

Apache Cassandra : un SGBD NoSQL hybride

Jonathan Lejeune

Sorbonne Université/LIP6-INRIA

DataCloud – Master 2 SAR 2020/2021



sources :

cours de Bases de données documentaires et distribuées, Philippe Rigaux

<https://www.tutorialspoint.com/cassandra>

https://en.wikipedia.org/wiki/Apache_Cassandra

<https://docs.datastax.com/en/cassandra/3.0/index.html>

Principales caractéristiques

- Une architecture distribuée pair à pair :
 - ⇒ aucun nœud central
 - ⇒ tout nœud est interchangeable avec un autre
- Des mécanismes de réPLICATIONS configurables sur plusieurs data-center
 - ⇒ tolérance aux pannes
 - ⇒ disponibilité permanente des données
- Passage à l'échelle linéaire et horizontale
 - ⇒ nombre de machines $\times 2$ = stockage et puissance de calcul $\times 2$
- Une cohérence configurable entre les réPLICAS
 - ⇒ compromis possible entre AP et CP du théorème CAP
- Interfaçable avec des traitements parallèles type MapReduce
 - ⇒ intégration naturelle dans l'écosystème Hadoop
- Cassandra Query Language (CQL)
 - ⇒ langage de requêtage haut niveau type SQL

Introduction à Cassandra

Bref historique

- 2007 : Conception originelle par les ingénieurs de Facebook pour des besoins internes
- 2009 : Cassandra est porté par l'Apache Incubator et sa communauté
- 2010 : promotion au rang des projet top-level
- juin 2011 (v0.8) : ajout du CQL
- depuis juin 2017 : version 3.11

État actuel

- **Développeur** : fondation Apache.
- **Principal distributeur et contributeur** : société Datastax
- **Principaux utilisateurs** : Apple, CERN, Cisco, Netflix, Uber, ...

Cassandra est une des technologies NoSQL open-source les plus utilisées dans le monde de l'industrie dans le domaine du Big Data

Un outil qui a beaucoup évolué

Initialement

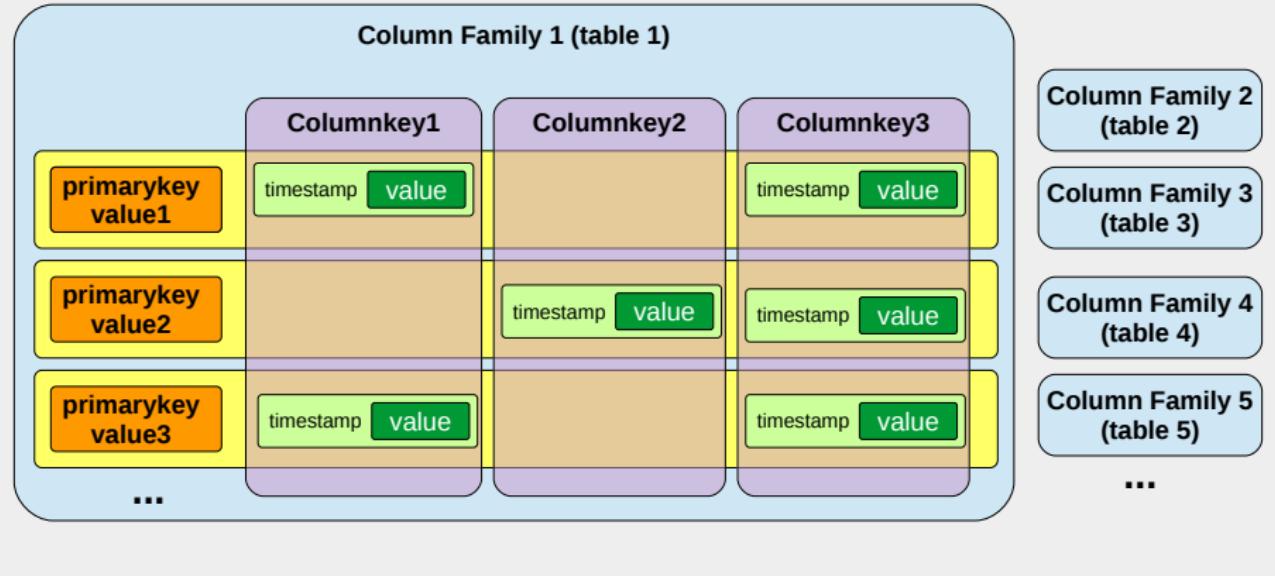
- Héritage :
 - du modèle de donnée de BigTable (orienté colonne)
 - de l'architecture de DynamoDB (orienté document)
- Interfaçage client Thrift exposant la représentation physique des données
- Un modèle de données très permissif

Aujourd'hui

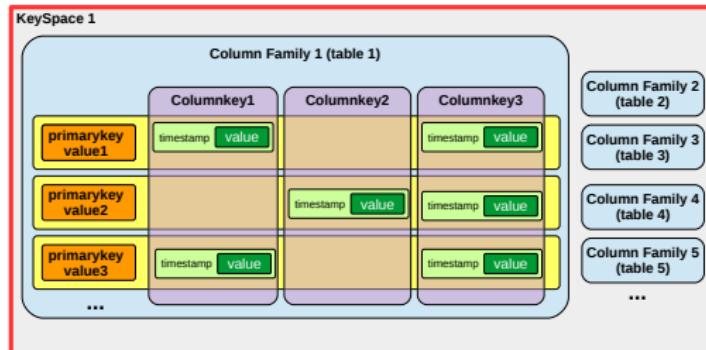
- Un modèle de données :
 - plus proche des SGBD relationnels (table, colonne, clé primaire, ...)
 - ayant une flexibilité intermédiaire entre NoSQL et SGBD classiques
 - avec un typage fort des données
- le langage CQL3 permet :
 - un requêtage type SQL sur un système NoSQL
 - une abstraction du stockage physique

