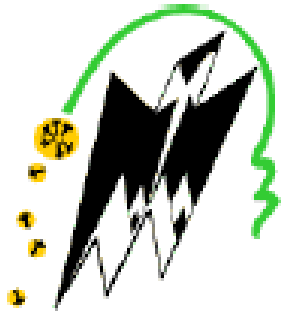
RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ MOULOUD MAMMERI DE TIZI-OUZOU



EN VUE DE L'OBTENTION DU DIPLÔME DE MASTER ACADEMIQUE
SPÉCIALITÉ : INFORMATIQUE
OPTION : SYSTÈMES INFORMATIQUES

THÈME

Mise en place d'un modèle de cohérence orienté Data-centric dans un environnement cloud

Présenté par :
Madjid BOUZOURENE

Devant le jury composé de :
Président(e) : M^{elle} YESLI Yasmine
Examinateur(trice) :M^{r} SAIDAINI Fayçal
Promoteur(trice) : M^{me} TAOURI Dalila

Année universitaire : 2019/2020

Remerciements

D'abord, nous remercions le bon ***DIEU*** *de nous avoir donné santé et courage pour réaliser ce travail.*

Nous tenons à exprimer notre profonde gratitude à notre encadreur ***Mme TAOURI Dalila****, pour nous avoir encadré et guidé et surtout pour ses judicieux conseils qui ont contribué à alimenter notre réflexion.*

Nous remercions chaleureusement les membres de jury pour l'honneur qu'ils nous ont fait en acceptant de juger notre travail.

Nos sincères sentiments vont à nos parents qui ont tant sacrifié jusqu'à aujourd'hui et leurs encouragements tout le long de notre parcours.

Yanis, Madjid.

Table des matières

Table des figures

Liste des tableaux

Première partie

Etat de l'art

Chapitre 1

Le Big Data

Introduction

Ces dernières années la quantité de données générées qui transite chaque jours sur le Web n'a fait que s'accroître de manière gigantesque. Plus de 29.000 Gigaoctets (Go) d'informations sont publiés dans le monde chaque seconde, soit plus de 2,5 trillions d'octets quotidiennement ou 2,5 Exaoctets. Cela regroupe les données d'entreprises (courriels, documents, bases de données, historiques de processeurs métiers...) les données issues de capteurs, les contenus publiés sur le web (images, vidéos, sons, textes), les transactions de commerce électronique, les échanges sur les réseaux sociaux (Facebook, Twitter, Linkedin...), les données transmises par les objets connectés (étiquettes électroniques, compteurs intelligents, smartphones...), les données géolocalisées, etc...

Aujourd'hui, plus de 5 milliards d'individus utilisent chaque jour des données. D'ici 2025, ils seront 6 milliards, soit 75% de la population mondiale. En 2025, chaque personne connectée exploitera des données au moins une fois toutes les 18 secondes. La plupart de ces interactions sont rendues possibles grâce aux milliards d'appareils IoT [1] connectés à travers le monde, lesquels devraient générer plus de 90 Zo de données en 2025.

La croissance explosive du volume de données numériques actuellement en circulation et les demandes toujours croissantes d'extraction de ces dernières nous ont plongés dans une nouvelle ère informatique ont poussé les chercheurs à trouver de nouvelles manières de voir et d'analyser cette quantité phénoménale de données. Il s'agit de découvrir de nouveaux ordres de grandeur concernant la capture, la recherche, le partage, le stockage, l'analyse, l'exploitation et la présentation des données. Ainsi est né le " Big Data ".

Comme chaque domaine de connaissance en expansion, la terminologie naissante Big Data et la science des donnée sont utilisées comme mots à la mode et sont des composites de nombreux concepts explique le NIST [2]. Dans ce chapitre nous allons présenter quelques statistiques ainsi que les concepts et les définitions se rapportant au domaine du "Big Data", les intérêts, contraintes et caractéristiques.

1. Internet Of Things.
2. National Institute of Standards and Technology des États-Unis.

1.1 Historique et statistiques

L'expression " Big Data " apparaît fréquemment dans la presse et dans les revues universitaires, et des programmes de "Data Science» ont vu le jour dans le monde universitaire au cours des six dernières années. Le 29 mars 2012, WHOSTP [3] a annoncé la "Big Data Research and Development Initiative" qui s'appuie sur des initiatives fédérales "allant de l'architecture informatique et des technologies de mise en réseau aux algorithmes, à la gestion des données, à l'intelligence artificielle, apprentissage automatique, développement et déploiement de cyber infrastructures avancées" [1].

Les "Big Data" sont apparues environ 560 fois par an dans JSTOR [4] de 2014 à 2017, même si elles ont été mentionnées moins d'une fois par an au siècle avant 2000 et seulement en moyenne environ huit fois par an entre 2001 et 2010.

Au cours des six dernières années, au moins 17 programmes de science des données ont commencé dans les principales universités de recherche américaines et Internet regorge de publicités pour des livres et des cours de science des données.

Selon l'étude Data Age 2025 [5] La sphère de données mondiale passera de 33 zettaoctets en 2018 à 175 Zo d'ici 2025. Près de 30% des données mondiales devront être traitées en temps réel et le stockage réalisé sur le Cloud public représentera 49% du volume total de données [2].

La figure suivante permet de représenter la quantité de données générée en 60 secondes d'Internet en 2020 :

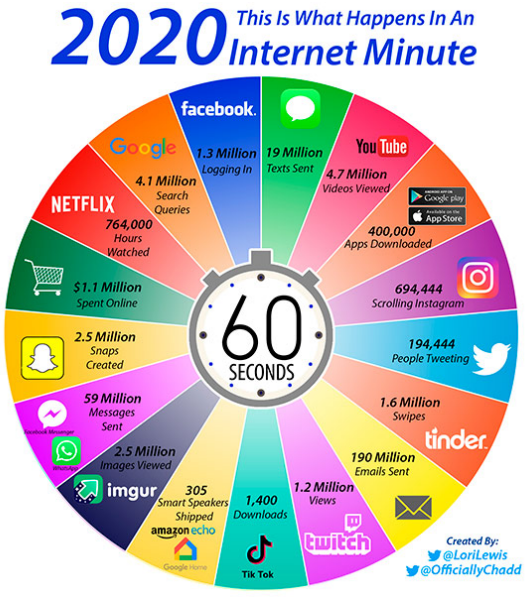


FIGURE 1.1 – 1 minute d'internet en 2020

3. White House Office of Science and Technology Policy en français Le Bureau de la politique scientifique et technologique de la Maison Blanche

4. JSTOR est à la fois un système d'archivage en ligne de publications universitaires et scientifiques et une bibliothèque numérique payante. Fondé en 1995, JSTOR est aujourd'hui une société américaine à but non lucratif basée à New York

5. L'étude Data Age 2025 est une étude sur la digitalisation dans le monde réalisée par l'IDC, établi un comparatif de la digitalisation dans quatre régions (Asie/Pacifique, dont Japon, mais hors Chine) ; Chine ; États-Unis ; EMEA (Europe, Moyen-Orient et Afrique).

En analysant la figure 1.1 on constate l'énorme quantité de données qui circule. En effet, cette année à chaque minute d'Internet, sont envoyés plus de 59 millions de messages. Chaque minute, Google enregistre près de 4,1 millions de requêtes différentes sur son moteur de recherche. Facebook enregistre près de 1,3 de logins.

1.2 Quelques définitions liées au Big Data

Aucune définition précise ou universelle ne peut être donnée au Big Data. Etant un objet complexe polymorphe, sa définition varie selon les communautés qui s'y intéressent en tant qu'usager ou fournisseur de services.

Définition 1. Le Big Data désigne l'ensemble des données numériques produites par l'utilisation des nouvelles technologies à des fins personnelles ou professionnelles. Cela regroupe les données d'entreprise ? des contenus publiés sur le web, des transactions de commerce électronique, des échanges sur les réseaux sociaux, des données transmises par les objets connectés des données géolocalisées, ...etc [3].

Définition 2. Le "Big Data" désigne les technologies et les initiatives qui impliquent des données trop diverses, en évolution rapide ou massives pour que les technologies, les compétences et les infrastructures conventionnelles puissent être traitées efficacement. Autrement dit, le volume, la vitesse ou la variété des données est trop important [4].

Définition 3. Le terme Big Data fait référence aux données dont le coût de stockage, de gestion et d'analyse dans des systèmes de base de données traditionnels (relationnels et/ou monolithiques) serait généralement trop élevé. Habituellement, ces systèmes ne sont pas rentables, car ils ne disposent pas de la flexibilité nécessaire pour stocker des données non structurées (comme des images, du texte et des vidéos), pour accommoder des données "à haute vélocité" (en temps réel) ou pour s'adapter automatiquement à de très gros volumes de données (de l'ordre du pétaoctet) [5].

1.3 Intérêt du Big Data

L'information est l'arme fatale du XXIème siècle. Celui qui détient l'information, a le pouvoir. C'est ce que nous répètent tous les experts géopolitiques et tous les économistes. Le Big Data en est maintenant la plus grande source et il est désormais généré tout autour de nous.

Avec la croissance de l'Internet des objets, l'utilisation des smartphones, des médias sociaux des applications dans le Cloud et autres technologies d'intelligence artificielle, le Big Data est devenu plus important que jamais et cela au sein de plusieurs secteurs d'activité mais plus principalement au niveau des services marketing. Plusieurs entreprises et organisations de toutes tailles utilisent maintenant le Big Data comme moyen d'obtenir davantage d'informations pour :

- Améliorer les offres de l'entreprise en offrant une analyse complète du comportement et des attentes du consommateur.

Exemple : *Google Analytics*[6] *permet par exemple d'optimiser un site web par une analyse en temps réel des données liées : nombre de visites, comportement de navigation, taux de rebond, nombre de pages lues, taux de clics..etc*

- Anticiper les besoins et la demande des clients et adapter ainsi les campagnes marketing.

  Exemple : *La consultation d'appareils photos sur le site de la Fnac*[7] *et le fait de retrouver des bannières d'appareils photo sur tous les sites que l'on fréquente est un avantage du Big Data (certains diront que c'est plutôt un inconvénient... !).*

- Prédire des faits ou générer des statistiques futur.

  Exemple : *Target*[8] *, aux Etats-Unis, parvient avec le Big Data à prédire l'accouchement prochain de femmes enceintes.*

- Optimiser la logistique et l'organisation.

  Exemple : *Il permet, par exemple, de suivre les ventes en temps réel et donc d'optimiser la gestion des stocks. Amazon*[9] *teste même régulièrement des fonctionnalités de livraison intuitive, visant à proposer la livraison à la commande. .*

- Le Big Data permet aussi de réduire les coût d'organisation et d'augmenter la productivité, de fidéliser la marque en analysant les comportement de prospects et des clients et améliore l'expérience de ces derniers.

pour résumer, Le Big Data permet de construire de meilleurs modèles, qui produisent des résultats plus précis avec des approches extrêmement innovantes concernant la manière dont :

- Les entreprises se commercialisent et vendent leurs produits.
- La gestion des ressources humaines.
- La réaction au catastrophes naturelles.

Ces exemples ne sont finalement qu'une poignée des opportunités qu'offre le Big Data. Les entreprises, et pas seulement, devront faire preuve d'imagination, d'organisation et d'un énorme sens d'analyse pour prendre la pleine mesure du phénomène. De cette maîtrise découle de nouveaux usages qui bouleverse notre façon de concevoir Internet.

6. Google Analytics est un service gratuit d'analyse d'audience d'un site Web ou d'applications utilisé par plus de 10 millions de sites, soit plus de 80 % du marché mondial
7. La Fnac « Fédération nationale d'achats », puis « Fédération nationale d'achats des cadres ») est une chaîne de magasins française spécialisée dans la distribution de produits culturels et électroniques à destination du grand public
8. Target Corporation, située à Minneapolis dans le Minnesota est une entreprise de grande distribution. Elle est le deuxième plus gros distributeur discount et le cinquième distributeur général aux États-Unis
9. Amazon est une entreprise de commerce électronique américaine basée à Seattle. Elle est un des géants du Web, regroupés sous l'acronyme GAFAM5, aux côtés de Google, Apple, Facebook et Microsoft.

Mais le Big Data, ce n'est pas que des avantages. C'est aussi des contraintes, nous verrons ces dernières dans la section suivante.

1.4 Les contraintes du Big Data

L'idée est de comprendre que les entreprises et les organisations collectent et exploitent de grands volumes de données pour améliorer leurs produits finaux, que ce soit la sécurité, la fiabilité, les soins de santé ou la gouvernance. En général, dans les affaires, l'objectif est de transformer autant de données en une forme d'avantage commercial. La question est de savoir comment utiliser des volumes de données plus importants pour améliorer la qualité de notre produit final ? Malgré un certain nombre de défis qui y sont liés.

Certaines publications [6] discutent des obstacles au développement d'applications de mégadonnées. Les principaux défis sont énumérés comme suit :

- **Représentation des données :** De nombreux ensembles de données présentent certains niveaux d'hétérogénéité dans le type, la structure, la sémantique, l'organisation, la granularité et l'accessibilité. La représentation des données vise à rendre les données plus significatives pour l'analyse informatique et l'interprétation des utilisateurs. Néanmoins, une représentation incorrecte des données réduira la valeur des données originales et peut même empêcher une analyse efficace des données.

- **Réduction de la redondance et compression des données :** En général, il existe un niveau élevé de redondance dans les jeux de données. La réduction de la redondance et la compression des données sont efficaces pour réduire le coût indirect de l'ensemble du système en partant du principe que les valeurs potentielles des données ne sont pas affectées. Par exemple, la plupart des données générées par les réseaux de capteurs sont hautement redondantes.

- **Gestion du cycle de vie des données :** Par rapport aux progrès relativement lents des systèmes de stockage, la détection et le calcul omniprésents génèrent des données à des taux et des échelles sans précédent. Nous sommes confrontés à de nombreux défis urgents, dont l'un est que le système de stockage actuel ne peut pas supporter des données aussi massives. De manière générale, les valeurs cachées dans le Big Data dépendent de la fraîcheur des données.

- **Mécanisme analytique :** Le système analytique des mégadonnées traitera des masses de données hétérogènes dans un temps limité. Cependant, les SGBDR traditionnels sont strictement conçus avec un manque d'évolutivité et d'extensibilité, ce qui ne pourrait pas répondre aux exigences de performance. Les bases de données non relationnelles ont montré leurs avantages uniques dans le traitement des données non structurées et ont commencé à se généraliser dans l'analyse des mégadonnées. Même ainsi, il existe encore quelques problèmes de bases de données non relationnelles dans leurs performances et applications particulières. Des recherches supplémentaires sont nécessaires sur la base de données en mémoire et des échantillons de données basés sur une analyse approximative.

- **Confidentialité des données :** La plupart des fournisseurs ou propriétaires de services de mégadonnées ne pouvaient actuellement pas maintenir et analyser efficacement des ensembles de données aussi énormes en raison de leur capacité limitée. Ils doivent s'appuyer sur des professionnels ou des outils pour analyser ces données,

ce qui augmente les risques potentiels pour la sécurité. Par exemple, l'ensemble de données transactionnelles comprend généralement un ensemble de données d'exploitation complètes pour piloter les processus métier clés. Ces données contiennent des détails et certaines informations sensibles telles que les numéros de carte de crédit.

- **Gestion de l'énergie :** La consommation d'énergie des systèmes informatiques a beaucoup attiré l'attention du point de vue économique et environnemental. Avec l'augmentation du volume de données et des demandes analytiques, le traitement, le stockage et la transmission de données massives consommeront inévitablement de plus en plus d'énergie électrique.

- **Expendabilité et évolutivité :** Le système analytique du Big Data doit prendre en charge les ensembles de données présents et futurs. L'algorithme analytique doit être capable de traiter des ensembles de données de plus en plus étendus et plus complexes.

- **Coopération :** L'analyse du Big Data est une recherche interdisciplinaire, qui nécessite la coopération d'experts dans différents domaines pour exploiter le potentiel du Big Data. Une architecture de réseau Big Data complète doit être mise en place pour aider les scientifiques et les ingénieurs dans divers domaines à accéder à différents types de données et à utiliser pleinement leur expertise, afin de coopérer pour atteindre les objectifs analytiques.

1.5 Les caractéristiques du Big Data

Les mégadonnées sont un terme générique utilisé pour désigner toute collection de données volumineuse et complexe qui peuvent dépasser la capacité de traitement des systèmes et techniques de gestion de données conventionnels. Les applications du Big Data sont infinies.

Les mégadonnées sont souvent caractérisées par **le volume** extrême des données, la grande **variété** de types de données et **la vitesse** à laquelle les données doivent être traitées.(Ces caracteristique sont dites les 3V)

Ces caractéristiques ont été identifiées pour la première fois par l'analyste **Douglas Laney's** membre du Gartner[10] dans un rapport publié en 2001. Plus récemment, plusieurs autres caractéristiques(autres V) ont été ajoutées aux descriptions des mégadonnées, notamment **la véracité**, **la valeur** et **la variabilité**. Bien que les mégadonnées ne correspondent à aucun volume de données spécifique, le terme est souvent utilisé pour décrire des téraoctets, des pétaoctets et même des exaoctets de données capturées au fil du temps.

10. Le Gartner est une entreprise américaine de conseil et de recherche dans le domaine des techniques avancées. Elle mène des recherches, fournit des services de consultation, tient à jour différentes statistiques et maintient un service de nouvelles spécialisées.

Figure 1.2 – Les 3V du Big Data

Certaines personnes attribuent encore plus de V aux Big Data ; data scientist et les consultants ont créé diverses listes contenant entre sept et 10 V.
On donne dans ce qui suit 10 caractéristiques "10V" sur les megadonnées sachant que Les 3 premier critères basiques du Big Data sont **le volume la vitesse** ainsi que **la variété** :

1. **Le Volume** Volume :
   Le volume fait référence aux vastes quantités de données générées chaque seconde, en minutes, en heures et en jours dans notre monde numérisé et qui peuvent provenir de vastes ensembles de données partagés ou de nombreuses petites données et événements collectés au fil du temps. En résumé, le volume est la dimension du Big Data liée à sa taille et à sa croissance exponentielle.

   Les défis liés à l'utilisation de volumes de mégadonnées incluent le coût, l'évolutivité et les performances liées à leur stockage, leur accès et leur traitement [7].

   Exemple : *300 heures de vidéo sont mise en ligne sur YouTube chaque minute.*

2. **La Variété** Variety :

   Elle fait référence aux différentes formes toujours croissantes que les données peuvent prendre, en effet les données Big Data ne sont pas seulement des nombres, des dates et des chaînes. Les mégadonnées englobent également une grande variété de types de données, y compris des données structurées dans des bases de données SQL et des entrepôts de données, des données non structurées, telles que des fichiers texte et document conservés dans des clusters Hadoop ou des systèmes NoSQL, et des données semi-structurées, telles que des journaux de serveur Web ou la diffusion des données à partir de capteurs [4].

