chacune un sous ensemble du résultat final : c'est la fonction Map.

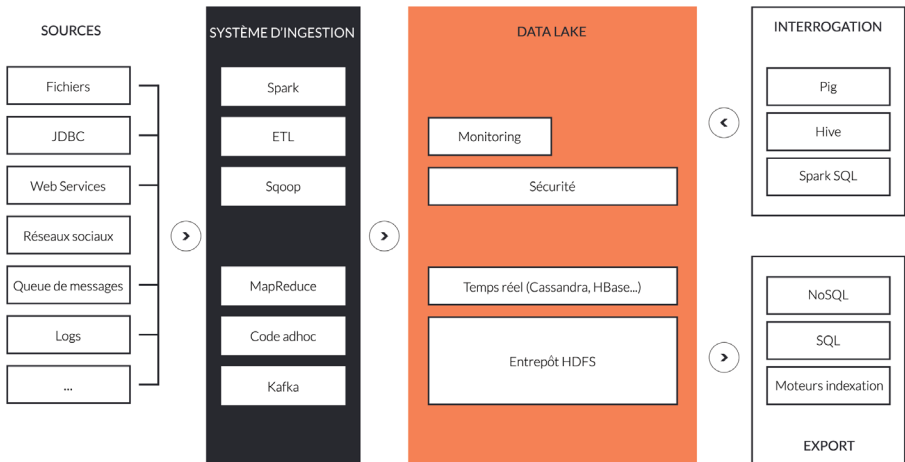
L'ensemble des résultats est traité (agrégation, filtre) : c'est la fonction Reduce.

MapReduce est idéal pour les traitements batchs, mais il n'est pas itératif par défaut. YARN est un gestionnaire de ressources (CPU et RAM) qui va permettre au cluster Hadoop de gérer la charge des serveurs.

Un processus MapReduce (ou Spark, Flink, ...) va utiliser un certain nombre de ressources, il sera lancé sur les nœuds disponibles ou bien mis en attente si les ressources sont insuffisantes.

Un cas d'utilisation très répandu actuellement est le "Data Lake".

## ~~DATA LAKE HADOOP~~



## 11.1.2. Synthèse

### 11.1.2.1. Points forts

- distribution des traitements au plus près de la donnée Hadoop (parallélisation),
- reprise automatique sur erreurs,
- coût de stockage très concurrentiel,
- écosystème très important.

### 11.1.2.2. Inconvénients

- performance (par rapport à NoSQL, Grille de données, ...),
- fait pour traiter de très gros volume de données.

## 11.2. Traitements de type Batch (incrémentaux)

### 11.2.1. Définition

Une architecture de type batch permet de traiter un ensemble de données en entrée jusqu'à épuisement de la source.

Tant que des données seront présentes les traitements vont se poursuivre et l'on aura un résultat cohérent et accessible uniquement à la fin des traitements.

Afin d'éviter cet effet tunnel il est possible de découper les données en entrée et c'est là que la notion incrémentale est importante.

Elle va permettre de prendre en compte les nouvelles données sans la nécessité de retraiter l'ensemble des données déjà traitées. Un parfait exemple de traitement Big Data de type Batch est Map Reduce dans sa version Hadoop.

D'autres frameworks peuvent aussi implémenter des traitements de type Batch :

- Apache Flink,
- Apache Spark,
- Apache Tez.

Les données sont d'abord sélectionnées par un traitement principal et



souvent unique.

Les données sont distribuées entre différents nœuds afin d'être traitées. Une fois les données traitées par l'ensemble des nœuds un traitement réalise les opérations globales:

- tri,
- agrégation,
- ...

## 11.2.2. Synthèse

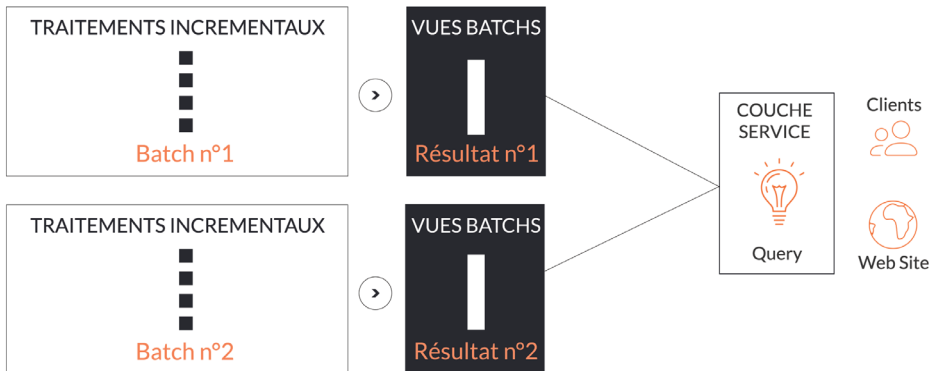
### 11.2.2.1. Points forts

- simplicité de mise en œuvre.

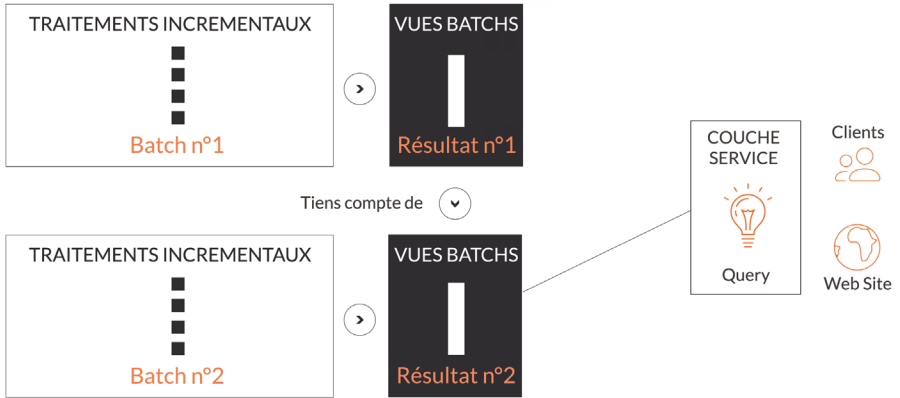
### 11.2.2.2. Inconvénients

- temps de traitement,
- les données arrivées en cours de traitement ne sont pas prises en compte.

