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 architectures Big Data

Une architecture Big Data est un regroupement d'outils permettant de gérer des données de leurs ingestion à leurs mise en valeur via des analyses. Il faut savoir qu'il existe plusieurs architectures dans le domaine du Big Data, et qu'elles répondent à des besoins différents. Nous allons nous intéresser aux trois architectures les plus importantes, étant donné que les autres sont des dérivées des trois architectures principales. Avant de voir en détail ces trois architectures, nous allons voir de manière plus générale les différents composants qui peuvent se retrouver dans des architectures Big Data [1][2]. Les différents composants pouvant se retrouver dans une architecture Big Data sont illustrés sur la figure 1.1.

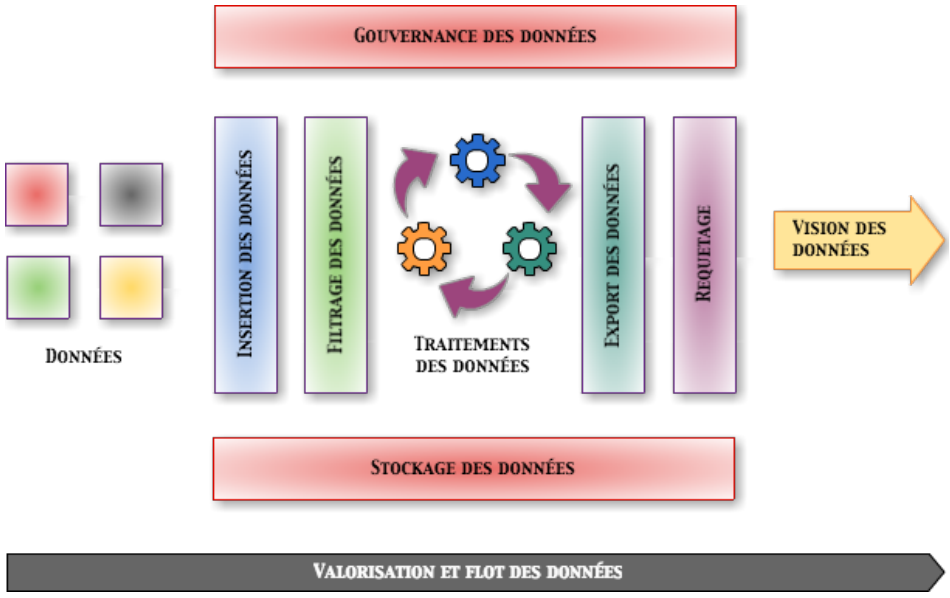

Figure 1.1 – Composants d'une architecture Big Data

Nous allons maintenant détailler le rôle de chaque composant présenté dans le digramme ci dessus.

- **Source de données :** N'importe quelle solution Big Data a besoin d'une source de donnée en entrée. Voici quelques exemple de données qui peuvent être utilisée dans une architecture Big Data
  - Des données issues de bases de données relationnelles.
  - Des fichiers statiques produits par des applications, des fichiers de logs par exemple.
  - Des sources de données en temps réel, par exemple des données de capteurs récupérés via des appareils IoTs.

