Archipelago : Un framework de peristence de graph de données.

https://github.com/GillesBodart/Archipelago

Gilles Bodart

July 15, 2017

Contents

Part I

State of the art

Chapter 1

Introduction

1.1 Historique

Un SGBD[1] est par définition un ensemble de procédés permettant d'organiser et de stocker des informations (potentiellement de gros volumes). Si stocker et retrouver l'information est un des plus grand challenge d'un SGBD, une communauté de développeur, pensent que ces système devraient pouvoir offrir d'autres fonctionnalités.

A partir des années 1980, le modèle relationnelle supplante les autres formes de structures de donnée.

Les évolutions logicielles suivant naturellement les évolutions matérielles, la généralisation des interconnexion des réseaux, l'augmentation de la bande passante, la diminution du cout des machines, la miniaturisation des espaces de stockage, ... de nouvelles opportunités sont arrivé au XXI^{e} siècle.

Les entreprises comme Google, Amazon, Facebook, Twitter, ... sont tour à tour arrivés aux limites du modèle Relationnel. Que ce soit a cause de volumes astronomiques (plus de 100 pétaoctets) ou du nombre de requêtes par secondes, il fallut développer une nouvelle façon de gérer les données.

Le NoSql[2] découle de ce genre de problèmes, ces modèles arrivent avec des approche optimisée pour des secteurs spécifiques.
Comme les modèles NoSql représentent ce qui n'est pas Relationnel, par soucis de classification, nous allons distinguer 4 usages principaux :

- Performances : L'objectif du SGBD sera d'augmenter au maximum les performances de la manipulation des données.
- Structures simples : Pour s'affranchir de la rigidité du modèle relationnel, la structure sera généralement simplifiée, en utilisant une représentation plus souple comme le JSON par exemple.
- Structures spécifiques : Certain moteur NoSql sont liés a des besoins spécifiques, la structure de représentation de donnée sera dès lors focalisée sur un cas d'utilisation.
- Volumétries : Un des principal aspect important des SGBD NoSql est leur capacité de

[1] Système de gestion de base de données
[2] Not Only Sql

gérer la montée en charge de données. La distribution des traitements au travers de plusieurs clusters est un facteur très important dans la plupart des applications BigData.

Et nous allons aussi distinguer 4 grandes familles de représentation de Schéma de données :

- Document : L'utilisation de format spécifiques tels que le très rependu JSON permet de stocker les données sur base de fichier.
- Clé / Valeur : Le système le plus simple, il manipule des paires de clé/valeurs, ou accède à un élément en fonction d'une table de hachage.
- Colonne : Inspiré de Google BigTable, la structure ressemble à la table relationnelle. On peut la comparer à une table de hachage qui va référencer une ou plusieurs colonnes.
- Graph : La famille Graph se distingue du fait que les entités ne sont pas considéré comme des entités indépendante, mais que la relation être ces objets est tout aussi important que le contenu.

1.2 De nos jours

Les implémentations de bases de données de types graph sont de plus en plus nombreuses, les relations entre les éléments permettent de parcourir le graph de manière très performante les rendent de plus en plus intéressante pour les entreprises possédant des millions de données. L'utilisation de ce genre de SGBD est dès lors tout a fait recommandé pour des entreprises intéressé entre les relations de ces donnés tels que des profils sociaux, des liens de cause a effet, des liens géographiques et bien d'autre.

1.2.1 Neo4J

Créé par Neo Technologie, une sociétée suédo-américaine, elle est actuellement (selon db-engines.com) la base de donnée orienté graph la plus utilisée dans le monde. Développé en java sous licence GPL V3, AGPL ou licence commerciale, Neo4J représente les données sous formes de "Noeuds" et de "Relations", chacun de ces éléments peuvent contenir une ou plusieurs propriétés. Les propriétés sont des couples clés/valeurs de type simple, comme des chaines de caractères ou des valeurs numériques, des coordonnées spatiales, ...

Une des particularité de Neo4J est l'absence de structure définie, un noeud peut être labellisé afin de permettre de travailler sur un ensemble d'éléments, mais il n'y aura aucune contrainte sur les propriétés du noeud. Cette particularité rend ce SGBD bien adapté pour les modèles évoluant fréquemment.