Modèle de données : Aperçu

KeySpace 1



Modèle de données : les keyspaces



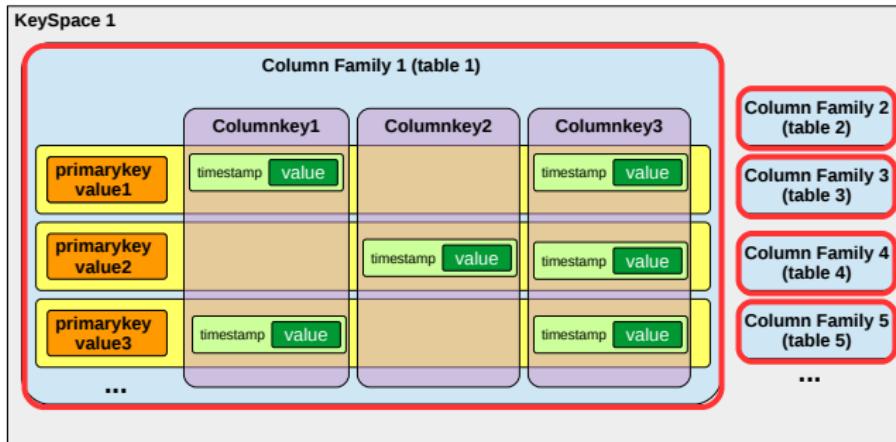
Caractéristiques

- Espace de nom regroupant un ensemble logique de tables
- Définit un facteur de réPLICATION (**RF**) + une stratégie de réPLICATION :
 - SimpleStrategy : le même RF pour tout le système (cluster)
 - NetworkTopologyStrategy : un RF par datacenter

Les keyspaces prédéfinis pour les métadonnées du système

`system, system_auth, system_distributed, system_schema, system_traces`

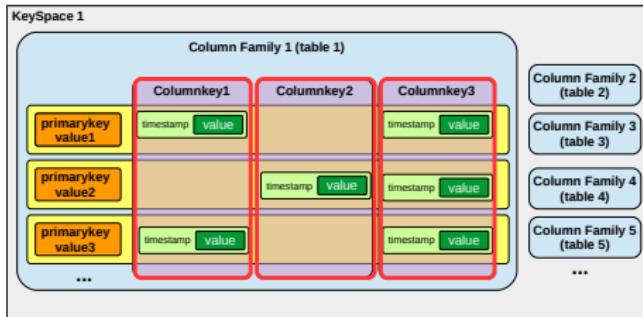
Modèle de données : les tables (ou column family)



Caractéristiques

- Ensemble de lignes de données
- Définit une clé primaire pour identifier les lignes
- Définit l'ensemble des colonnes de chaque ligne de la table
- Possède des propriétés internes : politique de cache, compactage, etc.

Modèle de données : les colonnes



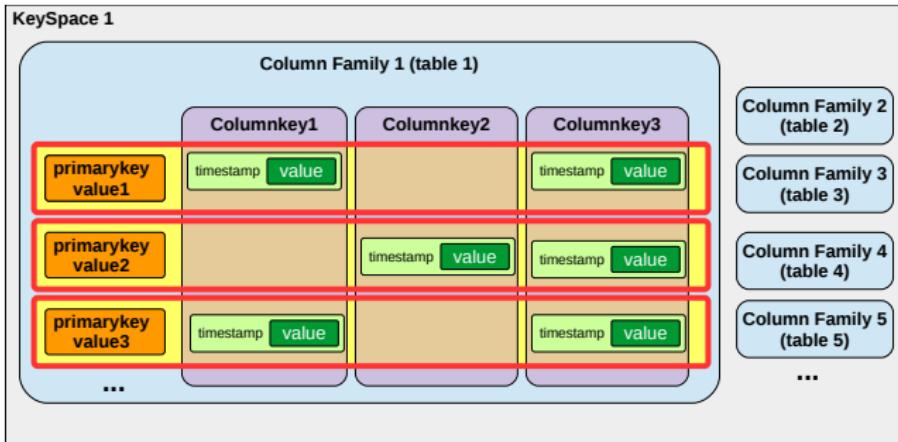
Caractéristiques

- Association d'une clé (nom de la colonne) à une donnée d'une ligne
- Possède un type

Exemples de type

- Types simples : chaîne, date, entier, flottant, adresse IP, booléen, timestamp
- Types collection : list, set, map
- Type tuple

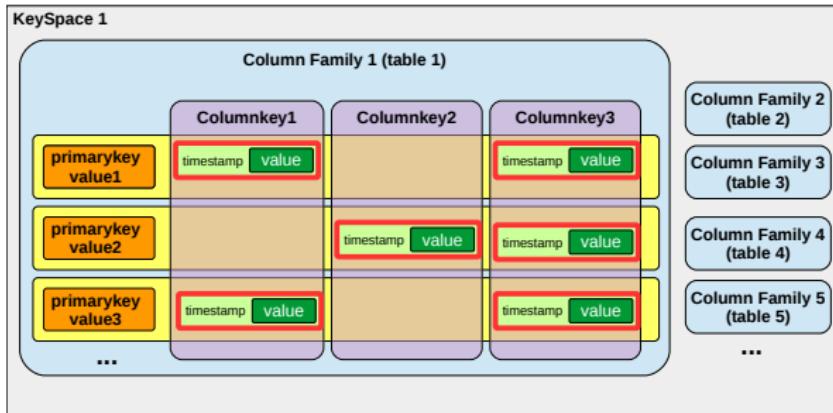
Modèle de données : les lignes



Caractéristiques

- Autres noms : document, partition
- Ensemble de valeurs typées
- Grain de réPLICATION ⇒ chaque ligne de la table est répliquée géographiquement
- Identifiée par une clé primaire (= rowkey ou clé de partition)

