MASTER MIAGE 2ème année
Université Paris Nanterre

Mémoire de fin d'études

Méthodes d'analyse des processus métier pour le choix d'une architecture Big Data adaptée

Auteur :
Ludwig SIMON

Tuteur :
Mcf. Emmanuel HYON

Février 2019 — Juin 2019

Remerciements

Remerciements

Résumé

Résumé

Motivations

Le Big Data est un domaine très vaste, il y a une multitude d'outils pour effectuer les "mêmes" tâches. Il est donc très difficile de faire le bon choix lorsqu'on se lance dans ce domaine.

Objectifs

Dans ce mémoire, nous allons dans un premier temps rappeler brièvement ce qu'est le Big Data et quand il est vraiment utile de l'utiliser. Dans un second temps nous allons devoir établir des critères permettant d'évaluer quelle catégorie d'outil convient le mieux et par la suite quel outil conviendra le mieux. Pour le choix de l'outil on utilisera en plus de critères de cas d'utilisation des benchmarks afin de récupérer des informations sur la consommation de ressources et de temps pour chaque outil. Ensuite, à l'aide de ces critères définit précédemment, nous allons définir des méthodes d'analyse afin de sélectionner les outils adéquat. Et pour finir nous allons appliquer nos méthodes sur un cas concret afin de vérifier l'efficacité de notre solution.

Sommaire

Introduction

Le Big Data

Quand peut-on utiliser le Big Data ?

Chapitre 1

Les différents paradigmes du Big Data

Dans un premier temps, il est nécessaire de présenter les paradigmes du Big Data. Cela permettra d'avoir une meilleure connaissance de ce domaine et fourniras une vue d'ensemble. Cette présentation a aussi pour but de dégager des solutions correspondantes aux paradigmes actuels du Big Data et fournir des critères de choix par rapport à ces différentes solutions.

1.1 Architecture Réactive

Le nombre de données augmentant très rapidement et les clients demandant un service le plus rapide possible et disponible à tout moment, il est de ce fait important d'avoir une infrastructure facilement adaptable à ce flux. C'est le but de l'architecture réactive [1]. Elle repose sur quatre principes :

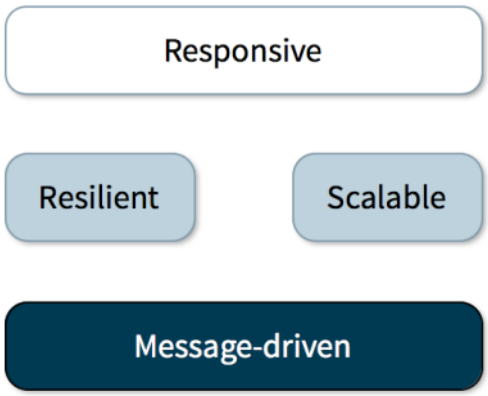

Figure 1.1 – Schéma de l'architecture réactive

- Responsive : C'est l'objectif à atteindre.
- Scalable et Resilient : L'objectif ne peut pas être atteint sans remplir ces deux conditions.
- Message Driven : C'est la base qui permettra d'accueillir tous les composants afin d'obtenir une architecture réactive

Responsive : Un système responsive doit être en mesure de réagir rapidement à toutes les requêtes peut importe les circonstances, dans le but de toujours fournir une expérience utilisateur positive. La clé afin de proposer ce service, est d'avoir un système élastique et résilient, les architecture "Message Driven" fournissent les bases pour un

système réactif. L'architecture Message Driven est aussi important pour avoir un système responsive, parce que le monde fonctionne de manière asynchrone tout comme elle. Voici un exemple : Vous voulez vous faire du café, mais vous vous rendez compte que vous n'avez plus de crème et de sucre.

Première approche :

- Mettre du café dans votre machine.
- Aller faire vos courses pendant que la machine fait couler le café.
- Acheter de la crème et du sucre.
- Rentrer chez vous.
- Boire votre tasse de café.
- Profiter de la vie !

Seconde approche :

- Aller faire vos courses.
- Acheter de la crème et du sucre.
- Rentrer chez vous.
- Mettre du café dans votre machine.
- Regarder impatiemment la cafetière se remplir.
- Expérimenter le manque de caféine.
- Et enfin, boire votre tasse de café.