Le langage de requête propre à Neo4J se nomme "Cypher", il a pour but de réaliser plus simplement que SQL les opérations de parcours ou d'analyse de proximité.

Exemple de requête Cypher :

```
CREATE (mo:Person {name:"Mamours"})
CREATE (mc:Person {name:"Mamyco"})
CREATE (g:Person {name:"Gilles"})
CREATE (m:Person {name:"Marie"})
CREATE (b:Person {name:"Enfant"})
```

```
CREATE (mo)-[:PARENT_OF]->(g)
CREATE (mc)-[:PARENT_OF]->(m)
CREATE (m)-[:PARENT_OF]->(b)
CREATE (g)-[:PARENT_OF]->(b)
```

Ces requêtes vont créer 5 noeud et 4 relations PARENT_OF, nous pouvons aisément comprendre que "Mamyco" est parente de "Marie"

```
MATCH (n:Person)-[:PARENT_OF]->(c:Person)
RETURN DISTINCT (n)
```

Cette query va retourner tout les nœuds distinct qui ont une relation :PARENT_OF avec un autre noeud.

```
MATCH (n:Person)-[:PARENT_OF*2]->(c:Person)
RETURN DISTINCT (n)
```

Celle-ci quant à elle va retourner toutes les personnes qui sont parent de parent et donc grand parents.

ces deux exemples peuvent montrer la force de l'utilisation d'un SGBD de type graph pour représenter un ensemble hierarchique de données par rapport au SGBD relationnelles qui nécessiterai une double jointure sur la Table "Person"

1.2.2 OrientDB

OrientDB est un SGBD initialement développé en C++(Orient ODBMS) ensuite repris en 2010 en Java par Luca Garulli dans une version multi-modèle sous licence Apache 2.0, GPL et AGPL. actuellement 3ème mondial (selon db-engines.com) il offre de nombreuses fonctionnalités intéressante.

OrientDB est base de donnée associant Document et Graph. Elle combine la rapidité et la flexibilité du type document ainsi que les fonctionnalités de relations des bases de données graph.

Ce SGBD est composé de trois grands éléments

- Document & Vertex : Source de contenu, ces élements peuvent être considéré comme des container de données, on peut le comparer avec la ligne d'une base de données relationnelle.
- Links & Edge : Une arrête orientés reliant deux éléments non nécessairement distinct.
- Property : Typée ou embarquée dans un document JSON, ceci va représenter le contenu de l'information. Ces propriétés sont bien entendu primordiales pour ordonner, rechercher, ...

Chaque Document ou Vertex appartient à une "Class", celle-ci peut être strictement définie ou plus laxiste. Comme dans la programmation orientée objet, OrientDB offre le principe de polymorphisme avec un système d'héritage entre les classes.

OrientDB utilise une sorte de SQL avancée pour interpréter les requêtes. On peut de plus

utiliser le langage Gremlin.

Voici quelques exemples d'utilisation du SQL avancé dans OrientDB.

```
CREATE CLASS Person EXTENDS V
CREATE CLASS Company EXTENDS V
CREATE CLASS WorkAt EXTENDS E
CREATE PROPERTY Person.firstname string
CREATE PROPERTY Person.lastname string
CREATE PROPERTY Company.name string
INSERT INTO Person(firstname, lastname) VALUES ("Gilles","Bodart"),("
ou
INSERT INTO Company set name = "ACME"
```

Cet ensemble de requête ressemblant au langage SQL permet de créer deux vertex, Person et Company , un Edge WorkAt et leur associe certaine propriétés. Les deux types d'insert différent permettent comme en SQL d'ajouter un noeud.

```
SELECT FROM V
```

Metadata			Properties		
@rid	@version	@class	firstname	lastname	name
10:0	1	Person	Marie	Van Cutsem	
10:1	1	Person	Gilles	Bodart	
11:0	1	Company			ACME

Part II

Analyse technique

Chapter 2

Application possibles des BDOG

Cette section se concentrera sur l'analyse des besoins utilisateurs, il tentera de répondre aux questions suivantes:

- Pourquoi utiliser une BDOG plutot qu'une BD relationnelle comme Oracle ou MySql ?
- On parle de Base de donnée orientée graph mais quelle est la différence entre un lien entre deux noeud et une relation entre deux table ?
- Si nous devions choisir un exemple qui nécessiterait l'utilisation d'une BDOG, quel serait il ?

