Simulation d'algorithmes d'équilibrage de charge dans un environnement distribué

Architecture

Kevin Barreau Guillaume Marques Corentin Salingue

14 février 2015

Sommaire

1 Architecture de Cassandra

Cassandra est une base de données distribuées, écrite en langage Java. Dans sa version 2.1.2, elle est composée de 963 fichiers, répartis dans 62 dossiers, pour un total de 126 502 lignes de codes et 36 287 lignes de commentaires.

Cassandra est un projet riche et complet. Nous ne nous intéresserons qu'aux parties de son architecture sur lesquelles nous allons travailler.

1.1 Staged event-driven architecture (SEDA)

Cassandra est basée sur une architecture de type Staged Event Driven Architecture (SEDA). Cela permet de séparer des tâches dans différents emplacements, appelés *stages*, qui sont connectés par un service de messages. Chaque stage possède une file d'attente pour les messages (un message correspondant à une tâche à traiter), ainsi qu'un ensemble de threads pour traiter les tâches (voir figure 1).

Dans le cas de Cassandra, la gestion des stages se fait dans le package `org.apache.cassandra.concurrent`. Les stages sont -pour la plupart- énumérés dans *Stage*. Ils sont ensuite gérés par le *StageManager*. Nous nous intéresserons principalement aux stages :

- **READ** : lectures locales
- **GOSSIP** : communications sur l'état des noeuds

Ces stages permettent de répondre à différents besoins.

Le stage "READ" permet de répondre aux besoins des protocoles d'affectation. L'idée est de créer un nouveau stage "QUEUE_MANAGER" pour gérer les file d'attentes du stage de lecture. Ainsi, les tâches traitées par le stage "READ" entraîneraient un message vers le stage "QUEUE_MANAGER", envoyant un message pour supprimer un message d'une file d'attente dans les autres noeuds ayant à traiter la même tâche. Les noeuds recevant le message le transmettraient au stage "QUEUE_MANAGER" qui s'occuperait alors de supprimer le message voulu de la file d'attente s'il y est encore présent. La figure 2 montre un exemple avec une requête de lecture arrivant sur le noeud 5, et avec les données à lire sur les noeuds 1 et 2.

Le stage "GOSSIP" permet de répondre aux besoins qui concernent la communication des données locales d'un noeud. Les noeuds de la base de données s'échangent des informations sur leur état toutes les secondes. L'ajout de données locales entraînera la modification du stage "GOSSIP" pour permettre l'envoi de ces nouvelles données.

1.2 Gossip

Cassandra utilise le protocole Gossip pour les communications entres les noeuds. Chaque noeud envoie les informations qu'il possède -sur lui et sur les autres noeuds- à au plus 3 autres noeuds du réseau. Cela permet d'avoir pour chaque noeud une connaissance globale du réseau avec un minimum d'interaction.

Les classes en rapport avec Gossip se situe dans package `org.apache.cassandra.gms`. La classe chargée de traiter les tâches de Gossip est `org.apache.cassandra.gms.Gossiper`.

Gossiper maintient une liste de noeuds "vivants" et "morts" (des noeuds inatteignables). Toutes les secondes, le module démarre un tour. Un tour entier de Gossip est composé de trois messages. Un noeud X envoie un message syn à un noeud Y pour initialiser Gossip. Y, à la réception de ce message syn, renvoie un message ack à X. Pour répondre à ce message ack, X envoie un message ack2 à Y pour compléter le tour (voir la figure 3).

1.3 Modèle de données

Le modèle de données de Cassandra s'appuie sur un schéma dynamique, avec un modèle de données orienté colonne (voir figure 4). On retrouve les classes gérant la modélisation de la base de données au sein du package `org.apache.cassandra.db`.

- **Keyspace** : le conteneur des données de l'application
- **Row** : une ligne dans le Keyspace, composée d'une clé et d'un ensemble de colonnes
- **DecoratedKey** : un token identifiant le positionnement d'une ligne dans la base de données
- **ColumnFamily** : un ensemble de colonnes pour une ligne donnée
- **Column** : un tuple contenant un nom, une valeur et un *timestamp* (la date de la mise à jour la plus récente de cette colonne)

Nous allons nous intéresser dans notre projet à la classe **ColumnFamily**, afin de pouvoir garder une trace de la popularité des objets. Les objets étant définis par un token, porté par la classe **DecoratedKey**, ColumnFamily est l'objet dont nous souhaitons connaître la popularité.

1.4 Métriques

Nous allons devoir sortir des données. Pour ça, Metrics ! Parler de JMX, de la librairie Metrics, et des possibilités pour notre projet grâce à ça (je pense aux histogrammes). Tout est surveillés dans Cassandra, facile d'avoir des infos sur un peu tout du coup.