Grâce à la première approche, on peut clairement voir la différence de gestion de l'espace et du temps. C'est grâce à cela que cette architecture est un atout primordiale pour assurer un système responsive.

Resilient :

La plupart des applications sont conçus par rapport à un fonctionnement idéal. C'est à dire qu'elle ne prévoient pas de gérer correctement les erreurs qui peuvent intervenir plus souvent que ce qu'on pourrait croire. Cela à pour effet de na pas garantir une disponibilité continu et peut provoquer la perte de crédibilité d'un service. La résilience est là pour pallier à ce problème, elle à pour but de récupérer les erreurs émises et de les traiter correctement afin de ne pas provoquer d'interruption de service et de ne pas impacter l'expérience utilisateur.

Scalable :

La résilience et l'élasticité vont de pair dans la construction d'une architecture réactive. L'élasticité permet d'adapter facilement le système à la demande et d'assurer sa réactivité peu importe la charge qu'il subit. Il existe deux moyens différents de rendre élastique une application :

- La mise à l'échelle verticale
- La mise à l'échelle horizontale

1.2 Architecture Répartie

1.3 Traitement des données

Après avoir récupérer des données, nous devons passer à l'étape du traitements des données. Celui à plusieurs rôles, en effet il peut servir à formater les données, leurs apportés de la cohérence en les combinant à des données déjà présentent. Et pour finir les rediriger vers le stockage souhaité. Le traitement des données peut se faire de deux manière différentes. La première solution est le traitement par Batch, et la seconde est le traitement en temps réel [2]. Chacune possède ses avantages et inconvénients, nous allons voir ça plus en détails.

1.3.1 Batch

Le traitement par Batch, consiste à traiter un important volume de données à un instant T. Le traitement par batch est surtout utilisé dans les cas ou nous avons des données stockés de manière journalière, et que nous avons besoin de tout traités en fin de journée. Il n'est pas rare de voir des tâches de traitements par Batch s'exécuter dans la nuit, étant donnée que l'on traite une masse de donnée importante, on sollicite la machine pendant une longue période. Réaliser ce traitement durant des périodes creuses permet de largement diminuer l'impact sur l'utilisation de la plateforme.

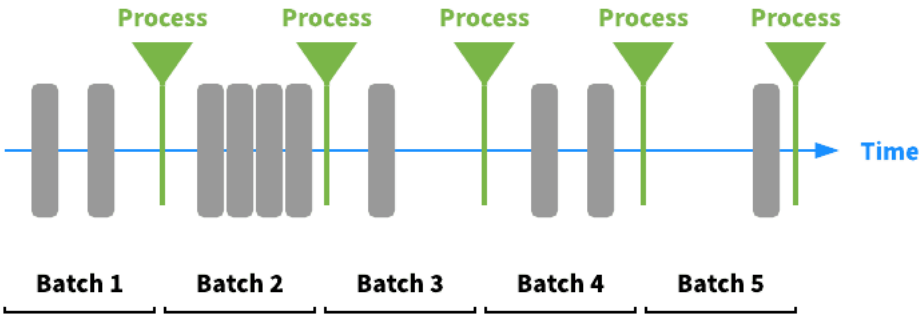

Figure 1.2 – Schéma du traitement par batch

1.3.2 Temps Réel

Un traitement de données est considéré comme étant en temps réel si il s'effectue en une seconde ou moins après la réception de la donnée. Il peut être de deux types, soit des micro batchs soit en streaming.

1.3.2.1 Micro-Batch

Le traitement par micro batch est basé sur le même principe que le traitement par batch, à l'exception qu'il s'exécute beaucoup plus régulièrement (Toutes les secondes ou moins) et que le nombre de données à traiter est donc significativement plus faible. Le micro batch est surtout utilisé dans les cas où notre système ne peut pas directement réagir lorsqu'une donnée arrive, on va donc récupérer les données très régulièrement afin de garantir un traitement en temps réel ou du moins dans le délai le plus bref possible.

1.3.2.2 Streaming

Le traitement en streaming s'appuie sur l'architecture réactive. En effet, contrairement aux traitements par batch et micro batch, ici on ne vas pas récupérer des données de temps en temps. Dès qu'une donnée arrive on va la récupérer et la traiter immédiatement. De par son fonctionnement le traitement en streaming ne nécessite pas de stockage en amont contrairement aux autres type de traitement.