- **Ingestion des données :** La solution doit être capable d'aller chercher directement les données dans les différentes sources. Il est possible que les données soient directement envoyées dans la chaîne de traitement des données, mais c'est rarement le cas. Afin d'aller récupérer les données à la source, il y a plusieurs solutions. On peut écrire un programme permettant d'extraire les données. Cette solution est la plus efficace si le programme répond bien aux contraintes du Big Data, c'est à dire, il faut qu'il soit le plus réactif possible et le plus facilement adaptable aux flux de données entrant (Architecture réactive A.1). La seconde solution la solution la plus simple, est l'utilisation d'un ETL (Extract Transform Load). C'est un outil graphique permettant de configurer l'extraction des données et leurs insertions dans la chaîne de traitements ou dans la solution de stockage. Néanmoins, cette solution est plus gourmande que l'écriture d'un programme car elle doit être capable de gérer énormément de sources et d'opérations différentes.
- **Traitement par lot (??) :** Les jeux de données étant trop lourd lors d'un traitement par lots, il est nécessaire de pouvoir exécuter une tâche de traitement de longue durée afin de filtrer, agréger et préparer les données en vue de leur analyse. En général, ce genre de traitement implique une lecture de fichier source et une écriture dans des nouveaux fichiers. Le traitement des données peut s'effectuer par des programmes java, scala ou Python ou encore via des outils spécialisé comme MapReduce ou Spark.
- **Ingestion des données en temps réel :** Si la solution doit interagir avec des sources de données en temps réel, l'architecture doit impérativement implémenter un moyen de récupérer ces données et de les stocker temporairement dans une file d'attente afin de pouvoir les temporiser. Cela permet d'éviter la perte de données entrante, et permet d'envoyer les données à la solution de traitement quand elle est disponible et de ne pas la surcharger. Généralement on utilise des messages brokers pour ce genre de tâches.
- **Traitement de flux (??) :** Une fois les données récupérées, elles doivent être filtrées, agrégées puis préparer pour l'analyse. le traitement est similaire au traitement par lot, seul les outils sont différents, car ils doivent être capable de gérer le traitements en temps réel.
- **Stockage des données :** Une solution de stockage de données est indispensable dans le cas de traitement des données par lot (**??**), et peut s'avérer utile lors de traitements des données en temps réel (**??**) si l'on souhaite conserver les données reçu en plus de les traiter. Le deuxième cas ou un stockage de donnée puet être utile pour le traitement en temps réel, est si l'on a besoin d'agréger des données statiques avec les données en temps réel. Ces solutions de stockage doivent être capable de gérer divers formats de données, et surtout ils doivent être distribués (A.2).
- **Requêtage :** Afin de pouvoir fournir les données stockées aux outils d'analyse et de visualisation, l'utilisation d'un outil de requetâge peut être requis. Dans certains cas votre base de données et votre outil de visualisation communiquerons directement entre eux, mais dans d'autre cas vous aurez besoin d'utiliser un outil de requetâge afin de fournir les données au bon format et de pouvoir faire une sélection des données à envoyer à l'outil.

- **Analyse et visualisation des données :** La dernière étape dans une architecture Big Data est la visualisation/analyse des données. la plupart des solutions Big Data ont pour but de faire de la valorisation de données et de fournir au minimum une visualisation des données et au mieux d'effectuer des analyses dessus. Cela peut se faire par l'écriture de rapport ou bien par application d'algorithme afin de détecter et de montrer différentes corrélation entre des données par exemple. Le but principal d'avoir une visualisation intelligente et facilement compréhensible de données étant à la base illisible par l'homme.
- **Orchestration :** La majorité des solutions Big Data consistent a effectuer des traitements de données répétés, ayant pour but de transformer les données sources puis de les stocker ou bien les envoyer directement à un outil d'analyse ou de visualisation des données. Il est donc important d'avoir un outil permettant de paramétrer les différentes actions que l'on souhaite effectuer sur nos données

1.1 Architecture Data Lake

1.2 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.

