Apache - Cassandra

Philippe Rigaux & Nicolas Travers

nicolas.travers (at) cnam.fr

Plan

Qu'est-ce que Cassandra ?

Cassandra [1], conçu à l'origine par *Facebook*, maintenant un projet de la fondation *Apache* (2008), distribué par la société *Datastax*.

- Initialement basé sur le système *BigTable* de Google (stockage orienté colonnes) ;
- Maintenant basé sur le système *DynamoDB* (hachage) $\Rightarrow$ stockage **orienté "clé/valeur"** ;
- À fortement évolué vers un *modèle relationnel étendu* (N1NF =*Non First Normal Form*) ;
- Un des rares systèmes NoSQL à proposer un typage fort ;
- Un langage de définition de schéma et de requêtes, CQL (pas de jointure).

1. `https://github.com/apache/cassandra/tree/trunk/lib`

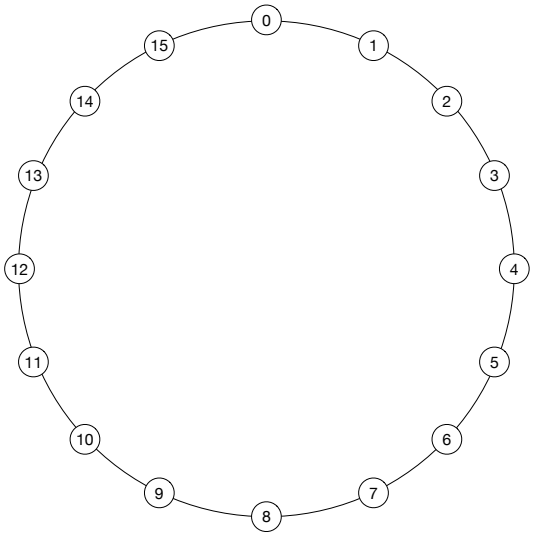
Scalable et distribué

Construit dès l'origine comme un système scalable et distribué.

- Propose des méthodes de passage à l'échelle inspirées du système Dynamo (Amazon) ;
- Distribution par hachage (*consistent hashing*) ;
- Tolérance aux pannes par réplication en mode multi-nœuds.

Très utilisé, réputé très performant.

Table de hachage distribuée



2^{64} noeuds, parcours par sauts (jusqu'à la moitié de l'anneau), réplication, tolérance aux pannes...

Modèle = relationnel + types complexes

- *Bases de données* : Keyspace
- *Tables* : Table ou Column Family
- *Lignes* : Row (valeurs simples ou complexes)

Perspective A : c'est du relationnel étendu en rompant la première règle de normalisation (type atomique).

Perspective B : chaque ligne est un document structuré (imbrication)

⚠Attention ⚠

Vocabulaire confus et parfois déroutant :
Keyspace, *columns*, *column families*...

Paires et documents

La structure de base est la paire (clé, valeur)

- **Clé** : un identifiant
- **Valeur** atomique (entier, chaîne de caractères), ou Valeur complexe (dictionnaire, ensemble, liste)

Une ligne (*row*, document) est un identifiant (*row key*) associé à un ensemble de paires (clé, valeur).

Les lignes sont typées par un schéma, y compris les données imbriquées.

Cassandra n'autorise pas l'insertion de données ne respectant pas le schéma.

Application !

Création d'un **keyspace** :

```
CREATE KEYSPACE IF NOT EXISTS Compagnie
    WITH REPLICATION={'class':'SimpleStrategy','replication_factor':3};
```

Création d'une **Column Family** / table (sans imbrication) :

```
CREATE TABLE Vols (
    idVol INT, dateDepart DATE, distance INT,  duree FLOAT,
    depart CHAR(3), arrivee CHAR(3), pilote INT,
    copilote INT, officier INT, ChefCabine1 INT, ChefCabine2 INT,
    primary key (idVol)
);

CREATE INDEX vol_pilote ON Vols (pilote);
```

Insertions

A la SQL

```
INSERT INTO Vols (idVol, dateDepart, distance,  duree, depart,
        arrivee, pilote, copilote, officier, ChefCabine1, ChefCabine2)
    VALUES (1, '2016-10-15', 344, 1.3, 'CDG', 'LCY', 1, 2, 3, 4, 5);
```

