Chapitre 3

Base de données NoSQL

Introduction

Plusieurs modèles de stockage de l'information sont explorés à la naissance de l'informatique. Mais c'est le modèle relationnel qui a eu le dessus en 1970 grâce à un modèle théorique puissant et simple qui permet d'assurer au mieux l'intégrité des données . Les bases de données NoSQL, signifiant « Not only SQL » ont commencé à émerger depuis 2009, notamment par les géants du WEB pour répondre aux nouveaux besoins en performance lors de traitement de gros volumes de données ,Ce dernier ne vient pas remplacer les bases de données relationnelles mais proposer une alternative ou compléter les fonctionnalités des SGBDs relationnels pour donner des solutions plus intéressantes dans certains contextes. En effet, le langage SQL a atteint ses limites dans certaines situations que nous verrons plus en détail dans ce chapitre. Tout au long de ce chapitre nous allons présenter d'une part les SGBDR, leurs contraintes, leurs points forts ainsi que leurs limites qui ont été le point de départ du mouvement NoSQL que nous allons aussi détailler. D'autre part, la comparaison entre ces deux notions s'impose afin de mieux comprendre l'apport du Nosql par rapport au SQL. Enfin, nous achevons ce chapitre en citant quelques exemples des bases de données NoSQL les plus utilisées avec leurs caractéristiques et avantages.

3.1 Systèmes de gestion de base de données relationnels

Les bases de données ont pris aujourd'hui une place essentielle dans l'informatique, plus particulièrement en gestion. Au cours des trente dernières années, des concepts, méthodes et algorithmes ont été développés pour gérer des données sur mémoires secondaires ; Ils constituent l'essentiel de la discipline « Bases de Données ». Cette discipline est utilisée dans de nombreuses applications. Il existe un grand nombre de Systèmes de Gestion de Bases de données (SGBD) qui permettent de gérer efficacement de grandes bases de données. Les bases de données constituent donc une discipline s'appuyant sur une théorie solide et offrant de nombreux débouchés pratiques. Les SGBD ont bientôt quarante ans d'histoire. Les années 60 ont connu un premier développement des bases de données sous forme de fichiers reliés par des pointeurs. Les fichiers sont composés d'articles stockés les uns à la suite des autres et accessibles par des valeurs de données appelées clés. Les systèmes IDS.I et IMS.I développés respectivement à Honeywell et à IBM vers 1965 pour les programmes de conquête spatiale, notamment pour le programme APOLLO qui a permis d'envoyer un homme sur la lune, sont les précurseurs des SGBD modernes. Ils permettent de constituer des chaînes d'articles entre fichiers et de parcourir ces chaînes.

Un système de gestion de base de données relationnels est un type particulier des

SGBD base sur le modèle relationnel introduit par Edgar Frank Codd Il permet en particulier de synthétiser n'importe quel lot d'informations en exploitant le contenu des différentes tables de la base de données par application des opérations de l'algèbre relationnelle telles que la jointure, la sélection et la projection.

3.1.1 Les bases de données relationnelle

Une base de données relationnelle est un type de base de données où les données sont liées à d'autres informations au sein des bases de données. Les bases de données relationnelles sont composées d'un ensemble de tables qui peuvent être accessibles et reconstruites de différentes manières, sans qu'il soit nécessaire de réarranger ces tables de quelque façon que ce soit. Le langage de requête structuré (SQL) est l'interface standard pour une base de données relationnelle. Les instructions SQL sont utilisées à la fois pour interroger de façon interactive les données contenues dans la base de données relationnelle et pour collecter les données dans le cadre de rapports.

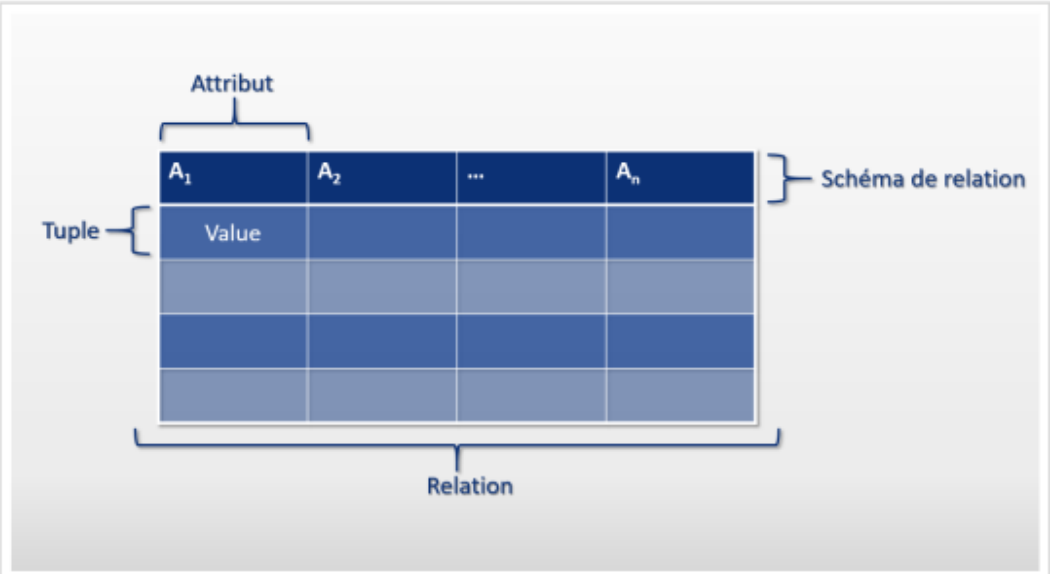

FIGURE 3.1 – Elements d'une table d'une BDDR.

3.1.2 Les règles CODD

Les douze règles de Codd (numérotées de 0 à 12, donc en fait treize...) formalisent la notion de SGBD relationnel. C'est le docteur Codd, père fondateur de l'algèbre relationnelle qui grave ainsi dans le marbre les 13 commandements de la Loi pour tout SGBD se prétendant relationnel. Ces règles sont basées sur ses travaux originaux effectués à partir de 1970, et ont été publiées dans deux articles de vulgarisation du magazine Computerworld (octobre 1985) : Is your DBMS really relational ? et Does your DBMS run by the rules ? Ces règles constituent les fondements des bases de données relationnelles et permettent de connaître le niveau relationnel du produit d'un éditeur. Ne pas s'y contraindre implique plus d'inconvénients que d'avantages. En un sens elles gravent dans le marbre "la Loi" que tout bon SGBD relationnel devrait suivre [40] :

— **Règle 0 :** Toutes les fonctionnalités du SGBDR doivent être disponibles à travers le modèle relationnel et le langage d'interrogation.

— **Unicité :** Toutes les informations de la base de données sont représentées d'une et d'une seule façon, c'est-à-dire par la valeur dans le champ de la

colonne de table.

— **Garantie d'accès :** Toutes les données doivent être clairement accessibles. La règle est essentiellement un ajustement des conditions de base de la clé primaire. Cela signifie qu'en spécifiant le nom de la table contenant, le nom de la colonne contenant et la valeur principale de la ligne contenante, chaque valeur scalaire dans la base de données doit être logiquement accessible.

— **Traitement des valeurs nulles :** Le système de gestion de la base de données doit permettre à chaque champ de rester nul (ou vide). Plus précisément, il doit prendre en charge la représentation systématique des « informations manquantes et des informations inappropriées » différentes de toutes les valeurs conventionnelles (par exemple, pour les valeurs numériques, « différentes de zéro ou de tout autre nombre »), indépendamment de Quel est le type de données. Cela implique également que ces représentations doivent être gérées par le système de gestion de base de données de manière systématique.

— **Catalogue lui-même relationnel :** Le système doit prendre en charge un catalogue en ligne intégré et associé auquel les utilisateurs autorisés peuvent accéder via des langages de requête réguliers. Par conséquent, l'utilisateur doit pouvoir accéder à la structure de la base de données (répertoire) en utilisant le même langage de requête que les données de la base de données.

— **Sous-langage de données :** Le SGBDR doit implémenter un langage relationnel qui supporte la manipulation des données et Métadonnées, définir les contraintes de sécurité et gérer les transactions

— **Mise à jour des vues :** En théorie, toutes les vues pouvant être mises à jour doivent pouvoir être mises à jour par le système.

— **Insertion, mise à jour, et effacement de haut niveau :** Le système doit prendre en charge les opérations par lots pour l'insertion, la mise à jour et la suppression. Cela signifie que les données peuvent être extraites d'une base de données relationnelle et que l'ensemble de données se compose de données provenant de plusieurs tuples et / ou de plusieurs tables. Cette règle stipule que les tuples par lots dans plusieurs tables et les tuples uniques dans une seule table doivent prendre en charge les opérations d'insertion, de mise à jour et de suppression.

— **Indépendance physique :** Les modifications au niveau physique (comment les données sont stockées, qu'il s'agisse de lignes ou de listes liées, etc.) ne nécessitent pas de modifications des applications basées sur la structure.

— **Indépendance logique :** Les modifications logiques (tableaux, colonnes, lignes, etc.) n'ont pas besoin d'être modifiées dans les applications basées sur la structure. L'indépendance des données logiques est plus difficile à atteindre que l'indépendance des données physiques.

