

# Maîtrisez les bases de données NoSQL

🕒 15 hours 📶 Medium

License 

Last updated on 6/4/21



## Maîtrisez le théorème de CAP



### Propriétés fondamentales d'une base de données relationnelle

Et pourquoi donc faire encore de la base de données relationnelle ? Le NoSQL est très tentant lorsque l'on voit toutes les possibilités qu'il peut offrir. Mais il ne faut pas négliger certains fondamentaux qui rendent les SGBDR incontournables.

Cela se résume dans une combinaison qu'il ne faut pas négliger :

1. Faire des jointures entre les tables de la base de données
2. Faire des requêtes complexes avec un langage de haut niveau sans se préoccuper des couches basses
3. Gérer l'intégrité des données de manière infaillible



Et c'est surtout ce dernier point qui attire l'attention dans le cadre du NoSQL. Gérer l'intégrité des données veut dire être capable de garantir le contenu de la base de données, quelles que soient les mises à jour effectuées sur les données, ainsi que sa pérennité. Nous allons voir que la gestion de la cohérence d'une donnée n'est pas forcément garantie.


### ACID vs BASE


Pour mettre en rapport les problématiques de la base de données relationnelle, on parle de propriétés **ACID** pour les transactions (séquences d'opérations/requêtes) :

- **Atomicité** : Une transaction s'effectue entièrement ou pas du tout
- **Cohérence** : Le contenu d'une base doit être cohérent au début et à la fin d'une transaction
- **Isolation** : Les modifications d'une transaction ne sont visibles/modifiables que quand celle-ci a été validée
- **Durabilité** : Une fois la transaction validée, l'état de la base est permanent (non affecté par les pannes ou autre)

Toutefois, ces propriétés ne sont pas applicables dans un contexte distribué tel que le NoSQL. En effet, prenons l'exemple d'une transaction de cinq opérations (lecture/écriture) : cela implique une synchronisation entre cinq serveurs pour garantir l'atomicité, la cohérence et l'isolation. Au final, cela se traduit par des latences dans les transactions (en cours et en concurrence). Ce qui n'est pas tolérable lorsque justement on veut éviter ces


 **Immergez vos données dans le NoSQL** 

 1. Choisissez votre famille NoSQL


 **2. Maîtrisez le théorème de CAP**

3. Passez à l'échelle


4. Choisissez votre base de données NoSQL

 Quiz: Savez-vous vraiment ce qu'est le NoSQL ?




**Created by**

  
CentraleSupélec

Grande École d'ingénieurs :  
cycle ingénieur, Master et  
École Doctorale, Mastère  
Spécialisé et formation  
continue



OpenClassrooms, Leading  
E-Learning Platform in  
Europe

latences en distribuant les calculs.

Le problème s'aggrave encore lorsque l'on distribue les données car il va falloir répliquer chaque donnée. Pourquoi ? Tout simplement parce que si un serveur tombe en panne, il faut pouvoir garantir de retrouver toutes les données présentes sur ce serveur, donc on fait de la réplication. Mais cela veut dire également qu'il va falloir synchroniser toutes mises à jour avec tous les réplicas de la donnée !



Du coup, les propriétés **BASE** ont été proposées pour caractériser les bases NoSQL :

- **Basically Available** : quelle que soit la charge de la base de données (données/requêtes), le système garantie un taux de disponibilité de la donnée
- **Soft-state** : La base peut changer lors des mises à jour ou lors d'ajout/suppression de serveurs. La base NoSQL n'a pas à être cohérente à tout instant
- **Eventually consistent** : À terme, la base atteindra un état cohérent

Ainsi, une base NoSQL relâche certaines contraintes, telles que la synchronisation des réplicas, pour favoriser l'efficacité. Le parallèle ACID / BASE repris du domaine de la chimie permet d'appuyer là où ça fait mal : la concurrence. L'*enfer* des transactions gérées par les bases de données relationnelles est transformé en *paradis* pour le temps de réponse en relâchant cette contrainte impossible à maintenir.

## Théorème de Brewer dit "théorème de CAP" ▼

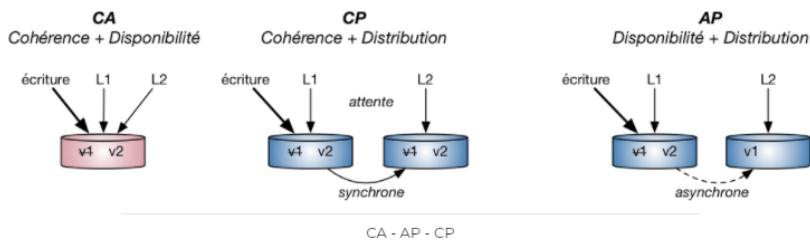
En 2000, [Eric A. Brewer](#) a formalisé un théorème très intéressant reposant sur 3 propriétés fondamentales pour caractériser les bases de données (relationnelles, NoSQL et autres) :

1. **Consistency** (Cohérence) : Une donnée n'a qu'un seul état visible quel que soit le nombre de réplicas
2. **Availability** (Disponibilité) : Tant que le système tourne (distribué ou non), la donnée doit être disponible
3. **Partition Tolerance** (Distribution) : Quel que soit le nombre de serveurs, toute requête doit fournir un résultat correct

Le **théorème de CAP** dit :

Dans toute base de données, vous ne pouvez respecter au plus que 2 propriétés parmi la *cohérence*, la *disponibilité* et la *distribution*.