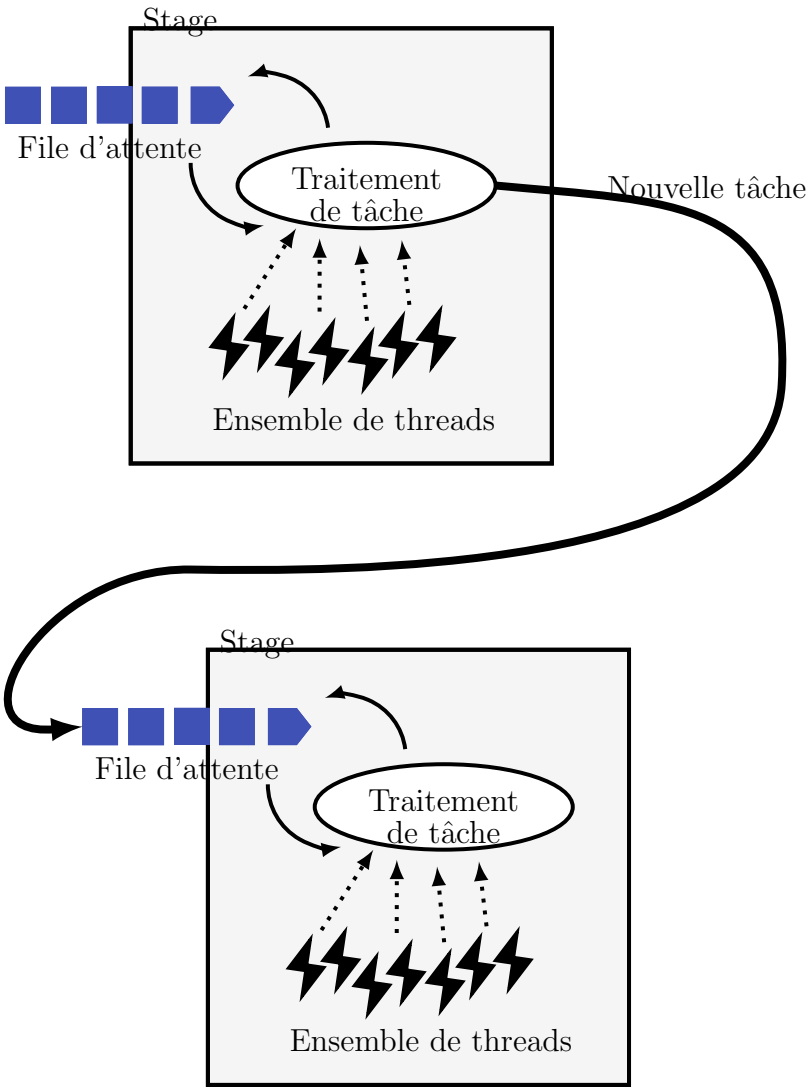

FIGURE 1 – Schéma de deux stages dans une architecture SEDA

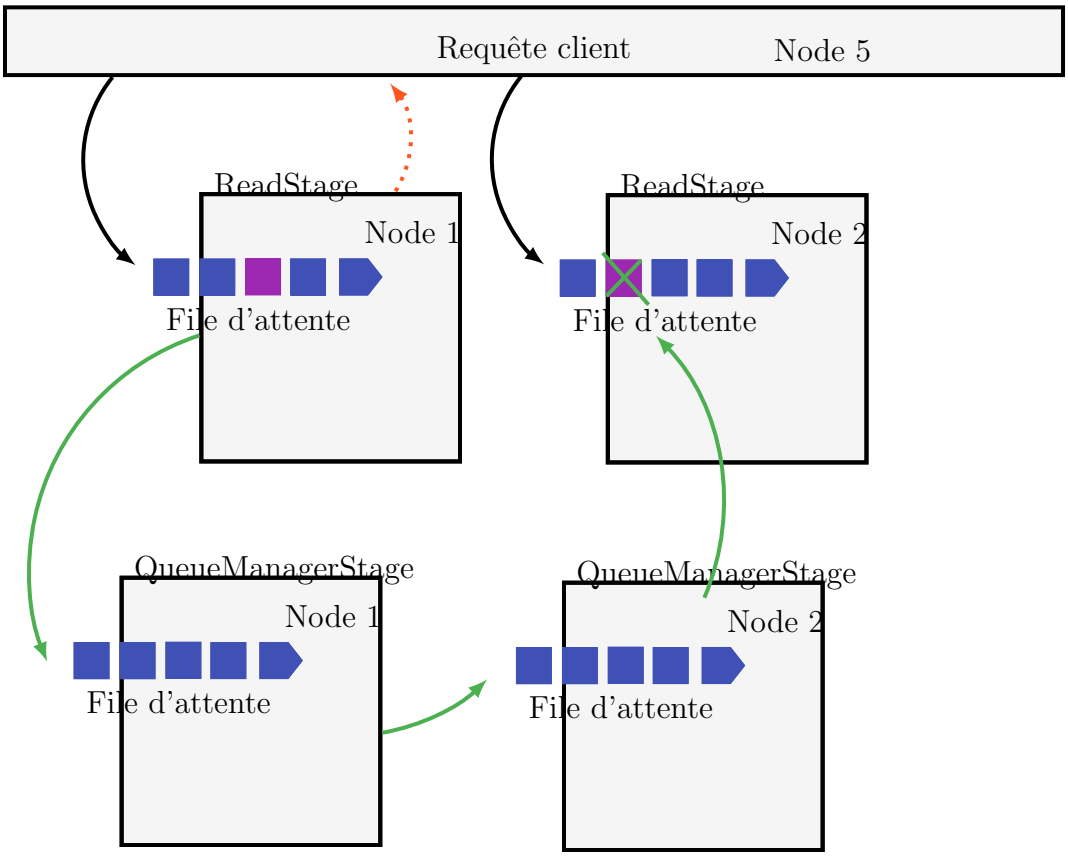

FIGURE 2 – Exemple simplifié d'une requête de lecture au niveau des stages, avec