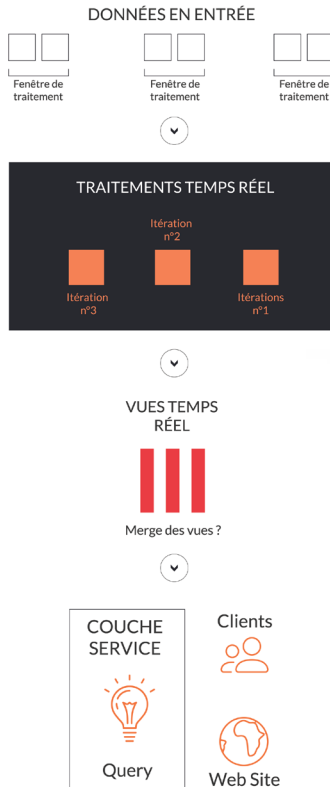
## ~~ARCHITECTURE BATCH INCRÉMENTALE~~



## ARCHITECTURE BATCH



## ARCHITECTURE LAMBDA



## 11.3. Traitements temps réel

### 11.3.1. Définition

On parle d'architecture temps réel ou streaming par opposition aux architectures de type batch.

Il n'est pas nécessaire d'attendre la fin des données en entrée pour émettre un résultat.

La notion de temps réel est toute relative et dépend du contexte : millisecondes, secondes ou encore minutes.

On distingue les solutions de type micro-batch des solutions de streaming

Micro-batch :

- un résultat est produit toutes les n secondes.

Streaming :

- chaque entrée est traitée immédiatement et produit un résultat.

Apache Spark est par exemple une solution de micro-batch alors que

Storm et Flink sont des solutions de streaming.

### 11.3.2. Synthèse

#### 11.3.2.1. Points forts

- temps de traitement modulable,
- performances (Améliore les temps de traitements),
- simplicité de mise en œuvre,
- peut être la base de solutions évolutives.

#### 11.3.2.2. Inconvénients

- fusion des vues pour produire un résultat plus complet,
- ne concerne que le traitement et doit donc être complété d'autres solutions (stockage, interrogation, ...).

## 11.4. Architecture Lambda

### 11.4.1. Définition

L'architecture Lambda a été imaginée par Nathan Marz et James Warren afin de résoudre des problématiques complexes mélangeant temps réel et batchs. L'architecture Lambda permet de stocker et de traiter de larges volumes de données (batch) tout en intégrant dans les résultats des batchs les données les plus récentes.

Marz et Warren sont partis du constat qu'il est plus simple de mettre en œuvre des traitements de type batchs que des traitements itératifs (plus de tolérance à l'erreur humaine, aux problèmes matériels ou encore aux bugs applicatifs).

Cette architecture permet la conservation des principes du Big Data :

- scalabilité,
- tolérance aux pannes, ...

Une architecture Lambda est composée de trois couches:

- couche batch (Batch Layer),
  - > stockage de l'ensemble des données,
  - > traitements massifs et réguliers afin de produire des vues consultables par les utilisateurs,
  - > la fréquence des traitements ne doit pas être trop importante afin de minimiser les tâches de fusion des résultats afin de constituer les vues.
- couche temps réel (Speed Layer) :
  - > ne traite que les données récentes (flux),
  - > calcul des vues incrémentales qui vont compléter les vues batch afin de fournir des données plus récentes,
  - > suppression des vues temps réel obsolètes (postérieures à un traitement batch).
- couche de service (Serving Layer) :
  - > permet de stocker et d'exposer aux clients les vues créées par les couches batch et temps réel,
  - > aussi capable de calculer dynamiquement ces vues,
  - > n'importe quelle base NoSQL peut convenir.

L'architecture Lambda est générique mais complexe dans le nombre de composants mis en œuvre.

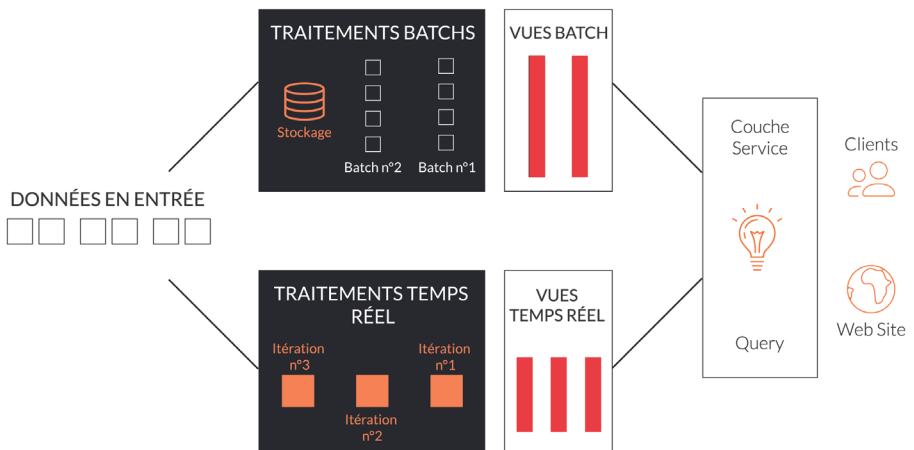
Il n'y a pas de solutions dédiées à cette architecture mais une multitude :

- stockage : NoSQL surtout mais aussi JMS, Kafka, HDFS,
- couche Batch : Hadoop MapReduce, Spark, Flink, ...
- couche Temps réel : Storm, Spark, Flink, Samza, Tez, ...
- couche de service : Druid, Cassandra, Hive, HBase, ElasticSearch, ...

Il existe toutefois des projets complets implémentant une architecture Lambda :

- Générique : Twitter Summingbird (<https://github.com/twitter/summingbird>)
- Dédicée au machine learning : Cloudera Oryx 2 (<http://oryx.io/>)

## ARCHITECTURE LAMDA



## 11.4.2. Synthèse

### 11.4.2.1. Points forts

- on conserve les données brutes afin de pouvoir les retraiter au besoin,
- la vision fournie aux clients est la plus fraîche possible,
- solution à tout faire,
- indépendant des technologies.

### 11.4.2.2. Inconvénients

- la logique métier est implémentée deux fois (dans la filière temps réel et dans la filière batch),
- plus de frameworks à maîtriser,
- il faut deux sources différentes des mêmes données (fichiers, web services),
- il existe des solutions plus simples lorsque le besoin est moins complexe, l'évolutivité des solutions Big Data permettra dans la plupart des cas de migrer vers une architecture Lambda lorsque le besoin l'exigera.

