Lambda+ Architecture

Un patron d'architecture haute performance pour le traitement des Big Data

Annabelle Gillet & Éric Leclercq & Nadine Cullot

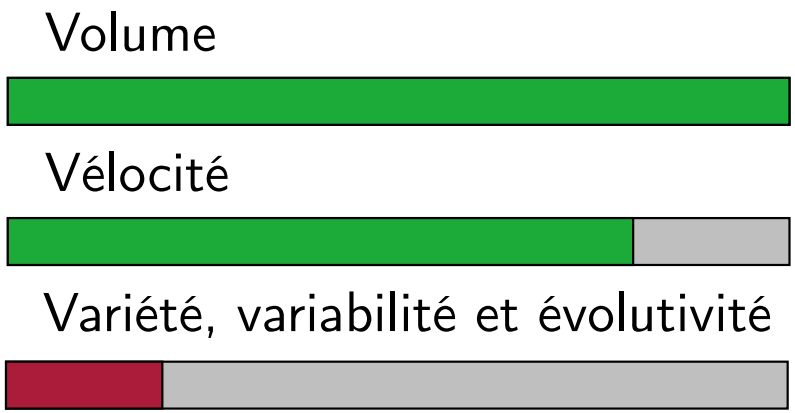
LIB - EA 7534 Université de Bourgogne Franche-Comté
Équipe Science des Données

JCAD 2020
2-4 Décembre 2020

Plan

Big Data

- L'exploitation des Big Data est devenue une préoccupation importante dans différents domaines comme le marketing, la politique, l'environnement, et des disciplines scientifiques telles que l'astronomie, la biologie, etc.
- L'augmentation de la puissance des machines et de leur utilisation en cluster prend en compte inégalement les problématiques induites par le traitement des Big Data :



Les architectures logicielles

La spécification d'architectures logicielles permet de maîtriser cette complexité :

- Les **besoins** et **propriétés** attendues (ex. : résistance aux pannes, passage à l'échelle, contrainte temps réel) sont les éléments fondamentaux à déterminer
- La définition des **composants** (ex. : SGBD, Data Lake, middleware, ETL, stream processing) et de leurs **interactions** (ex. : échanges de messages, transactions, web services) dépendent de ces éléments

Les styles et patterns d'architecture permettent de guider la conception.

Les styles et patrons d'architectures

Styles

Les styles ont une granularité élevée, et les styles guident les interactions entre les composants. Chaque style apporte certaines propriétés, tout en imposant des compromis sur d'autres propriétés.

Exemples de styles :

- Layered architecture
- Micro-services architecture
- Event-driven architecture

Patrons

Les patrons sont une abstraction d'un style d'architecture, qui permettent de répondre à des situations plus spécifiques.

Exemples de patrons :

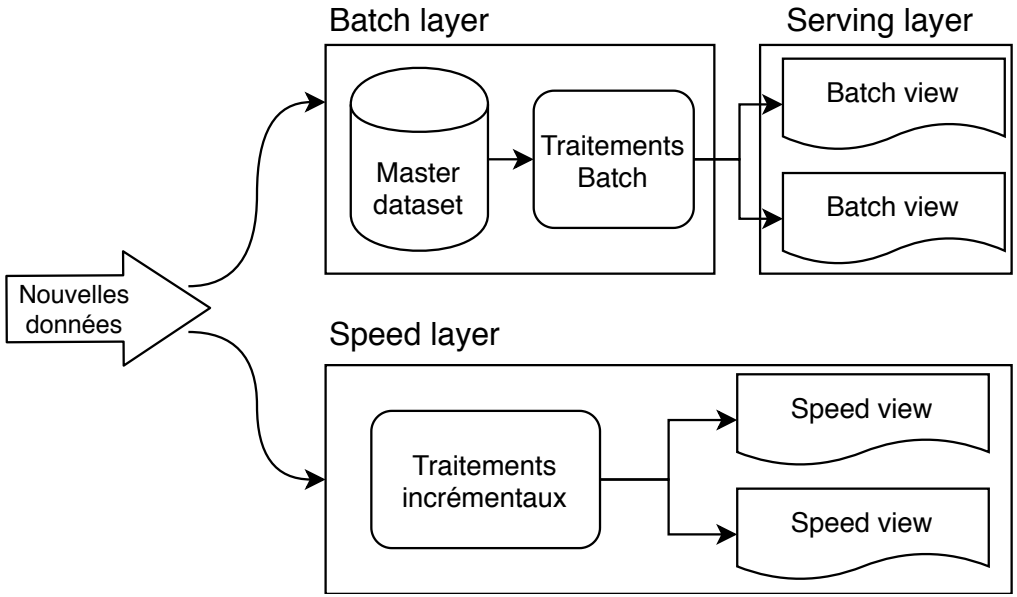
- Blackboard
- Model View Controller (MVC)
- Lambda Architecture

