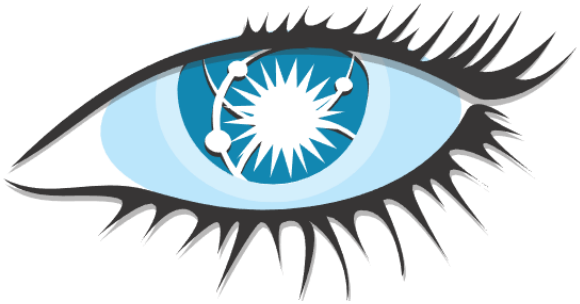
Cassandra et les bases de données NoSQL : architecture, performances et cas d’usage

Analyser les principes de Cassandra, ses avantages par rapport aux bases relationnelles, et ses cas d’application dans le Big Data.

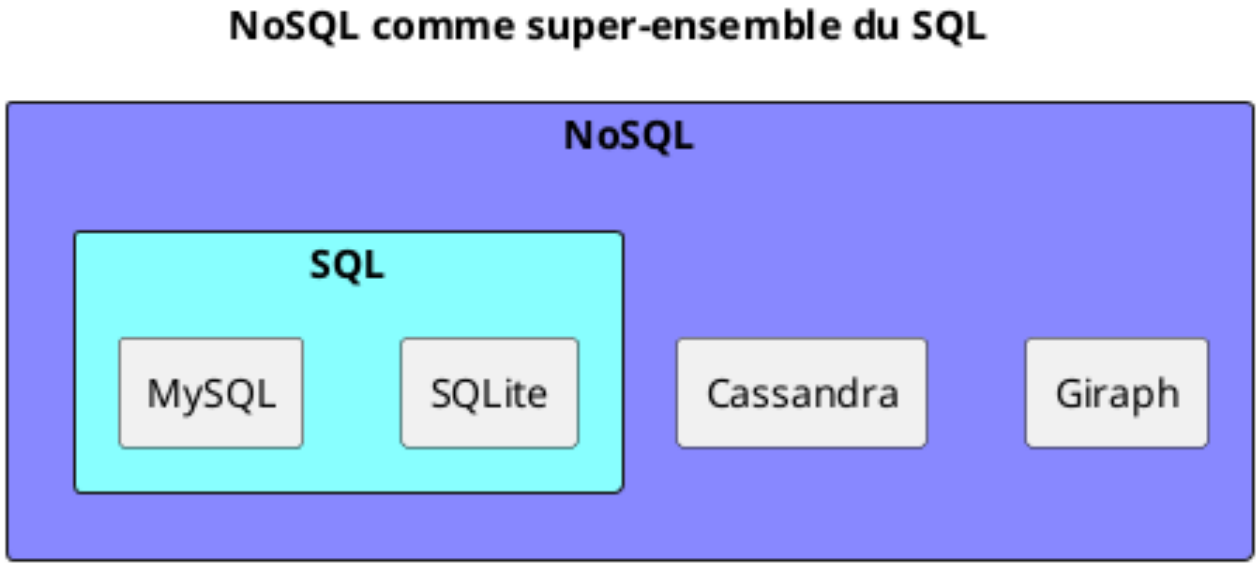
Matthieu Jolimaitre



Sommaire

Introduction aux bases NoSQL

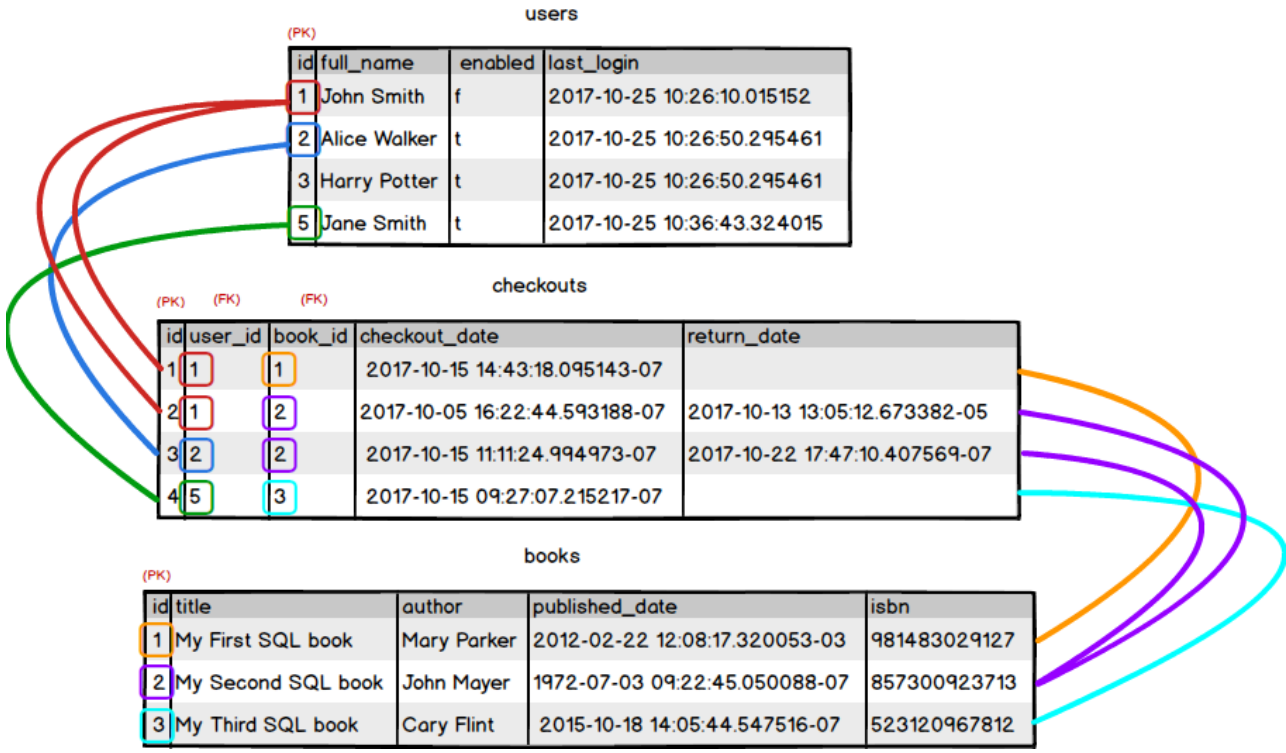
> Not Only SQL



Le SQL

Un modèle de donnée appuyé sur des tableaux et des scripts de modification.

```
INSERT INTO checkouts
  (id, user_id, book_id, checkout_date, return_date)
VALUES
  (1, 1, 1, '2017-10-15 14:43:18', NULL),
  (2, 1, 2, '2017-10-05 16:22:44', '2017-10-13 13:00:12'),
  (3, 2, 2, '2017-10-15 11:11:24', '2017-10-22 17:47:10'),
  (4, 5, 3, '2017-10-15 09:27:07', NULL);
```