l'ajout d'un stage "QUEUE_MANAGER"

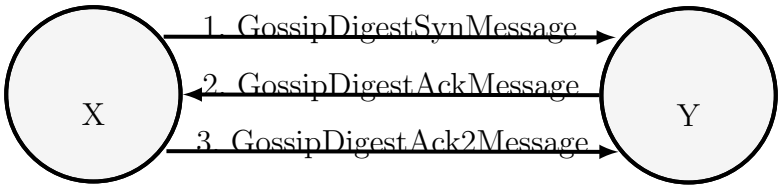

FIGURE 3 – Tour de messages Gossip entre les noeuds X et Y

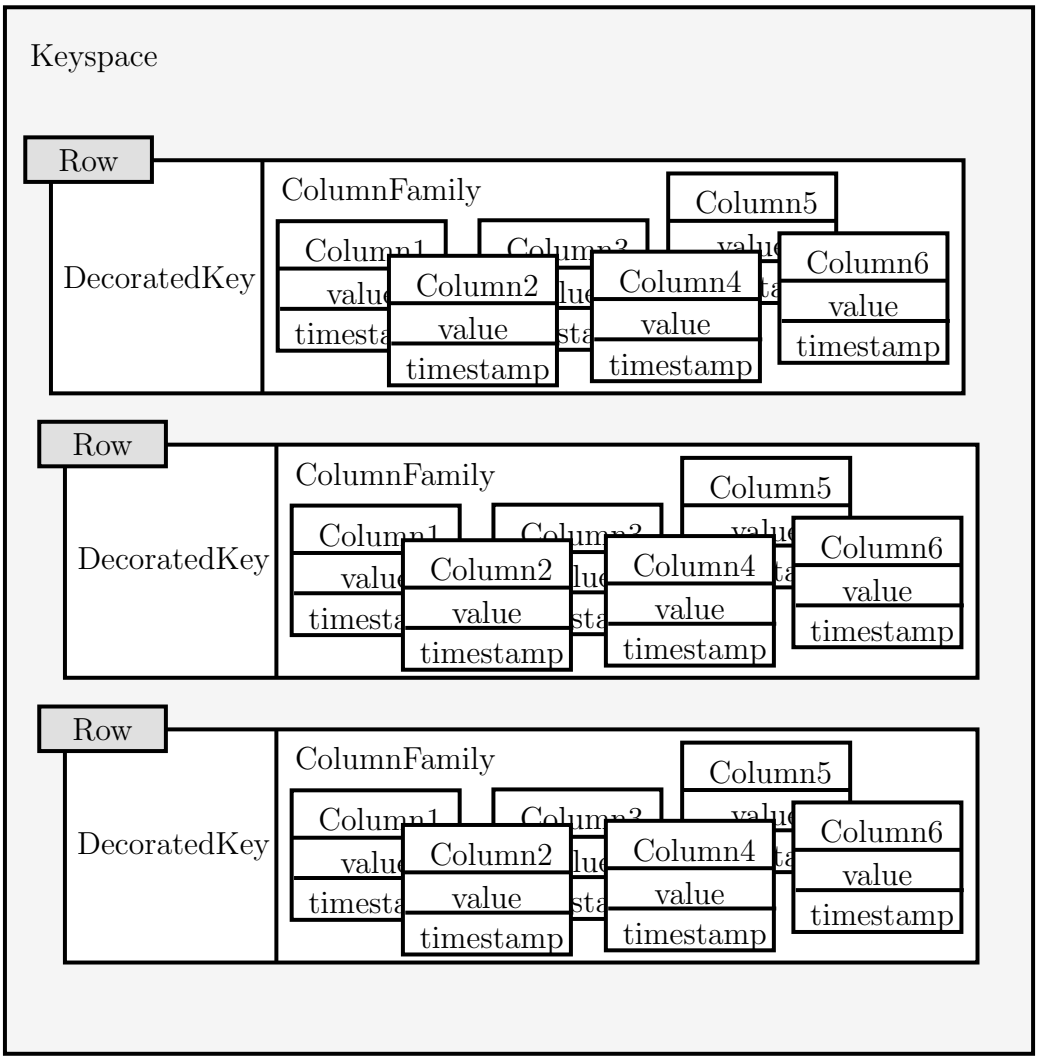

Figure 4 – Modèle de données de Cassandra