Plan

Définition

Lambda Architecture

- **Speed layer** : production des résultats approximatifs en continu (*speed views*), en temps réel grâce au *stream processing*
- **Batch layer** : production des résultats exacts périodiquement (*batch views*), grâce aux traitements par lots (*batch*)
- **Serving layer** : agrégation des vues *speed* et *batch*, pour répondre aux requêtes d'utilisateurs



Limites de la Lambda Architecture

- La **duplication des traitements** en *batch* et en *stream processing* induit une importante **complexité** pour obtenir des résultats identiques dans les couches *batch* et *speed*
- Les **cas d'utilisations sont limités** : il faut **connaître en amont** les traitements qui sont à effectuer, ce qui limite la prise en compte de la variabilité des Big Data
- La **définition manque de précision**, notamment concernant l'agrégation des données dans la *serving layer*
- Le contexte de création de la Lambda Architecture est fortement lié aux débuts des systèmes de *stream processing*

Les systèmes de *stream processing*

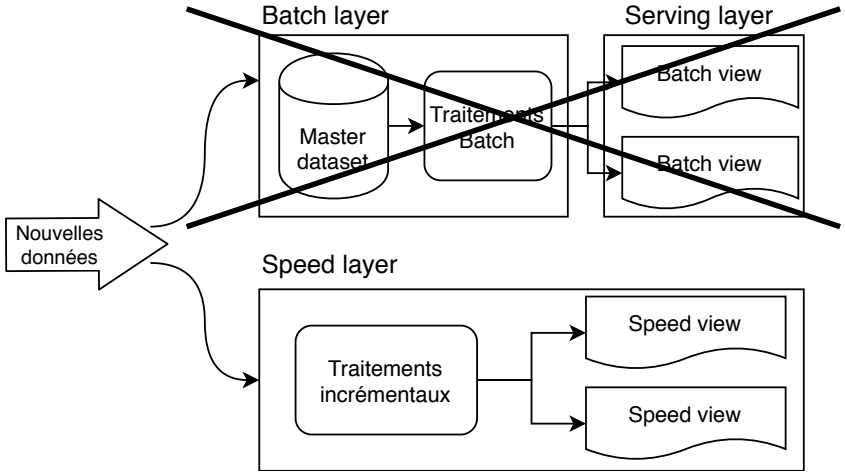
- Lors de l'émergence de la Lambda Architecture, les systèmes de *stream processing* n'en étaient qu'à leur début, et favorisaient les performances en proposant deux garanties de traitement des données :
  - ***At-most-once*** : les données sont traitées au plus une fois (elles peuvent ne pas être traitées)
  - ***At-least-once*** : les données sont traitées au moins une fois (elles peuvent être traitées plusieurs fois)

- Depuis, les systèmes de *stream processing* ont évolué, et proposent un nouveau niveau de garantie :
  - ***Effectively-once*** (souvent appelée ***exactly-once***) : les données sont traitées une et une seule fois

Effectively-once

Cette garantie est toutefois à nuancer : elle n'est effective que d'un point de vue du *stream processing*, ce qui veut dire que si un traitement possède des effets de bord (ex. : écriture dans un fichier), ils ne sont pas couverts par la garantie et peuvent être effectués plusieurs fois.

Kappa Architecture



- Simplification de la Lambda Architecture, en s'appuyant sur la maturité des systèmes de *stream processing*
- Perte de la capacité native de la forte résistance aux pannes
- Les cas d'utilisation restent restreints

Plan

Cas d'utilisation

Indicateurs en temps réel

Production d'indicateurs en temps réel, qui correspondent à des besoins bien identifiés.
De nouveaux indicateurs peuvent être ajoutés en fonction des découvertes dans les analyses exploratoires.