Exemple : *Un projet d'analyse des mégadonnées peut tenter d'évaluer le succès d'un produit et les ventes futures en corrélant les données de ventes passées, les données de retour et les données de révision des acheteurs en ligne pour ce produit.*

3. **La Vitesse** Velocity :

Elle Fait référence à la vitesse à laquelle les données sont générées et à la vitesse à laquelle les données se déplacent d'un point au suivant. Les progrès technologiques ont provoqué une production de données à grande échelle à grande vitesse. Les données en streaming au rythme sans précédent doivent être traitées en temps opportun. Le traitement de ces données en temps réel pour correspondre à leur taux de production est le principal objectif de l'analyse du Big Data.
La vélocité est également importante car l'analyse des mégadonnées s'étend à des domaines tels que l'apprentissage automatique et l'intelligence artificielle (IA), où les processus analytiques trouvent automatiquement des modèles dans les données collectées et les utilisent pour générer des informations.

Exemple : *Facebook est un exemple très courant d'un environnement qui traite des données à grande vitesse et à grand volume. Selon les enregistrements de janvier 2020, 5 milliards de commentaires sont laissés sur les pages Facebook chaque mois, Le bouton J'aime de Facebook a été enfoncé 1,13 billion de fois.Toutes les 60 secondes, 317 000 mises à jour de statut, 147 000 photos téléchargées, et 54 000 liens partagés [8] [9].*

Remarque : De plus en plus de **Vs** ont été introduits dans la communauté du Big Data au fus et à mesure que nous découvrons de nouveaux défis et de nouvelles façons de définir le Big Data. La **véracité** et la **valeur** sont deux de ces V supplémentaires.

4. **La Véracité** Veracity :

Elle fait référence aux biais, au bruit et aux anomalies dans les données. Ou, mieux encore, il fait référence aux incertitudes et à la fiabilité des données souvent incommensurables.

Exemple : *Dans le cadre d'un sondage réalisé par IBM, 27% des entreprises interrogées avouent ne pas être certaines de l'exactitude des données qu'elles collectent. De même, un chef d'entreprise sur trois utilise les données pour prendre des décisions, mais n'a pas vraiment confiance. Ce manque de véracité et de qualité des données coûte environ 3,1 trillions de dollars par an aux États-Unis.*

5. **Valeur** Value :

   Toutes les données collectées n'ont pas une valeur commerciale réelle et l'utilisation de données inexactes peut affaiblir les informations fournies par les applications d'analyse. Il est essentiel que les organisations utilisent des pratiques telles que le nettoyage des données et confirment que les données sont liées à des problèmes commerciaux pertinents avant de les utiliser dans un projet d'analyse de Big Data.

   On peut dire que les autres caractéristiques du Big Data n'ont pas de sens si on ne tire pas de valeur commerciale de ces données. Les Données massives offrent une valeur substantielle : comprendre mieux les clients. Les cibler en conséquence, optimiser les processus et améliorer les performances de la machine ou de l'entreprise. Avant de se lancer dans une stratégie Big Data, on doit comprendre le potentiel et les caractéristiques les plus difficiles [10].

   Exemple : *La mise en place d'une analyse Big Data a permis à la société de développement d'éoliennes Vestas*[11] *d'optimiser son processus d'identification des meilleurs emplacements pour implanter ses éoliennes. Le traitement Big Data a engendré une augmentation de la performance de production d'électricité et une réduction des coûts énergétiques associés.*

   Remarque : Certaines personnes attribuent encore plus de V aux Big Data ; les scientifiques des données et les consultants ont créé d'autres listes en ajoutant la **variabilité**, la **validité**, la **visualisation**, la **volatilité** ainsi que la **vulnérabilité**.

6. **Variabilité** Variability :

   La variabilité dans le Big Data fait référence à plusieurs sens. Dans un premier temps elle désigne le nombre d'incohérences dans les données. Celles-ci doivent être détectées par des techniques de détection d'anomalies et de valeurs aberrantes pour faciliter la création d'analyse significative.

   Les mégadonnées sont également variables en raison de la diversité de dimensions résultant de multiples types et sources de données. La variabilité peut également faire référence à la vitesse incohérente à laquelle les données volumineuses sont chargées dans la base de données [10].

   Exemple :*L'équipe d'IBM*[12] *fait participer Watson*[13] *au célèbre jeu télévisé américain Jeopardy, un jeu ou les candidats doivent trouver les réponses à des questions posées. Watson devait "être capable de comprendre l'énoncé des questions, buzzer pour prendre la main,disséquer une réponse dans son sens pour déterminer quelle était la bonne question". Les mots n'ont pas de définitions statiques et leur signification peut varier énormément dans le contexte.*

11. Vestas Wind Systems A/S, communément appelée Vestas, est une entreprise danoise fabricant d'éoliennes. Vestas a installé plus de 59 909 machines à travers le monde (82 GW) et a réalisé en 2017 un chiffre d'affaires de presque 10 milliards d'euros.
12. International Business Machines Corporation, connue sous le sigle IBM, est une société multinationale américaine présente dans les domaines du matériel informatique, du logiciel et des services informatiques.
13. Watson est un programme informatique d'intelligence artificielle conçu par la société IBM dans le but de répondre à des questions formulées en langage naturel.

7. **Validité** Validity :

   Similaire à la véracité, la validité fait référence à la précision et à la correction des données pour l'usage auquel elles sont destinées. Selon Forbes [14], environ 60 % du temps d'un scientifique est consacré au nettoyage de ses données avant de pouvoir effectuer une analyse. L'avantage de l'analyse des données massives est aussi primordiale que celui des données sous-jacentes. On doit donc avoir de bonnes pratiques de gouvernance des données pour garantir une qualité des données cohérente, des définitions communes et des métadonnées.

   Exemple : *La date d'une transaction est « 02/07/1994 » alors que l'activité de la société a débuté en 2000.*

8. **Volatilité** Volatility :

   On se pose les questions :«quel âge doivent avoir les données pour qu'elles soient considérées comme non pertinentes, historiques ou obsolète ?» «Combien de temps faut-il conserver les données ?»
   Avant l'ère Big Data, en général, on stockait les données indéfiniment. Quelques téraoctets de données ne pouvaient pas engendrer de dépenses de stockage élevées.

   En raison de la vitesse et du volume de ces données massives, leur volatilité doit être soigneusement prise en compte. Il est maintenant fondamental d'établir des règles pour la disponibilité et à la mise à jour des données afin de garantir une récupération rapide des informations en cas de besoin [10].

   Exemple : *Une entreprise e-commerce peut ne pas souhaiter conserver un historique des achats client d'un an. Parce qu'après un an la garantie par défaut sur leur produit expire, il n'y a donc aucune possibilité de restauration ces données.*

9. **Visualisation** Visualization :

   Une autre caractéristique du Big Data est la difficulté à les visualiser. Les logiciels de visualisation de données volumineuses actuels sont confrontés à des problèmes techniques en raison des limitations de la technologie en mémoire, de leur faible évolutivité, de leur fonctionnalité et de leur temps de réponse. Il est impossible de se fier aux graphiques traditionnels lorsque on essaye de tracer un milliard de points de données. Il est donc nécessaire d'avoir différentes manières de représenter des données. Telles que la mise en cluster de données ou l'utilisation de cartes d'arbres, de sunbursts, de coordonnées parallèles, de diagrammes de réseau circulaires ou de cônes.

   Si on associe cela avec la multitude de composante résultant de la variété et de la vélocité des données massives et des relations complexes qui les lient, il est possible de voir qu'il n'est pas si simple de créer une visualisation significative [10].

14. Forbes est un magazine économique américain fondé en 1917 par Bertie Charles Forbes.

Prenons l'exemple : *du tableau suivant qui fait apparaître deux séries de chiffres : le chiffre d'affaires "France" et le chiffre d'affaires "Reste du monde". La lecture de ce tableau et sa signification ne sont pas immédiates.*

kEur	Janv	Fevr	Mars	Avril	Mai	Juin	Juillet	Août	Sept	Oct	Nov	Déc
France	10000	12000	14000	13000	15444	17028	15000	15804	18000	17500	19000	20958
Reste du monde	3444	3816	4038	3558	3864	4074	3558	2000	3594	3498	3612	4140
	13444	15816	18038	16558	19308	21102	18558	17804	21594	20998	22612	25098

FIGURE 1.3 – Évolution du chiffre d'affaires par région.

Mais si nous représentons les séries de chiffres sous forme graphique (ci-dessous),on comprend en un coup d'œil que le chiffre d'affaires "France" progresse et que le chiffre d'affaires "Reste du monde" stagne.

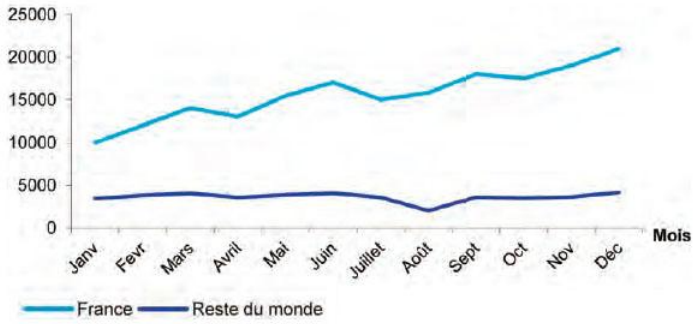


FIGURE 1.4 – Évolution du chiffre d'affaires par région.

10. **Vulnérabilité** Vulnerability :

Le Big Data apporte de nouveaux problèmes de sécurité. Malheureusement, il y a quotidiennement des violations de données massives.

Exemple :*Rapporté par CRN*[15] *: en mai 2016, "un pirate informatique appelé Peace a posté des données sur le web sombre pour les vendre, qui auraient inclus des informations sur 167 millions de comptes LinkedIn et 360 millions d'e-mails et de mots de passe pour les utilisateurs de MySpace*[16] *"[10].*

Remarque : Il existe d'autres sources qui parlent de 54 Vs pour les caractéristiques du Big Data tel que la **venue** (le lieu), **valence**, **vocabulaire**, **vagueness** (imprécision) ...etc .

15. CRN, anciennement Computer Reseller News, est un magazine commercial américain traitant d'informatique. Il cible principalement le milieu de la revente informatique
16. Myspace est un site web de réseautage social fondé aux États-Unis en août 2003, qui met gratuitement à disposition de ses membres enregistrés un espace web personnalisé, permettant de présenter diverses informations personnelles et d'y faire un blog

1.6 Les sources du Big Data

Pour réussir avec le Big Data, il est important que les entreprises disposent du savoir-faire pour passer en revue les différentes sources de données disponibles et classer en conséquence leur convivialité et leur pertinence [11].
Deux classifications des sources du Big Data sont donnés dans ce qui suit :

1. Selon qu'elles soient interne ou externe à l'entreprise.
2. Selon leur provenance.

1.6.1 Les données internes ou externes à l'entreprise

Les données sont internes si une entreprise les génère, les possède et les contrôle .

Exemple :

- Module ERP d'entreprise.
- Capteurs, contrôleurs.
- Centres d'appels internes.
- Journaux de site Web.

Les données externes sont des données publiques ou des données générées en dehors de l'entreprise ; en conséquence, la société ne la possède ni ne la contrôle . **Exemple :**

- Médias sociaux.
- Statistiques officielles.
- Prévisions météorologiques.
- Ensembles de données accessibles au public pour l'apprentissage automatique.

1.6.2 Les sources du Big Data selon leur provenances :

Les données les plus volumineuse sont utilisées aujourd'hui par les organisations et les entreprises dans le seul but d'effectuer des analyses. Toutefois, avant de pouvoir extraire des informations et des renseignements précieux à partir de données importantes, ces dernières doivent connaître plusieurs sources de données importantes. Les données, comme nous le savons, sont massives et existent sous diverses formes. Si elles ne sont pas bien classées ou si elles ne proviennent pas d'une source sure, elles peuvent finir par faire perdre du temps et des ressources précieuses. Afin de réussir avec les données volumineuses, il est important que les entreprises aient le savoir-faire nécessaire pour faire le tri entre les différentes sources de données disponibles et classer en conséquence leur utilité et leur pertinence.

Figure 1.5 – Les 5 sources du Big Data.

1.6.2.1 Les médias :

Les médias sont la source la plus populaire de mégadonnées, car ils fournissent des informations précieuses sur les préférences des consommateurs et l'évolution des tendances. Ces données sociales proviennent des mentions J'aime, des tweets et des retweets, des commentaires, des téléchargements de vidéos et des médias généraux qui sont téléchargés et partagés via les plateformes de médias sociaux préférées au monde. Ce type de données fournit des informations inestimables sur le comportement et le sentiment des consommateurs et peut être extrêmement influent dans les analyses marketing.

Exemple : *Les médias comprennent les médias sociaux et les plates-formes interactives, telles que Google, Facebook, Twitter, YouTube, Instagram, ainsi que des supports génériques tels que des images, des vidéos, des audios et des podcasts qui fournissent des informations sur l'interaction utilisateur.*

1.6.2.2 Le Cloud :

Aujourd'hui, avec la croissance accélérée du volume de données utilisé par les applications, de nombreuses organisations ont déplacé leurs données vers des serveurs Cloud pour fournir des services évolutifs, fiables et hautement disponibles [12].

Le stockage Cloud héberge des données structurées et non structurées et fournit aux entreprises des informations en temps réel et des informations à la demande. Le principal attribut du Cloud computing est sa flexibilité et son évolutivité. Comme les mégadonnées peuvent être stockées et sourcées sur des Clouds publics ou privés, via des réseaux et des serveurs, le Cloud constitue une source de données efficace et économique car les ressources sont fournis à la demande et on ne paye que les ressources utilisées [11].

1.6.2.3 Le Web :

Le Web public constitue une source répandu et facilement accessible pour le Big Data. Les données sur le Web ou «Internet» sont généralement disponibles pour les particuliers et les entreprises. De plus, des services Web tels que Wikipedia fournissent des informations gratuites et rapides à tout le monde. Le Web public est également une autre bonne source de données sociales, et des outils comme Google Trends peuvent être utilisés à bon escient pour augmenter le volume de données.

L'énormité du Web garantit sa polyvalence et est particulièrement bénéfique pour les start-ups et les PME, car elles n'ont pas à attendre pour développer leur propre infrastructure et référentiels de Big Data avant de pouvoir tirer parti du Big Data.

1.6.2.4 L'Internet Des Objets (IoT) :

Les données machine sont définies comme des informations générées par des équipements industriels, des capteurs installés dans des machines et même des journaux Web qui suivent le comportement de l'utilisateur. En logistique, il peut s'agir de capteurs qui servent à la traçabilité des biens pour la gestion des stocks et les acheminements.
En domotique, l'IoT recouvre tous les appareils électroménagers communicants, les capteurs , les compteurs intelligents et systèmes de sécurité connectés des appareils de type box domotique. Le phénomène IoT est également très visible dans le domaine de la santé et du bien-être avec le développement des montres connectées, des bracelets connectés et d'autres capteurs surveillant des constantes vitales [6].

Exemple : *En Tennis, la marque Zepp propose un système de capteur connecté qui analyse vos performances sur le terrain. Il suffit de fixer le capteur sur le manche de n'importe quelle raquette de tennis et de se mouvoir pour obtenir un retour d'information instantané sur iPhone, iPad ou appareil Android.*

1.6.2.5 Les bases de données :

Les entreprises préfèrent aujourd'hui utiliser une fusion de bases de données traditionnelles et modernes pour acquérir des mégadonnées pertinentes. Cela permet d'ouvrir la voie à un modèle de données hybride avec un faible coût d'investissement. De plus, ces bases de données sont également déployées à plusieurs fins de Business Intelligence. Le processus d'extraction et d'analyse de données parmi de vastes sources de mégadonnées est un processus complexe qui peut être frustrant. Ces complications peuvent être résolues si les organisations englobent toutes les considérations nécessaires des mégadonnées, et prennent en compte les sources de données pertinentes et les déploient d'une manière bien adaptée à leurs objectifs organisationnels.

Exemple : *Les bases de données les plus populaires Microsoft SQL Server, MongoDB, Oracle, MySQL et IBM Db2 ..etc.*

1.7 Les nouveaux métiers du Big Data

Les emplois Big Data, ou métiers de la donnée, sont de plus en plus nombreux. Les entreprises de tous les secteurs cherchent désormais à exploiter les données à leur disposition pour aiguiller leur stratégie et leur développement.Toutefois, pour être en mesure d'exploiter ces données, les entreprises doivent s'appuyer sur des compétences et du savoir-faire de professionnels hautement qualifiés capables d'utiliser les technologies analytiques. Ainsi, le Big Data a donné naissance à de nombreux nouveaux métiers : Data Miner, Data Analyst, Chief Data Officer, Data Architect, Data Scientist... Ces experts sont en mesure de collecter des données, de les organiser, de les traiter, et de les transformer en informations exploitables par tous les départements de l'entreprise.

Note : *Selon le Robert Half 2018 Salary Guide for Technology Professionals, les emplois Big Data seront les plus demandés au cours des années à venir. 67 % des exécutifs d'entreprises affirment que le Big Data est l'un des principaux facteurs de recrutement, au même titre que le marketing numérique et le Cloud.*

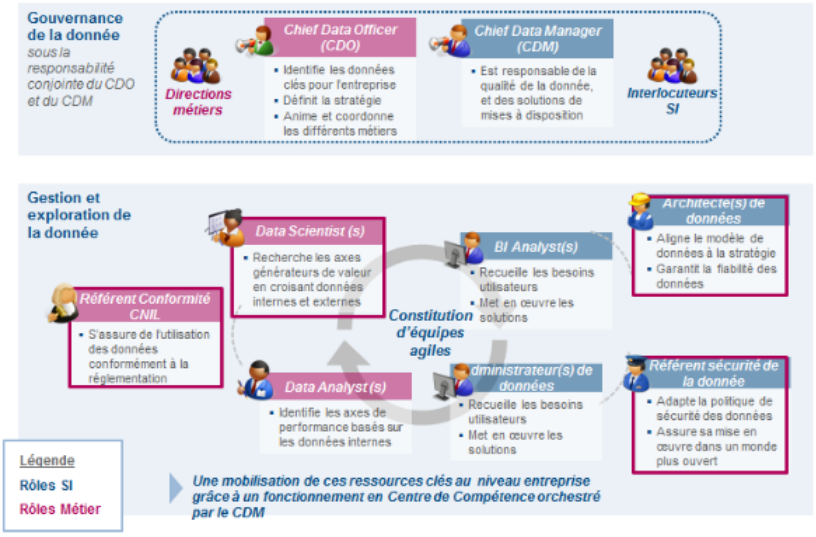


Figure 1.6 – Exemple d'emplois Big Data.