L'architecture Lambda est utilisée par des entreprises comme Metamarkets ou Yahoo

## 11.5. Architecture Kappa

L'idée de l'architecture Kappa a été formulée par **Jay Kreps** (LinkedIn). L'architecture Kappa est née en réaction à l'architecture Lambda et à sa complexité.

Elle est née d'un constat simple :

- la plupart des solutions de traitement sont capables de traiter à la fois des batch et des flux,
- l'architecture Kappa permet donc de simplifier l'architecture Lambda en fusionnant les couches Temps réel et Batch.

Elle apporte une autre évolution par rapport à l'architecture Lambda :

Le système de stockage des données est plus restreint et doit être un système de fichiers de type log et non modifiable (tel que Kafka).

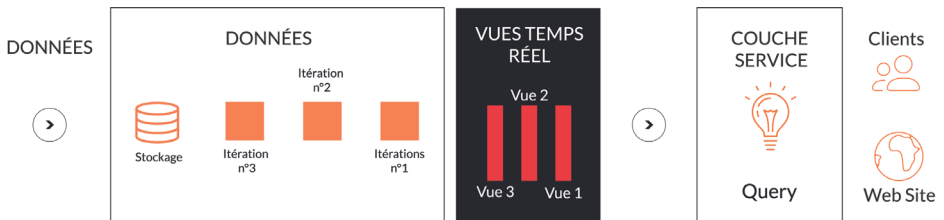
Kafka ou un autre système permet de conserver les messages pendant un certain temps afin de pouvoir les retraiter.

De fait, et encore plus que l'architecture Lambda, l'architecture Kappa ne permet pas le stockage permanent des données.

Elle est plus dédiée à leur traitement. Quoique plus restreinte, l'architecture Kappa laisse une certaine liberté dans le choix des composants mis en œuvre :

- stockage : Kafka, ...
- traitements : Storm, Spark, Flink, Samza, Tez, ...
- couche de service : Druid, Cassandra, Hive, HBase, ElasticSearch,...

## ~~ARCHITECTURE KAPPA~~



### 11.5.1. Synthèse

#### 11.5.1.1. Points forts

- solution à tout faire,
- indépendant des technologies,
- plus simple que l'architecture Lambda.

#### 11.5.1.2. Inconvénients

- pas de séparation entre les besoins,
- montée en compétence.

L'architecture Kappa est utilisée par des entreprises comme LinkedIn.

## 11.6. Architecture SMACK

L'architecture SMACK (pour Spark Mesos Akka Cassandra Kafka) est assez différente des architectures Lambda ou Kappa puisqu'elle est composée d'une liste de solutions.

Il faut donc bien comprendre les avantages et faiblesse des solutions avant de valider l'implémentation d'un cas d'utilisation.

Kafka est parfois remplacé par Kinesis sur le cloud (Amazon AWS)

Il est tout à fait possible d'implémenter une architecture Lambda ou Kappa avec ces solutions mais aussi adopter une architecture plus simple.

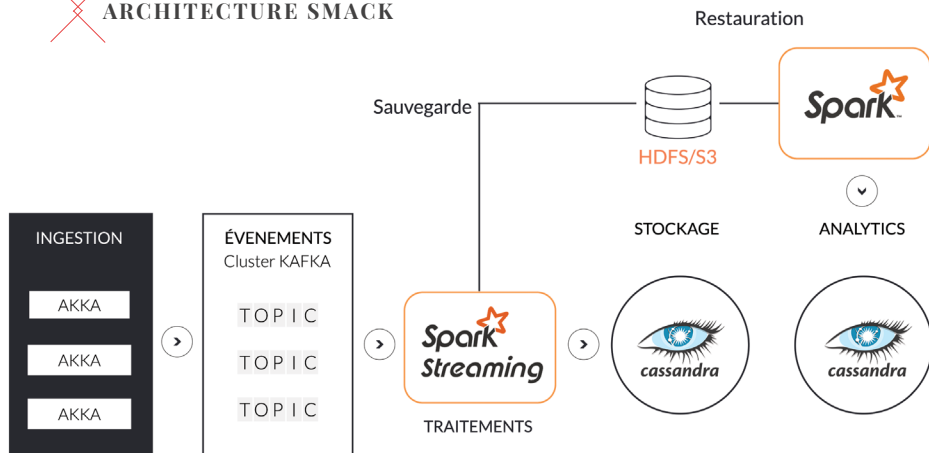
Choix de solutions matures, répondant aux exigences du Big Data :

- Spark : Framework de traitement des données
  - > Batch et streaming,
- Mesos : Gestion des ressources du cluster
  - > gestion des ressources du cluster (CPU/RAM), haute disponibilité grâce à Zookeeper,
- Akka : Implémentation du paradigme acteurs pour la JVM
  - > ingestion des données dans Kafka,
- Cassandra : Solution NoSQL
  - > stockage des données brutes mais aussi pour l'analyse des données,
- Kafka : Stockage des événements
  - > les événements de mise à jour sont stockés dans Kafka afin d'assurer leur persistance.

Certaines de ces solutions sont officiellement supportées par Mesos (Spark, Kafka) par contre l'intégration de Cassandra nécessite de s'appuyer sur des projets tiers comme ceux de Mesosphere (<http://mesosphere.github.io/cassandra-mesos/>)

Exemple d'implémentation de l'architecture SMACK





## 11.6.1. Synthèse

### 11.6.1.1. Points forts

- un minimum de solutions capable de traiter un très grand nombre de problématiques,
- solutions matures du Big Data,
- scalabilité des solutions,
- solution de gestion unique (Mesos),
- compatible batchs, temps réel, Lambda, ...

### 11.6.1.2. Inconvénients

- intégration de nouveaux besoins et donc de nouveaux frameworks,
- architecture complexe.

L'architecture SMACK est utilisée par des entreprises comme TupleJump ou ING.

## 11.7. Architecture acteurs

### 11.7.1. Définition

Le modèle d'acteurs est un modèle de programmation concurrente, dans lequel la charge de travail est répartie entre des entités s'exécutant en parallèle, les acteurs. C'est un modèle dans lequel il n'y a pas d'état partagé, les acteurs sont isolés et l'information ne peut circuler que sous forme de messages.

Ce modèle a été développé par Ericsson (1973) pour construire des systèmes télécoms hautement concurrents et disponibles (à l'origine lié au framework Erlang).

Le but est de fournir une abstraction pour les développeurs et la gestion des verrous et les threads de la programmation concurrente.

Les acteurs reçoivent ces messages et ne peuvent réagir qu'en manipulant les données dans le message (effectuer un calcul ou une transformation sur la donnée), en envoyant un message à d'autres acteurs ou en créant de nouveaux acteurs.