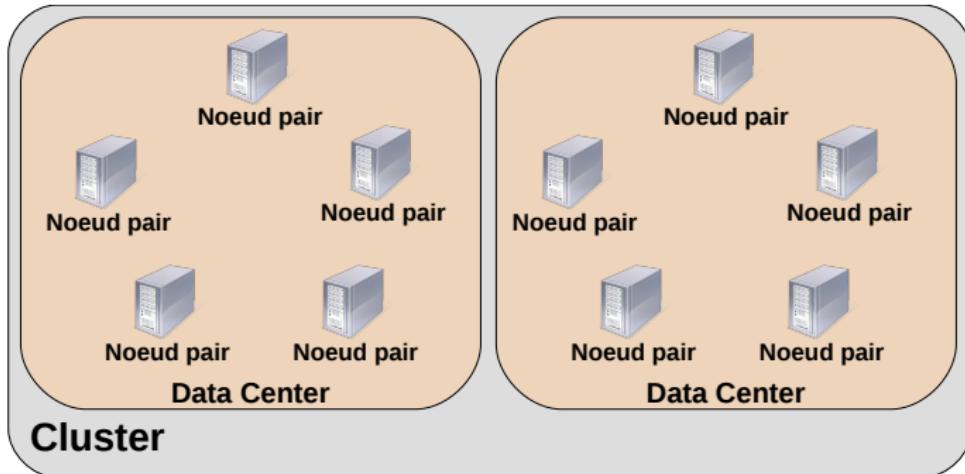
Modèle de données : les données



Caractéristiques

- valeur obtenue à partir d'une clé de ligne et d'une clé de colonne
- associée à un timestamp indiquant la date où la donnée a été écrite
 - ⇒ permet de dater les différents réplicas
 - ⇒ permet de conserver une série temporelle de valeurs
 - ⇒ utile lors de phases de resynchronisation entre réplicas suite à une partition réseau

Infrastructure physique : une architecture pair à pair



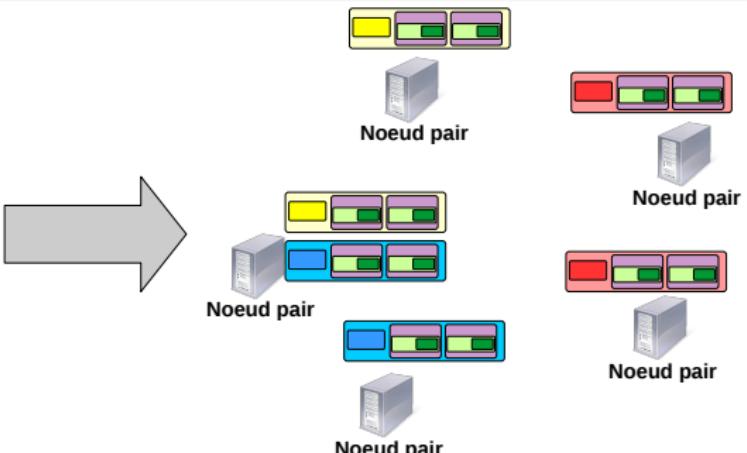
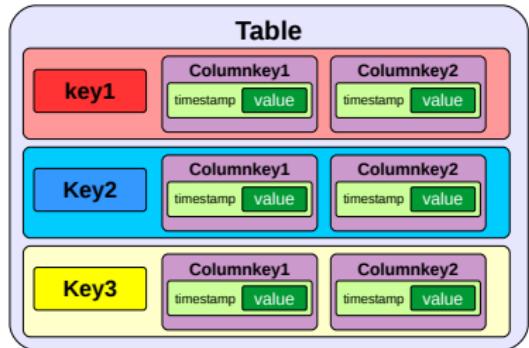
Composants

- **Nœud** : machine stockant les données
- **Cluster** : ensemble des nœuds de stockage
- **DataCenter** : sous-ensemble de nœuds de stockage
- **Rack** : regroupement physique de nœud au sein d'un Datacenter

Placement physique des données

Principes de distribution et de réPLICATION

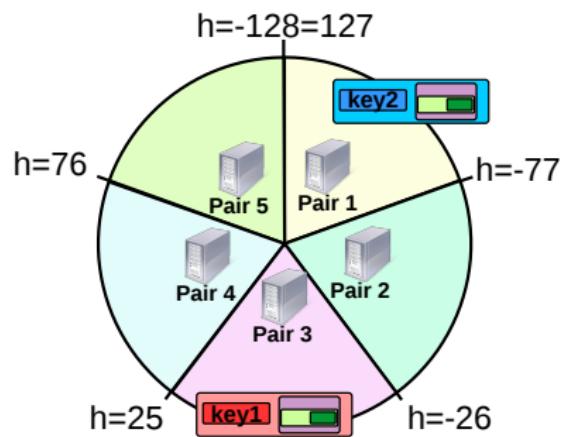
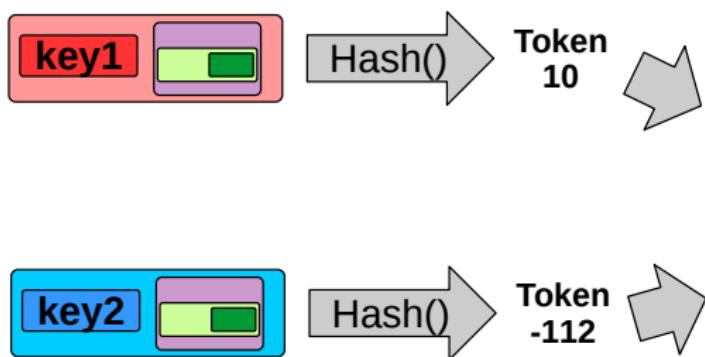
- La granularité de découpage d'une table est la ligne
- La distribution et la réPLICATION de lignes sur le cluster dépend :
 - d'une fonction de hachage (partitionnement)
 - du facteur de réPLICATION et la stratégie de réPLICATION du keyspace
 - de la correspondance de nœuds virtuels sur les nœuds physiques
 - du snitching