— **Indépendance d'intégrité :** Les contraintes d'intégrité doivent être indiquées séparément de l'application et stockées dans le répertoire. Au fil du temps, il devrait être possible de modifier ces contraintes sans affecter inutilement les applications existantes.

— **Indépendance de distribution :** Attribuer ou partitionner Les données ne devraient avoir aucun effet sur le programme client.

— **Règle de non-subversion :** Si le système fournit une interface de bas niveau, l'interface ne doit pas permettre le contournement du système (par exemple, la sécurité des relations ou les contraintes d'intégrité). L'ensemble de ces règles indique la voie à suivre pour les systèmes de gestion de bases de données relationnelles. Elles ne sont jamais totalement implémentées, à cause des difficultés techniques que cela représente.

L'ensemble de ces règles indique la voie à suivre pour les systèmes de gestion de bases de données relationnelles. Elles ne sont jamais totalement implémentées, à cause des difficultés techniques que cela représente.

3.1.3 Les contraintes des SGBDs relationnels

La plupart des moteurs de SGBD relationnels sont basés sur le concept de transaction Il s'agit d'une unité d'action qui les oblige à se conformer aux contraintes ACID, acronyme : Atomicité Consistance Isolation Durabilité et qu'on nomme donc acidifiante de la transaction :[41]

- **Atomicité :** Tout ou rien. Soit l'opération se fait en entier, soit elle ne se fait pas du tout. La notion d'atomicité sous-entend la possibilité de défaire une opération avortée qui consiste à interdire l'exécution partielle de la transaction. Quand une transaction en cours d'exécution est interrompue, le SGBD doit annuler toutes les opérations déjà exécutées.[42]

  Exemple : *sur 5000 lignes devant être modifiées au sein d'une même transaction, si la modification d'une seule échoue, alors la transaction entière doit être annulée. C'est primordial, car chaque ligne modifiée peut dépendre du contexte de modification d'une autre, et toute rupture de ce contexte pourrait engendrer une incohérence des données de la base.*

- **Cohérence :** L'opération doit assurer que la base de données sera dans un état valide après l'opération.[42]

- **Isolation :** L'opération doit se faire en toute autonomie sans dépendance à une autre opération Cohérence.[43]

  Exemple *si une requête est lancée alors qu'une transaction est en cours, le résultat de celle-ci ne peut montrer que l'état original ou final d'une donnée, mais pas l'état intermédiaire. De fait, les transactions doivent s'enchaîner les unes à la suite des autres, et non de manière concurrentielle.*
- **Durabilité :** Elle garantit que les transactions réussies survivront de façon permanente et ne seront pas affectées par d'éventuelles pannes ou problèmes techniques. Les changements apportés aux données doivent être permanents. Plus précisément, ce sont les effets logiques des données modifiées sur les futures transactions qui doivent être permanents [44].

3.1.4 Les forces des SGBDs relationnels

Le succès des SGBDs relationnels est dû essentiellement aux 12 règles de CODD et le modèle relationnel qui a permis de développer une vision de ce qu'est ou doit être une base de données. Parmi ces nombreux avantages nous citons [45] :

1. Cohérence transactionnelle forte : Assurer la gestion de la concurrence, l'isolation entre les utilisateurs, la reprise sur panne.
2. Optimisation très fine des requêtes, index permettant un accès rapide aux données.
3. Logiciels mûrs, stables, efficaces, riches en fonctionnalités et en interfaces.
4. Contraintes d'intégrité permettant d'assurer des invariants sur les données.
5. Séparation logique et physique : justifié par une forte indépendance entre :
   — Modèle de données et structures de stockage.
   — Requêtes déclaratives et exécution.
6. Capacité de gestion de requêtes complexes.

3.1.5 Limite des base de données relationnels

Bien que le SGBD relationnel soit utilisé avec succès depuis plus de 20 ans. Cependant, il existe de nombreuses restrictions importantes. Les premiers acteurs à atteindre ces limites sont les géants du web. Le constat est simple : les SGBD relationnels ne sont plus la solution nécessaire pour un environnement distribué requis pour l'énorme quantité de données n'est pas tant le langage SQL en lui-même qui est inadapté mais les grands principes qui l'ont établi (Le modèle de relation et de transaction que nous avons vu en haut). En effet, en Big Data, nous devons non seulement gérer de grandes quantités de données, mais nous espérons également obtenir des résultats rapides. Mais Les SGBDs relationnels traditionnels atteignent ici leurs limites à savoir

— Problème d'application des propriété ACID en milieu distribué : Une base de données relationnelle est construite en respectant les propriétés ACID, ses propriétés bien que nécessaires à la logique du relationnel nuisent fortement aux performances et en particulier la propriété de cohérence En effet, la cohérence est très difficile à mettre en place dans le cadre de plusieurs serveurs (environnement distribué), car pour que celle-ci soit respectée tous les serveurs doivent être des miroirs les uns des autres, de ce fait deux problèmes apparaissent :
   — Le coût en stockage est énorme car chaque donnée est présente sur chaque serveur.
   — Le coût d'insertion/modification/suppression est très grand, car on ne peut valider une transaction que si on est certain qu'elle a été effectuée sur tous les serveurs et le système [46].
— Problème de requête non optimale dû à l'utilisation des jointures : Imaginons une table contenant toutes les personnes ayant un compte sur Instagram, soit 1 milliards d'utilisateurs actifs par mois, les données dans une base de données relationnelle classique sont stockées par lignes, ainsi si on effectue une requête pour extraire tous les amis d'un utilisateur donné, il faudra effectuer la jointure entre la table des usagers et celle des amitiés (chaque usager ayant au moins un ami) puis parcourir le produit cartésien de ces

deux tables. De ce fait, on perd énormément en performances en raison du temps consommé pour stocker et parcourir une telle quantité de données [46].

— Problème de gestion des objets hétérogène : Au fur et à mesure du temps, les structures de données manipulées par les systèmes sont devenues de plus en plus complexes en contrepartie les moteurs de stockage évoluant peu. Le principal point faible des modèles relationnels est l'absence de gestion d'objets hétérogènes ainsi que le besoin de déclarer au préalable l'ensemble des champs représentant un objet [46].

— Surcharge sémantique : Le modèle relationnel s'appuie sur un seul concept (la relation) pour modéliser à la fois les entités et les associations entre ces entités. Il existe donc un décalage entre la réalité et sa représentation abstraite [47].

— Scalabilité limitée : On peut distinguer deux types de scalabilité que les SGBDs relationnels ne puissent gérer :

  — La scalabilité des traitements : représente la capacité de distribution des traitements sur un nombre de machines important afin d'être en mesure d'absorber des charges très importantes.
  — La scalabilité des données : représente la capacité de répartition des données entre un nombre important de machines pour être en mesure de stocker de très grands volumes de données.

— Contraintes d'intégrité : Les SGBDs relationnels sont limités à des contrôles de cohérence simples. Il est donc difficile de modéliser des contraintes réelles correspondant aux données d'une entreprise. Ce problème n'est que partiellement résolu avec SQL-2 qui permet d'exprimer des contraintes dans la partie langage de définition [47].

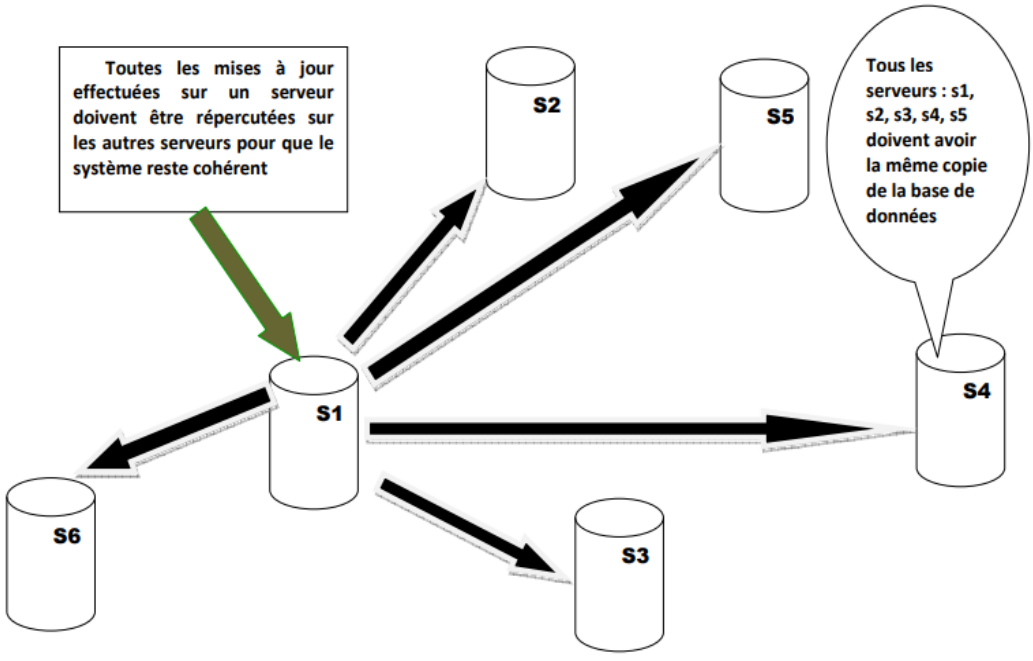

FIGURE 3.2 – Limites liées aux propriétés ACID.