La philosophie adoptée en termes de tolérance de panne pour ces systèmes est «let it crash». En effet, les acteurs sont des éléments légers, qu'il est facile d'instancier et de relancer en cas de crash (du fait qu'ils ne partagent pas leur état).

Pour cela, Akka impose une supervision afin de pouvoir gérer la mort prématurée d'acteurs, et ce dans une hiérarchie de supervision : tout nouvel acteur est supervisé par son parent. Cette supervision permet la création de systèmes qui s'auto-réparent.

De plus les acteurs sont purement asynchrones et peuvent être relocalisés de manière transparente, y compris dans une autre JVM (ce qui permet la création d'un véritable système distribué sur un cluster).

Un acteur possède les caractéristiques suivantes :

- il possède un état interne qui peut évoluer au cours du

temps,

- il traite séquentiellement des messages stockés dans une queue,
- il peut émettre des messages asynchrones à d'autres entités,
- utilisation de messages immuables,
- non partage d'état entre les acteurs (par contre il peut y avoir une dépendance entre les acteurs : traitements successifs, sous routine, etc.

Les acteurs fournissent un cadre qui simplifie les problèmes de concurrence à traiter par le développeur.

Tolérance à l'erreur : philosophie de type « Let It Crash ».

Un acteur particulier nommé Superviseur est en charge du bon fonctionnement des autres acteurs. Il peut redémarrer un ou plusieurs acteurs selon la stratégie :

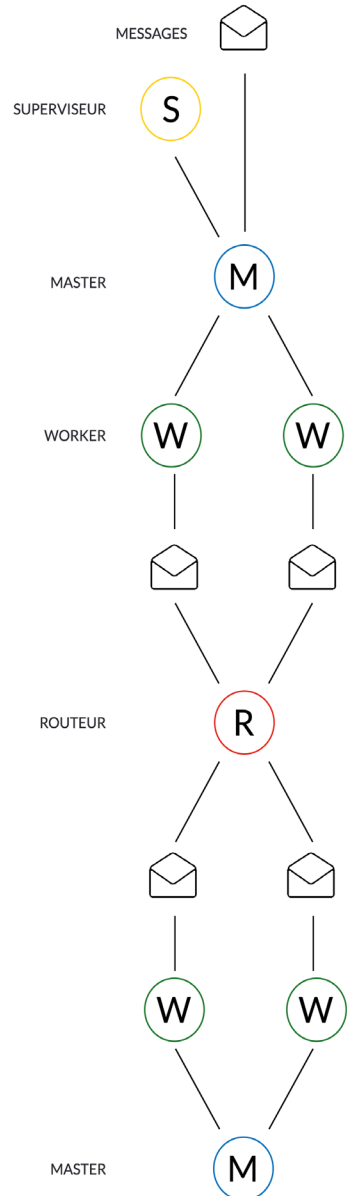
- for one : seul l'acteur levant une exception est redémarré,
- one for all : tous les acteurs supervisés sont redémarrés, cette stratégie est utilisée dans le cas de dépendances entre acteurs.

Il peut exister une arborescence de superviseurs avec un superviseur par domaine et un superviseur principal.

## 11.7.2. Synthèse

### 11.7.2.1. Points forts

- gestion native du multithreading,
- solutions matures et performantes (Akka).



### 11.7.2.2. Inconvénients

- architecture complexe.

## 11.8. Architecture Microservices

### 11.8.1. Définition

L'architecture microservices est souvent décrite comme l'architecture correcte de la SOA ("SOA done right"), je préfère utiliser le terme de Container-Oriented Architecture. Ce n'est pas une architecture complète et spécifique au Big Data.

Exposé par James Lewis et Martin Fowler, les principes fondamentaux en sont les suivants :

- un service ne traite qu'un périmètre restreint et clairement défini de fonctionnalités,
- chaque service est développé avec le langage et la base de données les plus adaptés pour répondre efficacement,
- un service doit être sans état,
- chaque service est autonome, s'exécute sur son propre serveur embarqué.

Le boom des microservices est dû à Netflix en 2010 en réponse aux problèmes engendrés par une architecture service traditionnelle (que l'on nomme par opposition au micro services : monolithes).

En effet les monolithes posent les problèmes suivants :

- dépendance technologique de tous les services
  - > stack technique,
  - > hardware,
- planning de déploiement partagé entre tous les services,
- SLA identique pour tous les services,
- si la charge d'un seul service augmente tous les services sont impactés,
- taille importante des équipes projets, ...

A l'opposé dans les microservices, chacun de ces services doit pouvoir exécuter une partie unique et spécifique de l'application, et être déployés

indépendamment des autres services.

De même pour éviter les problèmes des gros projets (taille des équipes, inertie, ...), il suffit de n'avoir que des petits projets.

Il ne s'agit pas de séparer les gros projets en sous équipes mais bien de projets indépendants : chacun a son organisation, son calendrier, sa base de code et ses données.

Le découpage se fait par domaine métier, en groupant les services et les types de données qui ont des liens forts, et en les séparant quand ils sont suffisamment indépendants.

L'architecture microservices est inspirée des technologies suivantes (et de leur maturité actuelle):

- architecture orientée service,
- mouvement DevOps,
- maturité des containers et de la virtualisation,
- NoSQL pour une mise en œuvre et déploiement simplifiés des solutions de stockage.

Des grands principes :

- scalabilité,
- expose des services avec un couplage lâche entre eux,
- pas de dépendance entre les services (Chaque service dispose de son serveur web, son datastore, ...),
- chaque service est hébergé et administré indépendamment.

Techniquement les microservices font la part belle aux web services (REST / JSON).

### **11.8.2. Testabilité**

Le fait de décomposer l'application en multiples sous applications complexifie les tests d'intégrations qui sont maintenant obligatoires avec cette architecture même si chaque service est théoriquement indépendant d'un point de vue métier.

Cela concerne essentiellement les services transverses (sécurité, frontal web, load balancing, transactions distribuées (Cf. point suivant, ...)).