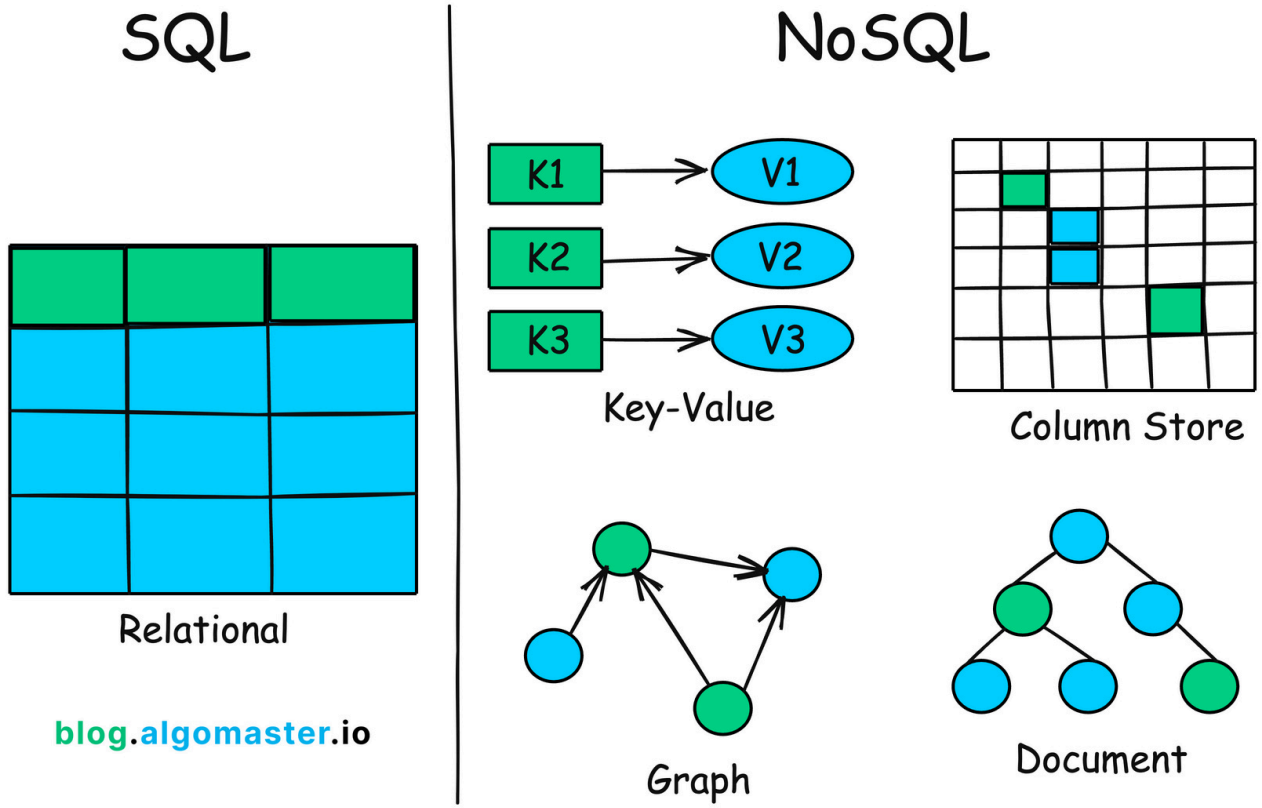


Pourquoi NoSQL ?

- Utiliser de nouvelles topologies.
  - Fournir des interfaces de modification plus adaptés à la donnée.
- Se détacher du modèle de données arrangées en tables.
  - Modéliser des états plus complexes.
  - Optimiser en fonction des données.
- Autoriser des transgressions du principe ACID.
  - Atomicité.
  - Consistance.
  - Isolation.
  - Durabilité.
- Apparition de nouveaux principes BASE.
  - Basiquement Disponible.
  - Souplesse de l'état.
  - Éventuellement consistent.