Avec JSon

```
INSERT INTO Vols JSON '{
    "idVol" : 1, "dateDepart" : "2016-10-15", "distance":344,
    "duree" : 1.3, "depart" : "CDG", "arrivee" : "LCY",
    "pilote" : 1, "copilote" : 2, "officier" : 3,
    "ChefCabine1" : 4, "ChefCabine2" : 5}';
```


Interrogation

Le langage **CQL** : *Cassandra Query Language*
Très proche du SQL (version simplifiée pour le NoSQL)

```
SELECT * FROM Vols WHERE idVol = 1;
```

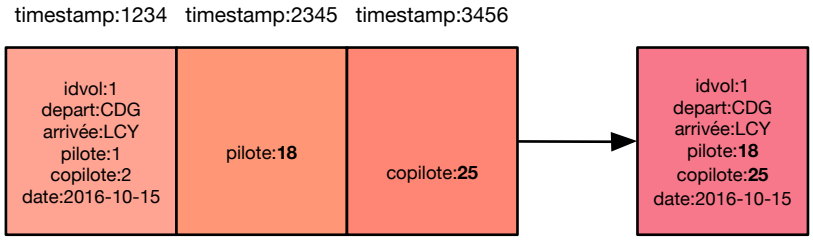
Résultat :

```
idVol | dateDepart  | distance |  duree | depart | arrivee | pilote | copilote | officier
-------------------------------------------------------------------------------------------
    1 |   2016-10-15|      344 |    1.3 |    CDG |     LCY |      1 |        2 |
```

Données temporelles

Toute valeur est associée à un TIMESTAMP.
Estampillage automatique (ms) lors de la mise à jour. Possible de spécifier avec la requête :

```
UPDATE Vols USING TIMESTAMP 2345
    SET pilote=18 WHERE idVol = 1;
```



Données temporelles

Récupérer la date de modification :

```
SELECT writetime(depart), writetime(pilote), writetime(copilote)
FROM Vols WHERE idVol=1;

writetime(depart) | writetime(pilote) | writetime(copilote)
------------------------------------------------------------
             1234 |              2345 |                3456
```

On peut supprimer une valeur définie un instant T :

```
DELETE pilote USING TIMESTAMP 1234 FROM Vols WHERE idVol=1;
```

Cette valeur n'est pas interrogeable (clause WHERE).

Données temporaires

Possible de gérer des données volatiles : TTL
Utilisation identique à TIMESTAMP, donne le nombre de secondes où la valeur est visible.

```
UPDATE Vols USING TTL 3600
    SET pilote=18 WHERE idVol = 1;
```

NoSQL : pas de jointure

Du fait de données distribuées, une jointure n'est pas envisageable. Il est donc préférable de regrouper les données le plus possible dans des lignes.

- Conception orientée sur les requêtes les plus fréquentes de l'application
- Souvent favoriser la dimension la plus grande (meilleure distribution)
- ⚠Le nouveau schéma génère de la redondance et des possibilités d'incohérence

Au pire, on représente les mêmes données sous plusieurs formes (vues matérialisées).

- Tous les vols, avec leurs hôtesses (solution envisageable)
- A chaque hôtesse, la liste de ses vols

Différentes solutions

Pour **un** attribut peut être associé :

1. **SET** : ensemble valeurs (non ordonnées)
2. **LIST** : liste de valeurs (ordonnées)
3. **MAP** : hashMap clé/valeur
4. **SubType** : ligne imbriquée

```
CREATE TABLE Vols (
    idVol INT, dateDepart DATE, distance INT,  duree FLOAT,
    depart CHAR(3), arrivee CHAR(3), pilote INT,
    copilote INT, officier INT, ChefCabine1 INT, ChefCabine2 INT,
    hotesses XXX,
    primary key (idVol)
);
```