### 11.8.3. Gestion des transactions

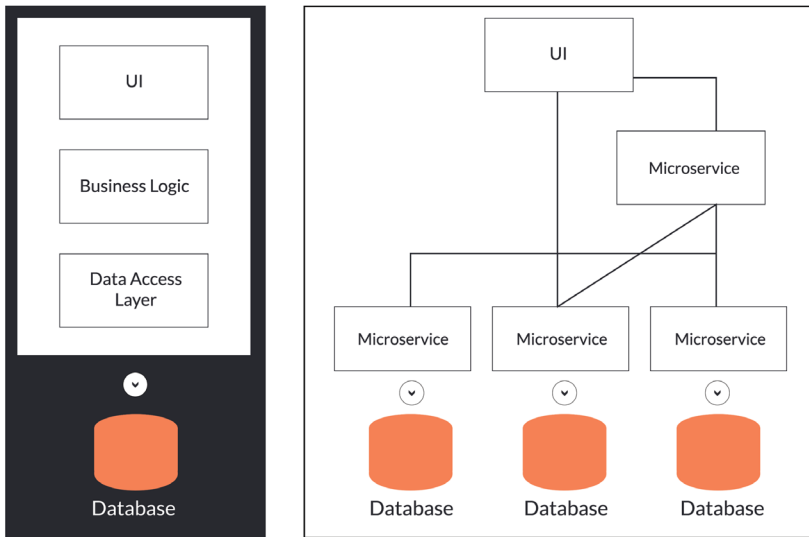
Si jamais une transaction est nécessaire entre deux services, il faut une transaction distribuée avec la complexité que cela induit.

Alternativement on peut implémenter des transactions de compensation afin d'éviter ces transactions distribuées.

Toutefois la meilleure pratique consiste à définir des services transactionnellement indépendants ou ne pas implémenter de transactions (sachant que la plupart des DataStores du Big Data ne les supportent pas).



### ARCHITECTURE MONOLITHE ET ARCHITECTURE MICROSERVICES



### 11.8.4. Synthèse

#### 11.8.4.1. Points forts

- scalabilité (Elasticité),
- découplage, ...

#### 11.8.4.2. Inconvénients

- Testabilité
- Gestion des transactions

## 11.9. Indexation (moteurs de recherche)

### 11.9.1. Définition

Le stockage et l'analyse des données ont souvent été deux besoins séparés et couvert par deux solutions différentes.

Objectifs des systèmes d'analyse des données :

- recherche en quasi temps réel,
- performances,
- navigation dans les données via les facettes, recherches par coordonnées géographiques,
- capacités d'analyse poussées via les facettes et le langage de requête.

Les systèmes de type SGBD-R ou DataWareHouse sont capables de répondre à ces exigences mais ils vont souffrir de certains maux :

- la capacité d'analyse est limitée (SGBD-R),
- scalabilité horizontale (SGBD-R et DWH),
- nécessite des données structurées ( SGBD-R et DWH),
- coût (SGBD-R et DWH), ...

Récemment nous avons vu l'émergence de systèmes capables de répondre à ces deux challenges : il s'agit de systèmes d'indexation de nouvelle génération tel qu'ElasticSearch.

Ces systèmes sont à la fois des solutions NoSQL pour le stockage et des moteurs d'indexation pour l'analyse des données.

ElasticSearch a été spécialement conçu pour :

- indexer de très gros volumes de données,

- assurer une forte montée en charge,
- assurer une grande tolérance aux pannes.

Des caractéristiques qui en font une solution Big Data.

Ces solutions s'affranchissent du besoin de schéma des données et vont stocker les informations au format brut afin d'offrir la possibilité de les réindexer au besoin.

Toutefois le système d'indexation est rarement la source de référence des données, les données sont souvent issues d'un autre système qui va pousser les données au fur et à mesure.

## **11.9.2. Synthèse**

### 11.9.2.1. Points forts

- rapidité,
- simplicité de mise en œuvre.

### 11.9.2.2. Inconvénients

- consommation mémoire importante,
- nécessite du paramétrage fin pour les fortes charges.

## **11.10. Architecture Data Lake**

### **11.10.1. Définition**

L'architecture "Lac de données" est un concept récent (2011) qui n'a réellement été rendu possible qu'avec l'apparition de systèmes capables de stocker des données non structurées.

On doit la paternité de ce terme à James Dixon, CTO de Pentaho.

Cette architecture est la conséquence directe des challenges du Big Data :

- une volumétrie exponentielle de données,
- des données sous exploitées,



- des données aux formats divers et non connu par avance.

En plus de cela intervient la notion de coût de stockage qui doit resté contenu. Un lac de données est un lieu de stockage universel dans lequel on peut mettre n'importe quel type de donnée, dans le format le plus proche possible de la source.

Il diffère donc des approches classiques de type Data Warehouse qui impose une structuration de la donnée au moment de l'écriture ("Schema on write"). Au moment de l'intégration des données, on trouve aussi différents traitements (qualité, filtrage, agrégation). Une fois intégrée, la donnée est donc figée dans un Data Warehouse.

A contrario dans un Data Lake, la donnée est stockée dans un format brute (souvent celui de la source) et ne subit aucune transformation. La donnée est donc brute et l'on va la "travailler" au moment de son utilisation ("Schema on read"), ce qui passe par un transfert des données dans un autre système.

C'est un atout de cette architecture étant donné que l'on ne connaît pas a priori les évolutions des sources de données et donc leur format. Le concept de Data Lake impose donc une séparation claire des responsabilités, le data Lake n'est en charge que du stockage des données, pas de leur traitement.

Les données sont mises à disposition pour des traitements de type :

- transformation
- mise en qualité,
- indexation,
- analyse, ...

Cas d'utilisation :

- analytics,
- vision 360,
- détection de fraude, ...

On trouve Hadoop et plus particulièrement HDFS et HBase.

Les avantages natif d'Hadoop dans ce cas d'utilisation sont le coût de stockage, la tolérance à la panne, la disponibilité de la plateforme et

surtout son évolutivité.

## 11.10.2. Synthèse

### 11.10.2.1. Parmi les solutions les plus fréquentes

- séparation entre stockage et exploitation de la donnée,
- coût de stockage réduit,
- disponibilité,
- évolutivité.

### 11.10.2.2. Inconvénients

- ne gère que le stockage,
- gouvernance des données obligatoire afin de suivre l'utilisation des données.

## 11.11. Critères de sélection d'une architecture

Le choix d'une architecture est en général difficile et le Big Data n'échappe pas à la règle.

Dans le cadre du Big Data cela est rendu encore plus difficile à cause de

certaines caractéristiques techniques :

- volumétries mises en œuvre,
- besoins temps réel,
- les évolutions constantes des solutions,
- on pose souvent les bases d'une architecture technique sans connaître tous les besoins.