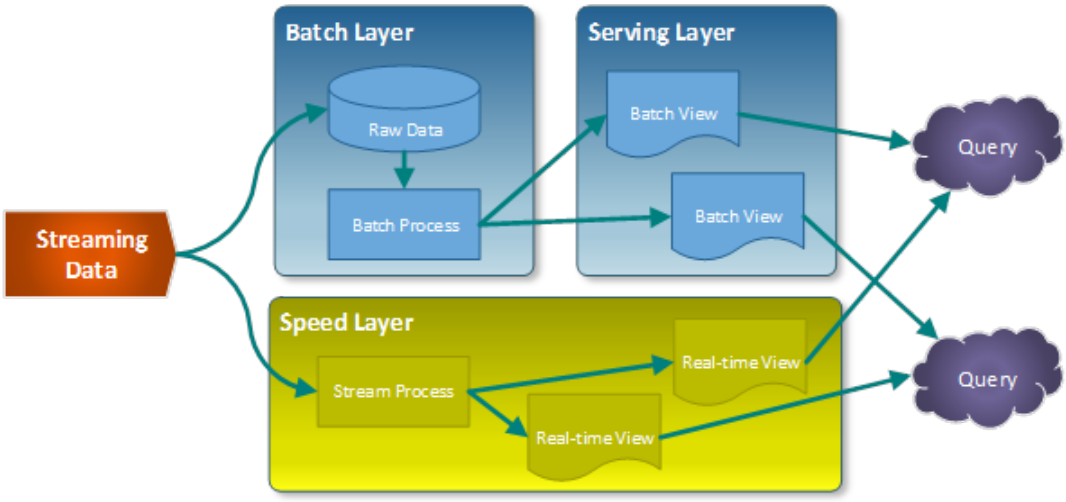

Figure 1.2 – Schéma de l'architecture Lambda

1.3 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.

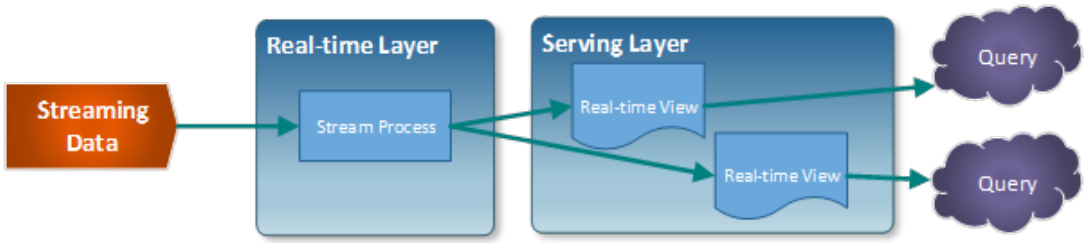

Figure 1.3 – Schéma de l'architecture Kappa

Chapitre 2

Analyse des solutions logicielles existantes

Maintenant que nous avons vu les différentes architecture Big Data existantes et que nous les avons décomposés, on va s'intéresser plus en détail à chaque composant de ces architecture. Pour chaque composant, nous allons présenter diverses solutions existantes permettant de remplir le rôle de ce dernier. Pour chacune de ses solutions, nous allons voir leurs avantages et inconvénients et leurs manière de fonctionner dans le but de pouvoir en dégager des critères de sélections.

2.1 Message Broker

Un Message broker [3], Agent de message en français, est un moyen de communication utilisant des messages entre deux applications (Ex : Communication entre un serveur et un client). Un message broker permet une communication asynchrone entre applications. L'utilisation de cette solution permet de pouvoir facilement filtrer les messages que l'on reçoit et de stocker temporairement les messages reçus afin d'éviter les pertes de données. Ce dernier cas, s'avère très utile dans le cas où l'application chargée de la réception des données n'est pas en fonctionnement pendant un certains temps. Il existe deux types de communications avec un message broker :

Publisher / Subscriber

Dans ce mécanisme, l'entité envoyant les données est nommé "Publisher" et l'entité les récupérant est nommé "Subscriber". Le publisher va envoyer des données dans des topics [1] afin que les Subscribers de ce topic puissent les récupérer. Un publisher peut envoyer des données dans un ou plusieurs topics, et les subscribers peuvent être abonné à un ou plusieurs topics (Voir figure 2.1).

