Projet :
Systèmes distribués pour le traitement de données

COURET Nathanael, HAGENBOURGER Maxime, CHENAL Vincent,
RAKOTOARISOA Jérémy, SERGENT Julien, TABOULOT Clément,
MATTON Valentin, VALLADIER Victor et LAFOREST Yann

Lundi 21 Novembre 2016

Table des matières

1 RabbitMQ

1.1 Présentation générale

RabbitMQ est un middleware de Pivotal facilitant l'échange de messages entre applications présentes sur un système informatique : message broker software. RabbitMQ implémente le protocole AMQP (conçu par un consortium international) qui standardise les échanges de messages. Écrit en Erlang, RabbitMQ est open-source et un support payant est disponible. Des librairies sont disponibles pour l'écrasante majorité des langages utilisés, est compatible avec les principaux systèmes d'exploitation et bénéficie d'une large communauté d'utilisateurs. Il est utilisé par de nombreuses entreprises pour répondre à leurs besoins en échange de messages, telles que Instagram, The New York Times, Nokia, SoundCloud...

1.2 Fonctionnalitées

RabbitMQ propose les fonctionnalités suivantes :
- Le transport de messages ;
- La communication asynchrone : L'émetteur d'un message et le récepteur du message n'ont pas besoin d'être activés en même temps. La file d'attente reçoit le message de l'application émettrice et le stocke jusqu'à ce que l'application réceptrice vienne lire le message ;
- Le routage : Les messages peuvent être routés entre machines ;
- Le clustering : Plusieurs serveurs RabbitMQ peuvent former un cluster ;
- La persistance des messages : Les messages d'une file peuvent être sauvegardé sur un support physique ; fiabilité : Chaque envoi ou réception par une application fait l'objet d'un accusé de réception. Couplé avec la persistance, ce mécanisme permet de garantir qu'aucun message ne sera perdu dans son transfert entre les applications ;
- La diversité des protocoles supportés : AMQP, STOMP, MQTT, AMQP, HTTP ;
- La diversité des clients : Java, Python, .NET, Ruby, PHP, etc... (plus de 20).

1.3 Principes de fonctionnement

RabbitMQ propose les différents cas d'utilisation :
- Point à point : Une seule application productrice et une seule consommatrice. Le message est retiré de la file d'attente lorsque le consommateur l'a lu ;
- Publish-subscribe : Les applications consommatrices des messages s'abonnent à un channel correspondant à une catégorie de messages qu'elles veulent recevoir. Tous les messages du channel restent disponibles tant que tous les abonnés ne les ont pas lu ;
- Work queue : Une application remplit une file de job à exécuter. Des serveurs viennent récupérer ces tâches pour les réaliser.
- Routing : Les messages sont distribués dans les queues en fonction de leur type.

2 HDFS - Hadoop Distributed File System

2.1 Présentation

HDFS est un système de fichiers distribué, redondant et fiable. Il est tolérant aux fautes, par conception. HDFS fait partie du framework Hadoop (HDFS, YARN, MR). Dans le cadre de la stack Mazerunner, YARN ne sera pas utilisé et Spark sera directement sur HDFS (Standalone Mode). Version utilisée : 2.7.3

2.2 Eléments d'architecture

Un cluster HDFS est composé d'un **NameNode** et de **DataNodes**. Les fichiers sont divisés en blocs, qui sont répliqués sur les DataNodes.

Le NameNode :
- centralisé, gére les métadonnées et le namespace.
- reçoit les requêtes client.
- est un *single point of failure*. Il peut être dédoublé (cf **Secondary NameNode** et **HA**)

Les DataNodes :
- stockent les blocs.
- n'ont aucune connaissance des fichiers d'origine.
- envoient régulièrement des **HeartBeats** au NameNode.
- envoient la liste des blocs stockés - **BlockReport**

2.3 failures

3 types de pannes,

1. NameNode failure :
2. DataNode failure :
   DataNode déclaré mort après 10min (par défaut) sans HeartBeat
3. Network partition :

Pour tester les pannes,

1. Trouver le pid du daemon concerné :

```
$ jps
```

2. Terminer brutalement :

```
$ sudo kill -9 <pid>
```

(à wrap dans un script à lancer dps la machine "manager")

2.4 Intégration

Installation du client :
- Installation classique d'Hadoop
- Copie des fichiers de configuration du cluster /etc/hadoop/
- accés via CLI :

```
$ hdfs dfs <commande>
```

- accés via API python avec l'URI du cluster (`hdfs://...`)

3 Neo4j

3.1 Présentation générale

Neo4j est un système de gestion de base de données orienté graphes. Le code source, écrit en Java, est développé par Neo Technology.

3.2 Caractéristiques

La base de données de graphes offrent de meilleures performances dans le traitement de requêtes utilisant des relations entre objets. Au lieu d'utiliser des jointures comme dans le bases de données relationnelles, Neo4j utilise des outils de parcours de graphes. Ces outils permettent également de faciliter des cas d'usages exploitant au maximum les relations.