Certaines caractéristiques fonctionnelles :

un des points attendus du Big Data est de réduire le Time To Market, ce qui laisse peu de temps pour repenser ou faire évoluer l'architecture.

Ainsi on peut soit se retrouver avec une architecture "jetable" (celle qui correspond parfaitement aux besoins exprimés) ou à l'opposé une architecture qui dans un premier temps s'avèrera trop complexe par rapport aux besoins réels (et qui anticipera les futurs besoins).

L'idéal est évidemment le compromis entre ces deux extrêmes, une

architecture Big Data devra être retenue en ayant en tête ces exigences contradictoires :

- être évolutive,
- faciliter son administration et la montée en compétences des équipes.

Le tableau suivant fait le lien entre une architecture, les critères de sélection principaux et les cas d'utilisation.

Architecture	Critère principal	Cas d'utilisation
Hadoop	Stocker de la donnée à un coût faible	Data Lake
Lambda	Construire une vision complète des données	Chaîne de traitement/ valorisation de la donnée
Kappa	Fournir une vision fraîche des données	Données métier à destination des utilisateurs
SMACK	Traiter de la donnée à un coût faible	Analyse des données (machine learning)

# 12 — SÉCURITÉ ET BIG DATA

## 12.1. Introduction

La sécurité avec le Big Data est rendue difficile car :

- de multiples solutions composent une plateforme Big Data et il manque encore des solutions globales de management,
- le principe même du Big Data est un système réparti sur plusieurs nœuds,
- hétérogénéité des solutions (stockage, traitements, analytics, indexation, ...).

De plus le Big Data est centrée sur les données et certaines sont naturellement sensibles :

- données bancaires (PCI-DSS),
- données médicales,
- données personnelles.

## 12.2. Challenges

Les enjeux de la sécurité :

- confidentialité des données
- contrôle des accès à la plateforme
- sécurisation des échanges
- audit.

Traiter les données à caractère personnel :

En France, l'usage des données à caractère personnel est réglementé par la loi "Informatique et Libertés".

Utilisation du Cloud : où sont stockées les données (dans quel pays et donc sous quelle réglementation ?)

Sécurité des logiciels/protocoles utilisés : régulièrement des failles de sécurité sont découvertes dans les produits et protocoles utilisés quotidiennement (java, SSL, etc.), les solutions Big Data ne sont évidemment pas épargnées.

Enfin la mise en œuvre des solutions de sécurité ne doit pas dénaturer la plateforme Big Data et notamment ses points forts :

- scalabilité,
- l'anonymisation ne doit pas perturber l'analyse sémantique et statistique des données,
- performances, ...

### **12.3. Implémentation**

A moins d'une architecture simple basée sur une seule solution, il n'existe pas de solutions globales permettant de sécuriser une architecture Big Data.

La sécurisation devra s'appuyer sur les possibilités offertes par les différentes solutions concernant la plateforme même si le système le plus sensible est bien évidemment le stockage.

Mais il ne faudra pas négliger :

- le transfert des données,
- le traitement des données,
- la visualisation des données, ...

En résumé la sécurisation d'une plateforme de Big Data est un chantier à part entière et demande une vision et une maîtrise de l'ensemble des solutions.

Enfin les mécanismes de sécurisation sont souvent inclus dans la version commerciales des solutions et peuvent à eux seuls justifier l'achat d'une licence et du support.

## 12.4. Tests et Big Data

### 12.4.1. Introduction

De nombreux points vont rendre les tests plus complexes dans les architectures Big Data :

- système hétérogène et distribué,
- volumétrie,
- données sensibles (anonymisation) .

Type de tests :

- données,
- infrastructure (résilience du système),
- validation des résultats.

### 12.4.2. Challenges

Voici les sources des dysfonctionnements dans les systèmes distribués d'après une enquête (Source : <https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-yuan.pdf>).

Le total dépassant les 100 % car il y a souvent plusieurs événements à l'origine du dysfonctionnement.

Événement à l'origine du dysfonctionnement	
Démarrage du service	58%
Écriture fichier/base depuis le client	32%
Nœud injoignable (réseau, crash, etc.)	24%

Changement configuration	23%
Ajout d'un nœud au cluster	15%
Lecture fichier/base depuis le client	13%
Redémarrage d'un nœud (planifié)	9%
Corruption données	3%
Autres	4%

On constate que seule une minorité des événements a pour origine l'utilisation du système :

- Écriture fichier/base depuis le client
- Lecture fichier/base depuis le client
- Corruption données

Les autres événements sont donc largement testables et peuvent être anticipés.

Les tests d'intégration et de robustesse sont donc essentiels dans une architecture Big Data.

Ce ne sont pas les seuls tests qui doivent être implémentés.

### 12.4.3. Catégories de tests

#### 12.4.3.1. Test intégration des données

Vérifier la capacité de la plateforme à intégrer les sources de données :

- qualité des données,
- intégration de multiples formats de données

#### 12.4.3.2. Test transformation des données

- vérifier la chaîne de valorisation des données de la plateforme,
- validation des résultats.

#### 12.4.3.3. Test chaîne d'intégration et de déploiement

- vérifier l'automatisation du déploiement de la plateforme,
- une plateforme automatisée diminue fortement les risques d'indisponibilité de la plateforme et améliorer la reproductibilité entre les différents environnements.

#### 12.4.3.4. Test scalabilité

- vérifier comportement de la plateforme en cas d'ajout/suppression de nœuds,
- infrastructure (résilience du système).

#### 12.4.3.5. Test de sécurité

- vérifier la vulnérabilité de la plateforme (données et traitements),
- qualifier la sensibilité des données et donc qui peut (et ne peut pas y avoir accès),
- en fonction de ces besoins valider que l'accès aux données, soit directement dans la zone de stockage ou bien au moyen de traitements, n'est pas possible.

#### 12.4.3.6. Test de performances

- vérifier les limites et le comportement de la plateforme sous



fortes sollicitations,

- les sollicitations doivent correspondre à des besoins réels (même futurs) afin d'éviter de surdimensionner la plateforme,
- ceci permettra de fixer les limites d'utilisation et les seuils qui nécessiteront une adaptation de la plateforme :
  - > ajout de machines,
  - > changement technologique.

#### 12.4.4. Solutions de tests

##### 12.4.4.1. Générateurs de données

Pas de tests sans données.

L'idéal est bien sûr d'utiliser des données réelles mais ce n'est pas toujours possible.