Métiers	Description
Chief data officer	Appelée Directeur de données ou bien Directeur de la transformation digitale est l'un des métiers du Big Data, Il cumule de nombreux savoir faire (marketing, stratégie, IT, business, innovation, logique d'entreprise, process) et savoir être (communication, ouverture aux autres, capacités à convaincre. . .) pour prendre en charge cette fameuse mutation numérique.
Big Data Architect	Tout comme les architectes créent des structures physiques, les architectes de données élaborent des schémas pour des systèmes de gestion de données. Il s'agit de l'un des métiers du Big Data qui combine les compétences en design et en implémentation de systèmes de gestion de données. C'est la personne qui se charge de collecter des données brutes pour l'entreprise. Leur quantité peut aussi varier énormément. Après avoir collecté les données brutes, l'architecte Big Data se charge de créer et d'optimiser des infrastructures de stockage, manipulation et restitution. Il doit élaborer une architecture de Data Management et concevoir un plan pour intégrer, centraliser, protéger et maintenir les données.
Data engineer	L'ingénieur Big Data collabore avec des entreprises pour développer, entretenir, tester et évaluer des solutions Big Data. La plupart du temps, ils utilisent leurs compétences en technologies Hadoop comme MapReduce, HiveMongo DB, ou Cassandra. L'ingénieur Big Data crée des systèmes de traitement de données à grande échelle. Il est également expert en solutions de Data Warehousing et en technologies de bases de données.
Data scientist	Le Data Scientist est chargé de développer les algorithmes nécessaires pour poser les bonnes questions et obtenir les bonnes réponses afin de tirer des informations pertinentes de vastes ensembles de données. Il est donc nécessaire d'avoir de solides connaissances en statistiques, en mathématiques, en modèles prédictifs et en stratégie d'entreprise.
Data Analyst	Le métier de Data Analyst a commencé à apparaître dans les appels d'offres ces derniers temps car les entreprises ont ressenti le besoin d'avoir les données valorisées et synthétisées sous forme d'indicateurs de performance , et de tableaux de bords. Le Data Analyst aide les entreprises à véritablement consommer les données mis en forme par le Data Engineer ou les résultats renvoyés par les modèles du Data Scientist pour la prise de décision effective. C'est un métier à l'intersection de la Business Intelligence et de l'ingénierie Big Data. Le Data Analyste maîtrise les outils de reporting, de visualisation , l'outil de suivi par excellence des décideurs , la programmation VBA, le SQL, et possède de très bonnes aptitudes communicationnelles pour échanger avec les décideurs de l'entreprise sur la signification des indicateurs calculés sur la base des données.
Data miner	Son rôle assez proche de celui de Data Analyst, il est responsable principalement de la fouille de données.
Data protection officer	C'est le chef d'orchestre de la conformité. Les responsables le consultent lors du traitement de données à caractère personnel, aussi bien en ce qui concerne la sécurité informatique que la sécurité juridique.
Business intelligence manager	Il gère une équipe de développeurs et d'analystes,Ce gestionnaire est dans un premier temps chargé d'identifier les besoins de l'entreprise en matière de Business Intelligence. Il doit analyser les informations pertinentes, et fournir des rapports détaillés basés sur ces analyses pour permettre à l'entreprise d'agir.

TABLE 1.1 – Métiers et description d'emplois dans Big Data.

1.8 Les technologies du Big Data

Le développement d'applications Big Data est devenu de plus en plus important ces dernières années. En fait, plusieurs organisations de différents secteurs dépendent de plus en plus des connaissances extraites d'énormes volumes de données. Cependant, dans le contexte du Big Data, les techniques et plateformes de données traditionnelles sont moins efficaces. Ils montrent une réactivité lente et un manque d'évolutivité, de performances et de précision. Pour faire face aux défis complexes du Big Data, plusieurs technologies ont été développées pour répondre à plusieurs problématiques comme l'analyse prédictive plus particulièrement dans le cadre de maintenance préventive ou encore de prédiction des ventes et gestion des stocks. L'analyse des données en temps réel est aussi une des applications du Big Data [13].

Nous décrivons ici tous les composants qui font partie des solutions Big Data sous de nombreux angles : matériel, méthodologies, logiciels et applications de base, etc.
Pour mieux catégoriser ces concepts, nous les avons a répartis en différentes sections selon l'objectif visé par chacun. Ces catégories sont : l'**infrastructure**, le **stockage**, le **traitement** et les **composants de haut niveau**.

1.8.1 Les Infrastructures

Le développement du Big Data commence avec les **cluster Big Data** [17] qui exécutent en parallèle les instructions d'un logiciel de haut niveau. Le cluster est partitionné en deux types de nœuds selon la fonction principale exercée [14] :

- Nœuds de données ou esclaves (informatique).
- Nœuds de gestion ou maîtres (gestion).

Outre leurs fonction, le maître et les esclaves peuvent être différenciés par leurs capacités de calcul et leur quantité dans le champ de nœuds.

Les esclaves sont chargés de surveiller les données partitionnées, de traiter et d'interroger les données locales. Les unités de données et de traitement doivent être aussi proches que possible pour éviter les retards introduits par les mouvements entre les partitions. Les nœuds de données sont gourmands en disque et standard en termes de capacités de calcul et de mémoire.

Les maîtres reçoivent et transforment les programmes des applications clientes en instructions parallèles qui peuvent être comprises par les esclaves. Une fois que les applications clientes ont atteint le démon maître, elles finissent par démarrer ou réveiller plusieurs processus dans les esclaves qui retournent finalement une sortie suivant la direction opposée. Parmi l'ensemble des responsabilités approuvées pour les nœuds de gestion figurent :

- La récupération après défaillance.
- La gestion des ressources.
- La planification des travaux.
- La surveillance ou la sécurité.

17. Un cluster Big Data est un type spécial de cluster de calcul conçu spécifiquement pour stocker et analyser d'énormes quantités de données non structurées dans un environnement informatique distribué.

Pour accomplir ces tâches, les maîtres nécessitent une puissance de calcul et de mémoire élevée. Dans les clusters Big Data standard, il suffit de garder deux maîtres support qui se surveillent mutuellement.

Les deux types de nœuds sont connectés via une connexion réseau, généralement LAN (Ethernet ou InfiniBand). Certaines configurations permettent également de connecter les maîtres de différents centres de données sur un réseau WAN pour éviter facilement les défaillances du système. Dans chaque centre de données, le maître et les esclaves sont interconnectés en privé pour ingérer des données, déplacer des données entre les nœuds et effectuer des requêtes. Il existe également un autre réseau public qui sert de façade entre le client et le service de gestion (ssh, VNC, interface web, ...).

1.8.2 Les technologies de traitement

Dans cette section nous parlerons de l'arrivée des technologies de traitement ajustées, plus spécialement sur la mise au point de modes de calcul à haute performance (MapReduce), nous parlerons de (Hadoop) une solution de Big Data très largement utilisée pour effectuer des analyses sur de très grands nombres de données, et enfin nous clôturons cette section avec le développement de nouvelles bases de données adaptées aux données non structurées (NoSQL) et nous verrons qu'est se que le NewSQL [15].

1. **Hadoop :**
Hadoop est un framework logiciel open source permettant de stocker des données, et de lancer des applications sur des grappes (cluster) de machines standards. Cette solution offre un espace de stockage massif pour tous les types de données, une immense puissance de traitement et la possibilité de prendre en charge une quantité de tâches virtuellement illimitée. Basé sur Java, ce framework fait partie du projet Apache, sponsorisé par Apache Software Foundation.

Grâce au framework MapReduce, il permet de traiter les immenses quantités de données. Plutôt que de devoir déplacer les données vers un réseau pour procéder au traitement, MapReduce permet de déplacer directement le logiciel de traitement vers les données.
Dans son principe, Hadoop se compose essentiellement de :

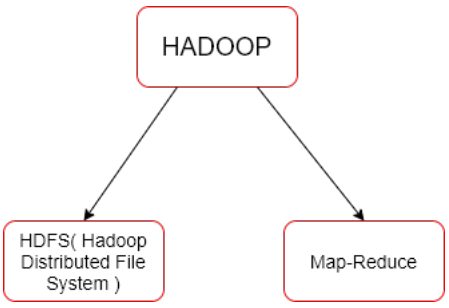


FIGURE 1.7 – Les composants d'Hadoop.

(a) **Système de gestion de fichiers HDFS :** (Hadoop Distributed File System)

HDFS est est un système de fichiers distribué, extensible et portable Inspiré

par GFS [18] et écrit en Java. Il est conçu pour être un système de stockage distribué, évolutif et résilient, conçu pour interagir facilement avec MapReduce. Il fournit une bande passante d'agrégation importante tout au long du réseau. Comme pour GFS, un réseau HDFS est composé d'un nœud maître appelé Namenode et des serveurs de données appelés Datanodes, comme expliqué dans la **Figure 1.8**. La structure du fichier HDFS est divisée en blocks d'octets de grande taille par défaut 64 Mo pour optimiser les temps de transfert et d'accès. Il est toutefois possible de monter à 128 Mo, 256 Mo, 512 Mo voire 1 Go[16].

Ces blocs sont ensuite répartis sur plusieurs machines, permettant ainsi de traiter un même fichier en parallèle. Pour garantir une tolérance aux pannes, les blocs de chaque fichier sont répliqués sur plusieurs machines. Notez que si la taille du fichier est inférieure à la taille d'un bloc, le fichier n'occupera pas la taille totale de ce bloc.

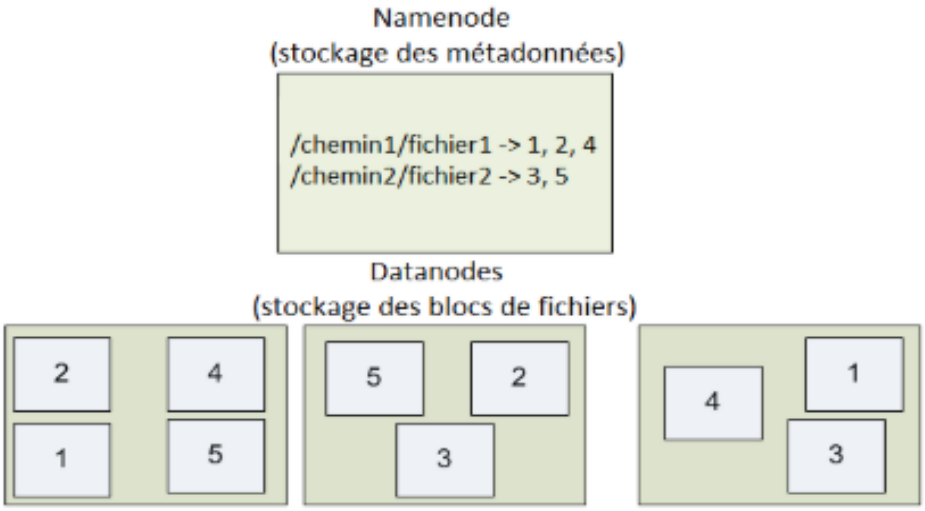


FIGURE 1.8 – Architecture HDFS.

(b) **Modèle de programmation Map-reduce :**

Le framework MapReduce permet de traiter les immenses quantités de données. Plutôt que de devoir déplacer les données vers un réseau pour procéder au traitement, MapReduce permet de déplacer directement le logiciel de traitement vers les données. Ce modèle fera l'objet de la deuxième technologie que nous allons présenter.

(c) Une collection d'outils spécifiques pour HDFS et Map Reduce comme des API et des frameworks.

2. **Map-reduce :** MapReduce a été introduit par Google et décrit en détails dans la publication «**MapReduce : Simplified Data Processing on Large Clusters** » publiée en 2004 et ça pour faciliter la mise en œuvre de ses workflows de traitement parallèle. L'objectif principal était de remplacer la programmation complexe et non intuitive sur l'informatique distribuée (préalablement abordée par les plateformes HPC) par une plateforme transparente moderne avec seulement deux

18. (Google file system) un système de fichier distribué conçu par Google

fonctions : Map et Reduce. Ces deux fonctions définies par l'utilisateur permettent aux utilisateurs d'utiliser les ressources distribuées sans se plaindre du réseau, de la planification, de la récupération après défaillance, etc.

Un modèle aussi intuitif que le MapReduce ne nécessite pas d'expertise concernant le parallélisme et les systèmes distribués. Son Framework Plug-and-Play embarque tous les détails pour implémenter les systèmes de calcul parallèle, la persistance et la résilience, l'optimisation et l'équilibre des ressources.

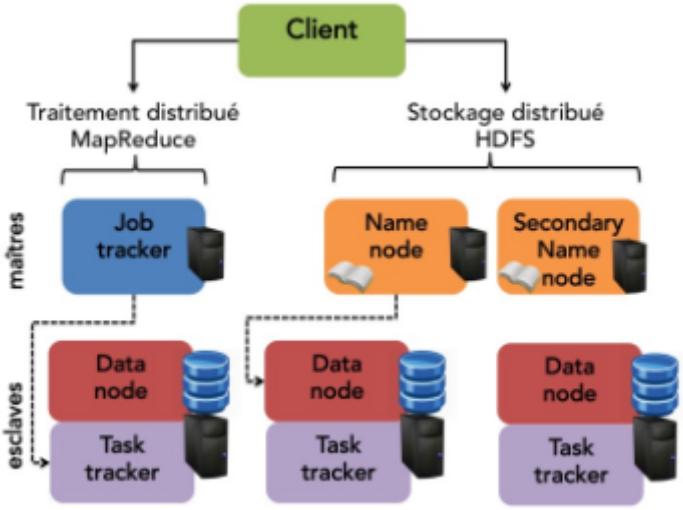


FIGURE 1.9 – L'environnement Hadoop.

(a) **Principe de MapReduce :**

Le Framework se décompose en deux parties. La fonction Map permet aux différents points du cluster distribué de distribuer leur travail. La fonction Reduce permet de réduire la forme finale des résultats des clusters en un seul résultat.Cela est rendu possible grâce au système de fichiers distribués HDFS d'Hadoop.Ce Framework est également constitué de plusieurs composants [11] :

- **Job tracker :** Est le nœud principal qui gère toutes les tâches et les ressources d'un cluster.
- **Les TaskTrackers :** Sont les agents déployés sur chaque machine d'un cluster pour lancer la map et réduire les tâches..
- **JobHistoryServer :** est un composant permettant de suivre les tâches complétées, généralement déployé comme une fonction séparée ou avec JobTracker.

Exemple d'utilisation de MapReduce : *Dans ce qui suit on vient mettre en clair le fonctionnement de MapReduce avec l'exemple typique "Word Count" schématisé afin de mieux cerner le r^ole de chaque fonction de ce framework.*
s'il est possible de compter manuellement le nombre de fois qu'un mot apparaît dans un roman, cela prend beaucoup de temps. Si l'on répartit cette tâche entre une vingtaine de personnes, les choses peuvent aller beaucoup plus vite. Chaque personne prend une page du roman et écrit

le nombre de fois que le mot apparaît sur la page. Il s'agit de la partie Map de MapReduce. Si une personne s'en va, une autre prend sa place. Cet exemple illustre la tolérance aux erreurs de MapReduce. Lorsque toutes les pages sont traitées, les utilisateurs répartissent tous les mots dans 26 boîtes en fonction de la première lettre de chaque mot. Chaque utilisateur prend une boîte, et classe les mots par ordre alphabétique. Le nombre de pages avec le même mot est un exemple de la partie Reduce de MapReduce.

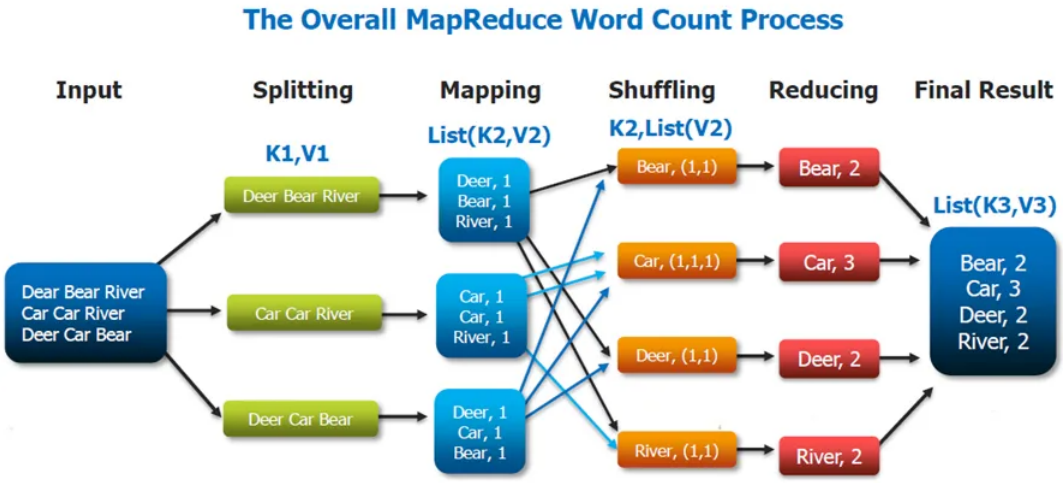


FIGURE 1.10 – Exemple de Word Count.

(b) **Les avantages de MapReduce :** Parmi les avantages de la programmation MapReduce nous citons :

- La scalabilité.
- La flexibilité.
- La sécurité et l'authentification.
- Le traitement parallèle.
- La disponibilité.
- Un modèle simple de programmation.

3. **Les bases de données NoSQL :**

De nos jours, l'ubiquité de la connexion Internet est une réalité (les voitures que nous conduisons, les montres que nous portons, nos petits appareils médicaux domestiques, nos réfrigérateurs et congélateurs, nos Smartphones et ordinateurs portables). De plus, les données numériques produites par les êtres humains, dont les séquences vidéo, les photos et autres, atteignent des volumes importants de plusieurs EO par jour. Ces données actuellement stockées dans des bases qui leur ont été conçues spécifiquement sont gérés par des logiciels de gestion de bases de données volumineuses, jouant le rôle d'intermédiaires entre les bases de données d'un côté et les applicatifs et leurs utilisateurs de l'autre. On parle ici des bases de données non-relationnelles, dites NoSQL.

Concrètement une base de données NoSQL est une approche de la conception des bases et de leur administration particulièrement utile pour de très grands

ensembles de données distribuées. Elle englobe une gamme étendue de technologies et d'architectures, afin de résoudre les problèmes de performances en matière d'évolutivité et de Big Data que les bases de données relationnelles ne sont pas conçues pour affronter. De plus elle est particulièrement utile lorsqu'une entreprise doit accéder, à des fins d'analyse, à de grandes quantités de données non structurées ou de données stockées à distance sur plusieurs serveurs virtuels du Cloud.

Pour mieux comprendre les bases de données NoSQL et leur application dans le Big Data, nous consacrons ***le chapitre 3 "Base de données NoSQL"*** à la description de cette technologie.

1.8.3 La technologie de Stockage : Le Cloud Computing

Le Cloud computing est une solution d'externalisation capable de louer de puissants moyens de calcul. Ceux-ci sont dotés de larges capacités de stockage extensibles et adaptés aux traitements des Big data. Au cœur des réflexions sur les infrastructures IT, de nouvelles offres Big data as a Service (BDaaS).