3.1.6 Exemple des base de données relationnels

Nous présentons ci-dessous les 4 systèmes de gestion de bases de données relationnelles (SGBDR) les plus utilisé en 2020 :[48]

- **Oracle Database :** Oracle Database est un système de gestion de base de données relationnelle (SGBDR) qui depuis l'introduction du support du modèle objet dans sa version 8 peut être aussi qualifié de système de gestion de base de données relationnel-objet (SGBDRO). Fourni par Oracle Corporation, il a été développé par Larry Ellison, accompagné entre autres, de Bob Miner et Ed Oates.
- **MySQL :** C'est un système de gestion de bases de données relationnelles (SGBDR). Il est distribué sous une double licence GPL et propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde, autant par le grand public (applications web principalement) que par des professionnels.
- **Microsoft SQL Server :** Microsoft SQL Server est un système de gestion de base de données (SGBD) en langage SQL incorporant entre autres un SGBDR (SGBD relationnel) développé et commercialisé par la société Microsoft. Il fonctionne sous les OS Windows et Linux (depuis mars 2016), mais il est possible de le lancer sur Mac OS via Docker, car il en existe une version en téléchargement sur le site de Microsoft1.
- **PostgreSQL :** PostgreSQL est un système de gestion de base de données relationnelle et objet (SGBDRO). C'est un outil libre disponible selon les termes d'une licence de type BSD. Comme les projets libres Apache et Linux, PostgreSQL n'est pas contrôlé par une seule entreprise, mais est fondé sur une communauté mondiale de développeurs et d'entreprises.

3.2 Les bases de données NoSQL

Depuis les années 2000, avec l'explosion des applications web et une croissance de leurs usages, les grands acteurs du Web ont dû faire face à de nouveaux enjeux concernant leurs infrastructures informatiques. Face à cette croissance, les solutions de base de données relationnelles Open Source ont montré rapidement leurs limites, en particulier dans le domaine du web.

3.2.1 Définition :

Définition 1 : NoSQL est un mouvement très récent, qui concerne les bases de données. L'idée du mouvement est simple : proposer des alternatives aux bases de données relationnelles pour coller aux nouvelles tendances et architectures du moment, notamment le Cloud Computing. Les axes principaux du NoSQL sont une haute disponibilité et un partitionnement horizontal des données, au détriment de la cohérence. Alors que les bases de données relationnelles actuelles sont basées sur les propriétés ACID (Atomicité, Consistance ou Cohérence, Isolation et Durabilité).

Définition 2 : Le NoSQL, désignant Not Only SQL , représente une catégorie de bases de données apparue au courant de l'année 2009 qui se différencie du modèle relationnel que l'on retrouve dans les systèmes de bases de données connues tels que MS SQL Server, Oracle ou PostgreSQL. Cette catégorie de produits fait le compromis d'abandonner certaines fonctionnalités classiques des SGBD relationnels au portfie de la simplicité, la performance et une forte scalabilité. La scalabilité est la capacité d'un système à répondre à une demande toujours grandissante de la part des utilisateurs en termes de requêtes. Il s'agit d'une capacité de montée en charge.

Remarque : *NoSQL est constitué de No et SQL. Le No est un acronyme qui signifie Not only, ce qui veut dire en français ce n'est pas seulement du SQL. Cela signifie donc qu'il y a plus que les bases de données relationnelles.*

3.2.2 Le théorème CAP

CAP est un acronyme qui signifie Consistency, Availability and Partition Tolerance , dans la langue français cela donne Cohérence, Disponibilité et Résistance au morcellement . Ce théorème est aussi connu sous le nom de théorème de Brewer , du nom d' Eric Brewer qui l'a inventé en 2000 Ce théorème explique qu'un système d'information distribué ne peut satisfaire à la fois plus de 2 contraintes suivantes :

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 (Tolérance au partitionnement) :** Quel que soit le nombre de serveurs, toute requête doit fournir un résultat correct.

Comme nous l'avons mentionner ci-dessus les trois propriétés a savoir la cohérence, la disponibilité et la tolérance au partitionnement ne peuvent être obtenues simultanément c'est pour cela que des couples assurant deux propriétés se sont formés :

— **Soucieux de la cohérence et de la disponibilité (CA)**
Consistency Availability, soit le couple AC : cluster de site unique, donc tous les nœuds sont toujours en contact. Lorsqu'une partition se produit, le système s'arrête. Ce cas est possible que dans le cadre de bases de données transactionnelles telles que les SGBDR.

— **Soucieux de la tolérance de partition et la cohérence (PC)**
Consistency Partition Tolerance, soit le couple PC : Certaines données peuvent ne pas être accessibles, mais les données restent cohérentes et le système est tolérant au panne.

— **Soucieux de la disponibilité et la tolérance de partition (AP)**
Availability Partition Tolerance, soit le couple AP : Le système est toujours disponible même sous partitionnement, mais certaines des données retournées peuvent être inexactes.

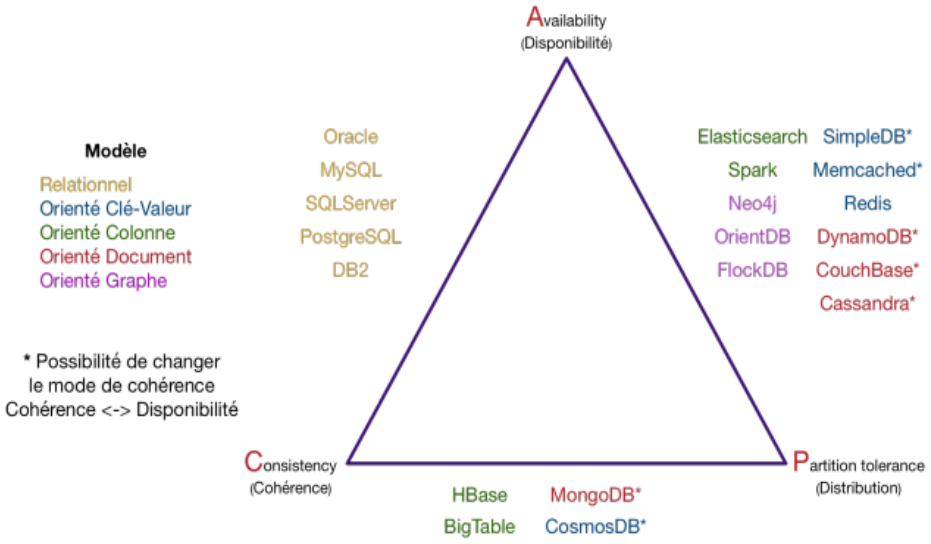

FIGURE 3.3 – Le théorème CAP.

3.2.3 Les types des base NoSQL

Le modèle de données de la base de données traditionnelle est principalement relationnel, spécifiquement pour prendre en charge les opérations de classe associées et les transactions ACID, mais dans les champs de la base de données NoSQL, le modèle de données traditionnel est le suivant :

3.2.3.1 Les bases de données orientées clé-valeur

Les données sont représentées par un couple clé valeur, ce type est le plus simple et est très adapté aux caches ou aux accès rapides aux informations. Son principe de base est le stockage d'une valeur associée à une clé unique. Celle-ci représente la seule manière de solliciter l'objet. La valeur pouvant être une simple chaîne de caractères ou un objet Elles ont de très bonnes performances car les lectures et écritures sont réduites à un accès disque simple. Il n'y a pas de schéma et les données ne sont pas liées entre elles. On les utilise quand le modèle de données est simple.[49]

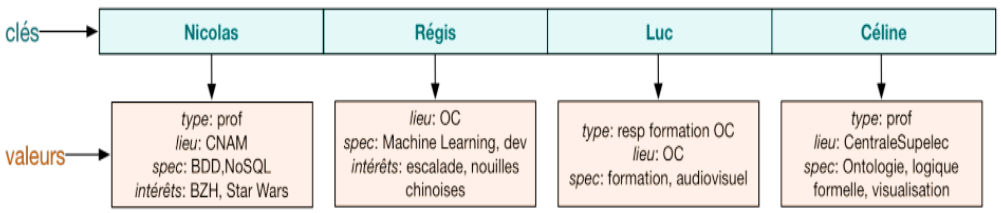

FIGURE 3.4 – Base de donnée orientée clé-valeu.

Ces bases permettent 4 opérations La création en effet seules opérations de type CRUD peuvent être utilisées :

- Create (key,value). Création d'un nouveau couple (clé, valeur).
- Read (key). La lecture : lire un objet en sachant sa clé
- Update (key,value). La modification : modifier et mettre à jour l'objet associé à une clé

- Delete (key). La suppression : supprimer un objet sachant clé.

Exemple de base de données clé-valeur

- Redis (VMWare) : Vodafone, Trip Advisor, Nokia, Samsung, Docker.
- Memcached (Danga) : LiveJournal, Wikipédia, Flickr, Wordpress.
- Azure Cosmos DB (Microsoft) : Real Madrid, Orange tribes, MSN, LG, Schneider Electric.
- SimpleDB (Amazon).