Partitionnement

Un principe similaire à une DHT

- Fonction de hachage calculant un token à partir de la clé de la ligne.
- Chaque nœud stocke un intervalle de valeur de token
- Dans un datacenter, les nœuds sont organisés logiquement en anneau sur l'espace de définition des valeurs de token



Politiques de partitionnement

Partitionnements à partir de fonctions hachages standards

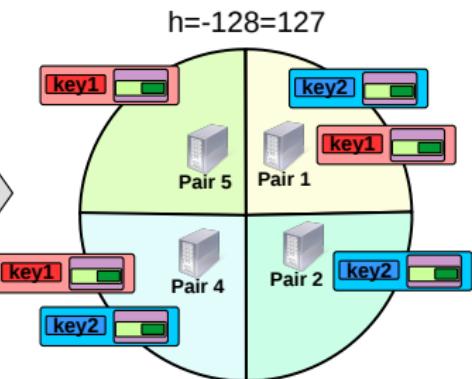
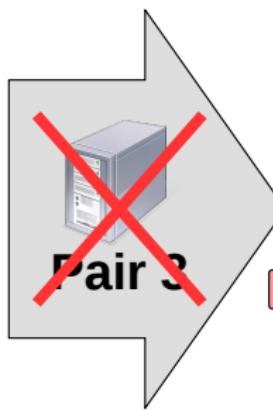
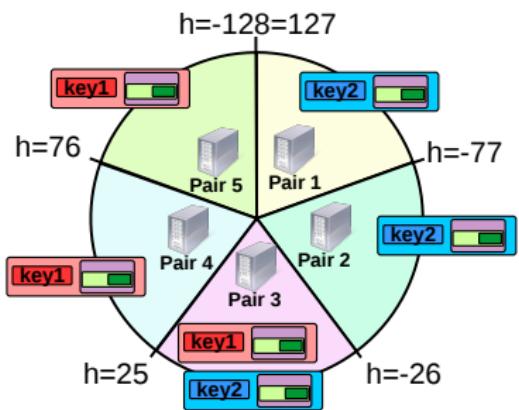
- Murmur3Partitioner (par déf.) : hachage selon l'algo Murmur3
 - RandomPartitioner : hachage selon l'algo MD5
- ✓ Répartition uniforme sur les nœuds
- ✗ Pas de stockage ordonné

Partitionnement ordonné

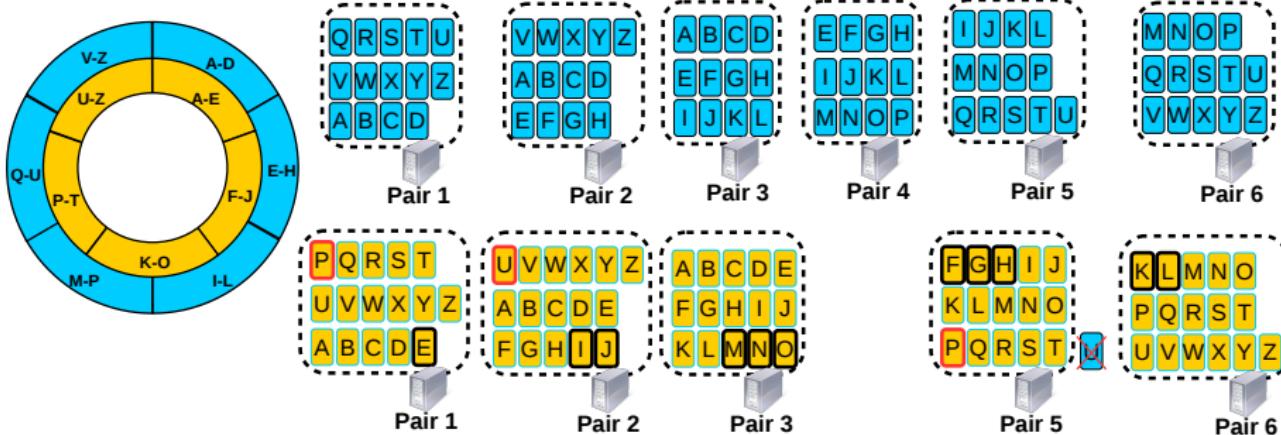
- ByteOrderedPartitioner : les clés sont triées octet par octet
- ✓ Les données sont triés et les intervalles correspondent directement à la valeur de clé
- ✗ Distribution potentiellement non uniforme

Principe

- Chaque keyspace définit un facteur de réPLICATION par datacenter
- Au sein d'un datacenter, les réPLICAS sont stockés sur les nœuds suivants dans l'espace des valeurs de token



RéPLICATION



Problématique

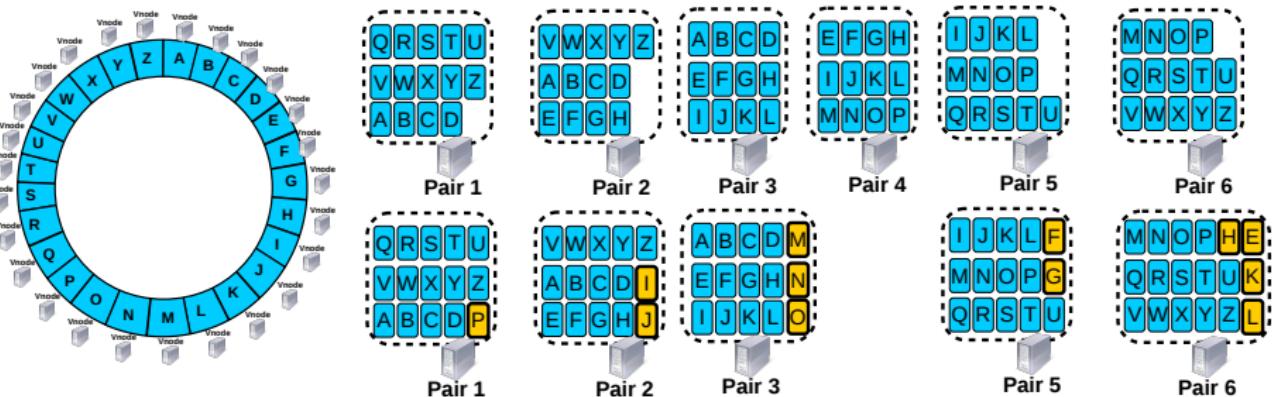
L'ajout ou la suppression d'un nœud (voulu ou pas) nécessite de recalculer les intervalles de responsabilité de chaque nœud physique.