SET : hotesses set<int>

- *Insertion* :
  INSERT INTO Vols (..., hotesses) VALUES (..., {6, 7, 8}) ;
- *Mise à jour* :
  UPDATE Vols SET hotesses = hotesses + {9} WHERE idVol = 1 ;
  UPDATE Vols SET hotesses = hotesses - {8} WHERE idVol = 1 ;
  UPDATE Vols SET hotesses = {10} WHERE idVol = 1 ;
  DELETE hotesses FROM Vols WHERE idvol = 1 ;
- *Liste des hotesses* :
  SELECT idVol, hotesses FROM Vols WHERE idVol = 1 ;

```
idVol |  hotesses
------------------
    1 | {6, 7, 8}
```


LIST : hotesses list<int>

- *Insertion* :
  INSERT INTO Vols (..., hotesses) VALUES (..., [6, 7, 8]) ;
- *Mise à jour* :
  UPDATE Vols SET hotesses = hotesses + [9] WHERE idVol = 1 ;
  UPDATE Vols SET hotesses[1] = 8 WHERE idVol = 1 ;
  UPDATE Vols SET hotesses = [10] WHERE idVol = 1 ;
  DELETE hotesses[0] FROM Vols WHERE idvol = 1 ;
- *Liste des hotesses* :
  SELECT idVol, hotesses FROM Vols WHERE idVol = 1 ;

```
idVol | hotesses
------------------
    1 | [6, 7, 8]
```

MAP : hotesses map<text, int>

- *Insertion* :
  INSERT INTO Vols (..., hotesses) VALUES (..., {"h1" : 6, "h2" : 7, "h3" : 8}]) ;
- *Mise à jour* :
  UPDATE Vols SET hotesses = hotesses + {"h4" : 9} WHERE idVol = 1 ;
  UPDATE Vols SET hotesses["h1"] = 10 WHERE idVol = 1 ;
  UPDATE Vols SET hotesses = {"h1" : 9} WHERE idVol = 1 ;
  DELETE hotesses["h2"] FROM Vols WHERE idvol = 1 ;
- *Liste des hotesses* :
  SELECT idVol, hotesses FROM Vols WHERE idVol = 1 ;

```
idVol |                        hotesses
-------------------------------------
    1 | {"h1" : 6, "h2" : 7, "h3" : 8}
```


SubType : hotesse frozen<hotesseType>

- Création du *type* à imbriquer :
  CREATE TYPE hotesseType (ID int, nom text, prenom text) ;
  Il faut "*figer*" le type : (frozen<hotesseType>).
- *Insertion* :
  INSERT INTO Vols (..., hotesses) VALUES (..., {"ID" : 6, "nom" : "Walthéry", "prenom" : "Natacha"}) ;
- *Mise à jour* :
  UPDATE Vols SET hotesse["nom"] = "Walter" WHERE idVol = 1 ;
  DELETE hotesse FROM Vols WHERE idvol = 1 ;
- *Liste des hotesses* :
  SELECT idVol, hotesse.nom, hotesse.prenom FROM Vols WHERE idVol=1 ;

```
idVol | hotesse.nom | hotesse.prenom
-------------------------------------
    1 |    Walthéry |        Natacha
```

CQL 3.3

- SELECT ...
  Attributs / sur clés et indexes : DISTINCT, COUNT(*)
- FROM
  Une seule table possible
- (WHERE ...)?
  Voir transparent suivant
- (ORDER BY ...)?
  Clé primaire (ASC/DESC), ou si prédicat sur clé primaire
- (LIMIT ...)?
  Clé primaire (ASC/DESC), ou si prédicat sur clé primaire
- (ALLOW FILTERING)?
  Prédicats sans indexes secondaires

Clause *WHERE*

- **Clé primaire = valeur** : Le plus efficace
- **token(Clé primaire) > valeur** : Liste de tokens par noeud
- **attribut = valeur + INDEX** : Moins efficace
- **attribut = valeur + ALLOW FILTERING** : Peu efficace
  test sur tous les noeuds
- **Données imbriquées** :
  - SET : *CONTAINS*
  - LIST : *CONTAINS*
  - MAP : *CONTAINS / CONTAINS KEY*
  - SubType : att = valeur (tuple imbriquée = BLOB)

  ⇒ Le *MAP* est à privilégier

Indexation

Création d'un index secondaire :