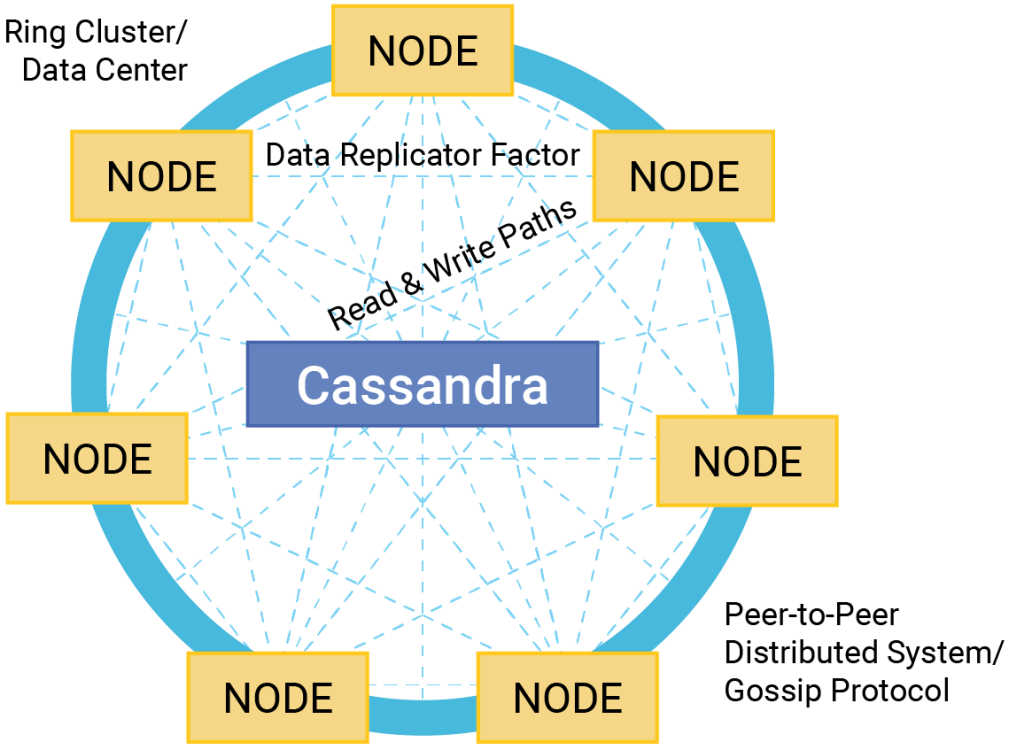
Catégories

- **clé-valeur** : Les données sont des valeurs accessibles par leur clé (souvent un texte).
- **document** : Clé-valeur optimisé pour les données structurées.
- **colonne larges** : Les données sont mises en tableau, mais le schéma d'un même tableau peut varier selon les lignes.
- **graphe** : Les données sont arrangées dans des nœuds pouvant être référencés dans d'autres nœuds.



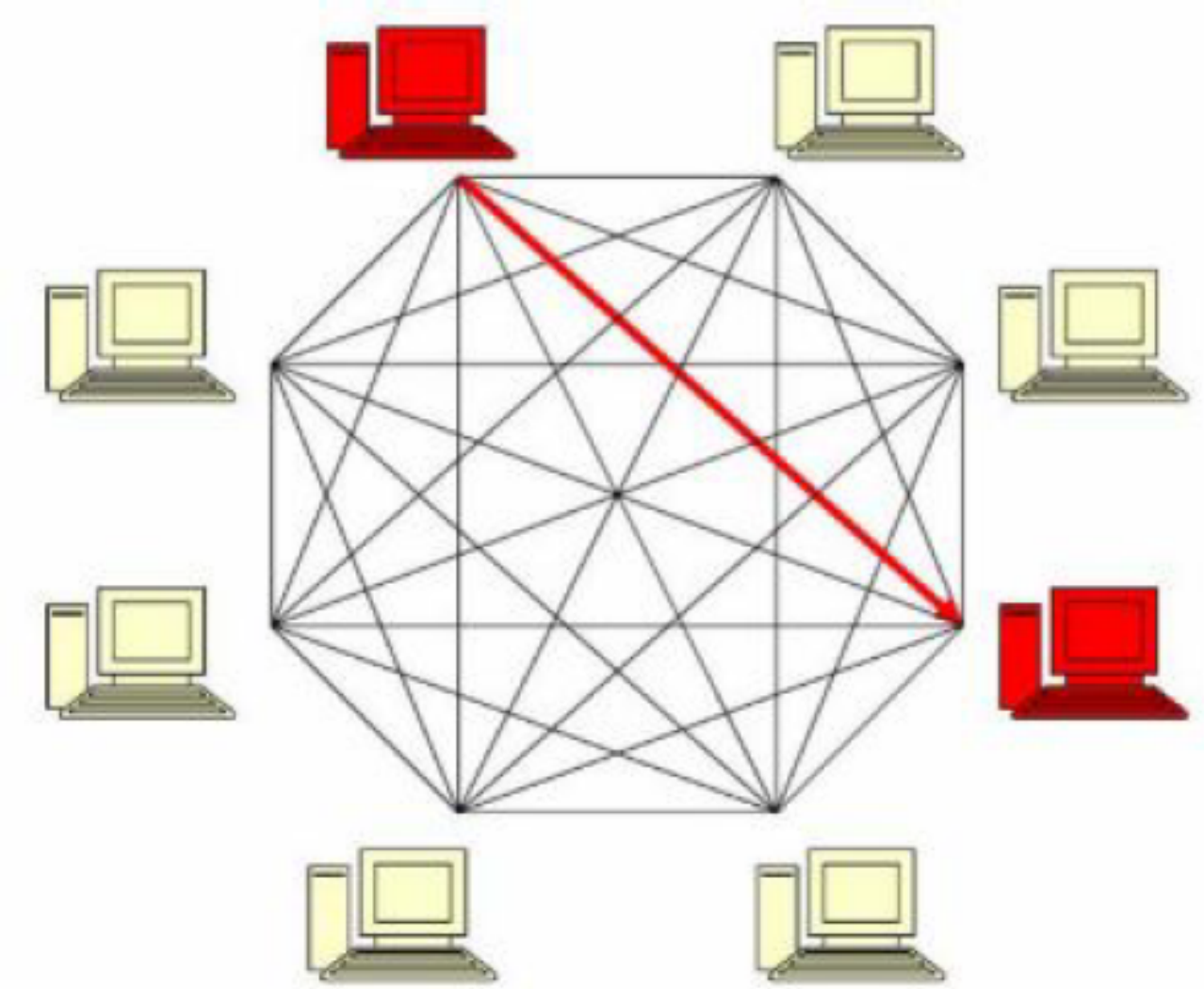
Architecture de Cassandra

- Suit un modèle de donnée en **Large colonnes**.
- **Distribuée** : Les données sont répliquées sur plusieurs 'nœuds' pour permettre une expansion horizontale et pour se rendre résilient aux pannes.
- Garanti une **consistance éventuelle**.