Point-to-point communication

La communication point à point est la forme la plus simple de Producteur/Consommateur. Le producteur envoie ses données dans une queue et le consommateur va lire les

1. Un topic est une catégorie dans laquelle les messages produit sont stockés.

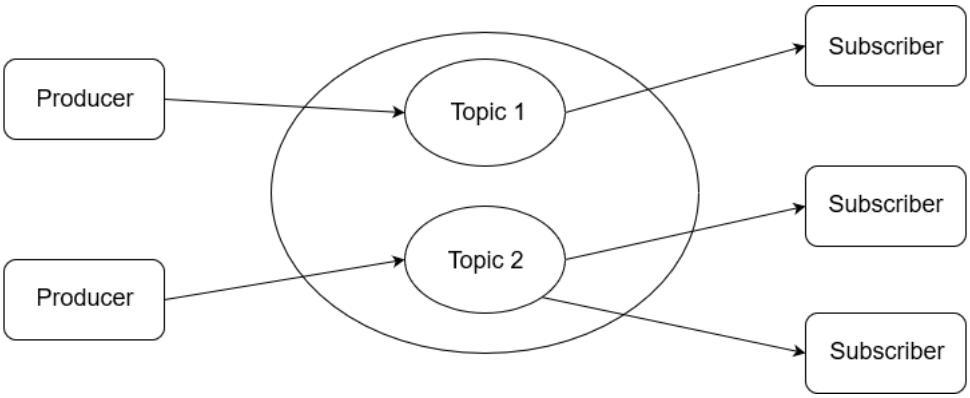

Figure 2.1 – Schéma du principe Producer/Subscriber

messages dans la queue. Tout comme le modèle précédent, il peut y avoir plusieurs producteur et consommateurs sur la même queue, mais si plusieurs consommateurs sont présents, ils ne recevront des portions différentes des messages afin de favoriser le traitement concurrentiel.

2.1.1 Kafka

2.1.2 ActiveMQ

2.1.3 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 Apache Nifi

2.2.2 Talend

2.2.3 Solution maison

2.3 Traitement des données

Après avoir récupérer des données, nous devons passer à l'étape du traitements des données. Celui ci 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 [4]. Chacune possède ses avantages et inconvénients, nous allons voir ça plus en détails.

2.3.1 Batch

Le traitement par Batch (Traitement par lot), 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 (2.2).

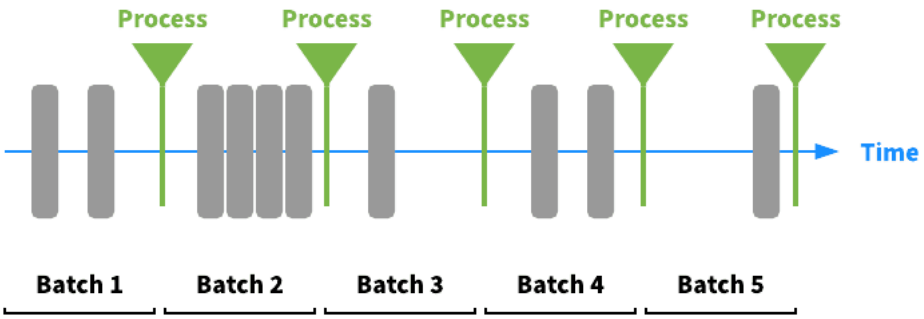

Figure 2.2 – Schéma du traitement par batch

Nous allons nous pencher sur les solutions existantes, implémentant un traitement par batch.

2.3.1.1 Spark

2.3.1.2 Hadoop MapReduce

2.3.2 Streaming

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.

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.

Streaming

Le traitement en streaming (Traitement de flux), s'appuie sur l'architecture réactive (A.1). 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 (2.3).

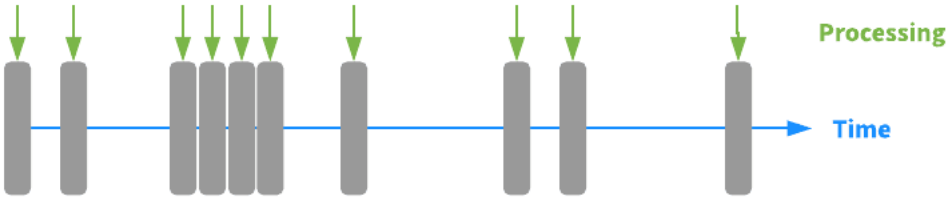

Figure 2.3 – Schéma du traitement en streaming

Nous allons maintenant nous intéresser aux différentes solutions proposant ce type de traitement.