- Classique : CREATE INDEX <index_name> ON <table_name> (<attribute>) ;
- Map :
  - Filtrage d'une valeur avec "attribute CONTAINS <value>"
  - Filtrage sur le nom de la clé :
    CREATE INDEX <index_name> ON <table_name> (keys(<attribut>)) ;
    "attribute CONTAINS KEY <key_name>"
- Un seul index par attribut

Installation du serveur Cassandra

- Docker : chercher l'image officielle (la première), télécharger, lancer et c'est tout !
- Windows : DataStax server [2]. ⚠Fichier de 1Go !
- Linux : Package à installer [3]
- MacOS : DataStax server [4] ou avec Python (pip) [5]

2. Guide complet windows : http://www.datastax.com/2012/01/getting-started-with-apache-cassandra-on-windows-the-easy-way
3. https://cassandra.apache.org/download/
4. Guide complet Mac : http://www.datastax.com/2012/01/getting-started-with-apache-cassandra-on-windows-the-easy-way
5. pip install : https://dbglory.wordpress.com/2015/02/22/installing-cassandra-on-mac-os-x/

Client

Interface client

- CQLSH : ligne de commande (disponible sous Docker, ou executable pour les autres)
- DevCenter : Application de DataStax[6] (*eclipse like*). Compte nécessaire.
- Autres interfaces mais licences limitées.

Connexion client

- port 9042 par défaut
- Attention à la redirection sous Docker

6. `https://academy.datastax.com/downloads/ops-center?destination=downloads/ops-center&dxt=DevCenter#devCenter`

Application

Drivers

- PlanetCassandra.org[7] : Java, Python, Ruby, C#, Node.js, PHP, C++, Spark, ...
- Connexion avec host+port
- Exec, Statement, PreparedStatement
- ResultSet

7. Drivers : `http://www.planetcassandra.org/apache-cassandra-client-drivers/`

Distribution & Réplication

Consistent Hashing

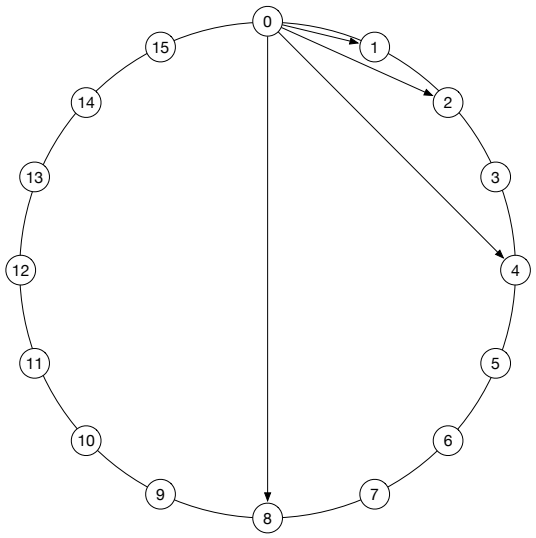
- Anneau avec table de hachage distribuée (sauts)
  Hachage continu pour les plages de valeurs (token)
- Réplication des données dans l'anneau : *Replication Factor* (3 par défaut)
  Réplicats répondent aux requêtes

Cohérence

- Nombre de réplicats à contacter : QUORUM
- `Anti Entropy Service` (AES) ⇒ garanti le taux de réplication

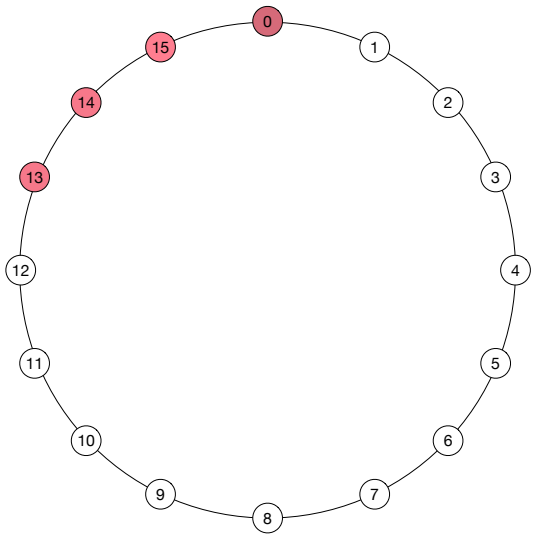
Consistent Hashing

- Distribution de données
- Rapidité (max $log_2(N)$ sauts)
- Réplication
- Tolérance aux pannes
- Elasticité
- Non centralisée