Peer-to-peer, partitionnement, réplication

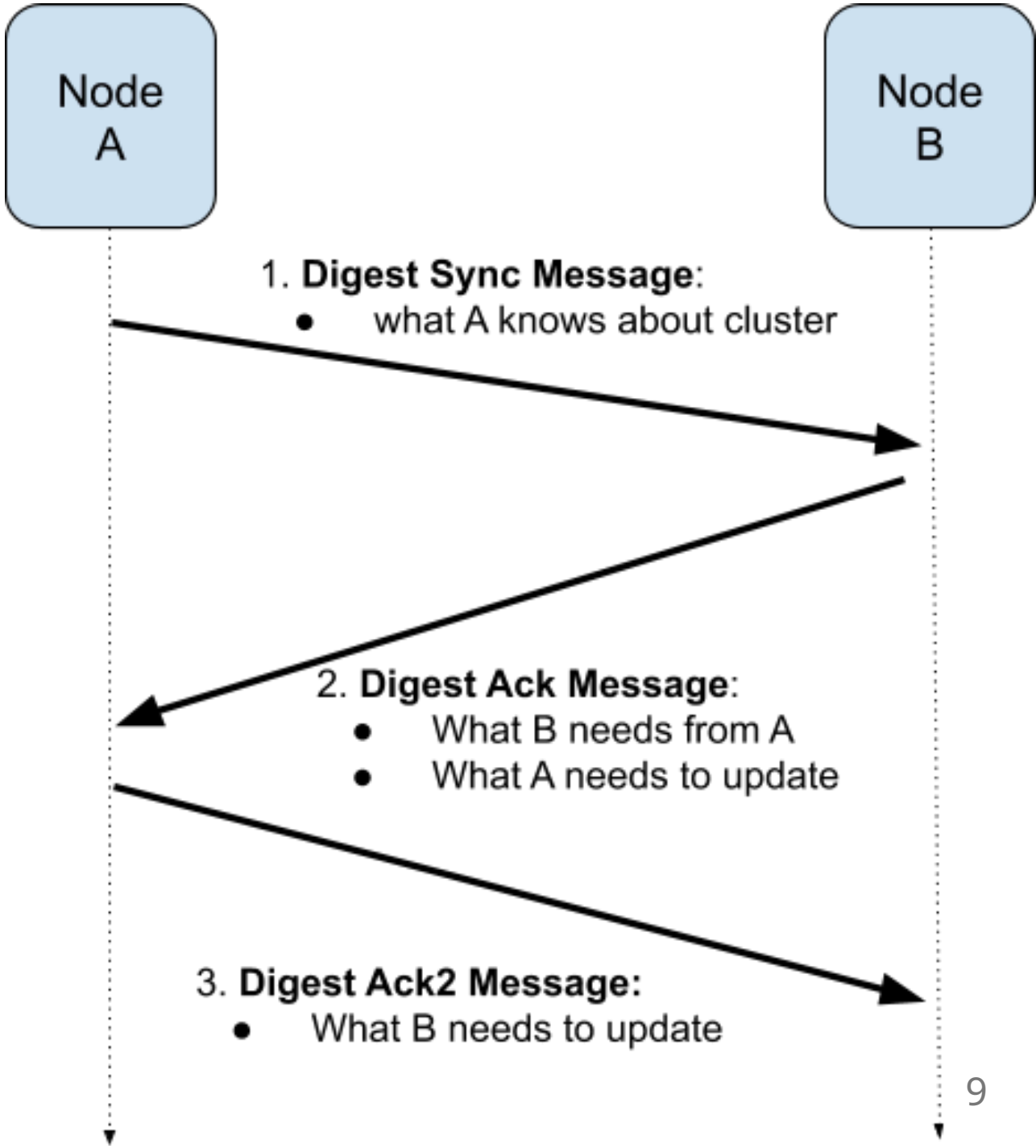
Différentes techniques pour répartir et synchroniser les données dans un système distribué.



pair-à-pair

Pour synchroniser les changements de données, **la grappe n’utilise pas un nœud autoritaire** orchestrant la synchronisation.

À la place, les nœuds communiquent périodiquement par deux, sans hiérarchie, en utilisant un **protocole de 'potins'**.

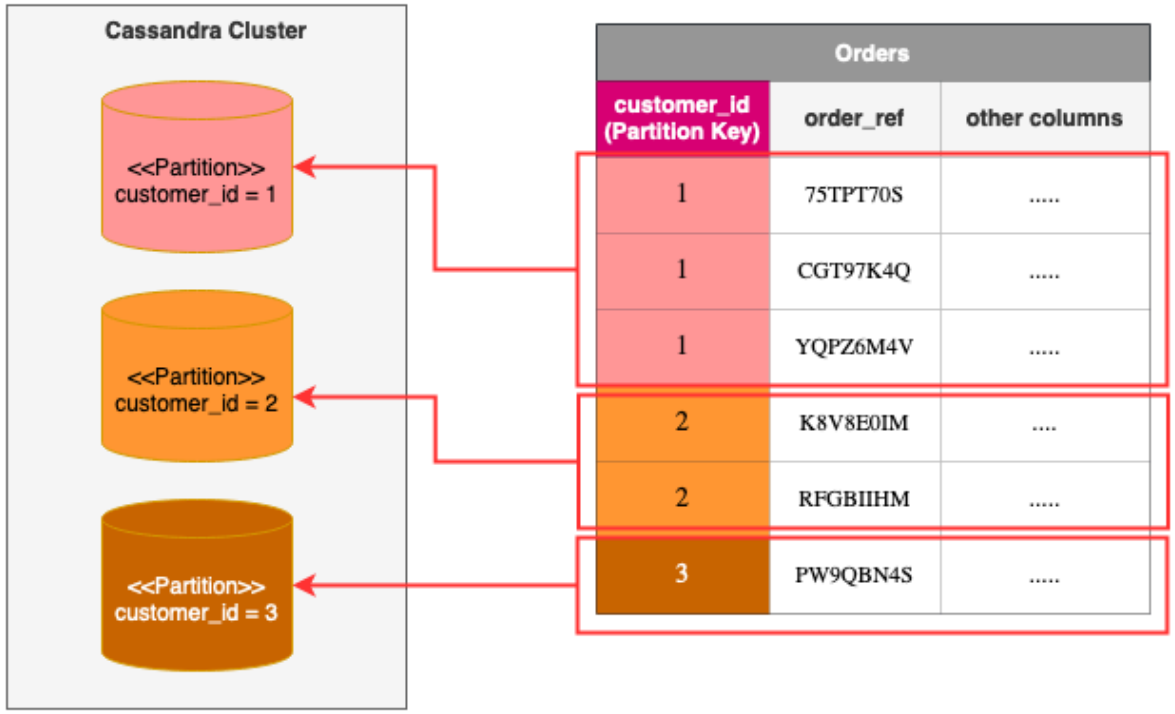


Partitionnement

Pour réduire la charge d'une requête sur la grappe, les données sont partitionnées entre les différents nœuds.

C'est à dire qu'**une donnée n'apparaitra que sur 'certains' nœuds** de la grappe, et qu'une requête accédant à cette donnée devra être traitée par un de ces nœuds.