2.1 Critères de comparaisons

L'établissement d'une liste non exhaustive de critères de comparaisons objectifs sera établie. Elle me permettra de comparer les différentes BDOG. Les possibilités de réponse à ces critère seront, dans les limites du possible ramenée au choix dual, Oui/Non. Cela permettra d'établir un arbre de décision binaire sur base de besoins clairs.
TODO Dessin arbre binaire

2.2 Comparaison des plus grandes BDOG

Bases de données / Critère	Neo4J	OrientDB	ArangoDB
Schema de donnée strict	X	X	✓
Format de donnée	JSON	JSON	
Principe d'héritage	X	✓	

Une idée d'arbre de décision devrait, à la fin de cette section, permettre à un utilisateur muni de ses besoins, de choisir la BDOG la plus adaptée à son projet.

Chapter 3

Le framework

3.1 Utilisation

Dans l'état actuele de l'avancement de ce mémoire, deux pistes sont envisagée:

- Création d'une API qui sera utilisée par l'application dans le but de simplifier les différentes oppérations sur la base de donnée. Exemple Hibernate pour JDBC.
  TODO Schema
- Création d'un systême d'abstraction qui va englober l'utilisation et de ce fait cacher l'implémentations des différentes oppérations.
  TODO Schema

3.2 Schema conceptuel

Les schémas conceptuels représentant une modèle exemple annoté des éléments du framework sera fourni et commenté dans cette section. Cela permettra au lecteur une meilleur compréhension.

3.3 Documentation

Une documentation claire et précise sur l'utilisation du framework Archipelago sera présente dans cette section. Un ensemble entre une documentation fonctionnelle et une documentation technique faite avec JavaDoc.

3.4 Processus

Description du processus implémenté sur base d'un exemple claire. Explications des différents choix d'implémentations et de chaque étape.

Chapter 4

Evaluation

4.1 Points forts

Autocritique du framework, sur base de test qualitatif et ou quantitatif. Evaluations : usability, performence, qualité, cohérence

4.2 Points faibles

Autocritique du framework, sur base de test qualitatif et ou quantitatif. Evaluations : usability, performence, qualité, cohérence

4.3 Retour d'information

Si le temps nous le permet, une analyse des retours utilisateur sera faite en fin de mémoire.

Chapter 5

Conclusion

5.1 Piste de réflexions

Une introspection sur le projet sera expliqué dans cette section, les idées innachevés y seront décrites en tant que piste de réflexions.

5.2 Archipelago en résumé

Le mémoire sera conclu avec un explication transversale et complète du framework, permettant au lecteur de garder une bonne impression sur le nouvl outil que sera ce framework.

Part III

Annexes

Chapter 6

Code Sources

Chapter 7

Bibliographie

- https://neo4j.com/ consulté à de nombreuses reprises (Neo Technology, Inc)
- https://www.arangodb.com/ consulté à de nombreuses reprises (ArangoDB GmbH)
- https://orientdb.com/ consulté à de nombreuses reprises (OrientDB LTD)
- https://snap.stanford.edu/data/
- https://networkx.github.io/
- http://igraph.org/redirect.html
- https://snap.stanford.edu/data/egonets-Facebook.html
- http://konect.uni-koblenz.de/
- https://icon.colorado.edu/#!/networks
- https://neonx.readthedocs.io/en/latest/
- J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. NIPS, 2012.
- J. Leskovec, K. Lang, A. Dasgupta, M. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. Internet Mathematics 6(1) 29–123, 2009.
- https://www.infoq.com/fr/articles/graph-nosql-neo4j
- http://www.silicon.fr/base-donnees-nosql-impose-sgbdr-93305.html
- https://prezi.com/4flswlgipwbo/nosql-not-only-sql/

LIVRE http://www.eyrolles.com/Chapitres/9782212141559/9782212141559.pdf

- https://db-engines.com