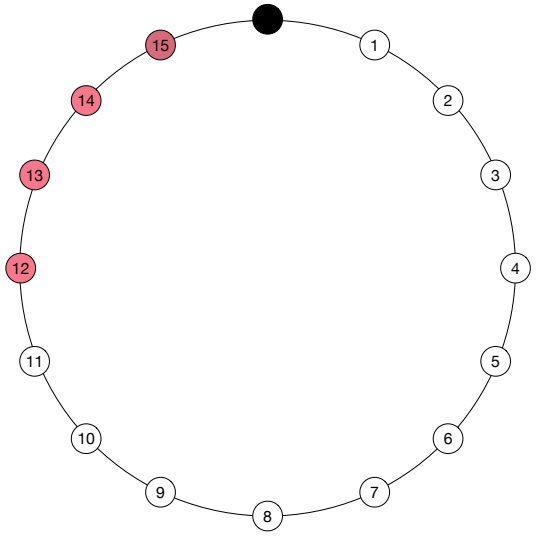
Consistent Hashing

- Distribution de données
- Rapidité (max $log_2(N)$ sauts)
- Réplication
- Tolérance aux pannes
- Elasticité
- Non centralisée



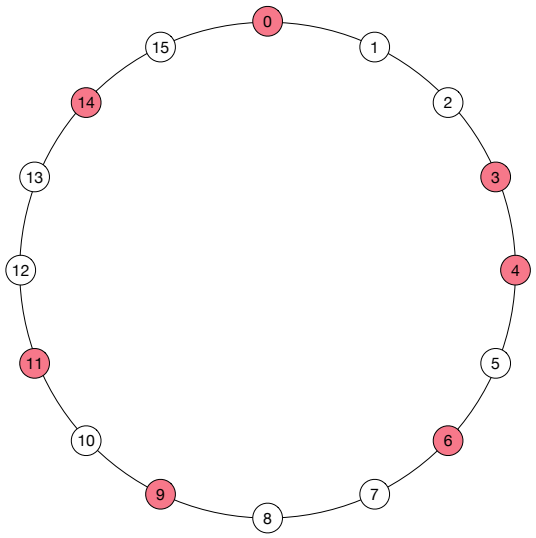
Consistent Hashing

- Distribution de données
- Rapidité (max $log_2(N)$ sauts)
- Réplication
- Tolérance aux pannes
- Elasticité
- Non centralisée



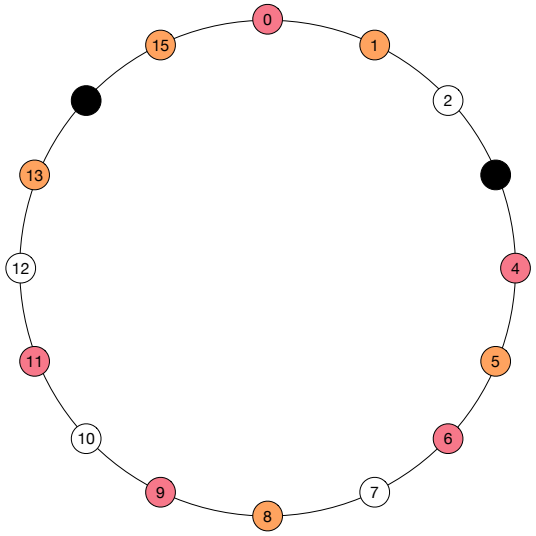
Consistent Hashing

- Distribution de données
- Rapidité (max $log_2(N)$ sauts)
- Réplication
- Tolérance aux pannes
- Elasticité
- Non centralisée



Consistent Hashing

- Distribution de données
- Rapidité (max $log_2(N)$ sauts)
- Réplication
- Tolérance aux pannes
- Elasticité
- Non centralisée