Pour une donnée partitionnée, l'adresse du nœud contenant une ligne est comprise **dans sa clé primaire**.



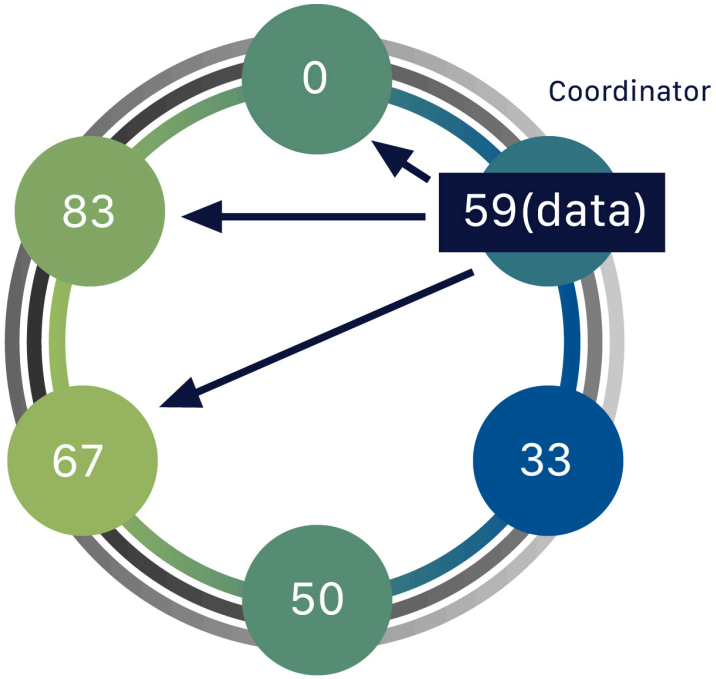
Réplication

Pour diminuer le **risque de perte** de donnée et augmenter la **disponibilité** d'une donnée, la réplication sert à contenir la même donnée dans plusieurs nœuds.

Concrètement, les partitions ont chacune un **facteur de réplication** RF, et **une partition est copiée vers RF nœuds** différents à partir d'un nœud responsable de cette partition.

Ce système est implémenté avec un mécanisme de jeton.

RF = 3



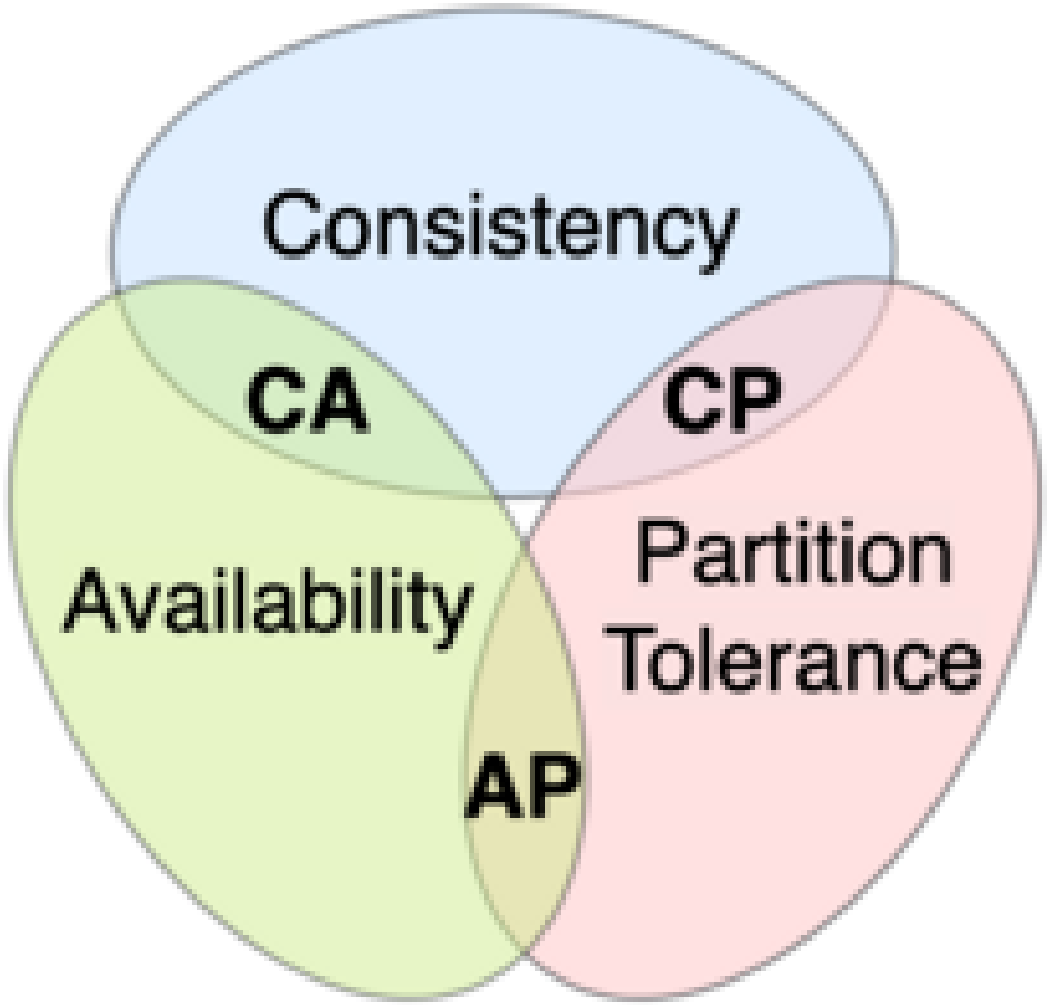
Consistency levels et CAP theorem

Les bases de données distribués ont trois aspects propres :

- **Consistance** : Toutes les lectures d'une même donnée retournent la version la plus récente de cette donnée.
- **Disponibilité** : Maximiser le taux de réponse aux requêtes saines.
- **Tolérance au partitionnement** : La dégradation de certains nœuds n'entraine pas une dégradation du système.

Consistency levels et CAP theorem

Le 'théorème de CAP' énonce qu’une base de donnée distribuée **doit se focaliser sur deux aspects**, car il est impossible d’en atteindre deux sans en sacrifier un.