Avantage

— Leur simplicité.
— scalabilité.
— disponibilité.
— Très bonnes performances car les lectures et écritures sont réduites à un accès disque.

Inconvénient

— Pas de requêtes sur le contenu des objets stockés.
— Pas de relations entre les objets.

3.2.3.2 Les bases de données orientées colonnes

Dans ce type les BDD ne sont plus stockées en lignes mais en colonnes. Le nombre de colonnes est dynamique. On pourrait se dire que ce type ressemble un peu à la représentation des tables dans les bdd relationnelles car on y retrouve le principe de table qui contient des lignes et aussi des colonnes, mais elles ont deux principales différences : Des colonnes dynamiques. Dans la même table deux éléments peuvent ne pas avoir le même nombre de colonnes car les valeurs nulles ne sont pas stockées (contrairement au SGBDR relationnel). Cette propriété permet un gain d'espace de stockage et amélioration des performances de traitement L'historisation des données se fait à la valeur et non pas à la ligne comme dans les SGBDR cela empêche le stockage d'informations en doublon et de ce fait allège considérablement la base de données et les temps de calcul [49].

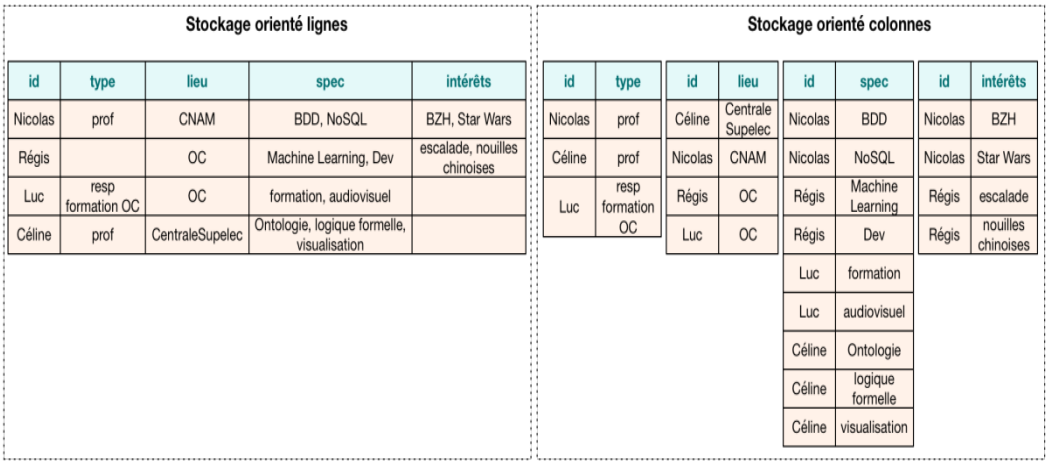

FIGURE 3.5 – Base de donnée orientée colonne.

Exemple de base de données orienté colonne :

— Hbase (Apache, Hadoop).
— Cassandra (Facebook , Apache)
— Big Table (Google).
— Hypertable.
— Apache Parquet.

Avantage

— Un temps de traitement réduit .
— Un gain d'espace de stockage grâce au stockage des valeurs nulles.

Inconvenant

— Pas adaptée aux données interconnectées.
— Pas adaptée pour les données non-structurées.

3.2.3.3 Les bases de données orientées documents

C'est une évolution de la base clé valeur, elle repose également sur l'association d'un couple [clé, valeur], et la valeur, dans ce cas, est un document. Ce document a une structure arborescente : Un document se compose d'un ensemble de couple clés/ valeurs. Les documents sont souvent de type JSON ou bien XML. Ces bases sont particulièrement adaptées au stockage de données semi structurées. Elles conviennent aux applications web pour leur simplicité d'utilisation et de déploiement [49].

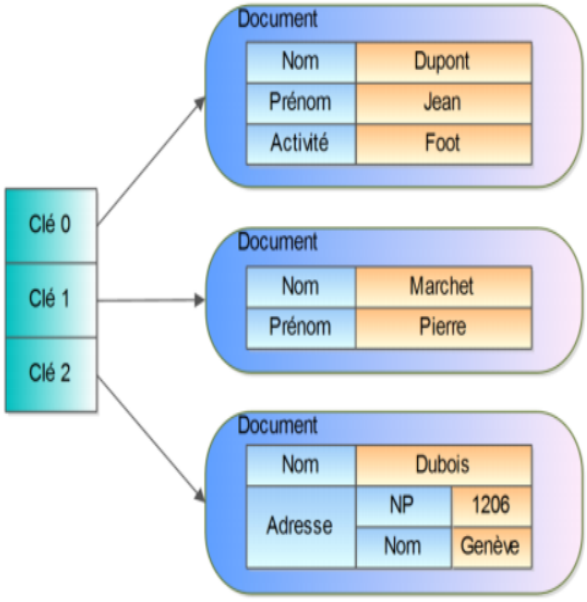

FIGURE 3.6 – Base de donnée orientée document.

Exemple de base de données orienté document

— MongoDB (MongoDB).
— RavenDB par Hibernating Rhinos.
— CouchBase (Apache, Hadoop).
— RethinkDB.
— DynamoDB (Amazon).

,

Avantage

- Les documents sont structurés mais aucune définition de structure préalable n'est nécessaire.
- On peut requêter et manipuler ces documents, et notamment récupérer, grâce une clé unique, l'ensemble des informations structurées de manière hiérarchique.

Inconvénient

- Pas de relations entre les documents Non adapter pour les données non-structurées.

3.2.3.4 Les bases de données orientées graphe

Ce sont des bases de données qui utilisent la théorie des graphes pour représenter et stocker les données , avec des nœuds et des arcs. Elles s'appuient sur les notions de : Noeuds qui ont chacun leur propre structure, Relations entre les noeuds ,Propriétés (de noeuds ou de relations). Le type orienté graphe est particulièrement bien adapté au traitement de certaines données comme celle des réseaux sociaux, les données géographiques, et de toutes les données fortement connectées de manière générale, par exemple la représentation des relations entre les abonnés sur Twitter ou Instagram.

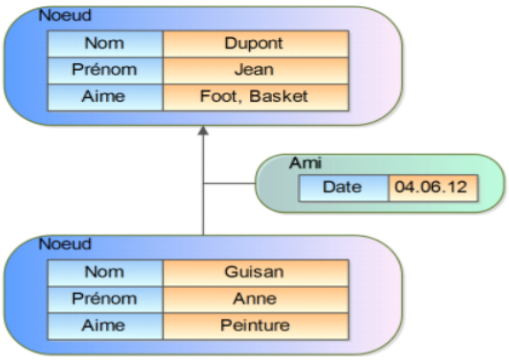

FIGURE 3.7 – Base de donnée orientée graphe.

Exemple de bases de données orienté graphes :

- Neo4j
- OrientDB (Apache)
- FlockDB (Twitter)
- inniteGraphe
- Titan
- ArangoDB

Avantage

- Adaptées aux objets complexes organisés en réseaux, aux données présentant des dépendances fortes.
- Permettent d'appliquer les algorithmes de théorie des graphes et la mise en place de visualisation de graphes nativement.
- Beaucoup plus rapides que les autres systèmes de stockage pour manipuler les données fortement connectées.

Inconvénient

— Non adaptées pour tous les autres contextes que celui des données fortement connectées.

3.2.3.5 Autres base de données

L'approche non relationnelle ne se limite pas aux base de données NOSQL, Il existe plusieurs autres suivant une approche non relationnelle, elles ne sont pas considérées comme des bases de données NoSQL de base, mais plutôt comme des bases de données NoSQL logicielles :

Bases de données d'objets : Les modèles de bases de données orientées objet regroupent des paquets de données très proches : un ensemble de données est regroupé avec tous ses attributs pour former un objet. Ainsi, toutes les informations sont immédiatement disponibles. Au lieu de tout répartir dans différentes tables, les données sont interrogeables par ensemble.

Bases de données XML : Une base de données XML Native (NXD en anglais) est une base de données qui s'appuie sur le modèle de données fourni par XML. Elle utilise typiquement des langages de requête XML comme XPath ou XQuery.

L'indexation dans une base de données XML nécessite d'indexer non seulement le contenu des éléments mais aussi la structure, les relations entre éléments pour que des requêtes XPath comme /foo/bar utilisent l'index.

3.2.4 Avantage du NoSQL

L'intérêt des systèmes de stockage NoSQL réside surtout dans les choix d'architecture logicielle qui ont été pris lors de leurs conceptions. Parmi les raisons principales qui ont mené à la création de ces systèmes.Le NoSQL possède un ensemble de fonctionnalités, notamment :

— La plupart des systèmes NoSQL sont open source.
— Les SGBD NoSQL évoluent horizontalement. Ainsi, pour gérer une grande quantité de données, il faut ajouter un serveur et non en augmenter sa capacité ou ses performances. La flexibilité et l'évolutivité horizontale offrent aux bases de données NoSQL une grande capacité d'adaptation au changement sans affecter le système ou les performances.
— Rapidité : flexibilité dans la gestion des données et rapidité , car ils n'utilisent pas de schéma de base avec les contraintes sur les champs.
— Les systèmes de gestion bases de données NoSQL peuvent s'adapter à tous les types de données car ils supportent les données structurées comme les non structurées.
— La réplication des bases de données NoSQL est pensée autrement que pour les bases de données classiques. Les réplications sont effectuées automatiquement dans les clusters (ensemble de plusieurs serveurs) et dans les centres de données
— Peut fonctionner sur des appareils de faible spécification.
— peut s'utiliser avec différents types de données.
— Permet de manipuler de grand volume de données
— Schéma flexible.