Mockaroo est un générateur online de données : <http://www.mockaroo.com/>

##### 12.4.4.2. NoSQL

NoSQL Unit <https://github.com/lordofthejars/nosql-unit> (MongoDB, Cassandra, HBase, Redis, Neo4j)

##### 12.4.4.3. Robustesse

**Toxiproxy** (Shopify) : Un proxy écrit en Ruby permettant de simuler des coupure ou des ralentissement dans les réponses des services.

**Jepsen** : Outil pour simuler un partitionnement réseau.

**Chaos Monkey** (Netflix) : Outil automatisé qui va permettre de simuler des pannes dans chacun des éléments de l'architecture.

## 13 — CONCLUSION ET ANNEXES

Ce White Paper dresse un état des lieux des technologies utilisées dans le Big Data mais aussi des opportunités qui sont offertes aux entreprises par une meilleure exploitation des données.

Nous avons vu qu'il existe de nombreuses solutions, directement induites par les cas d'utilisation.

La mise en place d'une architecture Big Data doit donc être liée à un besoin et il faut cibler les domaines dans lesquels il est le plus pertinent pour vous d'investir.

La meilleure organisation pour une entreprise orientée vers la valorisation des données est une équipe :

- agile (ce qui ne surprend plus personne en 2016),
- pluridisciplinaire (Data Scientist, Data Engineer, Graphistes, Marketing, ...),
- de taille modeste (20 personnes maximum).

Une bonne pratique est d'ouvrir l'accès à la donnée au sein de l'entreprise, au moyen d'un tableau de bord par exemple. Cet exercice facilite l'implication de tous les acteurs de la société.

En conséquence il faudra former un maximum d'utilisateurs et aussi se préparer à voire émerger des idées nouvelles qui devront impérativement faire l'objet d'un retour.

Enfin ce White Paper sera suivi de déclinaisons par secteur d'activité et par cas d'utilisation :

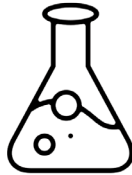
- taux d'attrition (churn) dans l'assurance,
- détection de la fraude dans le monde bancaire,
- traitement de très gros volumes en temps réel.

## 13.1. Matrice de recommandation

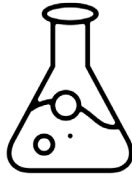
### ~~BIG DATA, INGRÉDIENTS~~



DATA  
DELUGE



DATA  
DIVERSITY



DATA  
FAST



DATA  
VALUE



DATA  
QUALITY

Techniquement les enjeux sont les suivants :

- scalabilité,
- latence,
- puissance de traitement,
- fiabilité,
- tolérance à la panne.

Idéalement le système retenu devra répondre à tous ces enjeux mais cela à un coût au niveau de la complexité et de la maintenabilité.

Il faut ajuster ces critères à ses besoins.

Cas d'utilisation	Caractéristiques	Architecture préconisée
Data Lake	Fort volume de données Données hétérogènes	Hadoop
Recommandations en temps réel	Vitesse de traitement des données	Lambda Zeta

Améliorer les offres/ produits existant	Fort volume de données Transformer et agréger les données Capacités analytiques puissantes	Batch Micro batch
Vue 360 client	Agrégation de silos	Batch Temps réel
Réduire coût de stockage	Fort volume de données	Hadoop NoSQL
Scoring	Fort volume de données Capacités analytiques puissantes	Temps réel

## 13.2. Lexique

Notions	Définitions
Arbres de décision	Algorithme permettant la résolution de problèmes en les représentant sous forme d'arbre dans lequel chaque feuille représente une solution possible, les branches les choix à suivre.
Churn	Le churn ou taux d'attrition correspond à la part des clients perdus sur une période.

Clickstream	<p>Il s'agit du flux de clics généré en permanence par les utilisateurs d'un site Internet.</p> <p>Il est donc la source principale de l'analyse de l'activité des utilisateurs au travers de leurs clics sur une page web. C'est une source précieuse d'information pour les algorithmes de Machine Learning.</p>
COA (Container Oriented Architecture)	<p>L'architecture Orientée Container vise à découper le Système d'Information en services indépendants qui seront déployés dans des containers.</p>
Data Analyst	<p>Maîtrisant les outils du Big Data et les statistiques, le Data Analyst code les algorithmes prédictifs sur la plateforme analytique.</p>
Data Cleansing	<p>Littéralement nettoyage des données. C'est une phase qui consiste à supprimer les données incohérentes, corriger les erreurs comme, par exemple, des données mal saisies. Disposer d'informations d'un bon niveau de qualité est un préalable à l'élaboration d'algorithmes de Machine Learning.</p>
Data Engineer	<p>C'est l'informaticien, spécialiste du Big Data, qui va mettre en œuvre tous les outils et solutions à destination des utilisateurs.</p> <p>Son périmètre de compétence s'arrête à l'analyse des données.</p>

Data Lake	L'approche Data Lake ou lac de données consiste à mettre en place un cluster Hadoop, le plus souvent, où vont converger toutes les données brutes que l'entreprise peut capter. Cela permet de s'affranchir des silos fonctionnels ou techniques d'une entreprise.
Data Management Platform	Une plateforme DMP permet de récupérer, centraliser, gérer et utiliser les données relatives aux prospects et clients. Les plateformes DMP sont utilisées pour optimiser le ciblage et l'efficacité des campagnes marketing et publicitaires.
Data Mining	Permet de trouver des corrélations, des modèles, des tendances parmi les données en utilisant des algorithmes statistiques et mathématiques.
Data Scientist	Poste à double compétence car il est capable d'utiliser les outils informatiques du Big Data (voire de coder en Python, R, ...) et de comprendre les enjeux business de ses analyses.
Datastore	Système de persistance des données (Hadoop, SGBD, NoSQL, ...).

Data Governance (Gouvernance des données)	Ensemble des techniques permettant de s'assurer de la gestion, de l'exploitation, de l'optimisation, et du contrôle des données des entreprises. L'activité consiste aussi à s'assurer que les bonnes pratiques du management de la donnée sont appliquées.
Data Visualisation	Aussi nommée « dataviz », il s'agit de technologies, méthodes et outils de visualisation des données. Elle peut se concrétiser par des graphiques, des camemberts, des diagrammes, des cartographies, des chronologies, des infographies, ou même des créations graphiques inédites. La présentation sous une forme illustrée rend les données plus lisibles et compréhensibles.
Dark data	Il s'agit de la donnée qui est collectée, utilisée et stockée par les entreprises mais qui n'est pas exploitée à des fins d'analyse et qui par conséquent n'est pas monétisée.
Dirty data	Données brutes, non filtrées et inutilisables pour les systèmes d'analyse.