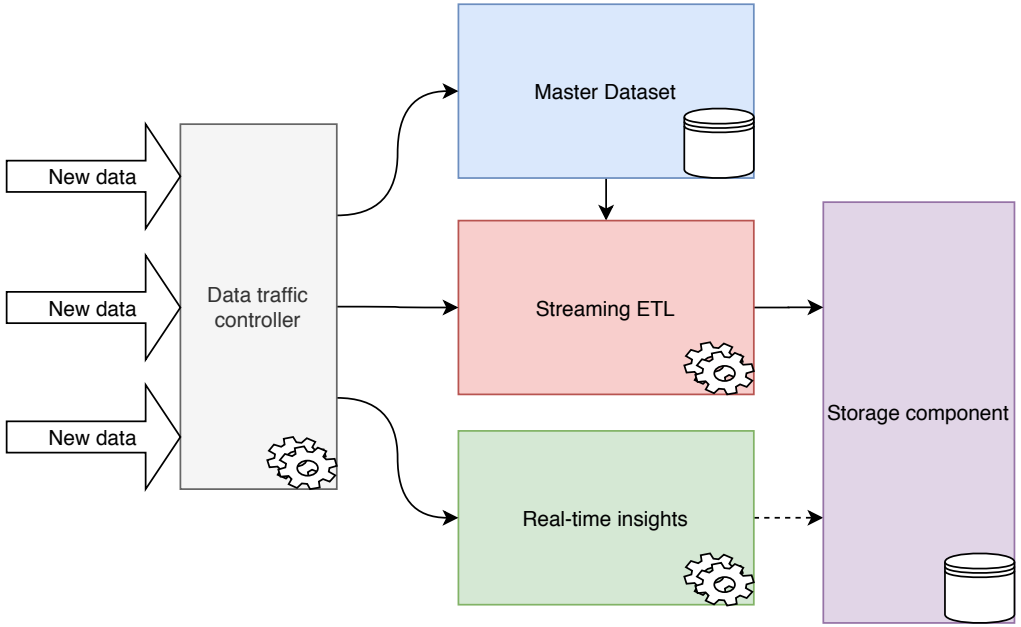
Analyses exploratoires

Analyses exploratoires, pour fouiller les données lors de situations ou besoins spécifiques.
Les analyses peuvent être guidées par des anomalies détectées dans les indicateurs en temps réel.

Les composants de la Lambda+

Lambda+ Architecture

- **Data traffic controller** : ordonne et réalise des transformations légères sur les flux de données sources
- **Master dataset** : conserve les données brutes, et les rend accessibles en cas de besoin
- **Streaming ETL** : insère les données qu'il reçoit en flux dans le *storage component*
- **Real-time insights** : calcul des indicateurs prédéfinis en temps réel directement sur le flux de données
- **Storage component** : conserve et met à disposition les données pour les analyses exploratoires