3.2.5 Inconvénients du NoSQL

Il faut néanmoins être conscient que les avantages apportés par ces systèmes ne sont pas sans contreparties, aucun système n'étant parfait. Les principaux inconvénients apportés par les choix de design des NoSQL sont les suivants :

— Fonctionnalités réduites : Les bases de données NoSQL sont principalement conçues pour stocker de la meilleure des façons qui soit des ensembles donnée mais en contrepartie nous avons très peu de fonctionnalités aussi bien faite que celle ci notamment le langage permettant d'effectuer des requêtes vers le système NoSQL est beaucoup moins riche.

— Normalisation et Open Source : Étrangement, le critère d'open source pour les bases de données NoSQL est à la fois sa plus grande force et sa plus grande faiblesse. La raison est qu'il n'existe pas encore de normalisation fiable pour NoSQL.

— Performances et évolutivité au détriment de la cohérence : En raison de la façon dont les données sont gérées et stockées dans ces bases, la cohérence des données pourrait bien être une préoccupation. Comme déjà vu précédemment, les bases de données NoSQL font l'impasse sur les propriétés dites ACID afin de mieux répondre aux besoins de performances et d'évolutivité. La cohérence des données est donc un facteur moins important. Selon le besoin, ces caractéristiques peuvent être un sérieux atout comme un paramètre irrecevable. S'il faut faire face à de très grosses montées en charge de très gros volumes de données, NoSQL est le système adéquat. Si on traite des données sensibles (transactions bancaires ou autres), la cohérence des données est indispensable. Et comme les systèmes NoSQL ne respectent pas l'ensemble des propriétés ACID, comme le font les systèmes relationnels classiques, cela se traduit en pratique par un effort supplémentaire du développeur dans certains cas pour s'assurer de la cohérence des données .

— Manque général de maturité : Bien que les bases de données NoSQL soient présentes depuis bon moment, leurs technologies sont encore immatures par rapport à celles des bases relationnelles. Cela se traduit également par un manque d'administrateurs et de développeurs ayant les compétences dans ce système. Les bases de données ont beau être « administrator-friendly » (simples à administrer), cela n'a pas de sens si les administrateurs en question ne savent pas comment les utiliser. A l'heure actuelle, les bases de données relationnelles sont beaucoup mieux implémentées dans les entreprises. Elles disposent d'un nombre plus grand de fonctionnalités et de professionnels qui comprennent comment les gérer. Les bases de données NoSQL ne sont donc, pas la solution miracle pour répondre à toutes les problématiques de stockage sur le web ou ailleurs. Il est surtout très important, de bien comprendre, ce que le choix d'une base de données de ce type va avoir comme conséquences en termes d'architecture logicielle et complexité de développement.

— difficulté d'administration : Un des buts des bases de données NoSQL était d'en simplifier l'administration, mais la réalité est différente. Aujourd'hui, les solutions NoSQL demandent beaucoup de compétences pour leur installation ainsi que des efforts conséquents lors de leur maintenance.

— Aucune contrainte et validation à exécuter dans la base de données.

— Interopérabilité : La passage d'une base de données NoSQL vers une autre n'est pas transparente pour une application.

— Peu de mises à jour sont prises en charge : Le problème qui se pose est que, dans NoSql, les données peuvent ne pas être cohérentes, ce qui signifie que les deux nœuds peuvent avoir des données différentes pour le même identifiant, tandis que la base de données SQL vous donne des propriétés ACID par lesquelles nous pouvons les résoudre .

3.3 Comparaison entre le SQL et NoSQL

— Les bases de données SQL sont des bases de données basées sur des tables tandis que les bases de données NoSQL sont des bases de données basées sur des paires clé-valeur, graphe , colonne ou encore Document . Cela signifie que les types de données représentées sont différents et que les relations entres elles sont aussi différentes
— Les requêtes SQL sont pour les données structurées alors que NoSQL traite les données non structurées.
— Les bases de données SQL sont destinées au traitement de données dont le volume est petit à moyen ou élevé par contre les bases de données NOSQL peuvent traiter un volume de données élevé.
— Les bases de données SQL conviennent parfaitement à l'environnement exigeant de nombreuses requêtes complexes , tandis que les bases NoSQL ne conviennent pas aux requêtes complexes
— Les bases de données SQL évoluent verticalement. Vous pouvez gérer l'augmentation de la charge en augmentant le processeur, la RAM, le SSD, etc. sur un seul serveur. D'autre part, les bases de données NoSQL évoluent horizontalement. Vous pouvez simplement ajouter quelques serveurs supplémentaires facilement dans votre infrastructure de base de données NoSQL pour gérer le trafic important.
— Dans les systèmes relationnels nous devons garantir l'acidité contrairement au NOSQL , les garanties ACID ne sont pas requises mais suivent le théorème Brewers CAP (Cohérence, Disponibilité et Tolérance de partition).

— Pourquoi utiliser les BDD SQL
SQL protège activement l'intégrité de votre base de données en fournissant la conformité ACID. En raison de ses données structurées, vous n'avez besoin d'aucun support système intégré pour utiliser différents types de données. De plus, SQL est l'option la plus recommandée par de nombreuses entreprises en raison de sa structure et de ses schémas prédéfinis.
— Pourquoi utiliser les BDD NOSQL
Le NOSQL vous permettant de stocker différents types de données ensemble et vous pouvez facilement évoluer en répartissant plusieurs serveurs. Si vous avez besoin de développer une application dans un délai imparti, vous pouvez opter pour NoSQL, qui améliorera vos performances grâce à sa phase de développement rapide

3.4 Comparaison entre les différentes base de données NOSQL

Comment faire son choix ?

— Apache Cassandra pour les acteurs du web : Initialement développée par Facebook (qui l'a publiée en open source en 2008), Apache Cassandra fait figure de star dans le monde web. Cette base NoSQL orientée colonnes a été adoptée par Apple, Netflix ou Spotify. Cassandra peut gérer de grands volumes de données et privilégie les performances notamment en lecture. Elle nécessite toutefois un réel travail de normalisation , mais elle peut également convenir aux jeunes pousses en raison de ses capacités de dimensionnement.

— Couchbase un outil de requêtage "SQL like" : Couchbase, un outil de requêtage "SQL like" lancé en 2010 par des anciens du projet d'infrastructure distribuée Memcached, Couchbase se caractérise par une architecture type maître-maître. Cette base orientée documents privilégie la cohérence des données aux performances pures. Couchbase dispose d'un outil de requêtage normalisé SQL baptisé N1QL (qui se prononce Nickel). Ce qui peut faciliter sa prise en mains par des développeurs rompus aux bases SQL. Avec une solution de cache intégrée, Couchbase vise les applications web. C'est la seule solution NoSQL de notre comparatif à proposer une offre pour mobiles (Couchbase Lite).

— Elasticsearch, pour la force du moteur de recherche : Elasticsearch est avant tout connu pour son moteur de recherche distribué. Il utilise la bibliothèque d'indexation open source Lucene tout comme Apache Solr à qui il est souvent comparé. Cet outil permet de stocker et analyser des données, structurées ou non, comme des textes libres ou des logs systèmes. Projet open source développé en Java sous licence Apache, Elasticsearch est associé à deux autres produits open source : le visualiseur de données Kibana et l'outil d'extraction, de transformation et de chargement de données (ETL) Logstash.

— HBase , pour les très gros volumes : Issue de la famille Apache, HBase est souvent opposé à Cassandra. Il s'agit là encore d'une base orientée colonnes. Basée sur une architecture maître-esclave et s'inspirant de BigTable de Google, elle peut gérer d'énormes quantités de données, plus encore que Cassandra. HBase est une base un peu à part car intimement liée à Hadoop dont elle est un sous-projet. Elle s'installe d'ailleurs sur son système de fichiers distribué HDFS. "Destinée aux fortes volumétries, HBase privilégie d'avantage les possibilités de requêtage à la cohérence des données", ajoute Christophe Parageaud. Produit complexe, HBase exige un gros travail de structuration.

— MongoDB : C'est l'une des plus populaires, développée depuis 2007 par la société du même nom, MongoDB est la base NoSQL la plus populaire selon le palmarès de DB-Engines. Basé sur une architecture de type maître-esclave, ce moteur orienté documents est reconnu pour la souplesse de sa structure. pas besoin de pré structurer les données, il suffit de créer des collections et d'y mettre des éléments Json sans avoir besoin de dire comment les organiser. MongoDB est une base générique qui répond à 80% des besoins couverts par une base relationnelle traditionnelle, à l'exception du transactionnel."