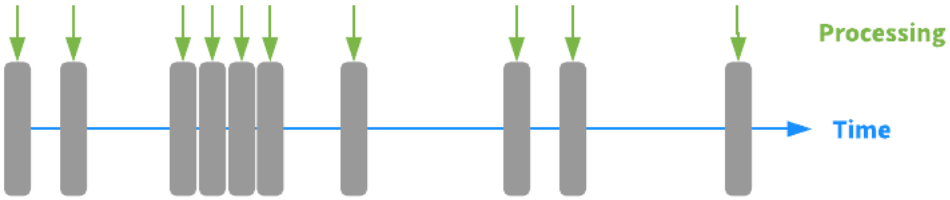

Figure 1.3 – Schéma du traitement en streaming

De manière générale, on peut pas forcément se permettre de n'utiliser que l'un de ces deux type de traitement de données, nous avons régulièrement besoin de les utiliser en même temps. Des architectures ont justement été pensées pour cela [3] [4].

1.3.3 Architecture Lambda

L'architecture Lambda a été crée dans le but de pouvoir à la fois traiter des données en batch et en streaming. Plus précisément, l'architecture lambda est composé de deux couches afin de gérer à la fois les batch et le streaming. Une couche est dédié pour traiter les données par batch, pour ensuite les rendre disponible (Ce qui est appelé une "view"). L'autre couche, s'occupe du streaming. Et à chaque donnée traitée, le résultat est mis à disposition. Grâce à ce fonctionnement lorsque l'on souhaite faire une requête, elle va pouvoir prendre en compte nos données récupérées par batch et par streaming.

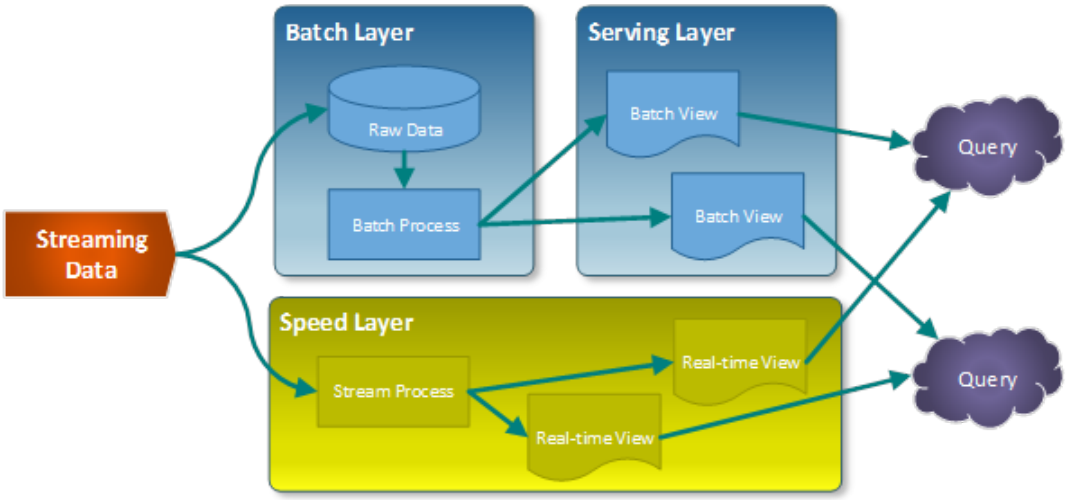

Figure 1.4 – Schéma de l'architecture Lambda

1.3.4 Architecture Kappa

L'architecture Kappa, se veut plus "simple". Le but étant de traiter toutes les données en streaming. Cela a pour conséquence de n'avoir qu'une seule couche contrairement aux deux nécessaire pour l'architecture Lambda. Mais cela ne permet pas de remplacer l'architecture Lambda, en effet il faut que les données que l'on récupère en Batch et la manière dont elles sont stockées, permettent de les traiter en streaming par la suite.