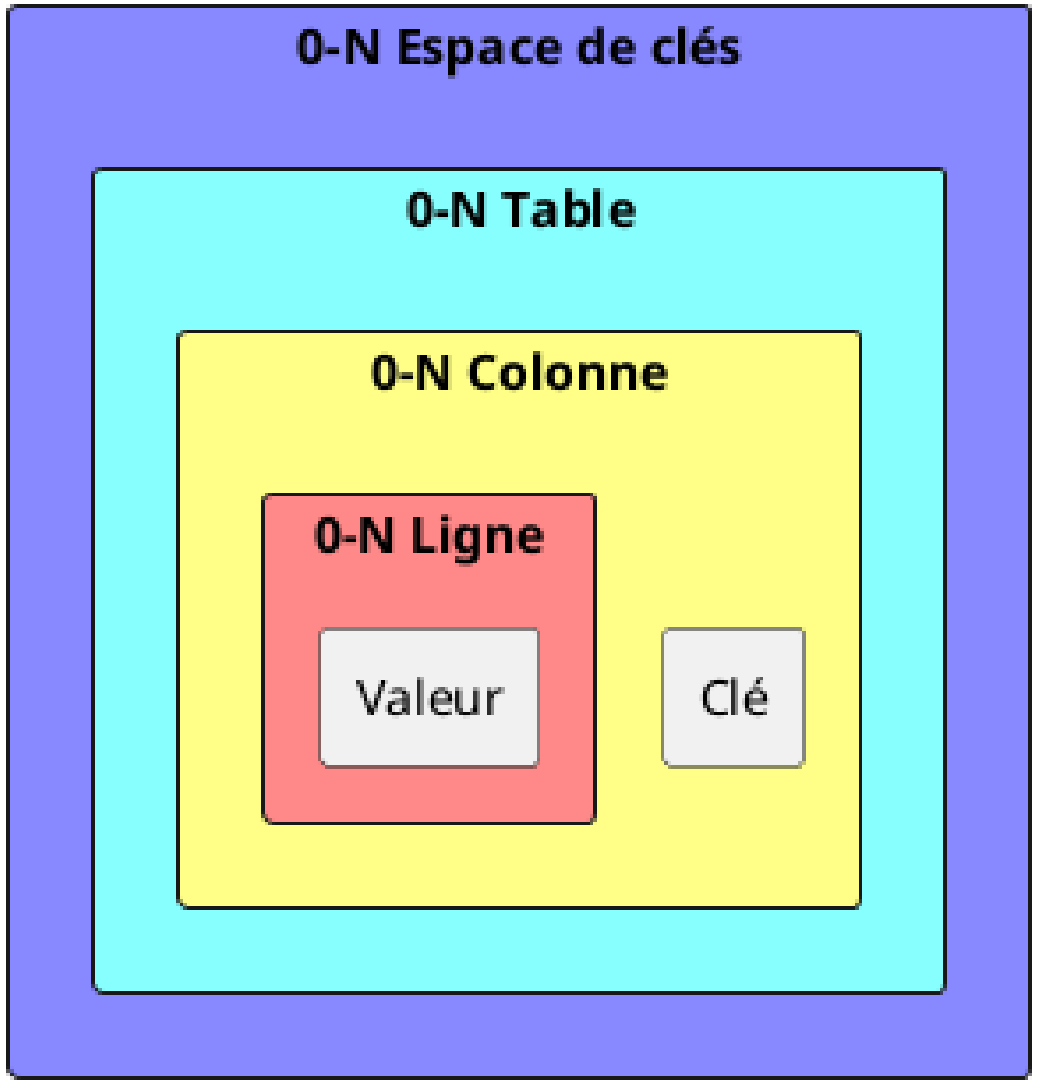


Modèle de données et requêtage

Cassandra suit un modèle de **tables à colonnes larges**, les colonnes sont de taille et structure dynamique : Le même tableau peut avoir des lignes de colonnes différentes.

Cela augmente la **flexibilité de modélisation** et diminue les **efforts de mise à jour du schéma** d'un tableau.

Modèle de donnée de Cassandra : Large colonne



Tables dénormalisées, CQL (Cassandra Query Language)

Le modèle de donnée est normalisé lorsqu'il réponds à plusieurs exigences :

- **UNF** : Aucune colonne n'est dupliquée.
- **1NF** : Les colonnes doivent contenir des valeurs primitives, et non des agrégats.
- **2NF** : Chaque colonne d’un tableau dépends de toutes les clés primaires de ce tableau.
- **3NF** : Les colonnes ne dépendent uniquement des clés primaires de leur tableau.

Normalisé

USERS BY EMAIL	
email	user_id
b@o.b	1
c@r.l	2

USERS		
id	name	age
1	bob	19
2	charlie	23

Dénormalisé

USERS BY EMAIL			
id	email	name	user_id
1	b@o.b	bob	1
2	c@r.l	charlie	2

USERS		
id	name	age
1	bob	19
2	charlie	23

- La dénormalisation :
  - Réduit le nombre d’opération par lecture.
  - Augmente le nombre d’opération par écriture.

Normalisé

USERS BY EMAIL

email	user_id
b@o.b	1
c@r.l	2

USERS

id	name	age
1	bob	19
2	charlie	23

Dénormalisé

USERS BY EMAIL

id	email	name	user_id
1	b@o.b	bob	1
2	c@r.l	charlie	2

USERS

id	name	age
1	bob	19
2	charlie	23

Cassandra Query Language : Un DSL pour Cassandra

```
/* Create a new keyspace in CQL */
CREATE KEYSPACE myDatabase WITH replication =
   {'class': 'SimpleStrategy', 'replication_factor': 1};

/* Create a new database in SQL */
CREATE DATABASE myDatabase;
```

```
/* Expiring Data in 24 Hours */
INSERT INTO myTable (id, myField) VALUES (2, 9) USING TTL 86400;
```

- Les écritures sont peu coûteuses, donc il est encouragé de créer des tables redondantes plutôt que des requêtes complexes.
    - Le filtrage est désactivé par défaut.