— REDIS, pour sa vitesse : Lancé en 2009, le projet est sponsorisé par VMware, Redis est une base de données reposant sur le principe de clé-valeur. En rendant les données facilement accessibles, elle privilégie la vitesse d'exécution. Revers de

la médaille, cette base n'est pas taillée pour les gros volumes et ses capacités de requêtage sont limitées. Elle ne gère pas la recherche multicritères. Redis se destine aux applications nécessitant une haute disponibilité et une faible latence, notamment dans l'e-commerce. Distribué sous licence BSD et écrit en code C, c'est le seul moteur de notre comparatif à gérer le transactionnel.

— Riak , pour sa tolérance aux pannes : Distribué sous licence Apache et inspiré de Dynamo, Riak est un système de gestion de base de données orienté clé-valeur. Sorte de version évoluée de Redis, il développe les capacités de requêtage en offrant la possibilité, via des index secondaires, d'aller requêter dans la valeur. Basho Technologies, l'éditeur qui développe Riak, fait de la tolérance aux pannes et de la haute disponibilité . Il propose des solutions taillées spécifiquement pour le stockage dans le cloud (Riak CS) et pour l'internet des objets avec la prise en compte des time series (Riak TS).

Deuxième partie

Synthèse

Problématique

Avec la croissance du volume de données qui doivent être enregistrés, traités et mis à jour de plus en plus vite. Le NoSQL s'est imposé ces dernières années comme étant le système de base de données le plus adéquat afin de stocker de grosses quantités de données, cependant les avantages apportés par ce système ne sont pas sans contreparties.

Quand ont parle de bases de données NoSQL en parles d'immenses masse de données hétérogènes qui proviennent de plusieurs endroits ce qui nous ramène au théorème de CAP qui dit qu'entre la cohérence des données et leurs disponibilité ainsi que le partitionnement du réseau, seul **trois** de ces contraintes peuvent être satisfaite en même temps, mais dans le cas des bases de données NoSQL le partitionnement du réseau est primordiales, ce qui nous obligent à choisir entre la disponibilité des données et leurs cohérences.[10]

Afin de bien illustrer cette situation nous proposons la figure ci-dessous :

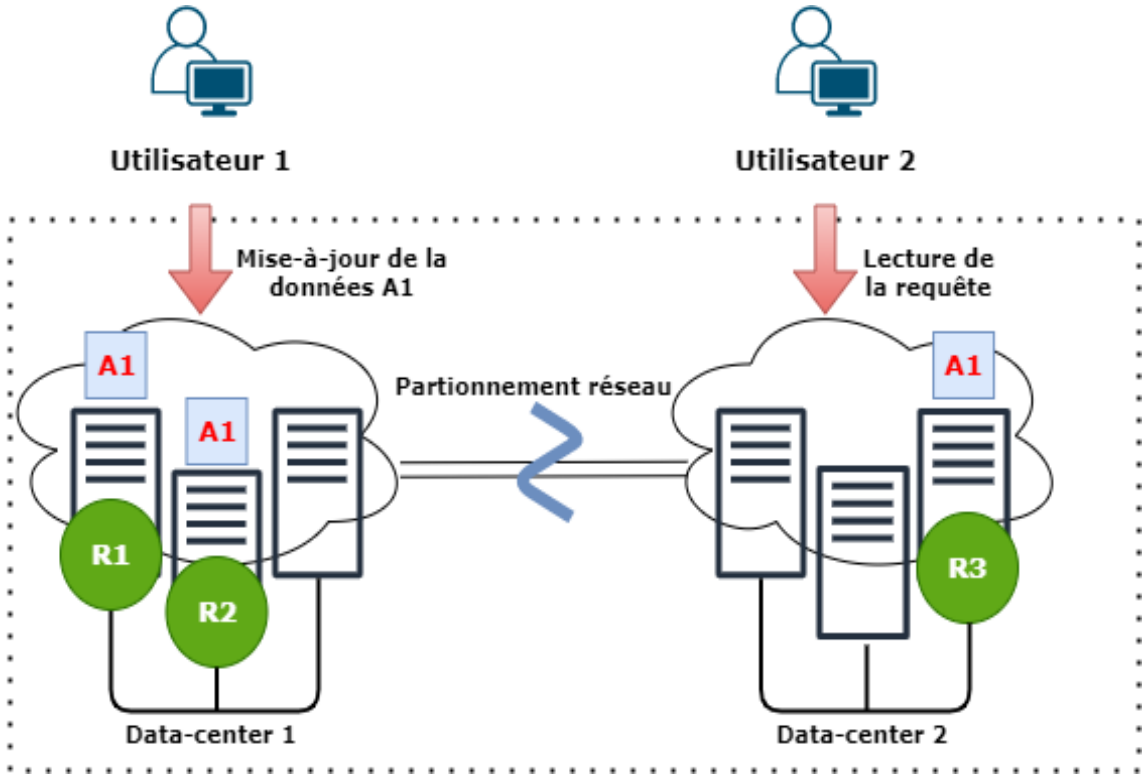

FIGURE 3.8 – Cohérence et disponibilité dans les systèmes distribuées.

Au niveau de la **figure 3.8** nous Pouvons observer que l'utilisateur 2 effectue une demande de lecture pour l'élément de données **A1** dans la réplique **R3** (Datacenter 2), après que l'utilisateur 1 a mis à jour l'élément de données **A1** dans la réplique **R1** (Datacenter 1) en présence d'une partition réseau qui isole les deux centres de données.

Étant donné la partition réseau, il existe deux scénarios possibles :

— Soit toutes les répliques peuvent être disponibles sans que la mise à jour de la donnée **A1** soit propagée vers la réplique **R3**, ce qui fait que l'utilisateur **2** lira des données obsolètes, violant ainsi la cohérence.
— Soit l'utilisateur **2** doit attendre que la partition réseau soit corrigée et que la mise à jour ait été propagée à la réplique **R3**, violant ainsi la disponibilité.

Pour aider à résoudre ce problème, des systèmes de base de données NoSQL ont vu le jour. Ces systèmes ont été créés avec une exigence standard à l'esprit, l'évolutivité. Les compromis causés par le théorème de **CAP** ont conduit à la prolifération de systèmes **non-ACID** pour la construction d'applications basées sur le cloud, appelés **BASE** [1] qui sont des systèmes qui sont essentiellement disponibles, reposent sur le maintien d'un état souple qui peut être reconstruit en cas d'échecs et ne sont finalement cohérents que pour survivre aux partitions réseau.[50]

Approche BASE

L'approche BASE selon E.Brewer [2] perd les propriétés ACID de cohérence et d'isolement au profit de "la disponibilité, la dégradation gracieuse et les performances". L'acronyme **BASE** est composé des caractéristiques suivantes :[51]

— Fondamentalement disponible.
— État doux.
— Finalement cohérent.

En ce qui concerne les bases de données, Brewer conclut que les bases de données actuelles sont plus cohérentes que disponibles et que les bases de données étendues ne peuvent pas avoir les deux, une notion qui est largement adoptée dans la communauté NoSQL. Les systèmes qui peuvent être caractérisés par les propriétés BASE incluent le Dynamo d'Amazon, qui est disponible et tolérant aux partitions mais pas strictement cohérent, c'est-à-dire que les écritures d'un client ne sont pas vues immédiatement après avoir été validées pour tous les lecteurs. BigTable de Google ne choisit ni ACID ni BASE mais la troisième alternative CAP étant un système cohérent et disponible et par conséquent incapable de fonctionner pleinement en présence de partitions de réseau.

Brewer souligne des traits et des exemples des trois choix différents qui peuvent être faits selon le theorem CAP. De plus, il oppose ACID à BASE tout en considérant les deux concepts.[51]

On peut résumé les propriétés BASE de la manière suivante : *une application fonctionne essentiellement tout le temps (essentiellement disponible) ; n'a pas besoin d'être cohérent tout le temps (soft-state) ; mais sera éventuellement dans un état connu (cohérence éventuelle)*

Le problème de cohérence et de disponibilité de données n'est pas le seul frein que rencontrent les bases de données NoSQL, bien qu'elles soient présentes depuis un bon moment, leurs technologies sont encore immatures par rapport à celles des bases de don-

1. Acronyme : Basically Available Soft-state Eventually-consistent
2. Eric Brewer est un scientifique américain, professeur émérite d'informatique à l'Université de Californie à Berkeley et vice-président de l'infrastructure de la société Google. Ses domaines de recherche comprennent les systèmes d'exploitation, l'informatique distribuée, le cloud computing et les réseaux de capteurs.

nées relationnelles qui résiste à l'épreuve du temps et qui continue d'ajouter de nouvelles innovations, De plus, elles sont maintenant capable d'absorber les nouveaux types de données (spatiales, semi-structurées, et les modèles de cohérence afin qu'ils puissent coexister dans un seul système). Il n'y a pas de problèmes d'évolutivité inhérents au modèle relationnel ou à la syntaxe de requête SQL. Il fallait juste une implémentation de stockage différente pour tirer parti d'une architecture évolutive. Cela a prouvé que, pour la majorité des cas d'utilisation, les bases de données relationnelles sont plus faciles à utiliser et généralement plus performantes que les systèmes NoSQL. Un autre problème confronté par les BDD NoSQL est la difficulté d'administration qui est un des points à ne pas négliger, si à la base elles ont été créées afin de simplifier l'administration, la réalité est différente. Aujourd'hui, les solutions NoSQL demandent beaucoup de compétences pour leur installation ainsi que des efforts conséquents lors de leur maintenance, et une infrastructure de stockage immense qui garantit la répartition de ce volume immense de donnée (tolérance au pannes),elles ont beau être « administrator-friendly » (simples à administrer), cela n'a pas de sens si les administrateurs en question ne savent pas comment les utiliser.