- ✗ Surcoût en déplacement de données pour reconstruire l'anneau

Vnode

Solution

- Définir des nœuds virtuels en associant un intervalle de token plus petit (voire un seul token) à chacun
- Un nœud virtuel est hébergé sur RF nœuds physiques
- A la perte d'un nœud physique, il suffit de réaffecter les vnodes concernés sur d'autres nœud physiques

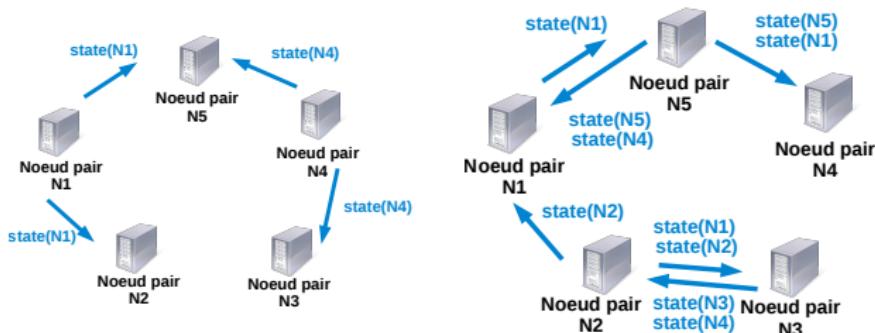


Gossip

Définition

Échange périodique d'information entre nœuds voisins :

- sur leur état
- sur ce qu'il savent de l'état des autres nœuds



Avantages

- ✓ Pas de diffusion globale, l'information se propage de voisin en voisin
- ✓ Envoi périodique = heartbeat = détection des pannes

Définition

Mécanisme se basant sur le protocole de gossip pour :

- propager assez d'information sur la topologie réseau afin de router efficacement les requêtes
- grouper automatiquement les nœuds en datacenter ou rack
- éparpiller les réplicas efficacement sur le cluster afin d'avoir au plus un réplica par rack

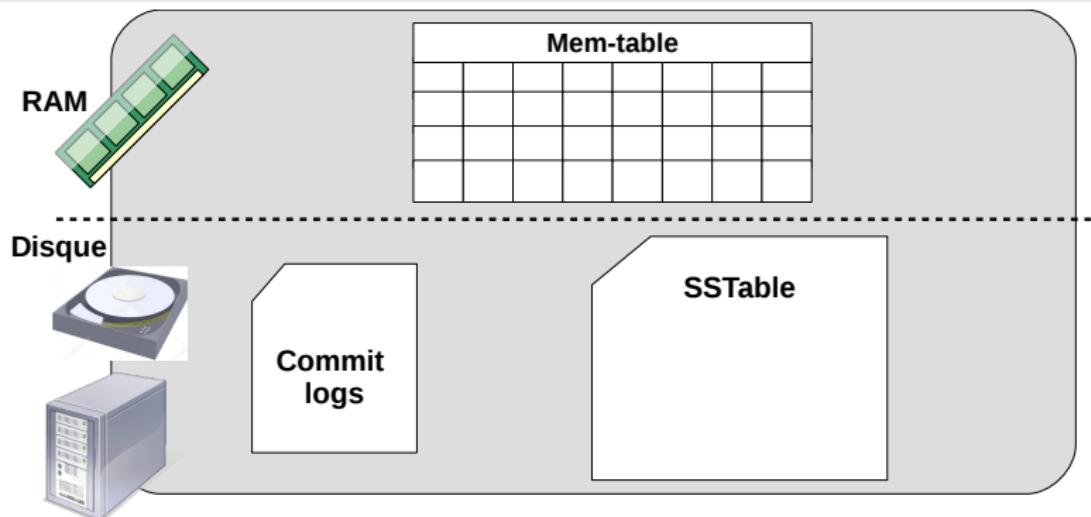
Quelques exemples de stratégies

- **Dynamic snitching** : choisit le réplica en fonction d'un historique de latences de lecture
- **SimpleSnitch** : utile si le cluster ne contient qu'un seul DC
- **RackInferringSnitch** : détermine les DC en fonction des IP des nœuds
- **PropertyFileSnitch** : détermine les DC à partir d'un fichier de conf

Stockage physique local

Composants internes pour une table

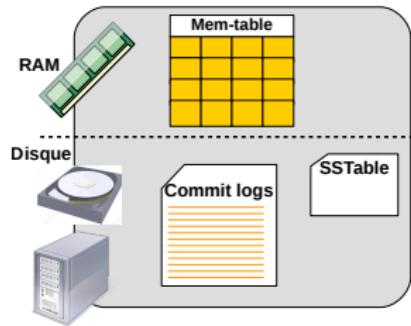
- **Commit log** : fichier pour journaliser les opérations d'écriture
- **Mem-table** : Cache write back RAM de la table trié par clé.
- **SSTable** : Fichier **immuable** contenant une version persistante et durable de la table