C'est un terme général employé pour désigner la livraison de ressources et de services à la demande par Internet. Il désigne le stockage et l'accès aux données par l'intermédiaire d'Internet plutôt que via le disque dur d'un ordinateur. Il s'oppose ainsi à la notion de stockage local, consistant à entreposer des données ou à lancer des programmes depuis le disque dur [17].

Conclusion

Au cours des dernières années un nombre important de sources de données est devenu disponible et à la portée de tous. Le Big Data en tant que discipline a déjà traversé son cycle d'implantation et est aujourd'hui transversal pour toute entreprise axée sur les données. Ainsi, les solutions et technologies Big Data représentent le moyen actuel d'atteindre la compétitivité et la croissance de notre société, où les appareils connectés augmentent rapidement, générant des données à des taux toujours croissants.

Jusqu'ici nous avons bien vu qu'avec le Big Data, les sources de données de plus en plus hétérogènes et localisées sur Internet sont produites de façon continue et à un rythme soutenu. Plusieurs nouvelles solutions ont vu le jour pour traiter ces volumes d'informations et ces flux de données en hausse constante, toutes ces solutions reposent sur un stockage distribué (partitionné) des données sur des clusters [19].

Comme nous l'avons vue dans la section 1.6 dans ce chapitre, les sources du Big Data sont nombreuses et varies. On s'intéresse dans le cadre ce travail à l'une des plus grand source du Big Data à savoir le Cloud qui fera l'objet du ***le chapitre 2 "La technologie de Stockage du Big Data : Le Cloud Computing"***.

19. Un cluster est une grappe de serveurs sur un réseau, appelé ferme ou grille de calcul.

Chapitre 2

La technologie de Stockage du Big Data : Le Cloud Computing

La plupart des systèmes Big Data reposent sur l'idée d'introduire d'abord des données dans le système sans dépenser trop d'efforts, puis imposer un schéma («schéma en lecture») une fois que les données sont accédées ou traitées. Ce modèle est beaucoup plus polyvalent que celui mis en œuvre par les bases de données relationnel, car le «schéma en lecture» est peu affecté par les mises à jour constantes, généralement présentes dans les systèmes dynamiques actuels des entreprises et des sciences. Dans ce qui suit nous verrons, les technologies de stockage, portées plus particulièrement par le déploiement du **Cloud Computing**.

2.1 Définitions

Plusieurs définitions sont apparus pour décrire cet espace de stockage immense qui n'est tout autre que le cloud computing :

1. **Définition 1.**

   Le "Cloud computing" en français "Informatique dans les nuages" ou "infonuagique" représente des ressources informatiques quelques part en dehors du réseau propre à l'entreprise ou à un particulier autrement dit c'est un concept d'organisation informatique qui place Internet au cœur de l'activité des entreprises, il permet d'utiliser des ressources matérielles distantes pour créer des services accessibles en ligne, L'accès au ressources se fait le plus souvent à l'aide d'un navigateur Web. En une phrase c'est une nouvelle façon de délivrer les ressources informatiques et ça, n'importe où dans le monde [18].

2. **Définition 2.**
   Un paradigme de calcul distribué émergeant dans lequel les données et les services sont disponibles dans des data centers extensibles et peuvent être accédés de manière transparente depuis des appareils (ordinateurs, téléphones, grappes, ...) connectés par Internet [19].

   > **Remarque :** *La notion de Cloud ne doit pas non plus être confondue avec celle du NAS*[1]*, utilisée par beaucoup d'entreprises via un serveur en résidence.*

1. Network Attached Storage est un serveur de fichiers autonome, relié à un réseau, dont la principale fonction est le stockage de données en un volume centralisé pour des clients réseau hétérogènes.Cette espace est offert pour l'ensemble du réseau local.

Ces réseaux locaux n'entrent pas dans la définition du Cloud. Cependant, certains NAS permettent d'accéder aux données à distance depuis Internet. [17]

De manière générale, on parle de Cloud Computing lorsqu'il est possible d'accéder à des données ou à des programmes depuis internet, ou tout du moins lorsque ces données sont synchronisées avec d'autres informations sur Internet. Il suffit donc pour y accéder de bénéficier d'une connexion internet.

2.2 Les caractéristiques du Cloud :

Selon le **NIST** [2], le cloud computing doit posséder 5 caractéristiques essentielles qui sont :

- **Accès réseau universel :** L'ensemble des ressources doit être accessible et à disposition de l'utilisateur universellement et simplement à travers le réseau.
- **Libre service à la demande :** Permet à l'utilisateur d'utiliser et de libérer des ressources distantes en temps réel en fonction des ses besoins, sans nécessiter d'intervention humaine du côté fournisseur.
- **Ressources partagées :** Les ressources matérielles du fournisseur sont partagées entre les différents utilisateurs.
- **Élasticité :** Les ressources allouées aux utilisateurs peuvent être augmentées ou diminuées selon l'usage.
- **Service mesurable et facturable (pay-as-you-use) :** Les utilisateurs paieront pour les ressources qu'ils ont utilisées et pour la durée de leur utilisation.

2.3 Les services Cloud :

Selon le type de ressource offerte aux utilisateurs, les services cloud ont tendance à être regroupés selon les trois modèles de base suivants :

- Logiciel en tant que service **(SaaS)**.
- Plate-forme en tant que service **(PaaS)**.
- Infrastructure en tant que service **(IaaS)** .

2. Le National Institute of Standards and Technology, ou NIST (qu'on pourrait traduire par « Institut national des normes et de la technologie »), est une agence du département du Commerce des États-Unis. Son but est de promouvoir l'économie en développant des technologies, la métrologie et des standards de concert avec l'industrie. Cette agence a pris la suite en 1988 du National Bureau of Standards fondé en 1901 avec substantiellement les mêmes missions.

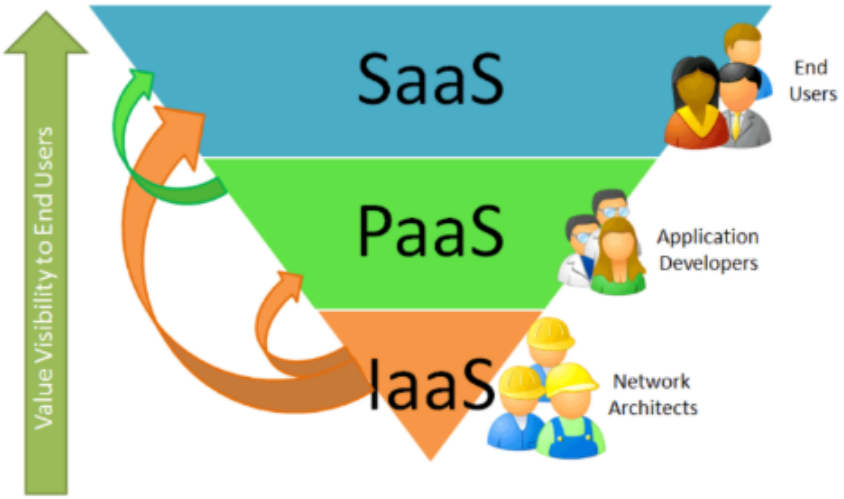


FIGURE 2.1 – Services cloud

Dans le prolongement de cette classification, lorsque le service fait référence à une base de données, le modèle est appelé Database as a Service (DBaaS) [20].

On retrouve dans le tableau suivant TABLE 1.1 les définitions de ces services ainsi que quelques exemple.

Service	**Définition**	**Exemple**
IaaS (Infrastructure as a service)	Fourni a l'entreprise différents composants informatiques comme des espaces de stockages, équipements réseaux, des unités centrales, etc... Ce service offre un accès à un parc informatique virtualisé ainsi que des machines virtuelles sur lesquelles l'utilisateur peut installer un système d'exploitation et des applications.	• Amazon EC2 Elastic Compute Cloud. • S3 / Simple Storage Service d'Amazon. • Google drive, DropBox qui sont gratuits.
PaaS (Platform as a service)	Il s'agit d'offrir aux utilisateurs des ressources machines , de l'espace de stockage et une plateforme de développement d'application. Ces plateformes sont spécifiques à un langage et à une base de données, et le système d'exploitation et les outils d'infrastructure sont sous la responsabilité du fournisseur.	• App Engine de Google qui se limite à Java et Python. • Windows Azure de Microsoft permet de travailler avec les langages comme .NET, PHP, Python, Ruby et Java.
SaaS (Software as a Service)	Ce service, met des applications à la disposition des utilisateurs. Ces applications peuvent être manipulées à l'aide d'un navigateur web. L'utilisation des applications reste transparente pour les utilisateurs, qui ne se soucient ni de la plateforme, ni du matériels et ni des mise à jour.	• Google apps avec Google Docs, Calendar et Gmail qui sont gratuites. • Facebook, Linkdin qui sont gratuits. • Offices 365 de Microsoft propose des applications web (Word,Excel, PowerPoint, Publisher...).

DBaaS (DataBase as a service)	Un tel modèle fournit des mécanismes transparents pour créer, stocker, accéder et mettre à jour des bases de données. De plus, le fournisseur de services de base de données assume l'entière responsabilité de l'administration de la base de données, garantissant ainsi la sauvegarde, la réorganisation et les mises à jour de version. L'utilisation de ce service permet aux fournisseurs de répliquer et de personnaliser leurs données sur plusieurs serveurs, qui peuvent être physiquement séparé [16].	• Amazon Web Services. • RackSpace. • IBM, Microsoft, Oracle.

TABLE 2.1 – Les services Cloud.

Remarque : *Avec le SaaS on utilise une application, avec le PaaS on construit ses applications et l'IaaS permet d'héberger le tout.*

2.4 Les composants clés du cloud computing :

Le cloud computing est étroitement lié aux mégadonnées. Les composants clés de l'informatique en nuage sont illustrés à la **Figure 2.2**. Les mégadonnées font l'objet d'une opération à forte intensité de calcul et mettent l'accent sur la capacité de stockage d'un système en nuage. L'objectif principal du cloud computing est d'utiliser d'énormes ressources informatiques et de stockage sous une gestion concentrée, afin de fournir aux applications Big Data une capacité de calcul finie. D'autre part, l'émergence du Big Data accélère également le développement du cloud computing. La technologie de stockage distribué basée sur le cloud computing peut gérer efficacement les Big Data ; la capacité de calcul parallèle et améliorer l'efficacité de l'acquisition et de l'analyse des mégadonnées.

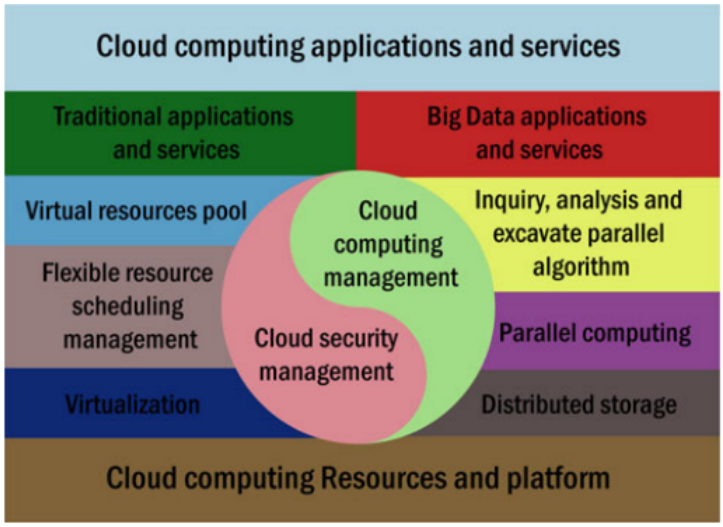


FIGURE 2.2 – Composants clés du cloud computing

Pour comprendre le fonctionnement du Cloud, il faut connaitre ses deux composants à savoir la notion de Virtualisation et de Data center.

1. **Virtualisation :**
   C'est l'ensemble des techniques matérielles et/ou logiciels qui permettent de faire fonctionner sur une seule machine, plusieurs systèmes d'exploitation (appelées machines virtuelles (VM), ou encore OS invité).

   Même s'il existe plusieurs types de virtualisation, la forme la plus populaire de virtualisation dans le cloud est la virtualisation des serveurs qui consiste à dématérialiser le comportement et les données d'un serveur ou d'une machine, de façon à faire tourner plusieurs de ces instances dématérialisées sur un même serveur physique. De cette façon, les différentes instances créées se partagent les ressources du serveur physique. Cela permet une plus grande modularité dans la répartition des charges, une facilité dans l'administration des serveurs et une réduction des coûts [21].

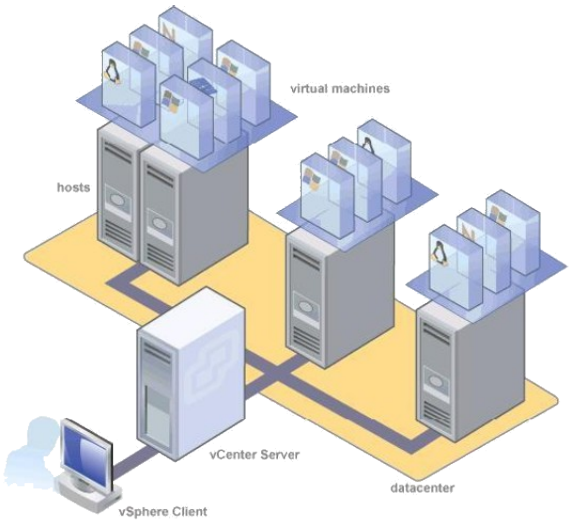


FIGURE 2.3 – Virtualisation De Serveurs

2. **Les Data Center :**
   Un data center ou centre de données, c'est un site physique qui a une infrastructure composée d'un réseau d'ordinateurs et d'espaces de stockage,Il peut être interne ou externe à l'entreprise. Ces sites sont des salle remplies de baies de stockage, utilisées par de nombreuses entreprises et autres organisations gouvernementales [22].

   - **Les composants du Data Center :** Un centre de données basique regroupe des serveurs, des sous-systèmes de stockage, des commutateurs de réseau, des routeurs, des firewalls, et bien entendu des câbles et des racks physiques permettant d'organiser et d'interconnecter tout cet équipement informatique.

FIGURE 2.4 – Exemple de deux data center, un de Facebook et l'autre de Google

- **L'architecture du data center**
  Théoriquement, n'importe quel espace suffisamment vaste peut servir de Data Center. Cependant, le design et l'implémentation d'un data center nécessite de prendre plusieurs précautions. Par-delà les problèmes basiques du coût et des taxes, les sites sont sélectionnés sur de nombreux critères, comme la localisation géographique, la stabilité météorologique, l'accès aux routes et aux aéroports, la disponibilité énergétique, les télécommunications ou encore l'environnement politique.

  Pour fonctionner correctement, un Data Center doit aussi abriter l'infrastructure adéquate : un système distribution d'énergie, un commutateur électriques, des réserves d'énergie, des générateurs dédiés au backup, un système de ventilation et de refroidissement, et une puissante connexion internet. Une telle infrastructure nécessite un espace physique suffisamment vaste et sécurisé pour contenir tout cet équipement.

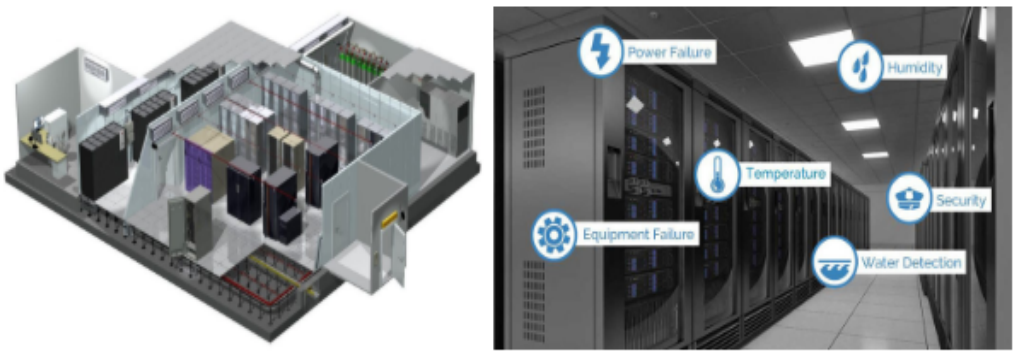


FIGURE 2.5 – Architecture des Data Cente

2.5 Les modes et type de stockage dans le Cloud

Le stockage dans le cloud englobe l'archivage, l'organisation et la distribution de données à la demande entre plusieurs volumes de stockage virtualisés, regroupés à partir d'équipements matériels physiques distincts. Plus simplement, le stockage dans le cloud est l'organisation des données stockées dans un emplacement accessible depuis Internet par toute personne qui dispose d'une autorisation [23].

Le stockage interne présente de nombreuses faiblesses, comme le risque de perdre des données. De plus, le cloud permet de partager très facilement des documents ou des fichiers multimédias trop larges pour être envoyés par mail.

Il existe **trois types de stockage** dans le cloud pour les entreprises :

1. Stockage dans le cloud public.
2. Stockage dans le cloud privé.
3. Stockage dans le cloud hybride.

En parallèle, il existe trois formats de stockage :

1. Mode bloc
2. Mode fichier
3. Mode objet.

Remarque : Chaque format présente ses avantages et ses inconvénients (les blocs sont plus rapides, les fichiers plus lisibles et les objets plus adaptés aux applications cloud-native et mises en paquet dans des conteneurs), mais certains produits de stockage logiciel dans le cloud sont capables de combiner les trois formats dans une solution unique et facile à déployer.

1. **Les formats de stockage dans le cloud :**

   Les trois formats de stockage de données dans le cloud :

   - **Stockage en mode bloc :**

     Le stockage en mode bloc permet de diviser un seul volume de stockage (par exemple, un nœud de stockage dans le cloud) en plusieurs instances individuelles appelées blocs. Il s'agit d'une solution de stockage rapide à faible latence, idéale pour les charges de travail hautes performances [23].

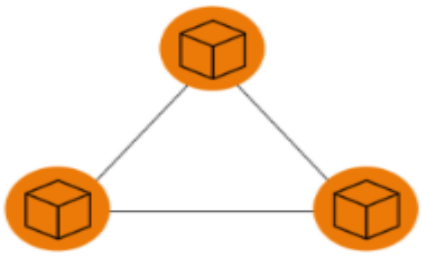


FIGURE 2.6 – Stockage en mode bloc

- **Stockage en mode objet :**

  Le stockage en mode objet implique d'attribuer à chaque donnée des identifiants uniques, que l'on appelle les métadonnées. Compte tenu du fait que les objets ne sont ni compensés ni chiffrés, ils sont rapidement accessibles à très grande échelle. Il s'agit donc d'une solution idéale pour les applications cloud-native [19].

FIGURE 2.7 – Stockage en mode objet

- **Stockage en mode fichier :**

  Le stockage en mode fichier est le plus utilisé sur les systèmes NAS. Il permet d'organiser les données et de les présenter aux utilisateurs. Sa structure hiérarchique permet de parcourir les données du haut vers le bas en toute simplicité, mais augmente le temps de traitement [23].