Sur les quantités de données à stocker, aucune contrainte et validation n'est faite sur elles en général, le principe c'est de sauvegarder toute donnée, car cette dernière est susceptible d'être utilisée, si ce n'est pas le cas à cet instant il le sera à un instant "t".

A l'heure actuelle, les bases de données relationnelles sont beaucoup mieux implémentées dans la majorité des entreprises, ces dernières ne veulent pas passer au base NoSQL en vu des problèmes liée à la migration d'une base relationnels à une base NoSQL et de l'interopérabilité (passage d'une base de données NoSQL vers une autre), la cause de cela c'est que ces bases de données on été crée par les géant du web (facebook, google, amazon, microsoft etc), et chacun à créer un système adapter pour ces propres besoins, y'a des modèle qui sont plus focalisé sur la contrainte de disponibilité de donnée et y'a ceux qui sont plus focalisé sur la contrainte de cohérence de données etc.À cause de cela malgré le critère d'open source pour les BD NoSQL qui est à la fois sa plus grande force et sa plus grande faiblesse il n'existe pas encore de normalisation fiable pour les BD NoSQL.

Si la migration ou l'interopérabilité est un problème liée à l'architecture des base NoSQL, l'extraction d'un modèle de donnée via une base de données Nosql n'en reste pas moindre, actuellement, les SGBD NoSQL ne disposent pas d'une fonctionnalité permettant d'afficher dynamiquement le modèle de la BD. Or, les utilisateurs (développeurs, Data scientists, Data analysts, etc.) ont besoin du modèle de données pour exprimer des requêtes d'interrogation et de mise à jour des données. En vu de la jeunesse de ce type de BD ce dernier n'est pas encore très utilisé dans toutes les entreprises, il y a donc un support limité (Documentation) ainsi qu'un support très faible de la communauté car le plus grand des travaux est consacré à la recherche sur le sujet. Interroger de tel système de BD n'est vraiment pas facile en vue de la quantité, les types de données à traiter ainsi que l'énergie nécessaire (CPU) à fin de retourner le résultat attendu en minimum de temps possible (idéalement en temps réel).

Dans les systèmes "not only SQL", le problème le plus critique est avant tout le partitionnement réseau c'est à dire que le système est distribuée et partagée sur différents réseau, ce dernier doit assurer doit pouvoir fonctionner même en cas de panne du réseau ce qui n'est pas évident à mettre en place dans de tel systèmes qui sont généralement temps réels .

Les bases de données NoSQL ne sont donc, pas la solution miracle pour répondre à tous les besoins . Il est surtout très important, de bien comprendre, ce que le choix

d'une base de données de ce type va avoir comme conséquences en termes d'architecture logicielle et complexité de développement, et ainsi en cas d'adoption de cette solution, il faudra savoir saisir quel est le l'élément le plus important entre la disponibilité et la cohérence de données.

Ce problème de cohérence et de disponibilité qu'illustre le théorème de CAP est particulièrement délicat dans le contexte des systèmes de stockage avec réplication de données, répartie géographiquement, car le but est de savoir comment atteindre un état cohérent dans toutes les répliques. Garantir une forte cohérence dans un tel environnement entraîne des surcoûts de performance importants en raison de la latence accrue du réseau entre les centres de données et du fait que les partitions réseau peuvent entraîner une indisponibilité du service . En conséquence, des modèles spécifiques ont été proposés pour offrir des garanties de cohérence plus faibles ou assouplies .Obtenir le juste équilibre entre des niveaux plus élevés de cohérence et de disponibilité est l'un des défis ouverts de cette décennie. Dans ce but, nous nous concentrerons sur les méthodes de pointe pour la cohérence et la disponibilité de données dans les environnements cloud.[10]

Plusieurs entreprises optent pour assurer un niveau de disponibilité et de performances élevés même en présence de partitions réseau plutôt que d'offrir un modèle de cohérence plus solide. Les environnements de stockage de données basés sur NoSQL fournissent des propriétés de cohérence en mode éventuel, ce qui signifie que toutes les modifications apportées à un élément de données répliquées atteignent finalement toutes ses répliques. Cependant, l'utilisation de ce type de cohérence augmente la probabilité de lire des données obsolètes, car les répliques auxquelles on accède peuvent ne pas avoir reçu les écritures les plus récentes. Cela a conduit au développement de solutions de cohérence adaptative, qui permettent d'ajuster le niveau de cohérence au moment de l'exécution afin d'améliorer les performances ou de réduire les coûts, tout en maintenant le pourcentage de lectures obsolètes à de faibles niveaux.

Chapitre 4

Disponibilité et cohérence des données dans les BD NoSQL

Introduction

La prolifération de sources de données allant des médias sociaux et de l'Internet des objets (IoT) aux données générées industriellement (par exemple, les transactions) a conduit à une demande croissante d'applications basées sur le cloud à forte intensité de données et a créé de nouveaux défis pour les bases de données de l'ère du Big Data.

Les bases de données NoSQL sont utilisées de nos jours par les développeurs d'applications cloud pour exploiter ces grandes quantités de données sans schémas spécifique. Ils ont un schéma flexible, impliquant la possibilité d'un schéma évoluant dynamiquement. Ces bases de données ne nécessitent pas de définition de schéma avant d'insérer des données et ne nécessitent pas de modification de schéma lorsque les besoins de gestion des données évoluent.[51]

L'Un des problème difficile qui se pose dans le contexte des systèmes de stockage en nuage avec réplication des données géographiquement distribuées est de savoir comment atteindre un état cohérent dans toutes les répliques. La mise en œuvre de la réplication synchrone pour garantir une cohérence élevée dans un tel environnement entraîne des surcharges de performances importantes en raison de la latence réseau accrue entre les centres de données et du fait que les partitions réseau peuvent entraîner une indisponibilité du service, Afin d'y remédier à ce problème plusieurs modèles spécifiques ont été proposés pour offrir des garanties de cohérence plus faibles ou assouplies.[10]

Dans ce chapitre on va voir tout d'abord les concepts généraux liée à la gestion de BDD Cloud puis nous allons étudier le conflit entre la cohérance et la disponibilité dans les systèmes réparti ainsi que les différentes méthodes de cohérences proposés afin de palier à ce compromis.

4.1 Concepts généraux

4.1.1 Réplication et cohérence de donnée

La réplication des données sur plusieurs nœuds est une solution efficace afin d'avoire de meilleurs performances et une disponibilité des données élevé. Le traitement distribuée des demandes d'accès améliore les performances en termes de temps de réponse et de Charge acceptable par le système (traitement parallèle).

Remarque *Il faut savoir qu'une donnée ne devient indisponible que si tous les nœuds qui en possèdent une copie tombent en panne simultanément.*

Le problème général lié à la réplication de ces données sur plusieur noeuds est celui de la cohérence des données au niveau des différentes répliquent. Afin de gérer cette dernière en a deux points de vue essentielle :

- Du point de vue des demandes d'accès, la gestion de la concurrence locale se préoccupe de leur isolation, c'est-à-dire de les exécuter en parallèle en évitant qu'une mise à jour soit visible par d'autres demandes d'accès tant qu'elle n'a pas été validée.
- Du point de vue des données, la gestion des copies doit assurer leur cohérence mutuelle, c'est-à-dire que toutes les copies d'une donnée soient identiques.

Sans contrôle, l'exécution simultanée des demandes d'accès peut conduire à des phénomènes indésirables. Néanmoins, accepter de vivre avec certaines imperfections peut améliorer la performance du système en améliorant la concurrence.

4.1.2 La réplication

La duplication des données est une solution très efficaces pour faciliter l'accès aux données tout en augmentant leur disponibilités, soit parce qu'un nœud voisin peut prendre la relève lorsque le serveur principal s'écroule (cas de panne), soit parce que les données sont copiées sur différents nœuds permettant de répartir les requêtes et de les traités en parallèles. Cela fournit une meilleur tolérance aux pannes et un meilleur temps de réponse.

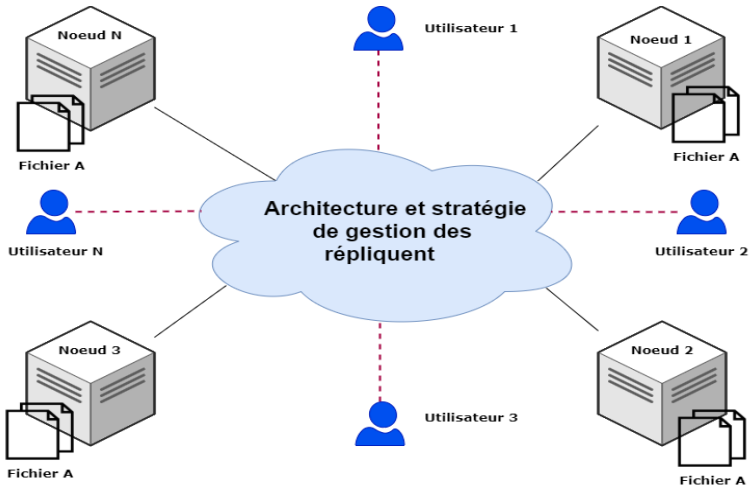

FIGURE 4.1 – Environnement d'utilisation de la technique de réplication.