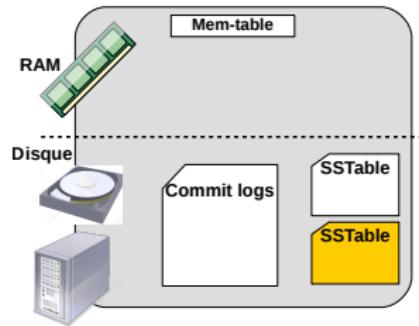
Stockage physique local : Flushing

Principes

- Transfert des données de la mem-table vers le disque
- Déclenché :
 - quand la mem-table atteint un certain seuil de taille
 - quand le fichier de Commit Log est plein
 - manuellement (outil nodetool)
- Recyclage de la mem-table et des logs.
- Création d'un nouveau fichier de SSTable



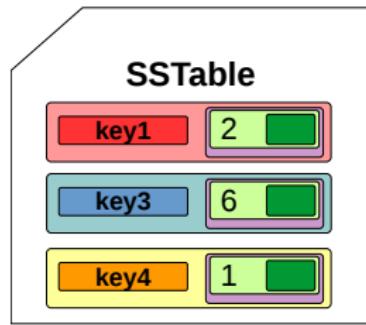
Flushing



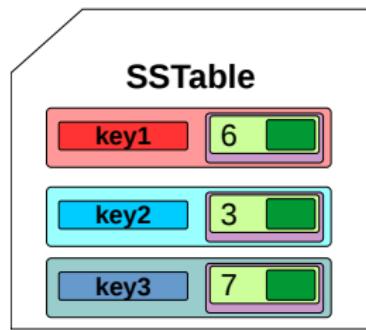
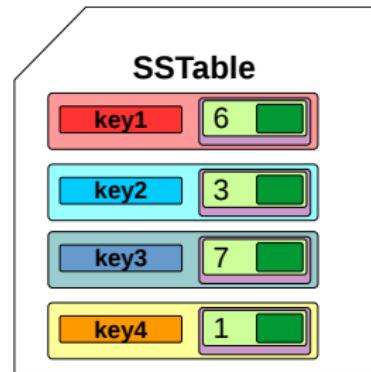
Compaction

Objectifs

Fusionner périodiquement les SSTables et effacer les données obsolètes



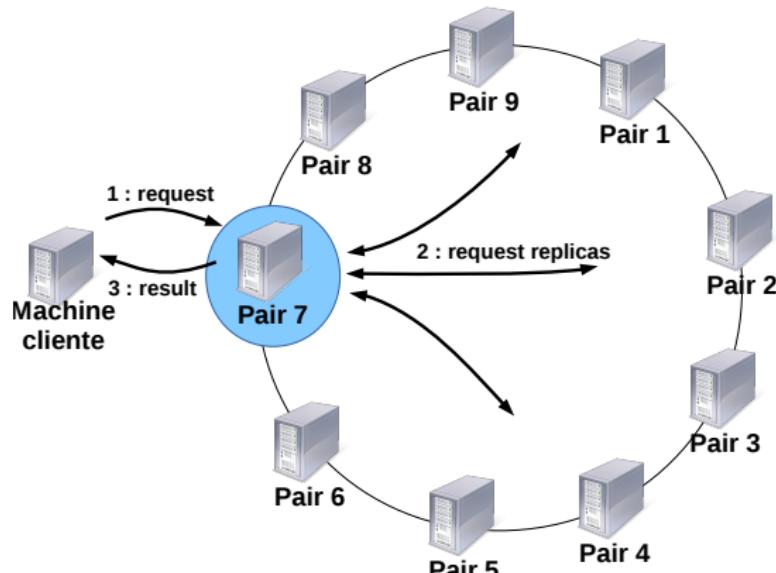
Compaction



Fonctionnement globale d'une requête cliente

Concepts associés à une requête cliente

- **Coordinateur** : Le nœud pair qui interagi avec le client
- **Niveau de cohérence** : indique un nombre de répliques auxquels il faut accéder pour que la requête termine



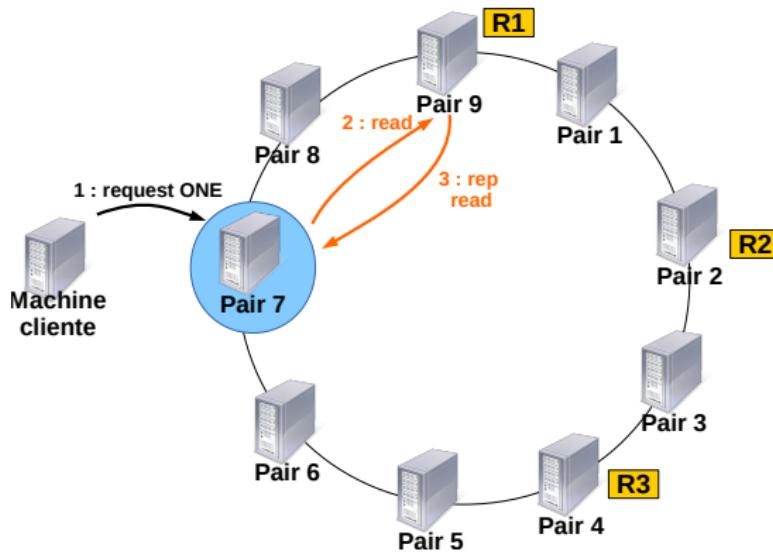
Principes

- Le coordinateur envoie la requête aux nombres de répliques voulus
⇒ on s'adresse aux répliques les plus proches d'après le mécanisme de snitching.
- Quand tous les résultats sont reçus, les comparer et vérifier leur cohérence
- Si il existe différentes versions, renvoyer au client la version avec le timestamp le plus grand
- En cas d'incohérence, le coordinateur envoie une requête `read_repair` aux répliques pour mise à jour des obsolésecences