Les données ne sont pas stockées de manière structurée dans Neo4j. Cela porte l'avantage d'adapter la base à la modification continue des données. En outre, le temps de développement de la base et le coût de sa maintenance s'en trouvent réduits.

Neo4j utilise le langage Cypher pour la description des requêtes.

3.3 Cas d'utilisation privilégiés

— Gestion de réseau
— Réseaux sociaux
— Recommandation
— Géo-spatial

3.4 Installation

Neo4j requiert Java 7.

```
wget -O - http://debian.neo4j.org/neotechnology.gpg.key | apt-key add -
echo 'deb http://debian.neo4j.org/repo stable/' > /etc/apt/sources.list.d/neo4j.list
apt-get update
apt-get install neo4j
```

Pour démarrer le serveur :

```
/etc/init.d/neo4j-service start
Apr s extraction du dossier,
```

4 Spark

4.1 Présentation générale

Spark est un framework open-source qui permet de traiter de larges volumes de données. C'est un framework qui est utilisé pour de grosse volumétries de données (supérieur en taille à 1 To de données). Pour des faibles volumétries de données, il existe différents outils plus simple et plus léger à mettre en place.

4.2 Usages

Spark permet de répondre à différents cas d'utilisations :

— Faire de l'analyse de logs
— Traiter des fichiers de texte
— Faire de la recommandations pour des produits, des articles ...
— Faire de la détection de fraude et de la sécurité
— Recherche distribuée (Google, Facebook ...)

4.3 Aspect fondamentale

Spark tourne au seins d'un cluster :

— Spark peut fonctionner de différentes façon au niveau de la gestion du cluster. Il pourra être en standalone ainsi se débrouiller de maniéres autonome ou alors être managé par les gestionnaires de cluster YARN ou Mesos. Dans un cluster qui contiendra on y trouvera un Master et plusieurs Workers. Dans cette configuration le master joue le role de distribuer les différentes traitement à effectuer aux différents Workers. Par ailleurs, si l'on souhaite être tolérants au pannes le cluster doit disposer d'un master en standby pour prévénir en cas de perte du premier master. Au fur et à mesure du traitement le master principale fait des sauvegardes dans le second master pour qu'il le master en stand-by soit capable de prendre le relais.
— Spark permet de faire du traitement de maniéres distribué en répartissant le travaille sur différents noeuds.
— Il se base sur un concept de Hadoop qui est Map Reduce. C'est un modéle de programation pour traiter de fortes volumétries. Map Reduce posséde plusieurs propriétes (équilibrage de charge, tolérance aux pannes, traitements distribués, traitement paralléle). Cela permet de découper la demande en plusieurs sous-problémes.
— Spark offre les possibilités de faire persiter les données entre plusieurs machines (données stockées en mémoire et/ou sur disque).

4.4 Resilient Distributed Dataset

Les RDD sont au centre du framework spark. Ils peuvent être vue comme des collections oû l'on y stocke des données. Ils possédent plusieurs propriétes :

— Persitante : Il peuvent être soit sur le disque ou/et sur la mémoire. Le faite de stocker les objects RDD en mémoire permet à spark de réaliser des traitements plus rapide que peut proposer Hadoop (Hadoop réalise des accées disque qui coûte chére).
— Resilient : Cela permet à spark d'être tolérant aux pannes. Car dans un cluster les risques qu'un noeud tombe en panne et ne soit plus accessible est bien réaliste (probléme de réseau, de disque ....). De ce faite spark est capable de détecter qu'un noeud est encore en vie ou pas. Ainsi, il va pouvoir être capble de relancer les traitements de l'objet RDD qui ont échoué sur un autre noeud.
— Partitioné : Au travers de spark on a la possibilité de découper les RDD en plusieurs partie. De ce fait, dans une RDD les données vont être partitioné. Soit spark le fait de maniére arbitraire ou on a aussi la possibilité de le faire en spécifiant la taille des partitions.
— Distribué : Du faite que les RDD sont partitionné on peut répartir le traitement sur plusieurs noeud. Ainsi, cela permet à chaque noeud de travailler sur des tailles de données reduites et ainsi d'être capables de répondres le plus rapidement possibles.

4.5 Performances

Pour montrer son efficacité au niveau du traitement des données spark à battu le record précedement détenue par Hadoop sur un trie distribué. Or comme spécifier avant les données sont en mémoires ce qui permet aux traitements d'être fait jusqu'à 100 fois plus vite qu'avec Hadoop.

Trie de 100 TB de données de maniéres distribué :

- Hadoop :
  - Temps de calcule : 72 minutes
  - Configuration : 2100 noeuds (50400 cores)
- Spark :
  - Temps de calcule : 23 minutes
  - Configuration : 206 noeuds (6592 cores)

5 Graphx

5.1 Présentation générale

GraphX est l'API de spark pour tout ce qui est traitement de graphes. GraphX étend l'objet RDD en introduisant le Resilient Distributed Property Graph.

5.2 Cas d'utilisation graphes

— Dans les réseaux sociaux
— Publicité ciblée
— Modélisation astrophysique