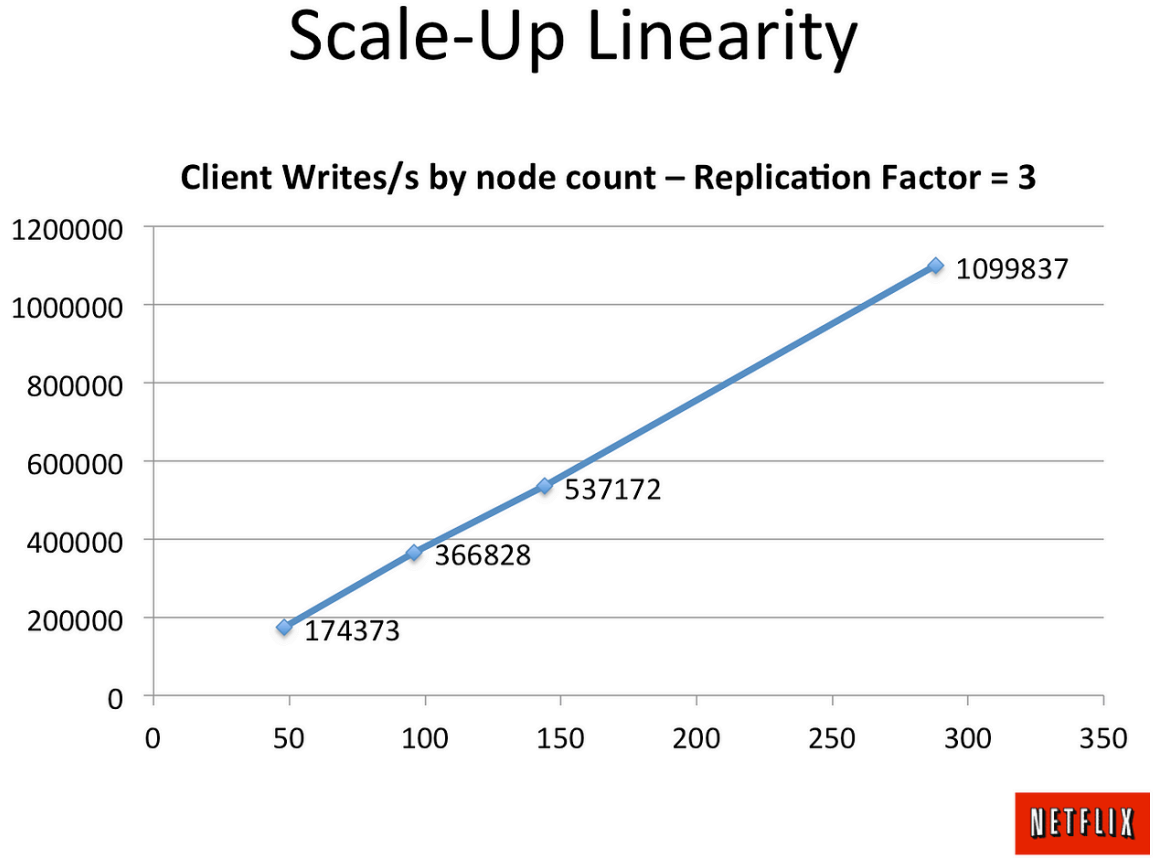
Performances et cas d’usage

- Distribué
  - **Répliqué** : Forte résilience.
  - **Partitionné** : Bonne mise à l’échelle.
- **NoSQL** : Flexibilité.
  - Modéliser sa donnée avec moins de contraintes.
  - Adapter les schémas pour améliorer les performances.

Écritures rapides, scalabilité horizontale

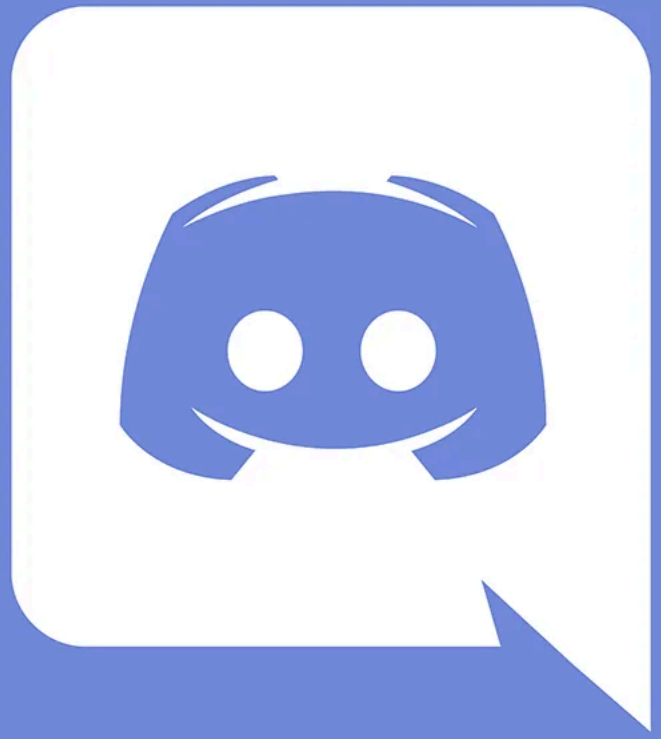
Cassandra est une base de donnée optimisée pour la **mise à l'échelle**, c'est à dire offrir une grande disponibilité au détriment de la **consistance forte des données**.

Donc elle est recommandée dans les cas où il y a une haute contention d'écritures, et où la consistance de la donnée n'est pas cruciale.



Cas d’étude : Discord

> July, we announced 40 million messages a day, in December we announced 100 million. How do we do it? Cassandra!
>
> https://discord.com/blog/how-discord-stores-billions-of-messages



Requirements

> We defined our requirements:
>
> - **Linear scalability** : We do not want to reconsider the solution later or manually re-shard the data.
> - **Low maintenance** : It should just work once we set it up. We should only have to add more nodes as data grows.
> - **Predictable performance** : We have alerts go off when our API's response time 95th percentile goes above 80ms. We also do not want to have to cache messages in Redis or Memcached.
> - **Not a blob store** : Writing thousands of messages per second would not work great if we had to constantly deserialize blobs and append to them.
>
> Cassandra was the only database that fulfilled all of our requirements.

Distribution

> Here is a simplified schema for our messages table (this omits about 10 columns).

```
CREATE TABLE messages (
  channel_id bigint, bucket int, message_id bigint, author_id bigint, content text,
     PRIMARY KEY ((channel_id, bucket), message_id)
) WITH CLUSTERING ORDER BY (message_id DESC);
```

- Nous notons une clé de partitionnement composite, utilisant l'ID d'un salon ET un nombre arbitraire pour contrôler la taille des partitions.