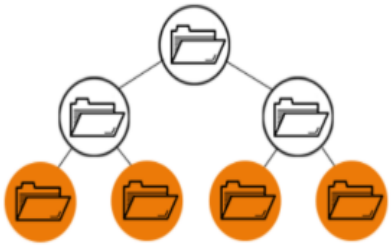


FIGURE 2.8 – Stockage en mode fichier

2. **Les types de stockage dans le cloud :**

   Il existe trois types de stockage de données dans le cloud :

FIGURE 2.9 – Type de stockage dans le cloud

- **Stockage dans le cloud public :**

  Il s'agit du stockage de données entre plusieurs pools de ressources virtuelles nommés clouds publics. Ceux-ci sont développés à partir de matériel détenu et géré par des entreprises tierces. Il n'est pas sans risque de stocker des données dans des systèmes qui ne nous appartiennent pas et qu'on ne gère pas. C'est pour cette raison que de nombreuses entreprises utilisent des conteneurs pour transférer des charges de travail et des applications entre les environnements de cloud public.

- **Stockage dans le cloud prive :**

  Il s'agit du stockage de données entre plusieurs pools de ressources virtuelles nommés clouds privés. Ceux-ci sont alimentés en données par des systèmes dédiés qui sont détenus et gérés par l'entreprise qui les utilise. À long terme, la mise en place manuelle d'un cloud privé à l'échelle de l'entreprise peut se révéler moins efficace que l'utilisation d'un logiciel existant. C'est pour cette raison que les entreprises utilisent des plateformes telles qu'OpenStack [3] pour réaliser la transformation numérique de leurs pools de ressources virtuelles en clouds privés.

- **Stockage dans le cloud hybride :**

  Il s'agit du stockage de données dans plusieurs environnements de cloud public et privé, connectés entre eux. Bien que les environnements de cloud privé et public qui composent le cloud hybride restent des entités distinctes, la migration de données entre les deux est facilitée par un réseau complexe de LAN, WPN, API, VPN ou conteneurs. Cette architecture, à la fois séparée et connectée, permet aux entreprises de stocker les données critiques dans leur cloud privé et les données moins sensibles dans le cloud public, tout en les déplaçant entre les différents environnements, selon les besoins.

  Remarque *La frontière entre le Local Computing et le Cloud Computing est parfois très fine. Pour cause, le Cloud est désormais ancré dans presque toutes les tâches que nous accomplissons sur ordinateur.*

3. OpenStack est un ensemble de logiciels open source permettant de déployer des infrastructures de cloud computing (infrastructure en tant que service)

Conclusion

Il n'est pas exagéré de dire que les différentes sources du Big Data(IoT, Cloud, Web, Media etc...) annonce une nouvelle ère économique pour l'ensemble de la planète. Les espoirs portés par les sources du Big Data ne portent pas simplement sur l'amélioration des processus et des modèles économiques existants, ils présentent en outre une dimension transformationnelle dans leur portée.

Cependant, rien de tout cela n'est possible sans une base de données fiable capable de gérer les énormes quantités de données générées par les appareils ces différentes sources.

Dans le cadre de ce travail on sait intéresser a l'une des sources du Big Data à savoir les Cloud qui génèrent d'immense quantités de données et qui nécessite une réelle réflexion sur les technologie de stockage à utiliser pour géré se gros volume de données croissant.

Le NoSQL se présente aujourd'hui comme une réelle solution pour faire face a la structuration, le stockage et a la gestion des bases de données (non relationnelles) distribuées sur le cloud .

La technologie NoSQL sera développer dans le chapitre suivant.

Chapitre 3

Base de données NoSQL

Introduction

Les bases de données SQL conventionnelles utilisent SQL (langage de requête structuré) comme interface principale pour gérer les bases de données et sont basées sur un modèle de base de données relationnelle. Le terme «bases de données NoSQL» est une expression générale émergente pour les bases de données dans le but de ne pas (ou presque) utiliser les fonctionnalités des bases de données SQL. Les bases de données NoSQL doivent être non relationnelles, distribuées, open-source et évolutives horizontalement.

3.1 Rappel de gestion de base de données relationnelles

Au début de l'ère des bases de données, chaque application stockait ses données au sein de sa propre structure unique. Lorsque les développeurs voulaient créer des applications pour utiliser ces données, ils devaient en savoir beaucoup sur la structure spécifique des données afin de trouver les données dont ils avaient besoin. Ces structures de données étaient inefficaces, difficiles à gérer et à optimiser pour obtenir de bonnes performances pour les applications. Le modèle de base de données relationnelle a été conçu pour résoudre le problème que présentent plusieurs structures de données arbitraires [24].

3.1.1 Le modèle relationnel

Le modèle relationnel est un moyen intuitif et simple de représenter des données dans des tables. Dans une base de données relationnelle SGBDR [1], chaque ligne de la table est un enregistrement avec un identifiant unique appelé clé. Les colonnes de la table contiennent les attributs des données, et chaque enregistrement a généralement une valeur pour chaque attribut, ce qui facilite l'établissement des relations entre les points de données.

1. SGBDR acronyme de Système de Gestion de Base de Données Relationnels

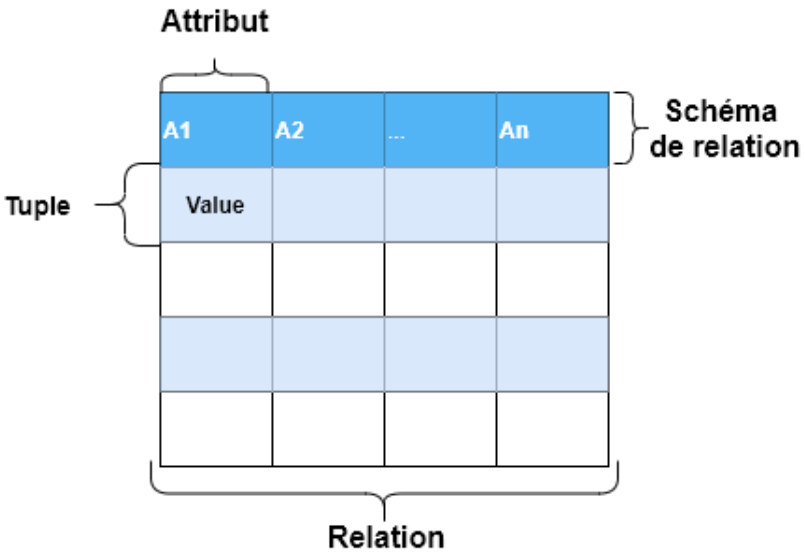


FIGURE 3.1 – Elements d'une table d'une BDDR.

3.1.2 Les règles CODD

Les douze (12) règles de Codd sont un ensemble de règles édictées par Edgar F. Codd [2] afin de définir les caractéristiques que doit présenter un système de gestion de base de données (SGBD) afin d'être considéré comme relationnel (SGBDR). On considère parfois une règle 0, qui stipule que l'intégralité des fonctions du SGBDR doit être accessible par le modèle relationnel [25].

1. **Unicité :** toutes les informations sur les données sont représentées au niveau logique (valeurs dans des colonnes de tables) et non physique.
2. **Garantie d'accès :** les données sont accessibles sans ambiguïté uniquement par la combinaison du nom de la table, de la clef primaire et du nom de la colonne.
3. **Traitement des valeurs nulles :** une valeur spéciale doit représenter l'absence de valeur, une information manquante ou une information inapplicable (valeur NULL).
4. **Catalogue lui même relationnel :** la description de la base de données doit être accessible comme les données ordinaires (un dictionnaire des données est enregistré dans la base).
5. **Sous langage de données :** un langage doit permettre de définir les données, définir des vues (visions particulières de la base, enregistrées comme des relations), manipuler les données, définir les contraintes d'intégrité, des autorisations et gérer des transactions.
6. **Mise à jour des vues :** toutes les vues pouvant théoriquement être mises à jour doivent pouvoir l'être par le système.
7. **Insertion, mise à jour, et effacement de haut niveau :** le langage doit comporter des ordres effectuant l'insertion, la mise à jour et la suppression de données, aussi bien pour des lots de tuples issues de plusieurs tables que juste pour un tuple unique issu d'une table unique.
8. **Indépendance physique :** indépendance vis à vis de l'implantation physique des données.

2. Edgar Frank « Ted » Codd est un informaticien britannique. Il est considéré comme l'inventeur du modèle relationnel des SGBDR.

9. **Indépendance logique :** indépendance vis à vis de l'implantation logique des données (tables, colonnes, etc.).
10. **Indépendance d'intégrité :** les contraintes d'intégrité doivent pouvoir être définies dans le langage relationnel et enregistrées dans le dictionnaire des données (catalogue).
11. **Indépendance de distribution :** indépendance de la répartition des données sur divers sites.
12. **Règle de non subversion :** on ne peut jamais contourner les contraintes (d'intégrité ou de sécurité) imposées par le langage du SGBD en utilisant un langage de programmation de plus bas niveau.

Remarque : 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

Les systèmes de gestion de base de données relationnels doivent se conformer à quatre propriétés de transaction : Atomicité , Cohérence , Isolation et Durabilité - appelées propriétés ACID [20][26] :

Remarque : *Une opération sur les données est appelée une transaction*

- **Atomicité :**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.

  Exemple : *Si 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 déprendre du contexte de modification d'une autre, et toute rupture de ce contexte pourrait engendrer une incohérence des données de la base.*

- **Cohérence :** qui consiste à assurer la cohérence des données. La transaction doit respecter les contraintes d'intégrité pour donner un résultat cohérent (passer d'un état valide à un autre état valide).

  Exemple : *Un système doit être capable de reconnaître qu'une facture est liée à un client et aux éléments factures. Il doit être capable d'éviter, par exemple, la suppression d'un client s'il existe encore des factures pour ce client, et la suppression d'une facture qui a des éléments associées.*

- **Isolation :** qui consiste à exécuter chaque transaction en isolation des autres transactions ; les modifications qu'apporte une transaction à la base de données durant son exécution ne seront visibles par les autres transactions qu'à sa terminaison. Cette propriété assure la cohérence des données dans le cas de l'accès concurrent entre les transactions.

  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'enchainer les unes à la suite des autres, et non de manière concurrentielle.

- **Durabilité :** qui signifie que toutes les actions entreprises par les transactions sur les données doivent avoir le caractère durable ; on ne peut pas refaire une transaction terminée. Pour cela, le SGBD doit enregistrer toutes les chronologies des interactions et modifications effectuées sur les données (dans des fichiers d'historique dit journaux). Le but est de se prémunir contre les éventuelles pannes informatiques.

3.1.4 Les limites des bases de données relationnels

Avec la quantité et le débit des données qui est en hausse chaque jours avec l'apparition du Big Data ces dernières années, les base de données relationnels sont confrontées a plusieurs problèmes :

- **Affichage tabulaire des données :**Tous les types de données ne peuvent pas être compressés dans un schéma rigide de tables bidimensionnelles liées. Les types de données abstraites et les données non structurées qui se produisent dans le cadre d'applications multimédia et de grandes solutions de données ne fonctionnent pas avec le modèle des bases de données relationnelles [27].

- **Pas de schéma de base de données hiérarchique :** Contrairement aux bases de données orientées objet, les bases de données relationnelles n'offrent pas la possibilité d'implémenter des schémas de base de données avec des classes hiérarchiquement structurées. Des concepts tels que les entités subordonnées qui héritent de propriétés d'entités supérieures ne peuvent pas être implémentés avec elles. Par exemple, on peut pas créer de sous-tuples avec eux. Tous les tuples d'une base de données relationnelle se trouvent au même niveau hiérarchique [27].

- **Segmentation des données :** Le principe de base des systèmes de bases de données relationnelles, qui consiste à diviser l'information en tables séparées (normalisation), conduit inévitablement à une segmentation des données. Tout ce qui est connexes n'est pas nécessairement stocké ensemble. Cette conception de base de données entraîne des requêtes complexes sur plusieurs tables au niveau de l'application. Le nombre plus élevé de segments interrogés qui en résulte a généralement un impact négatif sur la performance [27].

- **Performances limité :** Le modèle de bases de données relationnelles impose des exigences élevées en matière de cohérence des données, ce qui est préjudiciable à la vitesse d'écriture des transactions [27].

- Ces types de base de données sont incapable de gérer de très grands volumes de données (de l'ordre du péta-octet) ainsi que des débits extrêmes (plus que quelques milliers de requêtes par seconde).

- **Les contraintes ACID en milieu distribuée :** Dans le cas des systèmes distribuées, il est nécessaire de distribuer les traitements de données entre différents serveurs. Il devient alors difficile de maintenir les contraintes ACID

à l'échelle du système distribue entier tout en maintenant des performances correctes, ce qui entraînent de sérieux surcoûts en latence, accès disques, temps CPU.

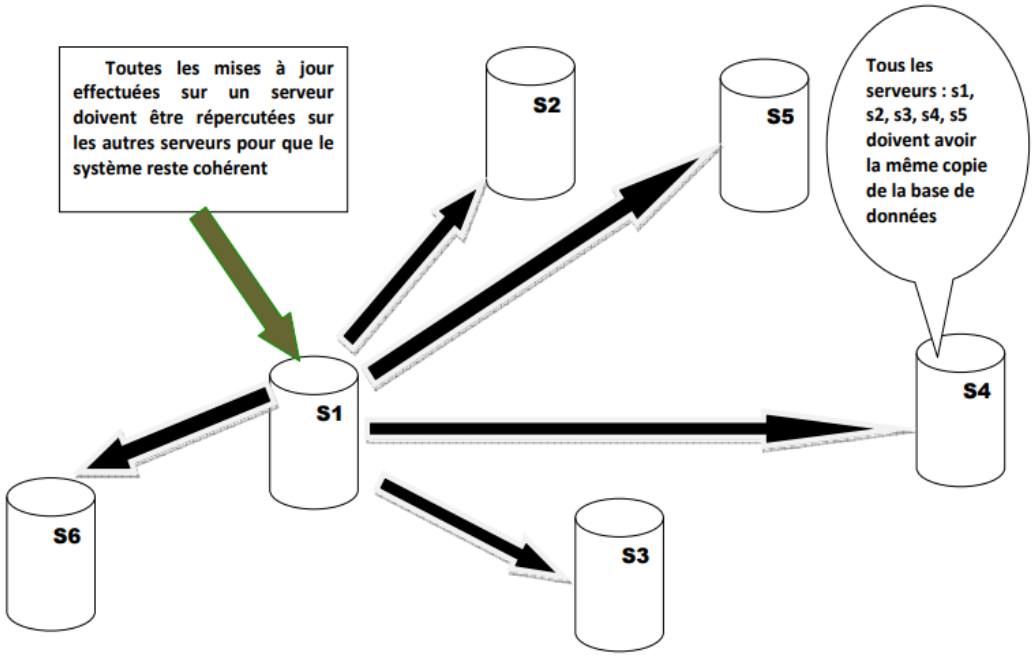


FIGURE 3.2 – Limites liées aux propriétés ACID.

Ces dernières années, d'importantes entreprises issues du domaine informatique ont décidé que les systèmes de gestion de bases de données relationnelles actuelles ne pouvaient plus répondre correctement à leurs besoins. Elles ont donc développé ou financé de nouveaux systèmes qui pourraient les aider à répondre à leurs besoins, à savoir les bases de données NoSql.

3.1.5 Exemples des base de données relationnels

Les systèmes de gestion de bases de données relationnelles (SGBDR) les plus couramment utilisés sont données comme suit [28] :

— **Db2 :**Est l'un des SGBD relationnelles propriétaire d'IBM [3] disponible aux utilisateurs sous licence commerciale.[28]

— **Microsoft SQL Server :** Le système de gestion de base de données de Microsoft [4] en langage SQL est disponible sous une licence par utilisateur payante.

— **MySQL :** Est le SGBDR open source [5] le plus utilisé dans le monde. Depuis son acquisition par Oracle, MySQL est commercialisé sous une double licence. La communauté des développeurs d'origine poursuit le projet sous le nom de MariaDB.

3. International Business Machines Corporation, connue sous le sigle IBM, est une société multinationale américaine présente dans les domaines du matériel informatique, du logiciel et des services informatiques.

4. Microsoft Corporation est une multinationale informatique et micro-informatique américaine, fondée en 1975 par Bill Gates et Paul Allen.

5. Un logiciel Open Source est un programme dont le code est distribué sous une licence permettant à quiconque de le lire, le modifier ou le redistribuer.

— **PostgreSQL :** avec PostgreSQL, les utilisateurs peuvent accéder gratuitement à un système de gestion de base de données relationnel-objet (SGBDRO). Le développement ultérieur est effectué par une communauté open source.

— **Oracle Database :** le système de gestion de base de données relationnelle de la société du même nom Oracle[6] est commercialisé sous licence propriétaire contre rémunération.

— **SQLite :**SQLite est une bibliothèque appartenant au domaine public contenant un système de gestion de bases de données relationnelles.

Résumé : *Le schéma relationnel des bases de données est clair, mathématiquement solide et a fait ses preuves dans la pratique depuis plus de 40 ans. Pourtant, le stockage des données dans des tables structurées ne répond pas à toutes les exigences des technologies modernes de l'information.*

3.2 Les bases de données NoSQL

Les exigences de développement d'applications moderne ont connu une profonde révolution ces 15 dernières années. Des volumes de données croissants doivent être enregistrés, traités et mis à jour de plus en plus rapidement. Pour ce qui est de la gestion de gros volumes de données, les bases de données NoSQL ou non relationnelles font depuis longtemps parler d'elles [29].

Tout le mouvement contre le monopole des bases de données SQL a commencé à la fin des années 1990. Cependant, ce n'est pas avant 2009 qu'il est devenu un sérieux concurrent du SGBDR. Il n'y avait pas de définition claire ni de terme pour ce mouvement, jusqu'à ce que l'expression NoSQL - signifiant «pas seulement SQL» - soit créée, suggérant d'éviter SQL. C'est alors que NoSQL est devenu très populaire grâce au terme existant et très connu SQL lui-même.

- **Définition :**

  NoSQL correspond à « not only SQL » et c'est en effet ce que ce modèle de base de données veut être : non pas une contrepartie, mais bien un enrichissement et complément utile des bases de données SQL relationnelles traditionnelles. Ce faisant, les bases de données NoSQL dépassent les limites des systèmes relationnels et exploitent un modèle de base de données alternatif. Cela ne veut toutefois pas dire qu'aucun système SQL n'est utilisé. Il existe de nombreuses variantes combinées au sein desquelles les deux solutions peuvent être utilisées et qui restent toutefois englobées sous l'étiquette NoSQL [29].