Requête de lecture : niveaux de cohérence

Niveau ONE

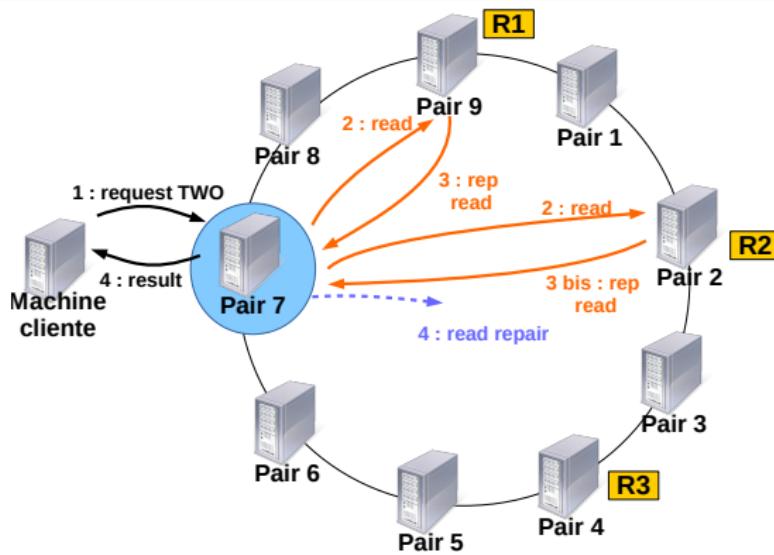
- Le coordinateur s'adresse au réplica le plus proche de lui
- Aucun read_repair envoyé
- Haute disponibilité, **Faible cohérence**



Requête de lecture : niveaux de cohérence

Niveaux TWO et THREE

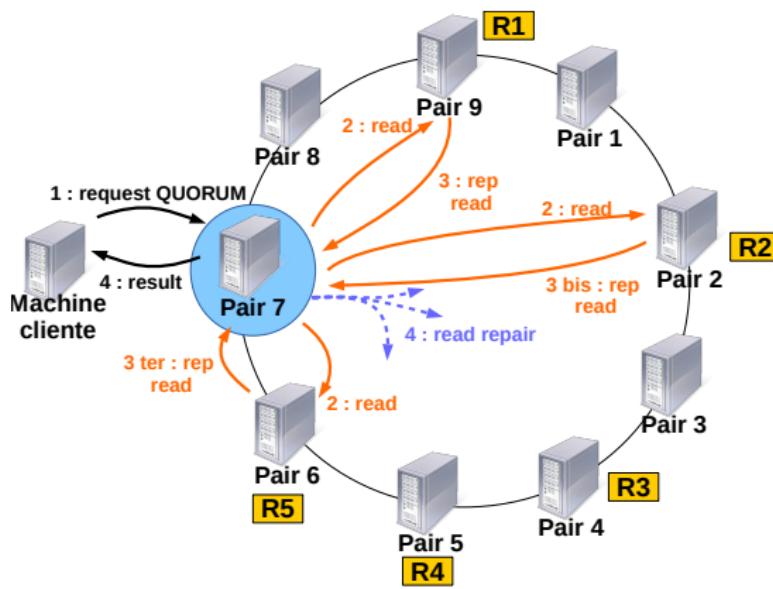
- Le coordinateur s'adresse au 2 (ou 3) répliques les plus proches de lui
- Un read_repair est éventuellement envoyé
- Haute disponibilité, **Faible cohérence**



Requête de lecture : niveaux de cohérence

Niveau QUORUM

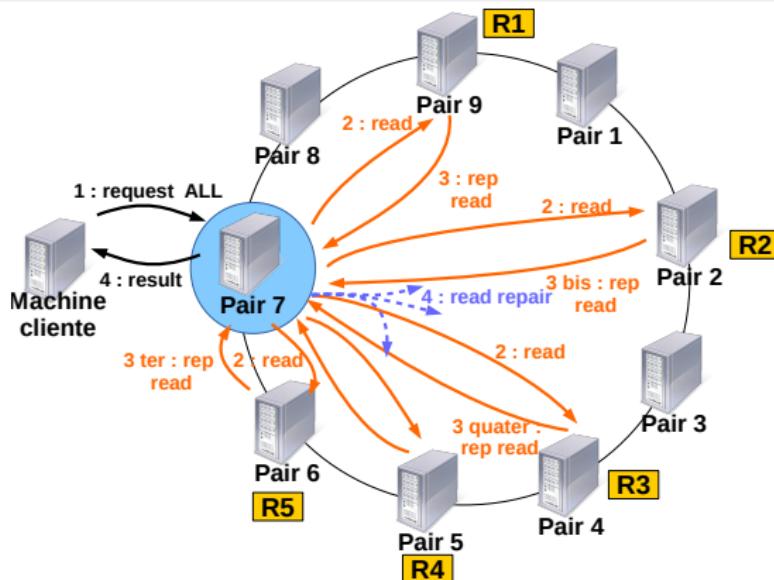
- Le coordinateur s'adresse à une majorité de répliques
- Un read_repair est éventuellement envoyé



Requête de lecture : niveaux de cohérence

Niveau ALL

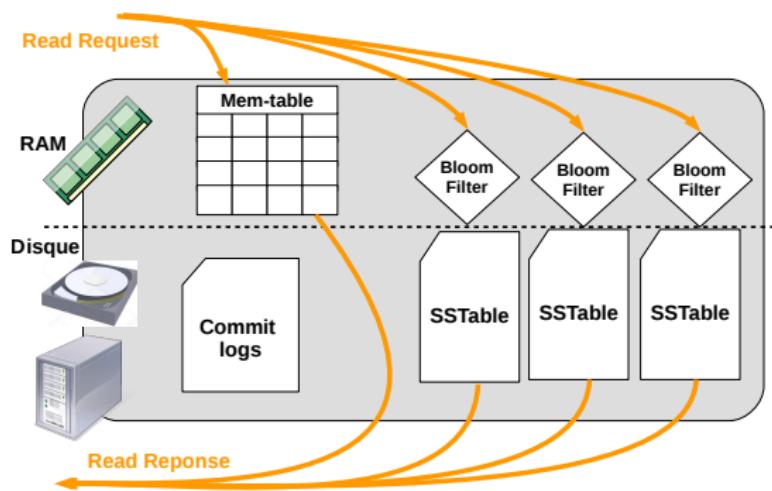
- Le coordinateur s'adresse à **tous** les réplicas
- si un réplica ne répond pas, la requête échoue
- Cohérence maximum, **Disponibilité faible**