Cela s'illustre assez facilement avec les bases de données relationnelles, elles gèrent la cohérence et la disponibilité, mais pas la distribution.



Prenons le couple **CA** (Consistency-Availability), il représente le fait que lors d'opérations concurrentes sur une même donnée, les requêtes L1 et L2 retournent la nouvelle version (v2) et sans délai d'attente. Cette combinaison n'est possible que dans le cadre de bases de données [transactionnelles](#) telles que les SGBDR.

Le couple **CP** (Consistency-Partition Tolerance) propose maintenant de distribuer les données sur plusieurs serveurs en garantissant la tolérance aux pannes (réplication). En même temps, il est nécessaire de vérifier la cohérence des données en garantissant la valeur retournée malgré des mises à jour concurrentielles. La gestion de cette cohérence nécessite un protocole de synchronisation des réplicas, introduisant des délais de latence dans les temps de réponse (L1 et L2 attendent la synchronisation pour voir v2). C'est le cas de la base NoSQL *MongoDB*.

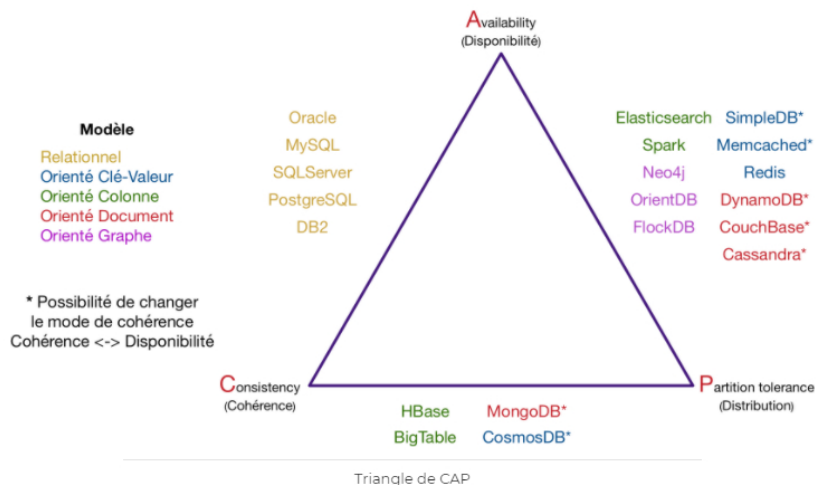
Le couple **AP** (Availability-Partition Tolerance) à contrario s'intéresse à fournir un temps de réponse rapide tout en distribuant les données et les réplicas. De fait, les mises à jour sont

asynchrones sur le réseau, et la donnée est "*Eventually Consistent*" (L1 voit la version v2, tandis que L2 voit la version v1). C'est le cas de *Cassandra* dont les temps de réponses sont appréciables, mais le résultat n'est pas garanti à 100% lorsque le nombre de mises à jour simultanées devient important.

Ainsi, la cohérence des données est incompatible avec la disponibilité dans un contexte distribué comme les bases NoSQL.

## Le triangle de CAP et les bases de données

Grâce à ce théorème de CAP, il est alors possible de classer toutes les bases de données en les plaçant sur le "triangle de CAP", tout en ajoutant des codes couleurs pour chaque modèle de stockage présenté dans le chapitre précédent.



Nous pouvons constater que les bases de données relationnelles se retrouvent sur la face CA du triangle, combinant disponibilité et cohérence. Nous retrouvons bien MongoDB pour le couple CP (cohérence et distribution) mais également les solutions orientées colonnes comme HBase ou BigTable.

**i** Cassandra est bien une solution orientée document.

Je vous invite à lire **The Definitive Guide - 2nd Edition** de Juin 2016, à la page 37.

So Cassandra is not really column-oriented, in that its data store is not organized primarily around columns.

Le couple AP (Disponibilité et distribution) regroupe le plus grand nombre de solutions NoSQL. En effet, la plupart cherchent les performances en relâchant volontairement la cohérence. Nous y retrouvons principalement des solutions orientées clé-valeur et graphes, mais également orientées documents (clé-valeur étendu).

Certaines solutions proposent également de modifier la politique de gestion de la concurrence (DynamoDB, CouchBase, Cassandra, MongoDB, CosmosDB...), dans ce cas, ils changent simplement de face sur ce triangle en passant de CP à AP.

L'avantage de ce triangle CAP est de fournir un critère de choix pour votre application. Vous pouvez ainsi vous reposer sur vos besoins en terme de fonctionnalité, de calculs et de cohérence. Le triangle servira de filtre sur les myriades de solutions proposées.

I FINISHED THIS CHAPTER. ONTO THE NEXT!

◀ CHOISISSEZ VOTRE FAMILLE NOSQL

PASSEZ À L'ÉCHELLE ▶

### Teachers



**Régis Behmo**

Expert en machine learning, développeur fullstack, grimpeur invétéré et gros, très gros amateur de nouilles chinoises.



**Nicolas Travers**

Maitre de Conférences en informatique au CNAM Spécialités : Bases de données, Optimisation BDD, SQL,

OPENCCLASSROOMS

What we do

Apprenticeship

Path experience

Our blog

BUSINESS SOLUTIONS

Business

CONTACT



FAQ

LEARN MORE

Work at OpenClassrooms


Become a mentor [↗](#)

Our store

Terms of use

Privacy policy

Accessibility

 English