- **Remarque :**

  Les systèmes NoSQL sont souvent décrits comme des mémoires structurées de stockage de données, ce qui met en évidence leur différence significative avec les bases de données SQL : contrairement à ces dernières, les bases de données NoSQL n'exploitent pas de schéma de tableau fixe dans lequel les données doivent être définies

6. Oracle (Oracle Corporation) est une entreprise américaine créée en 1977 par Larry Ellison. Ses produits phares sont Oracle Database (un système de gestion de base de données), Oracle Weblogic Server (un serveur d'applications), Oracle E-Business Suite (un progiciel de gestion intégré) et Oracle Cloud Infrastructure (une offre de Cloud Computing)

avant l'enregistrement. Elles utilisent des méthodes plus flexibles leur permettant d'enregistrer facilement de nouveaux jeux de données et d'assurer leur mise à jour en continu au sein de l'application. Les solutions NoSQL sont également adaptées au traitement de données non structurées ou inconnues, ce qui serait totalement impossible avec une base de données relationnelle.

- **Distinction des autres termes :**
  - **Non SQL :** Le terme « bases de données non SQL » est trompeur, pour ne pas dire faux. Il prédit que ces bases de données sont sans aucune utilisation de SQL et ce n'est pas toujours le cas dans NoSQL. Le terme «bases de données non SQL» existe, mais il est plutôt utilisé comme un terme vague que comme une expression professionnelle. Cela signifie que les données sont traitées par un autre langage que SQL, par ex. XQuery [7] pour les bases de données XML [8].

  - **Distributed storage :** Liés à NoSQL, il existe des termes tels que «stockage distribué» ou «stockage structuré distribué». Il est assez difficile de distinguer ces termes de NoSQL, car ils décrivent exactement une caractéristique de NoSQL. Le stockage distribué est un terme générique pour un système qui prétend être un stockage unique mais qui est en réalité une collection de nombreuses unités informatiques stockant des parties des fichiers. Certaines bases de données NoSQL prétendent être un système de stockage distribué, par exemple BigTable de Google.

3.2.1 Le théorème CAP

Le théorème CAP est une théorie soumise par Brewer [9] en 2000. Elle a été reprise en 2003 par Seth Gilbert et Nancy Lynch du MIT qui l'ont redéfinie et elle a pris le nom de théorème CAP. Cette dernière théorie stipule que dans le cadre d'un système distribué, plus spécifiquement dans l'utilisation d'une application web, une base de données ne peut pas garantir ces trois attributs en même temps [30].

Prenons trois exemples concrets afin d'illustrer l'explication de chaque attribut d'une base de données contenant une liste de clients :

1. Un serveur.
2. Deux serveurs sur deux machines distinctes contenant chacune la moitié de la liste des clients
3. Deux serveurs sur deux machines distinctes contenant chacune la liste des clients dans sa totalité, le second serveur étant un réplica.

1. **Consistency (Cohérence) :** Une donnée n'a qu'un seul état visible quel que soit le nombre de réplicas.

   Dans l'exemple 1, l'intégrité est totale, vu qu'il n'y a qu'un seul serveur, ce dernier aura toujours les dernières données à jour. Dans l'exemple 2, les deux serveurs retourneront toujours les dernières données, vu qu'ils contiennent chacun des données différentes. Par contre, dans l'exemple 3, si une modification est faite sur le premier

7. XQuery est un langage de requête informatique permettant non seulement d'extraire des informations d'un document XML, ou d'une collection de documents XML, mais également d'effectuer des calculs complexes à partir des informations extraites et de reconstruire de nouveaux documents ou fragments XML

8. L'Extensible Markup Language, généralement appelé XML, « langage de balisage extensible » en français, est un métalangage informatique de balisage générique

9. 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.

serveur, on ne peut être sûr que la propagation se fasse tout de suite sur le second. On ne peut donc garantir que l'intégrité soit totale dans ce cas.

2. **Availability (Disponibilité) :** Tant que le système tourne (distribué ou non), la donnée doit être disponible.

   Dans l'exemple 1, si le serveur venait à tomber en panne, la disponibilité serait nulle, car il n'y aurait plus aucune réponse de sa part. Dans l'exemple 2, si l'une des deux machines venait à tomber, un utilisateur ne pourra accéder qu'à la moitié des données. Alors que dans le dernier exemple, si l'un des deux tombait, les données resteraient complètement accessibles.

3. **Partition Tolerance (Tolérance au partitionnement) :** Garantit que le système continue de fonctionner même si un nœud de données répliquées tombe en panne ou perd la connectivité avec d'autres nœuds de données répliqués.

   La question ne se pose pas dans le cas 1, étant donné que ce dernier attribut ne fait état que dans le cas d'un système distribué. Par contre dans l'exemple 2, ce dernier n'est pas tolérant au partitionnement, car l'un comme l'autre les nœuds restants ne pourront espérer avoir les données de l'autre, ce qui coupera les clients de la moitié des données existantes. Dans l'exemple 3 par contre, les deux serveurs, même sans réseau entre eux, pourront continuer de renseigner les clients, vu que chacun possède la totalité des données.

Selon le théorème CAP et les différentes préoccupations des base de données NoSQL, une classification préliminaire des bases de données NoSQL est la suivante [31] :

— **Soucieux de la cohérence et de la disponibilité (CA)** Une partie de la base de données n'est pas concernée par la tolérance de partition, et utilise principalement l'approche de réplication pour assurer la cohérence et la disponibilité des données.

— **Préoccupé par la disponibilité et la tolérance de partition (AP)** Les bases ne sont pas forcément cohérente dans le temps mais la multiplicité des bases sur le réseau permet de garantir une réponse quoi qu'il arrive.

— **Préoccupés par la cohérence et la tolérance de partition (CP)** Un tel système de base de données stocke les données dans les nœuds distribués, mais assure également la cohérence de ces données, mais le support n'est pas assez bon pour la disponibilité. Pour notre mémoire on c'est intéressé sur ce dernier point.

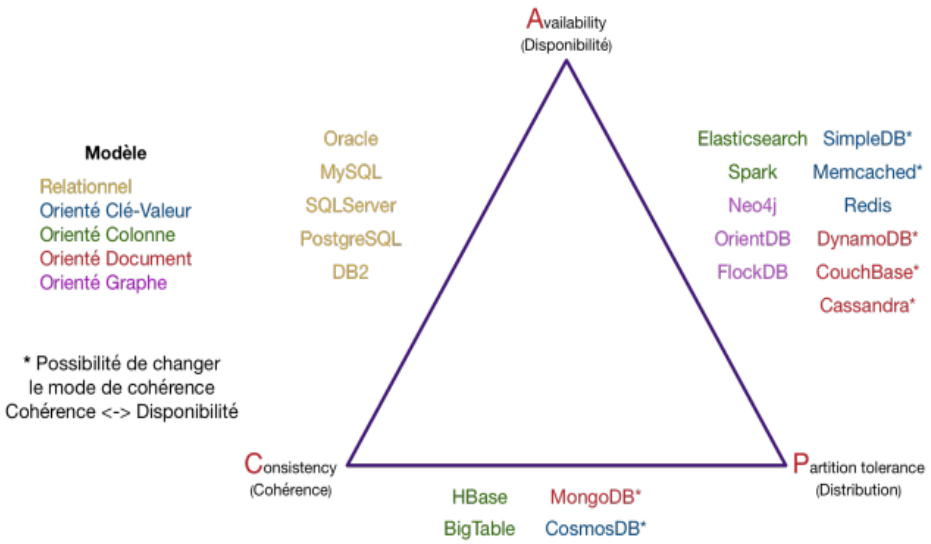


FIGURE 3.3 – Le théorème CAP.

3.2.2 Les types des bases de données NoSQL

Le modèle de données des base de données traditionnelles est principalement relationnel, spécifiquement pour prendre en charge les opérations de classes associées et les transactions ACID. Il existe différents modèles des bases de données NoSQL classés en quatre grandes catégories :

1. Les bases de données orientées clé-valeur.
2. Les bases de données orientées colonnes.
3. Les bases de données orientées documents.
4. Les bases de données orientées graphe.

3.2.2.1 Les bases de données orientées clé-valeur

Le but de la famille clé-valeur est l'efficacité et la simplicité. Un système clé-valeur agit comme une énorme table de hachage [10] distribuée sur le réseau. Tout repose sur le couple Clé/Valeur. La clé identifie la donnée de manière unique et permet de la gérer. La valeur contient n'importe quel type de données [32].

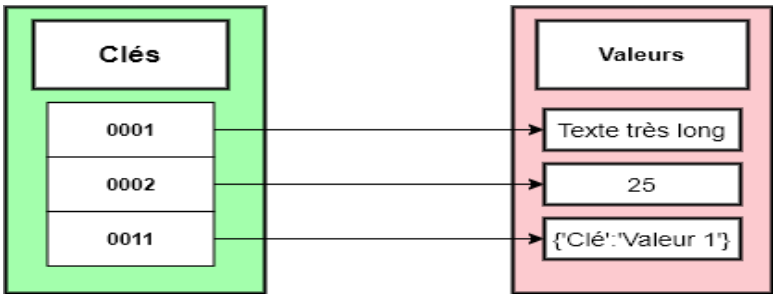


FIGURE 3.4 – Base de donnée orientée clé-valeur.

Les seules opérations de type CRUD peuvent être utilisées :

10. Une table de hachage est, en informatique, une structure de données qui permet une association clé–valeur, c'est-à-dire une implémentation du type abstrait tableau associatif; en particulier, l'implémentation d'une table des symboles lorsque les clés sont des chaînes de caractères.

- Create : créer un nouvel objet avec sa clé → create(key, value).
- Read : lit un objet à partir de sa clé→ read(key).
- Update :met à jour la valeur d'un objet à partir de sa clé → update(key, value).
- Delete :supprime un objet à partir de sa clé → delete(key)..

Exemple d'applications :

Détection de fraude en temps réel, IoT, e-commerce, gestion de cache, transactions rapides.

3.2.2.2 Les bases de données orientées colonnes

La représentation orientée colonnes est celle qui se rapproche le plus des tables dans une base de données relationnelles. Elles permettent d'être beaucoup plus évolutive et flexible puisqu'on peut disposer de colonnes différentes pour chaque ligne. Elles peuvent évoluer dynamiquement en nombre et en nom et contrairement à une table relationnelle (pas de champ « NULL »)[33].

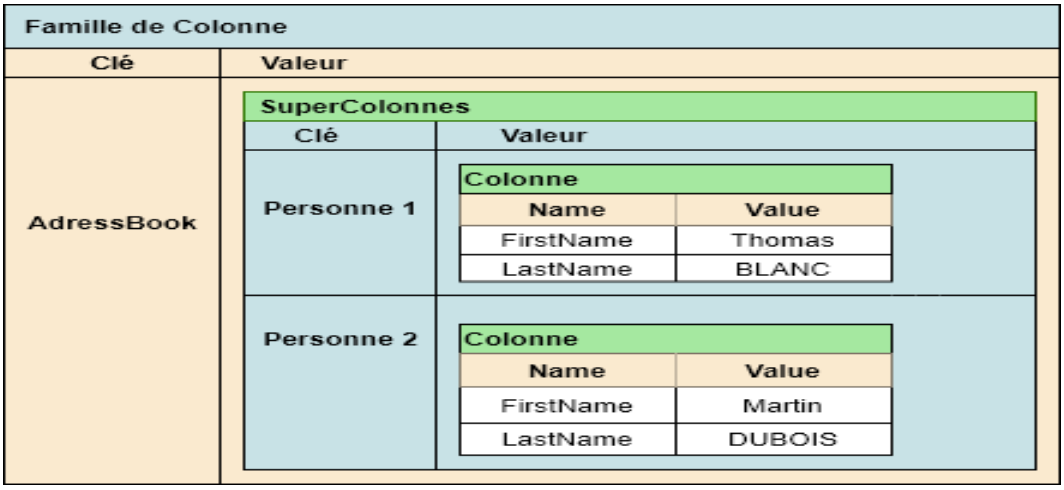


FIGURE 3.5 – Base de donnée orientée colonne.

- **Colonne :** chaque colonne est définie par un couple clé / valeur.
- **Super colonnes :** situées dans les familles de colonnes sont souvent utilisées comme les lignes d'une table de jointure dans le modèle relationnel.
- **Famille de colonnes :** permettent de regrouper plusieurs colonnes (ou supercolonnes).

Exemple d'applications :

Comptage (vote en ligne, compteur, etc), journalisation, recherche de produits dans une catégorie, reportage à large échelle.

3.2.2.3 Les bases de données orientées documents

Ces solutions reposent également sur le paradigme [clé, valeur], et la valeur, dans ce cas, est un document. Ce document a une structure arborescente : il contient une liste

de champs, un champs est associé à une valeur qui peut elle même être une liste. Ces documents sont principalement de type **JSON** [11] ou **XML** [34].

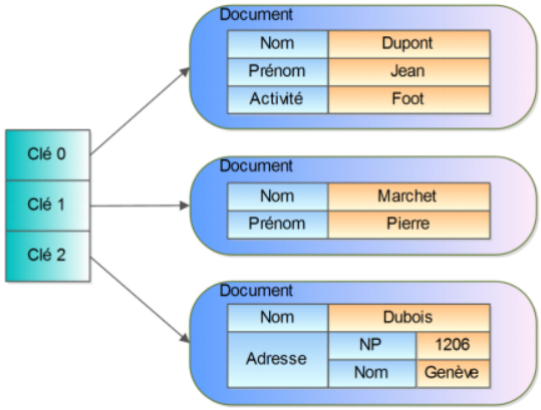


FIGURE 3.6 – Base de donnée orientée document.

Exemple d'applications :

Gestion de contenu (bibliothèques numériques, collections de produits, dépôts de logiciels, collections multimédia, etc.), framework stockant des objets, collection d'événements complexes, gestion des historiques d'utilisateurs sur réseaux sociaux.

3.2.2.4 Les bases de données orientées graphe

Une base de données graphe établit des relations entre les données à l'aide de nœuds et d'arêtes. Le réseau de relation des données est organisé par les points nodaux et leurs connexions les uns avec les autres. Dans le cas de volumes de données aux informations fortement interconnectées, les bases de données graphiques NoSQL présentent une performance considérablement supérieure à celle des bases de données SQL relationnelles. Elles sont principalement utilisées dans le domaine des réseaux sociaux, pour représenter, par exemple, les relations entre les abonnés sur Twitter ou Instagram.

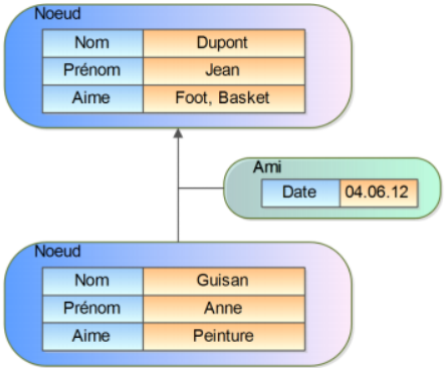


FIGURE 3.7 – Base de donnée orientée graphe.

11. JavaScript Object Notation est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML.

Exemple d'applications :

Réseaux sociaux (recommandation, plus court chemin, cluster...), réseaux SIG [12] (routes, réseau électrique, fret...), web social (Linked Data).

3.2.2.5 Autres base de données

Il existe plusieurs autres bases de données suivant une approche non relationnelle, mais 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 bases de données d'objets utilisent l'idée d'objets dans les langages de programmation et la transforment en systèmes de bases de données.

Exemple : *db4o est un système de gestion de base de données orientée objet Open Source pour des applications Java et .Net* [13].

Bases de données XML : Les bases de données XML sont vraiment des bases de données non SQL. Le langage de requête est XQuery, XPath ou XUpdate et non plus SQL. Ces bases de données permettent de stocker des données au format XML. Par conséquent, la structure de la base de données est hiérarchique. XML est largement utilisé dans les applications Internet, donc une transformation de (anciennement) SQL en XML n'est plus nécessaire.

Exemple : *eXist est une base de données XML basée sur Java open source prenant en charge XQuery et XPath et possède comme CouchDB une interface HTTP RESTful* [14].

3.2.3 Avantage du NoSQL

Les bases de données NoSQL ont été créées en réponse aux limitations du technologie de base de données relationnelle. Comparées aux bases de données relationnelles, les bases de données NoSQL sont plus évolutives et offrent des performances supérieures, et leur modèle de données corrige plusieurs faiblesses du modèle relationnel.
Les avantages de NoSQL sont notamment :

- Capable de gérer un volume important de données structurées, semi-structurées et non structurées.
- Rapidité : NoSQL n'est pas relationnelle. Pas de schéma de bases avec les contraintes sur les champs. Cela apporte de la flexibilité dans la gestion des données et la rapidité.
- Offre une conception de schéma flexible qui peut facilement être modifiée sans interruption de service.
- Programmation orientée objet facile à utiliser.

12. SIG :Système d'information géographique est un système d'information conçu pour recueillir, stocker, traiter, analyser, gérer et présenter tous les types de données spatiales et géographiques

13. .NET est un framework, Il a pour but de faciliter la tâche des développeurs en proposant une approche unifiée à la conception d'applications Windows ou Web, tout en introduisant des facilités pour le développement, le déploiement et la maintenance d'applications.

14. REST (representational state transfer) est un style d'architecture logicielle définissant un ensemble de contraintes à utiliser pour créer des services web. Les services web conformes au style d'architecture REST, aussi appelés services web RESTful, établissent une interopérabilité entre les ordinateurs sur Internet.

- Haute performance.
- Scalibilité facile : Elles offrent un niveau de scalibité excellent sur des environnements clusterisé.
- Faible complexité et latence.
- Tolérance aux pannes et haute disponibilité.
- Moins cher : Les solutions de gestion de base de données no relationnelles sont pour la plupart en Open Source. Les entreprises qui l'utilisent en retirent un bon rapport qualité/prix.

Le NoSQL est principalement adapté aux applications qui cherche :

- Schéma flexible.
- Aucune contrainte et validation à exécuter dans la base de données.
- Données temporaires.
- Pas besoin d'ACIDE.
- Différents types de données.
- Volume de données élevé.

3.2.4 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 de NoSQL sont les suivants :

- Les bases de données NoSQL n'ont pas les fonctions de fiabilité des bases de données relationnelles (ne prennent pas en charge les propriétés ACID).
- Afin de soutenir ACID, les développeurs devront implémenter leur propre code, ce qui rendra leurs systèmes plus complexes.
- Support limité : due à la jeunesse des bases de données NoSQL, le support de la communauté est parfois limité.
- Manque de standardisation : Pas de langage "NoSQL" standard sur les différentes bases de données.
- Aucune contrainte et validation à exécuter dans la base de données.
- Intéropérabilité : La passage d'une base de données NoSQL vers une autre n'est pas transparente pour une application.
- Impossibilité d'avoir en même temps la cohérence, disponibilité et tolérance de partition.