Requête de lecture : point de vue local

Étapes pour lire une ligne de clé K

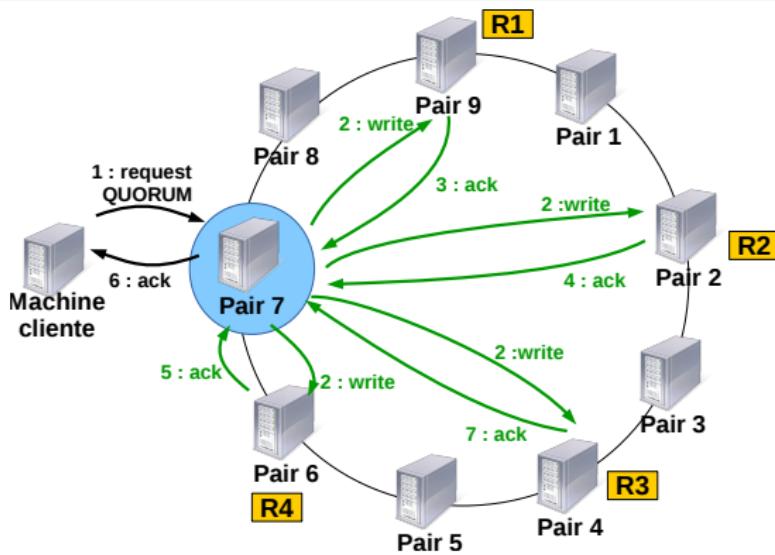
- Lire toute les données concernant K dans le memtable
- Pour tout fichier SSTable concerné :
 - tester la présence de la clé grâce à un filtre de Bloom
 - si présent lire les données dans le fichier



Requête d'écriture

Principes

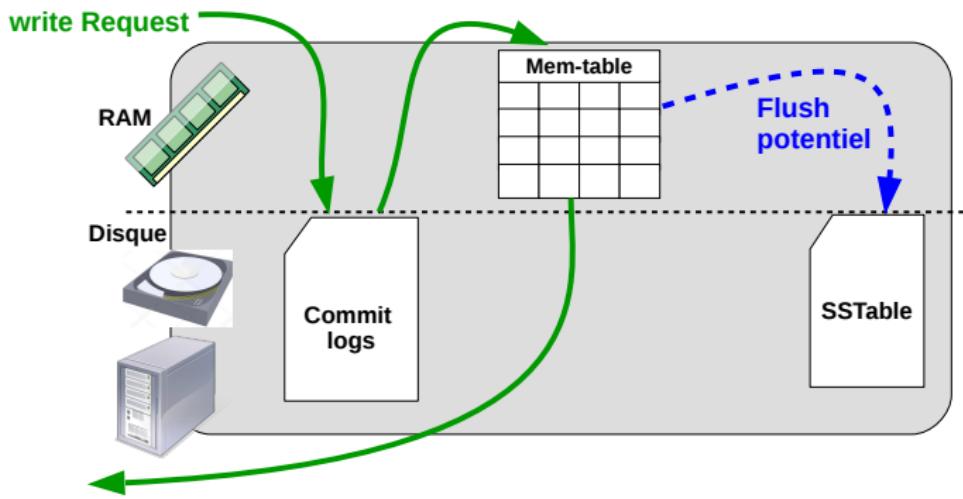
- Le coordinateur envoie la requête à tous les réplicas
- Le niveau de cohérence = le nb de réplicas devant acquitter l'écriture
- La requête est terminée lorsque tous les ack requis ont été reçus



Requête d'écriture : point de vue local

Étapes

- Écriture dans le commit log
- Écriture dans la Mem table
- Flush éventuellement la mem-table vers une SSTable



Interface client de Cassandra : le langage CQL3

Qu'est ce que c'est ?

- un langage haut niveau de requêtage
- hérite de la syntaxe de SQL

Similaires avec SQL

- **Data Definition Language** = opération sur le schéma de données
⇒ CREATE, DROP, TRUNCATE, ALTER
- **Data Manipulation Language** = modification des données
⇒ INSERT, UPDATE, DELETE
- **Data Retrieval Language** = interrogation des données
⇒ SELECT

Différences majeures avec SQL

- **Ni jointure ni clés étrangères**
⇒ nécessité de dénormaliser le schéma relationnel
- **Pas de GROUP BY**
⇒ doit être géré coté client (via traitement Map-Reduce)
- **Clause WHERE limitée** : pas de fonction de conversion ni de recherche contextuelle like %
- **Clause ORDER BY limitée**
- **Pas de contrainte d'intégrité/d'unicité** :
PRIMARY KEY précise juste la clé de partition de la table, mais aucune vérification n'est faite sur son unicité
- **Les clauses INSERT et UPDATE sont équivalentes**
- **Valeur null ou vide non matérialisée.**

Quels outils pour l'utiliser ?

- via un interpréteur de requête dédié cqlsh
- via un driver faisant l'interface entre CQL et un langage de programmation

Driver CQL par langage

	Datastax Java driver , Achilles, Astyanax, Casser, Kundera, PlayORM
	Datastax Python driver
	Datastax Ruby driver
	Datastax C# driver , Cassandra Sharp, Fluent Cassandra
	Datastax Nodejs driver , Node-Cassandra-CQL
	Datastax PHP driver , CQL PHP, PHP-Cassandra
	Datastax C++ driver , libQTcassandra
	Datastax Spark connector , Phantom, Quill
	Alia, Cassaforte, Hayt
	CQerl, Erlcass
	CQLc, Gocassa, GoCQL
	Cassy
	Rust CQL
	Cassandra : Client/DBD : Cassandra