<p>KPI (Key Performances Indicators)</p>	<p>Ceux-ci variant en fonction de ce que vous mesurez :</p> <ul style="list-style-type: none"> <li>- pour un site web,</li> <li>- visites,</li> <li>- visiteurs,</li> <li>- nouveaux visiteurs,...</li> <li>- pour une campagne email, taux</li> <li>- d'ouverture, de réactivité, de clics,...</li> </ul>
<p>LDAP (Lightweight Directory Access Protocol)</p>	<p>LDAP est un protocole permettant l'interrogation et la modification des services d'annuaire.</p>
<p>Machine Learning</p>	<p>Discipline issue de l'intelligence artificielle, le Machine Learning ou apprentissage automatique consiste au développement d'algorithmes qui apprennent un phénomène à partir des données. L'apprentissage est automatique, à la différence du Data Mining classique, où les analyses sont réalisées par le statisticien, a posteriori.</p>
<p>MDM (Master Data Management)</p>	<p>Ensemble de processus visant à gérer les données de références de l'entreprise au sein d'une base de données.</p>
<p>NoSQL</p>	<p>Système de gestion de bases de données fondé sur des principes de non relation entre les documents et de scalabilité horizontale.</p>



Open Data	Principe de donner accès à tous à des données numériques. Celles-ci sont mises en ligne et réutilisables par tous. Une “donnée ouverte” est publiée de manière “Complète, Primaire, Opportune, Accessible, Exploitable, Non-Discriminatoire, Non-Propriétaire, Libre de droits, Permanente, et Gratuite” (Srce Association Libertic).
Prédictif	Les algorithmes prédictifs constituent une application directe des techniques de Machine Learning dans le Big Data. A partir d'un historique d'achats, de sessions de navigation sur un site internet, ces algorithmes vont prédire quels seront les prochains besoins d'un consommateur. A partir de l'analyse des vibrations d'un moteur, un algorithme prédictif va diagnostiquer une panne avant qu'elle ne survienne.
Qualité des données	C'est l'un des problèmes clés du Big Data pour que les algorithmes fonctionnent correctement, ils doivent pouvoir s'appuyer sur des données fiables et cohérentes. Cela impose un gros travail de nettoyage en amont pour ne pas faire ce qu'on appelle du «Machine Learning on dirty data».
Régression logistique	Algorithme prédictif utilisé dans le scoring des clients.

Scalabilité	<p>Capacité d'un système à pouvoir supporter l'augmentation de charge.</p> <p>On distingue :</p> <ul style="list-style-type: none"><li>- horizontale : ajout de serveurs</li><li>- verticale : augmentation des capacités du serveur</li></ul>
Scoring	<p>Note attribuée à un prospect pour évaluer son appétence à une offre, le risque de perte de son client (attrition) ou encore un risque d'impayé ou de faillite d'une entreprise.</p>
Smart Data	<p>Principe d'analyse et de valorisation de la donnée brute.</p> <p>Ce principe sous entend que les données brutes ne sont "smart" par défaut. Seule une analyse permettra d'en extraire les informations les plus pertinentes dans le contexte de l'entreprise.</p>
SOA (Service Oriented Architecture)	<p>L'architecture Orientée Service vise à transformer le Système d'Information en services unitaires et métier.</p>

### 13.3. Liens et références intéressants

Survey finds Big Data experts spend up to 90% of their time preparing the data for analysis : <http://www.datamation.com/data-center/big-data-mostly-prep-work.html>

Java on GPU vs CPU : <https://medium.com/@jmaxg3/java-on-gpgpus-845d9ba58533>

Gartner status of Big Data : <https://www.gartner.com/doc/3115022/>

Gartner Big Data Industry Insights : <http://www.gartner.com/webinar/2931518>

Principles and best practices of scalable realtime data systems (Nathan Marz et James Warren) : <https://www.manning.com/books/bigdata>

Articles Big Data Ippon : <http://blog.ippon.fr/category/big-data/>

Algorithmes d'apprentissage : [a-tour-of-machine-learning-algorithms](#)

Théorème CAP : [cap-twelve-years-later-how-the-rules-have-changed](#)

Pattern microservices : [service-per-container](#)

La France et le cloud : [marche-du-cloud-en-france-en-2015](#)

Techniques de modélisation : [nosql-data-modeling-techniques](#)

## 14 — REMERCIEMENTS

L'auteur remercie l'ensemble des relecteurs pour leurs suggestions bienveillantes et critiques :

- Sébastien Bergia
- Bertrand Pinel
- Fabien Arrault
- Pierre Baillet
- Alexis Seigneurin
- Aurélien Rabache
- David Martin
- Geoffray Gruel
- Julien Dubois
- Michael Leneveut
- Raphaël Brugier
- Sébastien Mazade
- Stéphane Lagraulet
- Faten Habachi et toute l'équipe Marketing et Edition.



---

PARIS  
BORDEAUX  
NANTES  
LYON

RICHMOND, VA  
WASHINGTON, DC  
NEW-YORK

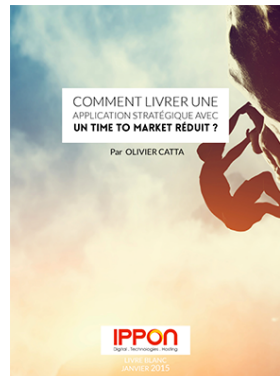
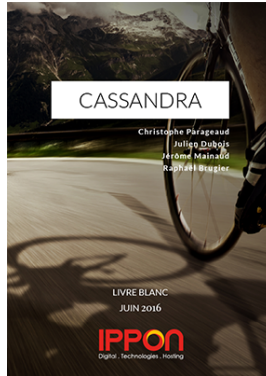
MELBOURNE  
MARRAKECH

*[www.ippon.fr/contact](http://www.ippon.fr/contact)  
[talents@ippon.fr](mailto:talents@ippon.fr)  
[blog.ippon.fr](http://blog.ippon.fr)*

+33 1 46 12 48 48  
*@ippontech*

# Découvrez aussi nos autres ressources

sur <http://www.ippon.fr/ressources/>





**IPPON**  
Digital . Technologies . Hosting

***www.ippon.fr***  
***blog.ippon.fr***

Paris  
Nantes  
Bordeaux  
Lyon  
Washington, DC  
Richmond, VA  
New York, NY  
Melbourne  
Marrakech