3.3 SQL vs NoSQL

SQL	NoSQL
Prend en charge un puissant langage de requête	Prend en charge un langage de requête très simple.
La base de données relationnelle a un schéma fixe.	La base de données NoSQL n'a pas de schéma fixe.
La base de données relationnelle suit les propriétés ACID. (Atomicité, cohérence, isolation et durabilité)	La base de données NoSQL suit le theoreme de CAP. (Cohérence, disponibilité et tolérance au partitionnement.
Prend en charge les transactions (également les transactions complexes avec jointures).	Prend pas en charge les transactions (ne prennent en charge que les transactions simples).
Utilisée pour gérer les données à faible vitesse.	Utilisée pour gérer les données à haute vitesse.
Les données SGBDR proviennent d'un ou de plusieurs emplacements.	Les données NoSQL arrivent de nombreux endroits.
Gère que les données structurées.	Gère des données structurées, non structurées et semi-structurées.
Les SGBDR ont un point de défaillance unique avec basculement.	Les bases de données NoSQL n'ont pas de point de défaillance unique.
Gère un volume de données modéré.	Gère des données volumineuses ou des données dans un volume très élevé.
Le SGBR a une structure centralisée.	NoSQL a une structure décentralisée.
La base de données relationnelle offre une évolutivité en lecture uniquement.	La base de données NoSQL offre une évolutivité en lecture et en écriture.
La base de données relationnelles sont déployées de manière verticale.	La base de données NoSQL est déployée de manière horizontale.

TABLE 3.1 – Tableau comparatif de SQL et NoSQL .

3.4 Examples BDD NoSql

Il existes différents technologies de base de donnée NoSQL on citera quelques-unes classés selon leurs catégories :

A/ **BDDs orientés clé-valeur :**

- REDIS.
- ORACLE NOSQL.
- VOLDEMORT.
- RIAK.
- INFINISPAN.
- HAZELCAST.

1. **REDIS**

   Redis est un magasin à valeur ajoutée. De plus, c'est le magasin de valeur clé le plus célèbre. Redis prend en charge certains langages C++, PHP, Ruby, Python, Perl, Scala, etc. Redis est fait en langage C. De plus, il est autorisé par BSD [15].[35]

15. La Berkeley Software Distribution ou BSD, « collection de logiciels de Berkeley » en français, est un système d'exploitation dérivé d'Unix et originaire de l'université de Californie à Berkeley.

Caractéristiques :

- Basculement automatique (sans intervention humaine).
- Conserve sa base de données entièrement en mémoire.
- Les transactions sont un moyen d'envoyer en une seule opération un ensemble d'actions.
- Répliquer les données à un nombre quelconque d'esclaves.
- Possibilité de contrôler la durée de vie d'une donnée dans la base.
- Prise en charge de la publication / abonnement.

AVANTAGES	INCONVÉNIENTS
— Prise en charge d'une grande variété de types de données que ce soit binaire, string, object, bitmap, etc.. — Facile à installer. — Très rapide (effectuer environ 11000000 SETs par seconde, environ 81000 GETs par seconde). — Les opérations sont atomiques. — Outil multi-utilitaires (utilisé dans plusieurs cas d'utilisation).	— Ne supporte pas les jointures. — Connaissances requises de Lua[16] pour les procédures stockées. — l'ensemble de données doit s'insérer confortablement dans la mémoire.

TABLE 3.2 – Avantages et incovénients de REDIS

2. **ORACLE NoSQL**

Oracle vient de démarrer la base de données NoSQL avec Oracle NoSQL. Il devient populaire en 2018. Il est moins populaire que les bases de données MongoDB et Casandra. Oracle NoSQL Database implémente une carte allant des clés définies par l'utilisateur aux éléments de données opaques. Bien qu'il enregistre les numéros de version internes pour les combinaisons clé/valeur, il ne gère que la dernière version dans le point de vente. La version 12c d'Oracle est conçue pour le cloud et peut être hébergée sur un ou plusieurs serveurs, et permet la gestion de bases de données contenant des milliards d'enregistrements. Certaines des caractéristiques de la dernière version d'Oracle incluent un cadre de grille et l'utilisation de structures physiques et logiques. Oracle Database 18c offre désormais aux clients une plate-forme performante, fiable et sécurisée pour moderniser facilement et à moindre coût leurs charges de travail transactionnelles et analytiques, que ce soit dans le Cloud, sur site ou en configuration Hybrid Cloud [35].

Caractéristiques :

- Oracle NoSQL Database gère les données volumineuses.
- Prend en charge SQL et est accessible à partir des bases de données relationnelles Oracle.
- Base de données Oracle NoSQL utilisant l'API Java/C pour lire et écrire des données.
- Permet d'accéder aux données par l'intermédiaire du nœud pour la clé demandée.

16. Lua est un langage de script libre, réflexif et impératif. Créé en 1993, il est conçu de manière à pouvoir être embarqué au sein d'autres applications afin d'étendre celles-ci.

AVANTAGES	INCONVÉNIENTS
— Basé sur le concept de programmation PL/SQL. — Les communautés de pairs à pairs aident à résoudre tous les problèmes. — La base de données Oracle est sécurisée et garantit que les données des utilisateurs ne sont pas altérées par des mises à jour rapides.	— Coût élevé pour les petites organisations. — Nécessitent des ressources importantes pour l'installation. — Des mises à niveau matérielles peuvent même s'avérer nécessaires pour implémenter Oracle. — Prend beaucoup de place.

TABLE 3.3 – Avantages et incovénients de ORACLE NOSQL

B/ **BDDs orientés colonnes :**

- HBASE.
- CASSANDRA.
- ACCUMULO.
- HYPERTABLE.

1. **HBASE**

   HBase est une base de données distribuée et non relationnelle qui est conçue pour la base de données BigTable par Google. L'un des principaux objectifs de HBase est d'héberger des milliards (de lignes x millions de colonnes). On peut ajouter des serveurs à tout moment pour augmenter la capacité. Et de multiples nœuds maîtres assureront une haute disponibilité des données. Il est composé en Java 8. Il est autorisé sous Apache et est accompagné d'une API Java simple d'utilisation pour l'accès des clients [35].

   Caractéristiques :

   - Prise en charge de l'échec automatique.
   - Linéairement évolutif.
   - Permet la réplication des données.
   - S'intègre à Hadoop, à la fois comme source et comme destination.

AVANTAGES	INCONVÉNIENTS
— Fournit des recherches rapides pour les grandes tables. — Fournit un accès à faible latence à des rangées individuelles à partir de milliards d'enregistrements. — API Java facile pour le client. — Auto-sharding. — Sans licence. — Gérer de grands ensembles de données sur le dessus du stockage de fichiers HDFS. — Flexible sur la conception des schémas. — Haute vitesse.	— Ne supporte pas la transaction. — Pas de permissions ou d'authentification intégrée. — Indexé et trié uniquement sur clé. — Point de défaillance unique (lorsqu'un seul HMaster est utilisé). — Ne supporte pas la structure SQL. — Problèmes de mémoire sur le cluster.

TABLE 3.4 – Avantages et incovénients de HBASE

2. **CASSANDRA**

   Il a été développé par Facebook pour la recherche dans les boîtes de réception. Cassandra est un système de stockage de données distribué pour le traitement de très grandes quantités de données structurées. Cassandra est utilisé par certaines des plus grandes entreprises telles que Facebook, Twitter, Cisco, Rackspace, eBay, Netflix, et plus [35].

   Caractéristiques :

   - Linéairement évolutif.
   - Maintient un temps de réponse rapide.
   - Prise en charge des propriétés ACID.
   - Supporte MapReduce avec Apache Hadoop.
   - Flexibilité maximale pour répartir les données.
   - Hautement évolutif.
   - Architecture peer-to-peer.

AVANTAGES	INCONVÉNIENTS
— Hautement évolutif. — Aucun point de défaillance unique. — Réplication Multi-DC. — Intégration étroite avec d'autres applications basées sur la JVM [17]. — Plus adapté aux déploiements de centres de données multiples, à la redondance, au basculement et à la reprise d'activité après sinistre.	— Soutien limité pour les agrégations. — Performances imprévisibles. — Ne prend pas en charge les requêtes ad hoc.

TABLE 3.5 – Avantages et incovénients de CASSANDRA

C/ **BDDs orientés documents :**

- MONGODB.
- COUCHDB.
- RAVENDB.
- JERRASTORE.

1. **MongoDB**

   MongoDB est la plus connue des bases de données NoSQL. Il s'agit d'une base de données Open-Source orientée document. MongoDB est une base de données évolutive et accessible. Il est en C++. MongoDB peut également être utilisé comme système de fichiers. Dans MongoDB, JavaScript peut être utilisé comme langage de requête. Des performances étonnantes et de nouvelles fonctionnalités ont propulsé cette base de données NoSQL à la première place.[35]

17. JVM acronyme de Java Virtual Machine (en français "la machine virtuelle Java") est un appareil informatique fictif qui exécute des programmes compilés sous forme de bytecode Java

Caractéristiques :

- Fournit un rendement élevé.
- Auto-sharding.
- Exécution sur plusieurs serveurs.
- Prise en charge de la réplication maître-esclave.
- Les données sont stockées sous forme de documents de style JSON.
- indexer n'importe quel champ d'un document.
- Il a une configuration d'équilibrage de charge automatique en raison des données placées dans des tessons.
- Facile à administrer en cas d'échec.

AVANTAGES	INCONVÉNIENTS
— Facile à installer. — MongoDB Inc.[18] offre un soutien professionnel à ses clients. — Prise en charge des requêtes ad hoc[19]. — Base de données haute vitesse. — Base de données sans schéma. — Base de données évolutive horizontalement. — La performance est très élevée.	— Ne supporte pas les jointures[20]. — La taille des données est élevée. — L'imbrication des documents est limitée. — Augmenter l'utilisation inutile de la mémoire.

TABLE 3.6 – Avantages et incovénients de MongoDB

2. **COUCHDB**

CouchDB est une base de données NoSQL Open Source qui utilise JSON pour stocker des informations et JavaScript comme langage de requête. CouchDB applique un type de système de contrôle multi-versions pour éviter le blocage du fichier DB pendant l'écriture. C'est Erlang. C'est autorisé sous Apache. Il est classé 1er sur la liste Best NoSQL Database 2016 pour sa popularité [35].

Caractéristiques :

- Cartographier/réduire la liste et afficher.
- Assurer la sécurité au niveau de la base de données.
- L'authentification s'ouvre via un cookie de session comme une application web.
- JSONP gratuitement.
- Suivre le stockage des documents.
- Prise en charge des propriétés ACID.
- Fournir la forme la plus simple de réplication.
- Interface utilisateur graphique basée sur un navigateur pour gérer vos données, vos autorisations et votre configuration.

18. MongoDB Inc. est une société de logiciels américaine qui développe et fournit un support commercial pour la base de données open source MongoDB

19. Ad hoc signifie en latin "à cet effet". on peut l'appeler une requête "à la volée" ou une requête "juste comme". C'est le genre de requête SQL que le tape simplement sans le savoir.

20. la jointure ou appariement est l'opération permettant d'associer plusieurs tables ou vues de la base par le biais d'un lien logique de données entre les différentes tables ou vues

AVANTAGES	INCONVÉNIENTS
— Utilise Map/Reduce, l'interrogation des données est quelque peu séparée des données elles-mêmes. — Stocker toutes les données JSON.	— Les requêtes arbitraires coûtent cher. — Un peu plus d'espace avec CouchDB. — Ne supporte pas XML.

TABLE 3.7 – Avantages et incovénients de COUCHDB

D/ **BDDs orientés graphes :**

- Neo4J.
- INFINATEGRAPHE
- INFOGRID.
- HYPERGRAPH DB
- ALLEGROGRAPH.

1. **Neo4J**

   Neo4j est considéré comme une base de données de graphes native car il implémente efficacement le modèle de graphes de propriétés jusqu'au niveau du stockage. Cela signifie que les données sont stockées exactement comme vous les enregistrez sur un tableau blanc et que la base de données utilise des pointeurs pour naviguer et parcourir le graphique. Neo4j dispose à la fois d'une édition Communauté et d'une édition Entreprise de la base de données. L'édition Enterprise inclut tout ce que l'édition Community a à offrir, plus les exigences supplémentaires de l'entreprise telles que les sauvegardes, la mise en grappe et les capacités de basculement[35].

   Caractéristiques :

   - Supporte les contraintes UNIQUES.
   - Prise en charge des règles ACID (Atomicité, Cohérence, Isolation et Durabilité).
   - Prend en charge les deux API Java : API de chiffrement et API Java native.
   - Prise en charge des index à l'aide d'Apache Lucence.
   - Langage de requête facile Neo4j CQL.
   - Contient une interface utilisateur pour exécuter les commandes CQL : Navigateur de données Neo4j.

AVANTAGES	INCONVÉNIENTS
— Facile à récupérer les détails de ses nœuds adjacents ou de ses relations sans jointures ou index. — Facile à apprendre les commandes du langage de requête Neo4j CQL. — Pas besoin de jointures complexes pour récupérer les données. — Représente des données semi-structurées très facilement. — Haute disponibilité pour les applications temps réel des grandes entreprises. — Réglage simplifié.	— Ne supporte pas Sharding [21]

TABLE 3.8 – Avantages et incovénients de Neo4J

D'autres base de données peuvent proposées plusieurs services en même temps comme Amazon DynamoDB :

Amazon DynamoDB

DynamoDB utilise un modèle de base de données NoSQL, qui n'est pas relationnel, ce qui permet d'avoir des documents, des graphiques parmi ses modèles de données. Chaque requête DynamoDB est exécutée par une clé primaire identifiée par l'utilisateur, qui identifie de manière unique chaque élément. Il libère également les clients du fardeau de l'exploitation et de la mise à l'échelle d'une base de données distribuée. Ainsi, le provisionnement matériel, l'installation, la configuration, la réplication, le patch logiciel, la mise à l'échelle des clusters, etc. sont gérés par Amazon [35].

Caractéristiques :

- Haute évolutivité.
- Plage de hachage pour l'indexation d'une plage de valeurs.
- Stockage des données dans les partitions.
- Utilise JSON comme protocole de transport et non comme format de stockage.

AVANTAGES	INCONVÉNIENTS
— Facile à installer. — Fournir une API DynamoDB AWS de bas niveau. — Mise à l'échelle automatique. — Réduit la complexité de la gestion de la haute disponibilité et de la mise à l'échelle pour les périodes de pointe d'utilisation. — Chiffrement au repos. — La sécurité de DynamoDB est régie par l'AWS Identity.	— Ne sauvegarde pas gratuitement vos tables. — Limite de taille.

TABLE 3.9 – Avantages et incovénients de Amazon DynamoDB

21. Le Sharding permet d'organiser de manière horizontale la répartition des données. Les data shards, les jeux de données partitionnés peuvent être distribué sur des serveurs aux coûts bien plus raisonnables.

3.5 Vers le NewSQL

Globalement les systèmes NoSQL ne respectent pas les propriétés ACID ou en tout cas pas complètement. Cet aspect ne permet pas d'offrir une grande sûreté dans l'accès aux données. Par ailleurs la base de données NoSQL reste très contraignante par certains aspects. Ainsi le traitement des requêtes de type OLAP [22] nécessite une programmation importante au niveau applicatif.
Pour l'environnement Big Data, la prise en charge de plus de clients ou d'un débit plus élevé nécessitait une mise à niveau vers un serveur plus grand. Cela signifiait que la mise en œuvre d'une architecture évolutive nécessitait un modèle de programmation NoSQL ou reposait sur le partitionnement et la réplication explicite. Il n'y avait aucune solution offrant une sémantique ACID complète. Cette tension est ce qui a inspiré le mouvement NewSQL.

3.5.1 Présentation des BDD modernes le NewSQL

NewSQL est un stockage distribué potentiellement entièrement en mémoire et pouvant être requêté classiquement par une interface SQL. NewSQL est tiré du monde NoSQL mais reste différent. Comme NoSQL il s'agit d'une nouvelle architecture logicielle qui propose de repenser le stockage des données. Elle profite des architectures distribuées, des progrès du matériel et des connaissances théoriques depuis 35 ans. Mais contrairement à NoSQL elle permet de conserver le modèle relationnel au cœur du système.

NewSQL est né de la rencontre de 3 types d'architecture, **relationnelle**, **rnon-relationnelle** et **rgrille de données** appelée également **rcache distribué**, comme indiqué dans la **Figure 3.8** . En effet il se positionne comme un stockage distribué conçu dans le prolongement des architectures NoSQL, pour des accès transactionnels à fort débit, au moyen d'une interface SQL. D'un point de vue évolutivité, il se situe en tant que concurrent direct des solutions NoSQL. Mais contrairement à ces solutions il conserve une interface relationnelle via le SQL, ce qui est l'une de ses forces. Enfin la plupart des solutions NewSQL proposent un stockage en mémoire. Ce stockage en mémoire distribué sur plusieurs machines sous forme de grille de données est largement utilisé depuis une dizaine d'années dans les environnements où une faible latence est critique, notamment dans certaines applications des banques d'investissement. Les solutions NewSQL partagent ainsi un positionnement intermédiaire entre les solutions NoSQL et les grilles de données [36].

22. OLAP, acronyme de Online Analytical Processing, est une technologie permettant d'effectuer des analyses de données multidimensionnelles au sein de bases de données créées à cet effet

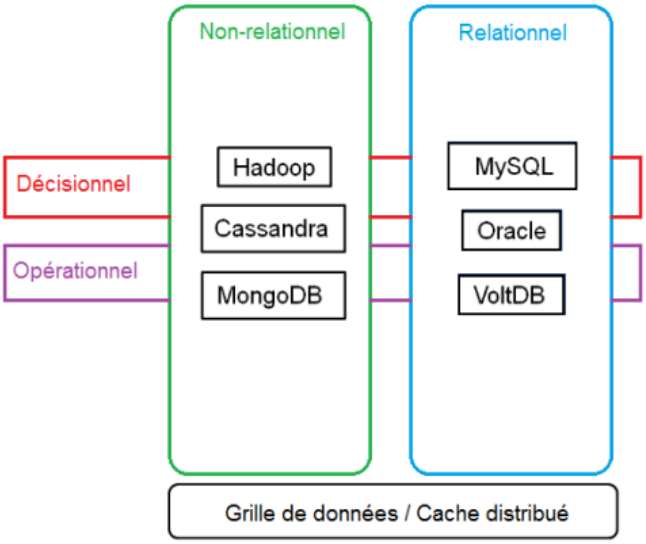


FIGURE 3.8 – Naissance du NewSQL à partir de 3 architectures.

3.5.2 L'architecture NewSQL

L'architecture NewSQL reprend des expériences antérieures du SQL relationnel et du NoSQL plusieurs caractéristiques, tout en ayant certaines particularités en termes de choix et d'avantages [36] :

1. Le choix d'une interface SQL et d'un schéma relationnel.
2. Le schéma relationnel avec des limitations pour faciliter la distribution des données et des traitements.
3. La distribution et la réplication des données pour assurer l'évolutivité et la résilience

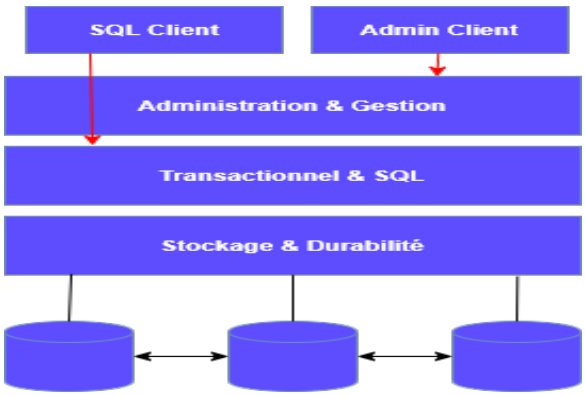


FIGURE 3.9 – L'architecture d'une base de données NewSQL populaire NueDB.