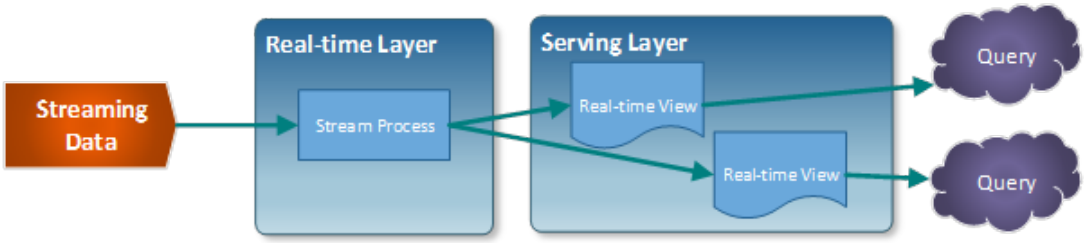

Figure 1.5 – Schéma de l'architecture Kappa

1.4 Stockage des données

1.4.1 BASE

1.4.2 ACID

1.4.3 Théorème de CAP

1.4.4 Les différents type de stockage

Chapitre 2

Analyse des solutions logicielles existantes

Une fois nos critères définit, il faut décomposer le Big Data qui est un domaine très vaste, en plusieurs catégories [5]. Une fois cette décomposition effectué, on va pouvoir s'intéresser aux outils existant permettant de réaliser chacune des tâches. On va noter leurs points faibles et leurs point forts ainsi que la manière dont ils sont censés être utilisés.

2.1 Système de Messaging

2.1.0.1 Kafka

2.1.0.2 ActiveMQ

2.1.0.3 RabbitMQ

2.2 Ingestion/Extraction de données

La première catégorie, qui est aussi la première étape d'une architecture Big Data, c'est le récupération de données. Plus précisément comment nous allons récupérer des données, soit via des requêtes sur des sources externes, soit des sources externes nous envoie directement des données.

2.2.1 VertX

2.2.2 Akka

2.3 Traitement des données

Une fois les données reçu, des traitements sont nécessaire afin de pouvoir stocker les données au format souhaité ou bien pour faire un tri des données utiles.

2.3.1 Spark Streaming

2.3.2 Spark

2.3.3 MapReduce

2.4 ETL

2.4.1 Apache Nifi

2.4.2 Talend

2.5 Stockage des données

Une partie très importante du Big Data est le stockage des nombreuses données que l'ont reçoit. Il existe énormément de manières différentes de stocker des données selon la manière dont nous voulons les utiliser par la suite.

2.5.1 Time Series

2.5.1.1 OpenTSDB

2.5.1.2 InfluxDB

2.5.2 Graph

2.5.2.1 Neo4j

2.5.2.2 JanusGraph

2.5.3 Données Structurées

2.5.3.1 Hive

2.5.3.2 Phoenix Framework

2.5.4 Clé-Valeur

2.5.4.1 HBase

2.5.4.2 Redis

2.5.5 Index

2.5.5.1 ElasticSearch

2.5.5.2 Apache Solr

2.6 Requêtage

2.6.1 Kibana

2.6.2 Banana

2.6.3 Grafana

2.6.4 Tableau

2.6.5 OLAP

2.6.5.1 Outil

2.6.6 OLTP

2.6.6.1 Outil

2.7 Solution complète

Chapitre 3

Critères d'analyse

3.1 Type de traitement des données

Streaming - Micro Batch - Batch

3.2 Format des données

3.3 Perte de données admissible

3.4 Volumétrie

3.5 Performance

Chapitre 4

Implémentation : Exemple avec un processus métier

4.1 Définition du processus métier

4.2 Application des méthodes sur le processus métier

4.3 Évaluation du résultat

Bibliographie

[1] Kevin Webber. *What is Reactive Programming ?*

[2] Laura Shiff. *Real Time vs Batch Processing vs Stream Processing : What's The Difference ?* url : https://www.bmc.com/blogs/batch-processing-stream-processing-real-time/.

[3] Michael Verrilli. *From Lambda to Kappa : A Guide on Real-time Big Data Architectures*. url : https://www.talend.com/blog/2017/08/28/lambda-kappa-real-time-big-data-architectures/.

[4] Mehdi Refes. *Architecture Lambda, Kappa ou Datalake : comment les exploiter ?*

[5] Christophe Parageaud. *Big Data, panorama des solutions*. url : https://blog.ippon.fr/2016/03/31/big-data-panorama-des-solutions-2016/.

Table des matières

Table des figures