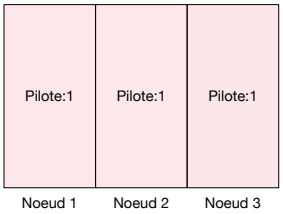
Cohérence : Gestion du Quorum

R + W > N

- **N** : Taux de réplication
- **W** : Nb minimum d'écritures devant accuser de réception
- **R** : Nb copies d'une donnée à consulter pour une requête
- Ecritures et lectures en parallèle
- Si W + R <= N ⇒ Cohérence éventuelle
  (pannes, charge, nb réplicas)

Exemple

- Soit N=3
- MAJ : pilote=18
  2 acquittements (**W=2**)
- 2 lectures en parallèle (**R=2**)
- 2+2>3 ⇒ retourne : 18



Introduction Modèle & Requêtes Installation Scalable Distribution & Réplication Cohérence Map/Reduce Création d'un cluster

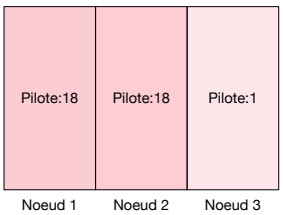
Cohérence : Gestion du Quorum

R + W > N

- **N** : Taux de réplication
- **W** : Nb minimum d'écritures devant accuser de réception
- **R** : Nb copies d'une donnée à consulter pour une requête
- Ecritures et lectures en parallèle
- Si W + R <= N ⇒ Cohérence éventuelle
  (pannes, charge, nb réplicas)

Exemple

- Soit N=3
- MAJ : pilote=18
  2 acquittements (**W=2**)
- 2 lectures en parallèle (**R=2**)
- 2+2>3 ⇒ retourne : 18



Cohérence : Options de Cohérence

- `CONSISTENCY <level> ;`

  level : ANY, ONE, TWO, THREE, QUORUM, ALL
- **ONE/ANY** : par défaut, au moins 1 acquittement (Bon compromis)
- **ALL** : W tous acquittés, 1 seul R (Système avec peu d'écritures)
- **QUORUM** : N/2 + 1 (pour R et W)
  (Véritable quorum, possibilité d'incohérence - faible)

Agrégats

Comment effectuer un "GROUP BY + COUNT" ?

- Concept de Map/Reduce pour le NoSQL.
  - Programme décomposer en deux parties : filtrage + regroupement
  - *Map* : prend une ligne et produit une **clé/valeur** en sortie
  - *Reduce* : prend une clé (provenant du map) avec la liste de valeur et produit une valeur en sortie (pour la clé).
- Sous *Cassandra* :
  - Programme Java appelé : **User-Defined Aggregate Function** (UDA)

`https://docs.datastax.com/en/cql/3.3/cql/cql_using/useCreateUDA.html`

User-Defined Aggregate Function

Map

```
CREATE OR REPLACE FUNCTION avgState ( state tuple<int,bigint>, val int )
    CALLED ON NULL INPUT RETURNS tuple<int,bigint> LANGUAGE java
    AS 'if (val !=null) { state.setInt(0, state.getInt(0)+1);
                          state.setLong(1, state.getLong(1)+val.intValue()); }
       return state;';
```

Reduce

```
CREATE OR REPLACE FUNCTION avgFinal ( state tuple<int,bigint> )
    CALLED ON NULL INPUT RETURNS double LANGUAGE java
    AS 'double r = 0;
          if (state.getInt(0) == 0) return null;
          r = state.getLong(1);
          r/= state.getInt(0);
          return Double.valueOf(r);';
```

UDA

```
CREATE AGGREGATE IF NOT EXISTS average ( int )
    SFUNC avgState STYPE tuple<int,bigint>
    FINALFUNC avgFinal INITCOND (0,0);

SELECT average (distance) FROM Vols;
```

Cluster

- **Bootstrapping :** L'ajout de noeuds (ou *bootstrapping*) utilise la notion de 'token'. Un noeud va devenir gestionnaire d'une section de l'anneau (par défaut 256 noeuds par cluster).