La réplication peut se faire à plusieurs niveaux et prendre plusieurs formes comme suit :

- **Aucune réplication :** Les données ne sont pas dupliquées entre les sources du système (cas de la centralisation).
- **Réplication partielle :** Les données sont répliquées sur quelques sources du système.
- **Réplication totale :** Les données sont répliquées sur toutes les sources du système.

Cependant Il faut bien distinguer entre la réplication et les autre type de partage de données tel que :

La sauvegarde : Les données sauvegardées ne changent pas dans le temps (état fixe des données), tandis que les données répliquées évoluent sans cesse à mesure que les données sources changent.

La copie : À la différence de copier, la réplication gère la cohérence des répliques, donc une réplique est plus qu'une copie.

4.1.2.1 Avantage de la réplication

— Une amélioration de la fiabilité ou bien une sûreté de fonctionnement.

Exemple :
— Si une copie tombe en panne, il est toujours possible d'obtenir les données à partir d'une autre copie.
— La redondance permet une meilleure protection contre la corruption de fichiers.

— Amélioration des performances.
Exemple :
— Diviser la charge de travail entre plusieurs serveurs.
— Extensibilité géographique en rapprochant les serveurs des clients.

— Une disponibilité de données.
Les données sont disponibles (dans le réseau locals) même en l'absence de toute connexion à un serveur central,de sorte que l'utilisateur n'est pas coupé de ses données en cas de défaillance d'une connexion réseau longue distance.

— Temps de réponse.
Exemple :
— Les requêtes sont traitées sur un serveur local sans accès à un réseau étendu, ce qui accélère le débit.
— Par ailleurs, le traitement local allège la charge du serveur de bases de données central, ce qui permet de moins solliciter le processeur.

4.1.2.2 Inconvénients de la réplication

Malgré tous les avantages qu'elle procure, la réplication soulève un certain nombre de problèmes que nous allons aborder dans ce qui suit :

- **Placement des répliques :** Ce problème consiste à choisir, des localisations physiques pour les répliques, qui réduisent les coûts de stockage et d'accès aux données en fonction des objectifs des applications et de la réplication.
- **Choix d'une réplique :** Ce problème consiste à sélectionner, parmi toutes les répliques d'une donnée, celle qui est la meilleure du point de vue de la consistance.
- **Degré de réplication :** Ce problème concerne la recherche du nombre minimal de répliques qu'il faut créer pour une donnée, sans pour autant réduire les performances des applications. Cela peut être déterminé d'une manière prédictive ou adaptative.
- **Cohérence des répliques :** Parmi les problèmes liés à la réplication, le problème de la cohérence des données est sans doute celui qui est le plus complexe, c'est sur ce dernier que nous allons nous intéresser dans la suite de ce chapitre.

4.1.3 La cohérence

Les avantages de la réplication des données sont contraints par le problème de cohérence mutuelle des copies, car cette réplication doit être transparente vis à vis de l'utilisateur se qui offre la garantie d'une vue cohérente des données dispersées.

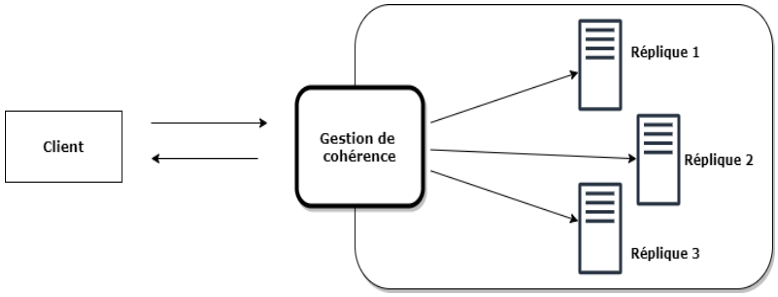

FIGURE 4.2 – Vue cohérente des donnée.

La gestion de cohérence est donc définie comme étant le contrôle des accès, afin de fournir une vision qui fait abstraction de la réplication, de la distribution et de la concurrence des accès aux données partagées.

4.1.3.1 Types de cohérence

La gestion des copies en termes de propagation des mises à jour est ainsi nécessaire. La charge induite par cette cohérence peut avoir un impact significatif sur le système notamment du point de vue performance. Il s'agira donc de garantir la cohérence mutuelle d'un ensemble de répliques dans des délais acceptables, on distingues deux type de cohérence :

- **Cohérence forte :** Une cohérence de répliques est forte lorsque toute interrogation d'une copie quelconque reflète le résultat de toutes les modifications antérieures.

- **Cohérence faible :** Une cohérence de répliqués est dite faible, si on tolère qu'une interrogation ne reflète pas toutes les modifications antérieures, avec la garantie que celles-ci seront toutes répercutées au bout d'un temps fini.

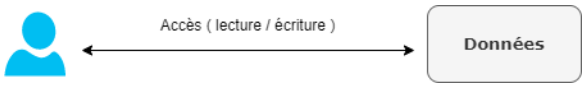

FIGURE 4.3 – Vue idéale des données.

FIGURE 4.4 – Vue réelle des données.

4.1.4 Exigences de stockage des données dans le cloud

Le stockage de données dans le cloud exige une infrastructure fiable et appropriée, afin que toutes les ressources puissent être utilisées et partagées efficacement et cela pour réduire les problèmes liées à la cohérence de données.

Pour garder la bonne gestion des bases de données dans les systèmes distribuées il faut veiller au respect de certain critères :[10]

Automatisation : Le stockage des données doit être automatisé pour pouvoir exécuter rapidement les changements d'infrastructure nécessaires pour maintenir la cohérence des répliques et cela sans intervention humaine.

Disponibilité : Le stockage des données doit garantir que les données continuent d'être disponibles à un niveau de performance requis dans des situations allant de normales à défavorables.

Élasticité : Non seulement le stockage de données doit pouvoir évoluer avec une charge croissante, mais il doit également pouvoir s'adapter aux réductions de charge en libérant des ressources cloud, tout en garantissant la conformité avec un accord de niveau de service (SLA [1]).

Tolérance aux pannes : Le stockage de données doit pouvoir récupérer en cas de panne, par exemple en fournissant une instance de sauvegarde de l'application qui sera prête à prendre le relais sans interruption.

Faible latence : Le stockage de données doit gérer les problèmes de latence en mesurant et en testant la latence du réseau, avant d'enregistrer les données modifiées

1. Service- Level Agreement : est un document qui définit la qualité de service, prestation prescrite entre un fournisseur de service et un client

par une application et avant de mettre ces données à la disposition d'autres applications.

Tolérance de partition : Le système doit continuer à fonctionner malgré les partitions réseau.

Performance : Le stockage de données doit fournir une infrastructure qui prend en charge un accès, une mise à jour et une récupération de données rapides et robustes.

Fiabilité : Le stockage des données doit garantir que les données peuvent être récupérées en cas de catastrophe.

Évolutivité : Le stockage de données doit évoluer rapidement pour répondre aux demandes de charge de travail, offrant ainsi une évolutivité horizontale et verticale.

Afin de garantir l'intégrité des données, la plupart des systèmes de bases de données classiques sont basés sur des transactions. Cela garantit la cohérence des données dans toutes les situations de gestion des données. Ces caractéristiques transactionnelles sont également appelées propriété ACID (atomicité, cohérence, isolation et durabilité).

Cependant, il n'est pas trivial d'assurer les propriétés ACID dans un stockage de données cloud, précisément parce que les données sont répliquées sur plusieurs serveurs. Malgré cette difficulté, des stratégies ont été proposées pour tenter d'émuler les propriétés ACID pour les transactions d'applications Web.

Exemple :
- *l'atomicité peut être garantie en implémentant le protocole de validation en deux phases (2PC).*
- *l'isolement peut être obtenu par un contrôle d'accès simultané multi-versions ou par un horodatage global.*
- *la durabilité en appliquant des stratégies de mise en file d'attente telles que FIFO (First-In, First-Out) aux transactions d'écriture simultanées, afin que les anciennes mises à jour ne remplacent pas les dernières.*

Cependant, la réplication représente un obstacle important pour garantir la cohérence, à fin de garantir cette dernière des stratégies de mise à jour et de propagation des données doivent être mises en œuvre, c.à.d si une copie est mise à jour, toutes les autres doivent également être mises à jour.

De ce fait plusieurs conflits surgissent entre les différents aspects de la haute disponibilité, de la cohérence ainsi que la tolérance de partition dans les systèmes distribués - connus sous le nom de théorème CAP.

Bien que la mise à l'échelle horizontale puisse sembler préférable, le théorème de CAP montre qu'étant donné les partitions réseaux qui sont inévitables dans un scénario géographiquement distribué, nous mettons en évidence le compromis entre cohérence et disponibilité.