Avantages de la Lambda+

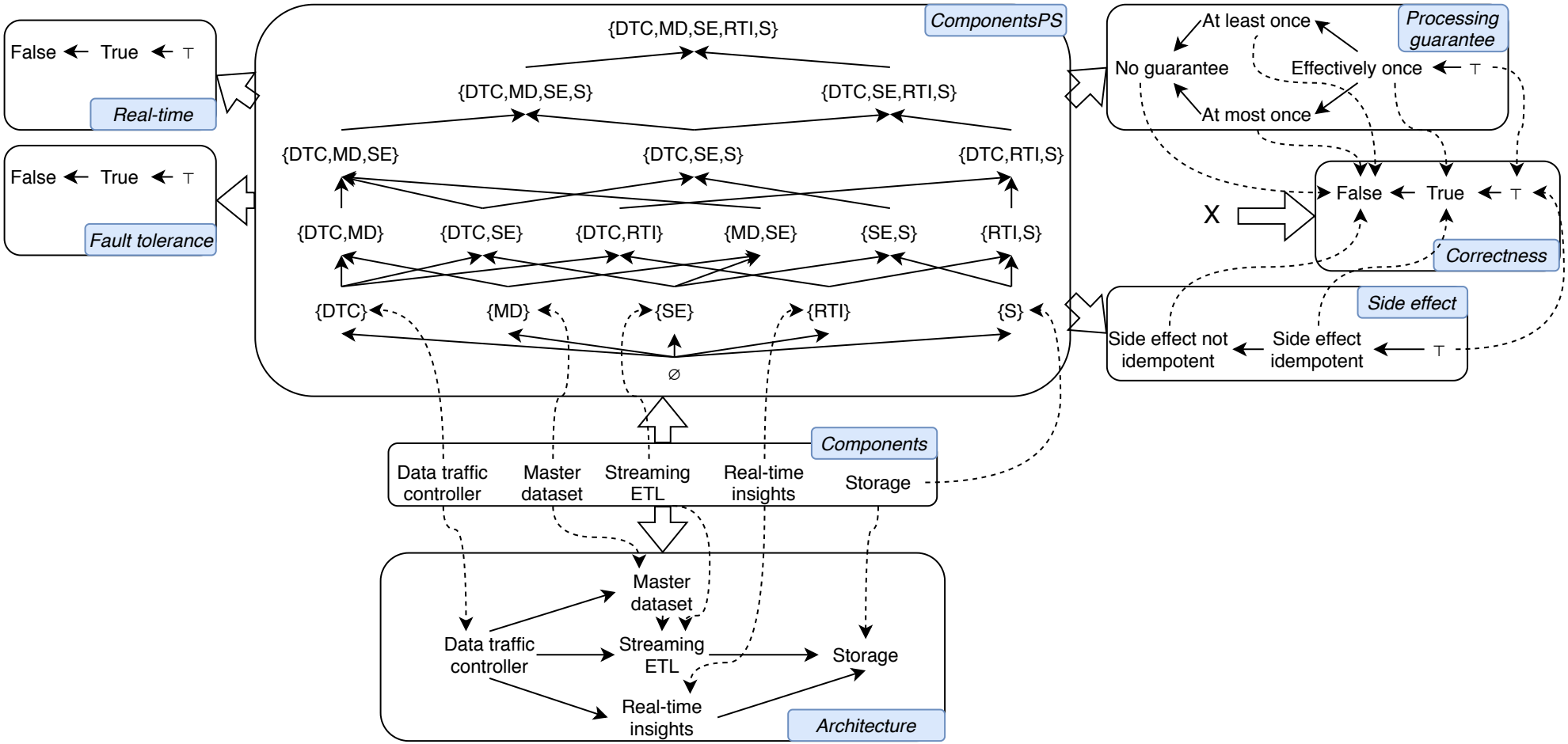
- Conservation de la capacité de **résistance aux pannes** de la Lambda Architecture, tout en **supprimant la duplication des traitements** :
  - le master dataset conserve les données brutes
  - l'extraction permettant de relancer le traitement en cas de besoin renvoie les données vers le composant de streaming ETL
- **Gain de flexibilité** : les analyses exploratoires permettent de trouver de nouveaux traitements pour extraire des indicateurs pertinent en temps réel

Formalisation

Basée sur la théorie des catégories, elle permet de :

- **déduire la formation des propriétés complexes** en connaissant les propriétés simples dont elles dépendent
- **déduire les propriétés qui sont conservées** ou non lorsque les composants sont considérés dans leur ensemble
- **naviguer entre les différents niveaux d'abstraction**, et détailler une partie de l'architecture
- vérifier qu'une **implémentation d'architecture respecte un style ou un patron**

Formalisation



Formalisation des composants, de leurs interactions et de leurs propriétés.