2.3.2.1 Spark Streaming

2.3.2.2 Apache Storm

2.4 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.

2.4.1 Time Series

2.4.1.1 OpenTSDB

2.4.1.2 InfluxDB

2.4.2 Graph

2.4.2.1 Neo4j

2.4.2.2 JanusGraph

2.4.3 Clés/Valeurs

2.4.3.1 Redis

2.4.3.2 RoxDB

2.4.4 Documents

2.4.4.1 CouchDB

2.4.4.2 CouchBase

2.4.4.3 MongoDB

2.4.5 Wide Column

2.4.5.1 HBase

2.4.5.2 Cassandra

2.4.6 Système de fichiers

2.4.6.1 Hadoop HDFS

2.5 Orchestration

2.6 Requetâge

2.7 Visualisation et Analyse des données

2.7.1 Kibana

2.7.2 Banana

2.7.3 Grafana

2.7.4 Tableau

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

Annexe A

Les différents concepts du Big Data

A.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 [5]. Elle repose sur quatre principes :

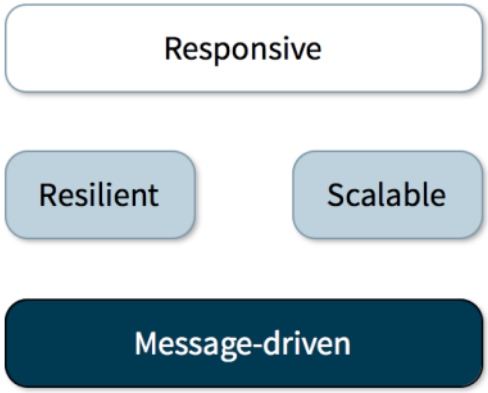

Figure A.1 – Schéma de l'architecture réactive

- 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. L'élasticité est un point très important, lorsque votre plateforme reçoit énormément de connexion cela veut dire que votre service connaît un succès important, c'est donc le pire moment pour avoir un service non disponible. Cela entrainerait une perte de crédibilité et de clients.

Il existe deux moyens différents de rendre élastique une application :

— La mise à l'échelle verticale
— La mise à l'échelle horizontale (Architecture Répartie)

La mise à l'échelle verticale consiste à maximiser l'utilisation des ressources de notre machine. Cela peut se faire par l'utilisation de programmes fonctionnant en asynchrone sur plusieurs threads. Tandis que la mise à l'échelle horizontale consiste à augmenter le nombre de machine afin d'avoir plus de ressources à notre disposition. Pour le moment on ne vas pas trop s'attarder sur la division horizontale car une partie lui est dédiée.

La mise à l'échelle verticale est certes la moins couteuse, mais ce n'est pas la plus simple à mettre en place. En effet le multithreading permet de tirer la maximum des performances de la machine, mais cela ajoute une certaine complexité au programme. Par exemple le fait de travailler avec des variables mutables entre différents threads n'est pas une tâche aisée, cela provoque donc des limitations à la mise à l'échelle

verticale. De plus, une fois avoir exploiter au maximum une machine en ayant mis en place la mise à l'échelle verticale, si on a encore besoin de ressources supplémentaire la mise à l'échelle horizontale devient indispensable.

Message-Driven :

Le Message-Driven Architecture

A.2 Architecture Répartie

Bibliographie

[1] Zoiner Tejada et olprod. *Architectures Big Data*. url : https://docs.microsoft.com/fr-fr/azure/architecture/data-guide/big-data.

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

[3] https ://www.tibco.com. *What is a Message Broker ?* url : https://www.tibco.com/reference-center/what-is-a-message-broker.

[4] 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/.

[5] Kevin Webber. *What is Reactive Programming ?* url : https://blog.redelastic.com/what-is-reactive-programming-bc9fa7f4a7fc.

Table des matières

Table des figures