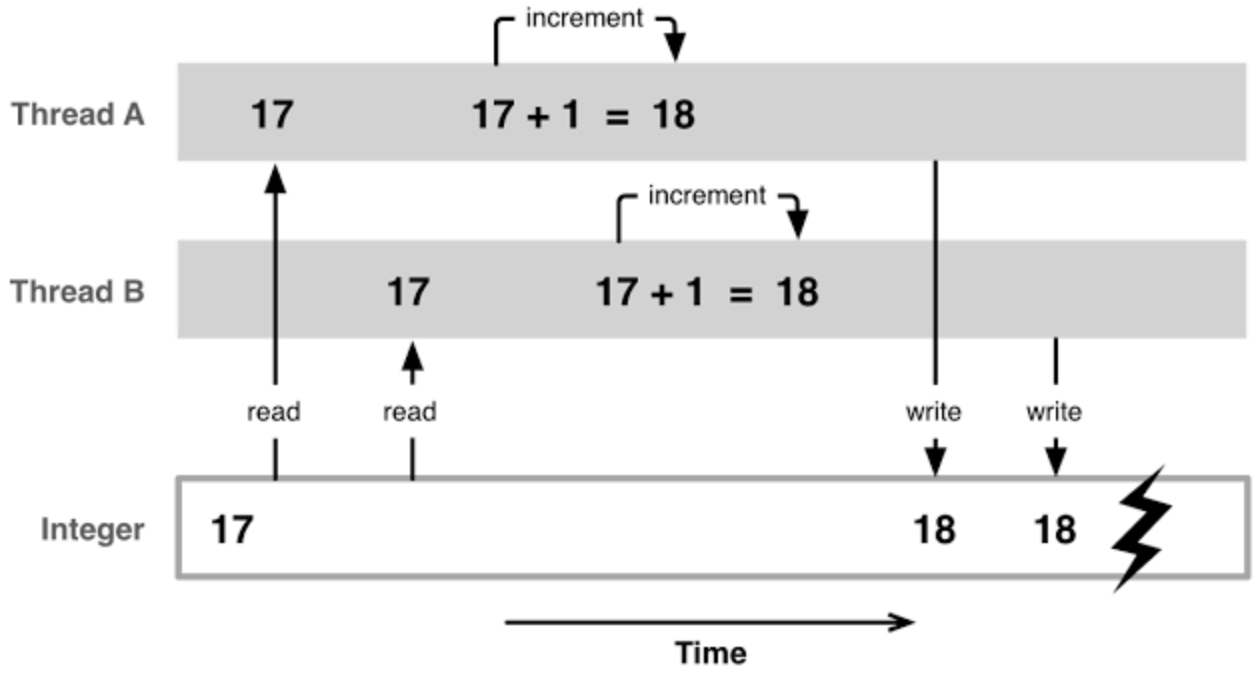
Résultats

> At the beginning of 2022, it had 177 nodes with trillions of messages

Limites et comparaison avec d’autres NoSQL

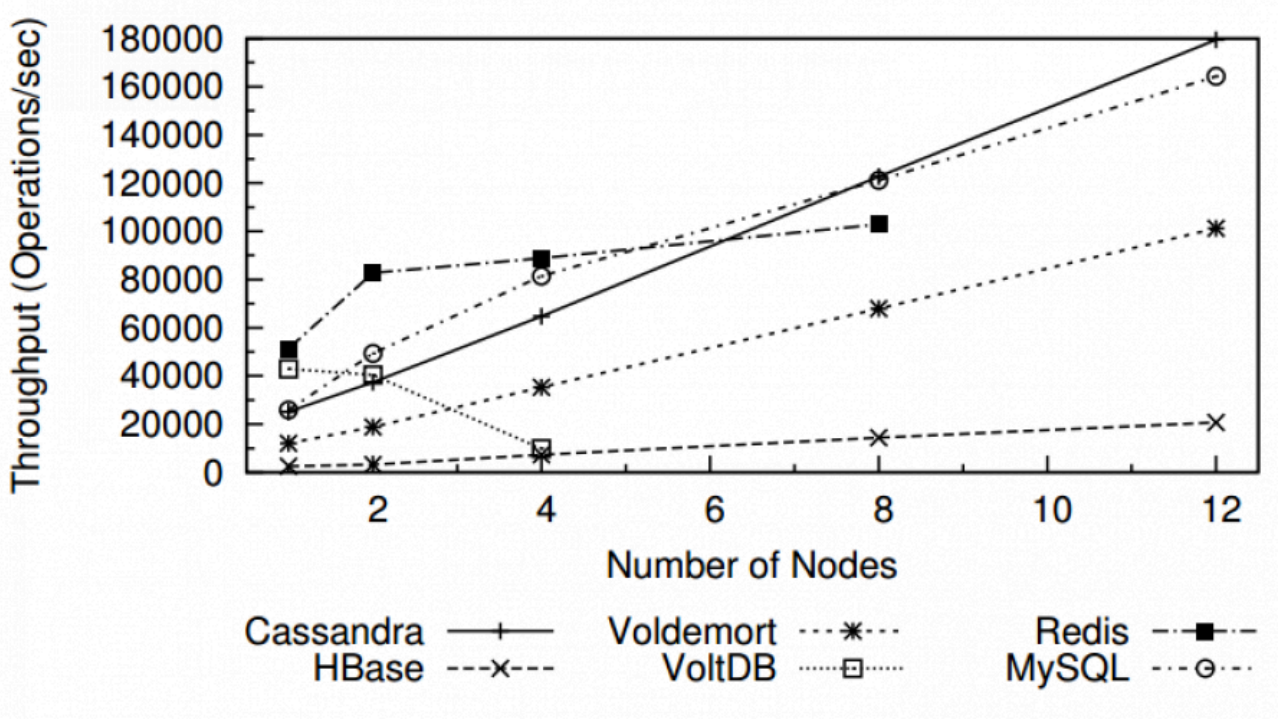
Faible consistance

- Une **consistance éventuelle** entraine un risque de **courses de données** pouvant rendre l'état du service invalide.



Coût de la mise à l'échelle

- Les performances sur un seul noeud sont relativement basses.



Fin