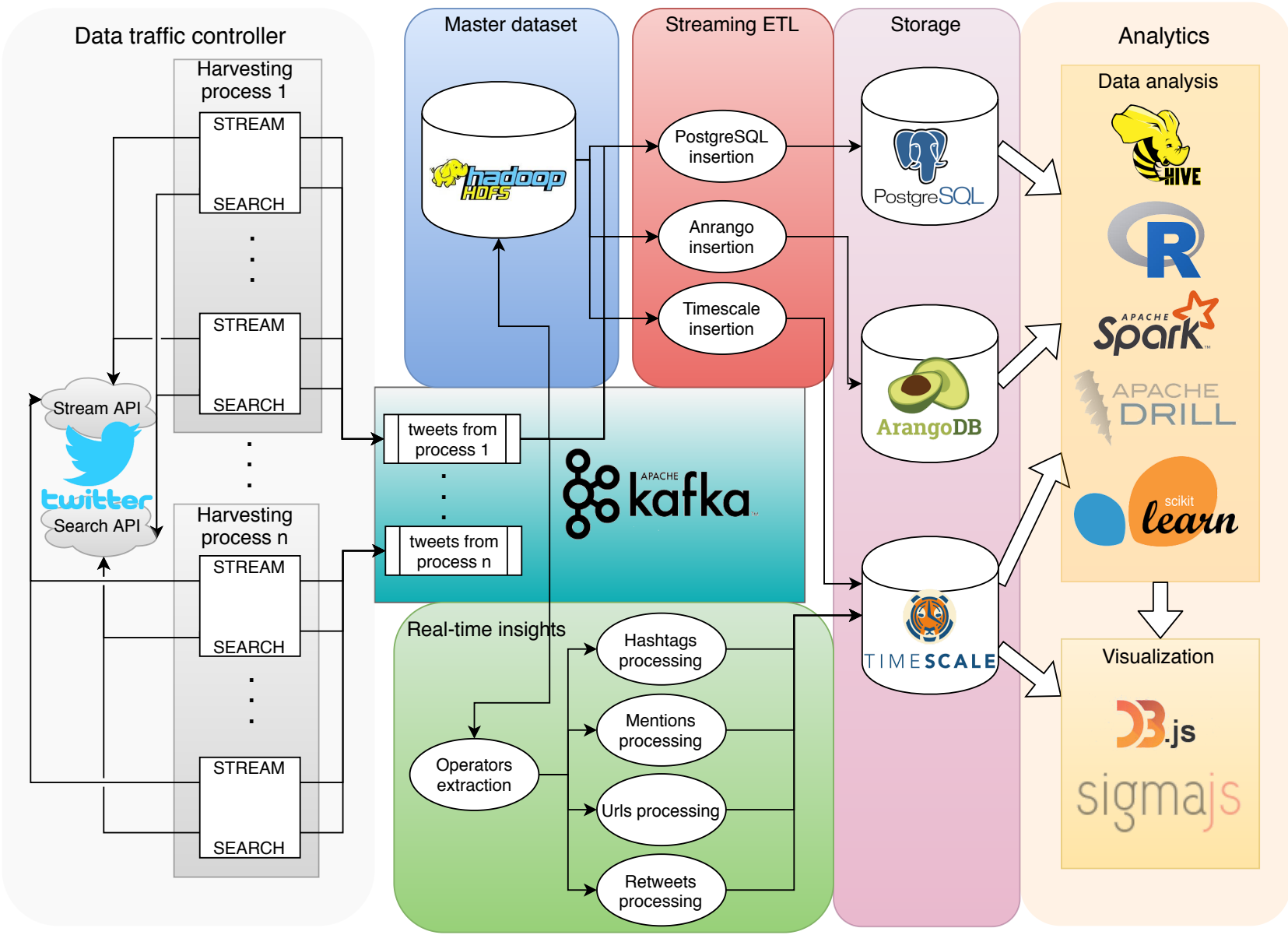
Plan

Projet Cocktail

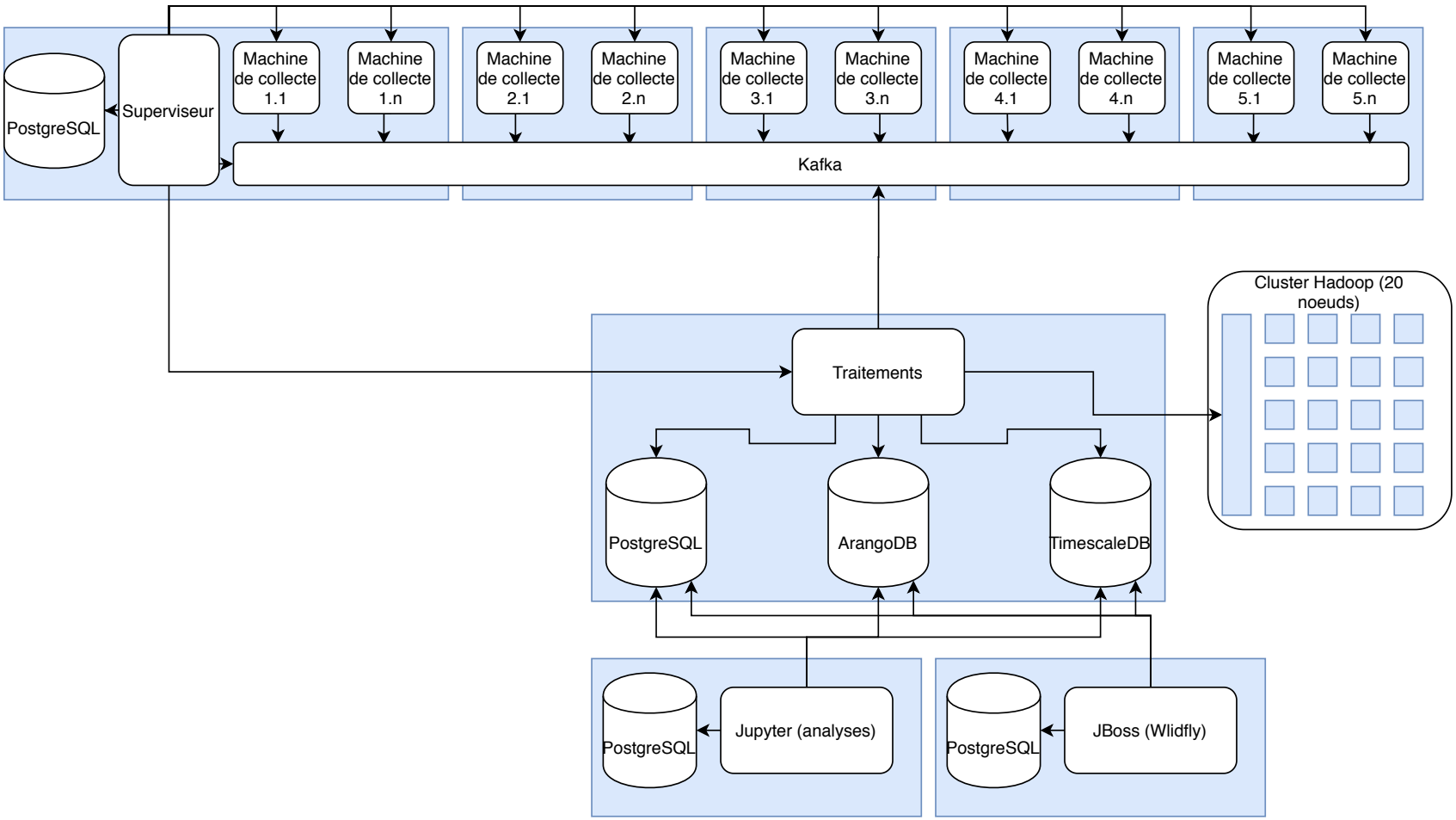
ISITE-BFC (ANR-15-IDEX-0003)

Le but du projet Cocktail est de créer un observatoire en temps réel des tendances, des singularités et des signaux faibles dans les discours à propos de l'alimentaire et de la santé sur Twitter.

Définition


Data traffic controller
Harvesting process 1
STREAM
SEARCH
STREAM
SEARCH
Stream API
twitter
Search API
Harvesting process n
STREAM
SEARCH
STREAM
SEARCH
Master dataset
hadoop HDFS
Streaming ETL
PostgreSQL insertion
Anrango insertion
Timescale insertion
tweets from process 1
tweets from process n
APACHE kafka
Real-time insights
Hashtags processing
Mentions processing
Urls processing
Retweets processing
Operators extraction
Storage
PostgreSQL
ArangoDB
TIMESCALE
Analytics
Data analysis
HIVE
R
APACHE Spark
APACHE DRILL
scikit learn
Visualization
D3.js
sigmajs

Architecture physique



Depuis le début de la collecte : 4To de données brutes, 185M de tweets

Plan

Conclusion

Proposition et mise en pratique de la Lambda+ Architecture qui :

- **évite la complexité** de la Lambda tout en conservant sa capacité de **résistance aux pannes** (avec un entrepôt)
- s'adapte à la **variété** des Big Data grâce à la **dualité analyses exploratoires/indicateurs**

L'architecture est utilisée depuis septembre 2019 pour un projet de recherche et elle intègre en plus la dimension analyse avec :

- des traitements stream processing spécifiques aux données des réseaux sociaux
- un polystore pour faciliter la mise en oeuvre des analyses
- un modèle intermédiaire utilisant les tenseurs pour faciliter les transformations de modèles, assurer le typage strict des compositions d'opérations (type safe)
- des notebooks développés avec Jupyter pour les experts métier