3.5.3 Les avantages de la solution NewSQL

La solution NewSQL présente des avantages intéressants en termes de performances par rapport à ses prédécesseurs [36] :

- Il améliore la cohérence des données.
- Similaire aux requêtes SQL.
- Elle présente une architecture qui a de meilleures performances par nœud que les solutions classiques de type SGBD relationnel.
- Elle minimise la complexité des applications tout en améliorant la consistance des données et en fournissant un support transactionnel complet.
- Pourrait utiliser des extensions SQL.
- Il fournit un style de clustering NoSQL mais utilise une modélisation et des requêtes SQL familières.
- Elle fournit des analyses plus riches des traitements.

3.5.4 Les limites des bases de données NewSQL

La solution NewSQL est confrontée à quelques limitations l'empêchant d'atteindre un niveau de maturité suffisant [36] :

- sont architecture n'est pas approprié dans les applications avec un volume supérieur à quelques téraoctets.
- Cette architecture nécessite un matériel spécifique avec des capacités importantes de stockage en mémoire, ce qui revient très onéreux.
- Malgré son attitude à vouloir intégrer le modèle relationnel, le NewSQL fournit un accès limité aux outils SQL traditionnels.

Résumé *Une base de données NewSQL conserve la structure classique en colonnes mais fait appel à différents procédés pour conserver la rapidité même sur de larges volumes. En revanche, cette technologie n'a pas le recul suffisant pour aller plus loin et n'a pas encore pu suffisamment faire ses preuves. Par conséquent les entreprises sont encore réticentes à l'adoption de cette toute nouvelle architecture.*

3.5.5 Exemples des bases de données NewSQL

NuoDB : C'est un SGBD distribué qui pourrait être décrit comme une base de données à faible latence. Il permet à l'utilisateur d'interagir avec lui de manière transactionnelle en prenant en charge les opérations ACID et la réplication des données.

FIGURE 3.10 – Logo NuoDB.

VoltDB Il s'agit d'une base de données distribuée en mémoire qui est considérablement plus rapide que les bases de données SQL. c'est une base de données flexible qui prend en charge le stockage JSON. Il est le mieux adapté aux applications à lecture fréquente et à faible fréquence d'écriture.

Figure 3.11 – Logo VoltDB.

Clustrix c'est une base de données distribuée qui prend en charge l'analyse en temps réel, elle est optimisée pour les transactions massives. Il prend en charge certains outils de BI[23] et une récupération rapide des données.

Figure 3.12 – Logo Clustrix.

Conclusion

Dans l'ensemble, les bases de données NoSQL, tout comme les bases de données relationnelles, présentent des avantages et des inconvénients les uns par rapport aux autres. Les bases de données NoSQL sont un composant important du Big Data pour stocker et récupérer de grandes quantités de données. Bien que les bases de données NoSQL offrent des gains de performances, certains chercheurs sont prudents et sceptiques quant à la cohérence des données.

La mousse dans l'espace de gestion des données est importante - et notre tendance à parler en termes de catégories (SQL, NoSQL, NewSQL) par rapport aux problèmes rend difficile pour les développeurs de logiciels de comprendre ce qui est dans la boîte à outils. Les offres actuelles de nouvelles bases de données ne sont pas toutes les mêmes - et la reconnaissance de la façon dont l'ADN derrière chacune aide ou entrave la résolution des problèmes est la clé pour choisir la meilleure solution.

L'un des problèmes difficile qui se pose dans le contexte des systèmes de stockage Cloud 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 de remédier à ce problème plusieurs modèles spécifiques ont été proposés pour offrir des garanties de cohérence plus faibles ou assouplies.

Dans le chapitre suivant on va se concentrer plus précisément sur le problème de disponibilité et de cohérence des données dans de tels bases de données, nous étudierons par la suite les différentes méthodes qui peuvent régler ce problème.

23. L'informatique décisionnelle (en anglais business intelligence (BI) ou decision support system (DSS)) est l'informatique à l'usage des décideurs et des dirigeants d'entreprises.

Deuxième partie

Analyse

Problématique de la cohérence et de la disponibilité des données

Alors que le marché du Cloud Computing continue de se développer, une variété de modèles de déploiement Cloud apparaît. Les modèles de Cloud sont généralement classés, d'une part, en fonction de l'emplacement du déploiement de l'environnement de Cloud Computing (qui distingue le Cloud public du Cloud privé et hybride) et, d'autre part, en fonction des services informatiques et de la couche applicative fournis par le Cloud (distinction entre Clouds IaaS, PaaS et SaaS).

La croissance constante des volumes de données, introduit plusieurs défis tel que, le stockage, l'accès, la gestion et le traitement incluant la recherche des données. En plus, les systèmes de stockage de données Cloud doit généralement fournir une qualité de service fiable pour satisfaire les engagements contractuels SLA (Service Level Agrement). Dans ce contexte, La réplication de données est une technique largement utilisée dans les environnements distribués qui stocke des données sur plusieurs sites, elle fournit la disponibilité et la performance. Si un site de stockage tombe en panne, le système peut continuer à fonctionner en utilisant des données répliquées, ce qui augmente la disponibilité et la tolérance aux pannes. En même temps, comme les données sont stockées sur plusieurs sites de stockage. La réplication des données permet de rapprocher les données des utilisateurs pour diminuer la latence, ou de créer de nombreuses copies pour des données extrêmement populaires afin de répartir la charge sur différents serveurs. Ce qui permet d'augmenter l'efficacité du système, de réduire la consommation de la bande passante et d'améliorer l'évolutivité du système. Donc, la réplication des données est très utile pour offrir de bonnes performances d'accès. La réplication est largement utilisée dans les systèmes parallèles, base de données, mobile et Cloud Computing.

Cependant, la réplication représente un obstacle important pour garantir la cohérence, pour ce faire des stratégies de mise à jour et de propagation des données doivent être mises en œuvre, autrement dit 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 aux partitions dans les systèmes distribués.

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

Pour mettre en évidence cette problématique, nous présentons un exemple illustré dans la figure ci-dessous :

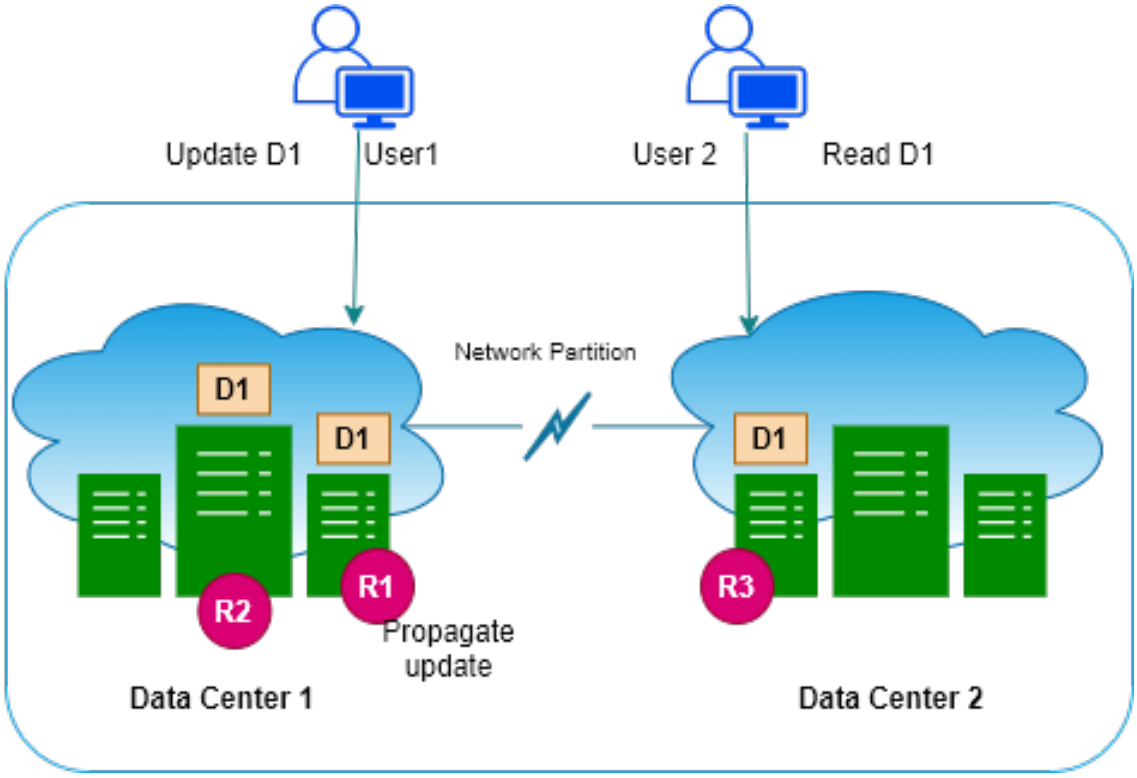


FIGURE 3.13 – Cohérence et disponibilité dans les systèmes distribuées.

Nous Pouvons observer que l'utilisateur 2 effectue une demande de lecture pour l'élément de données D1 dans la réplique R3 (Datacenter 2), après que l'utilisateur 1 a mis à jour l'élément de données D1 dans la réplique R1 (Datacenter 1) en présence d'un partition réseau qui isole les deux centres de données. Étant donné que la partition réseau signifie que la mise à jour effectuée par l'utilisateur 1 n'a pas été propagée à la réplique R3, il existe deux scénarios possibles :

1. **Scénarios 1** : Les répliques peuvent être disponibles et l'utilisateur 2 lira les données obsolètes, violant ainsi la cohérence.
2. **Scénarios 2** : L'utilisateur 2 attend 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 la CAP ont conduit à la prolifération de systèmes non-ACID pour la construction d'applications basées sur le cloud, appelés BASE (les 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).

Approche BASE

L'approche BASE selon Brewer 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 [37] :

- **Basically Available** (Disponibilité basique) : Même en cas de désynchronisation ou de panne d'un des nœuds du cluster, le système reste disponible selon le théorème CAP.
- **Soft-state** (Cohérence légère) : Cela indique que l'état du système risque de changer au cours du temps, sans pour autant que des données soient entrées dans le système. Cela vient du fait que le modèle est cohérent à terme.
- **Eventual consistancy** (Cohérence à terme) : Cela indique que le système devient cohérent dans le temps, pour autant que pendant ce laps de temps, le système ne reçoive pas d'autres données.

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.

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 comme un spectre au lieu d'alternatives qui s'excluent mutuellement.

On peut résumer 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)*

De tels systèmes **non ACID** offre des modèles de cohérence distincts,chacun essayant de trouver une soplution au compromis du théorème de **CAP** et chacun essayant de trouver un équilibre entre la disponibilité et la cohérence.Cette problématique est un sujet d'actualité la communauté des bases de données NoSQL .

Notre objectif dans se qui va suivre sera de recenser tout les travaux en cours de réflexion sur cette problématique, de les analyser, les comparer et d'en faire une synthèse .

Dans le prochaine chapitre on va voir les différent modèle de cohérence de donnée, puis par la suite les différentes méthodes de réplication de la cohérence.

Chapitre 4

Disponibilité et cohérence des données dans les BD NoSQL - Étude et synthèse -

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. 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 [38].

L'un des problème difficile qui se pose dans le contexte des systèmes de stockage Cloud 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 de remédier à ce problème plusieurs modèles spécifiques ont été proposés pour offrir des garanties de cohérence plus faibles ou assouplies [20].

Dans ce chapitre nous allons voir tout d'abord les concepts généraux liés à la gestion de BDD Cloud puis nous allons revenir sur le conflit entre la cohérence et la disponibilité dans les systèmes réparti et nous représenterons les différentes méthodes de cohérences proposés afin de palier à ce problème.

Partie A

4.1 Concepts généraux

4.1.1 Réplication et cohérence des données :

La réplication des données sur plusieurs nœuds est une solution efficace afin d'avoir de meilleurs performances et une disponibilité élevé des données. 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) [39] .

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 plusieurs noeuds est celui de la cohérence des données au niveau des différentes répliques. Afin de gérer cette dernière en a deux points de vue essentiels :

- 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.

4.2 La réplication :

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

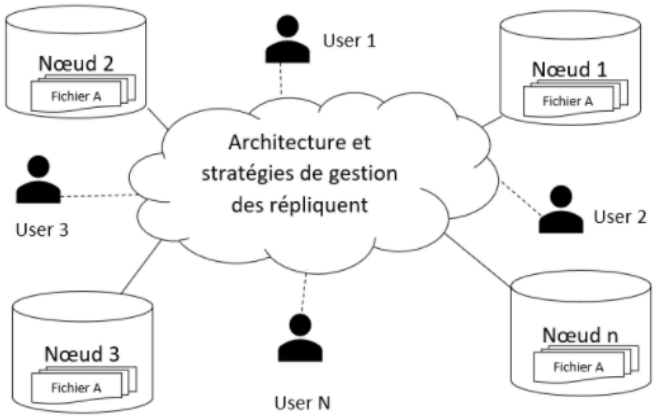


FIGURE 4.1 – Environnement d'utilisation de la technique de réplication.

Dans le cas de la centralisation des données la réplication n'existe pas comme la montre la figure 4.2 [40] :

- **Aucune réplication :** Les données ne sont pas dupliquées entre les sources du système (cas de la centralisation).

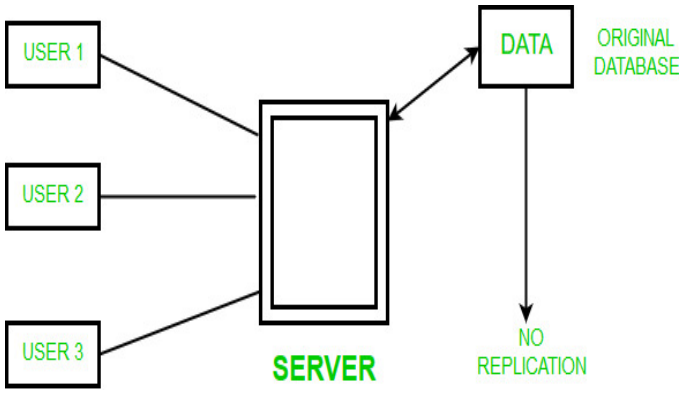


FIGURE 4.2 – Schéma représentant l'absence de réplication.

La réplication peut se faire sur deux niveaux [40] :

- **Réplication partielle :** Les données sont répliquées sur quelques sources du système.

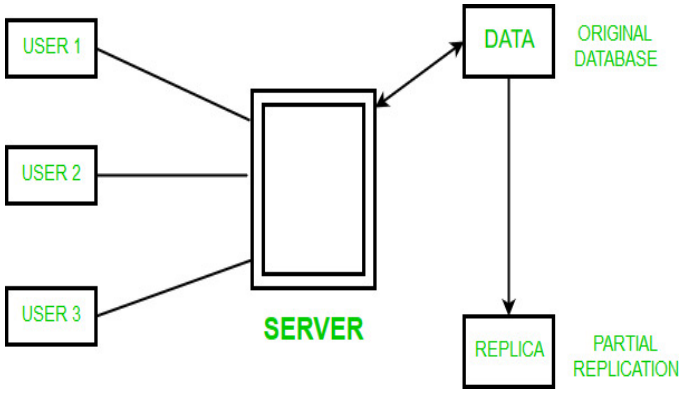


FIGURE 4.3 – Schéma représentant la réplication partielle.

- **Réplication totale :** Les données sont répliquées sur toutes les sources du système.

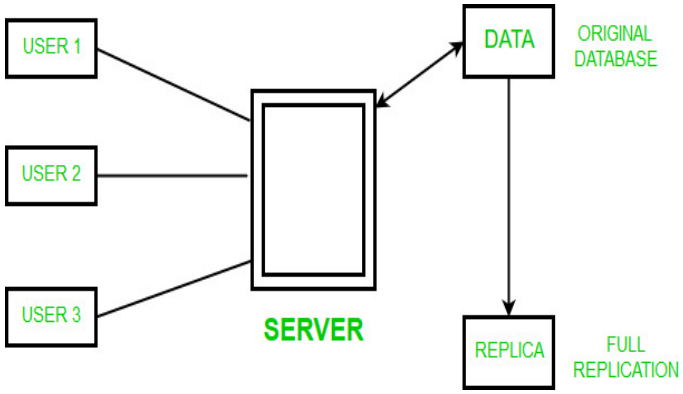


FIGURE 4.4 – Schéma représentant réplication total.

Remarque 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 [41].

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

4.2.1 Les avantages de la réplication :

Comme la réplication rend les données accessibles à plusieurs hôtes ou centres de données, elle simplifie le partage de données entre systèmes à grande échelle en répartissant la charge du réseau entre les systèmes hétérogènes [42].

- Une amélioration de la fiabilité ou bien une sûreté de fonctionnement.
  Exemple :
  - ✓ Si un nœud tombe en panne, il est toujours possible d'obtenir les données à partir d'un autre nœud.
  - ✓ 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 local) même en l'absence de toute connexion à un serveur central, de sorte que l'utilisateur ne soit 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.

4.2.2 Les 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 cohérence.
- **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, le probleme de la cohérence des répliques fera l'objet de notre travail.

4.3 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 ce qui offre la garantie d'une vue cohérente des données dispersées.

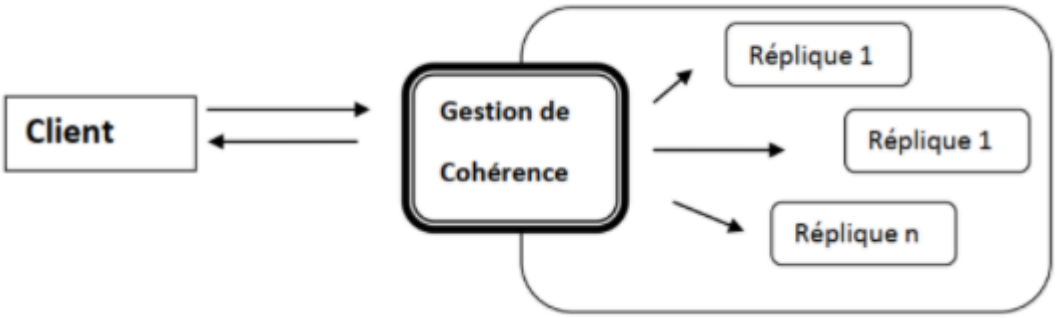


FIGURE 4.5 – Vue cohérente des donnée.

La gestion de la 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.

4.3.1 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 [43].
- **Cohérence faible :** Une cohérence de répliques 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 [43].

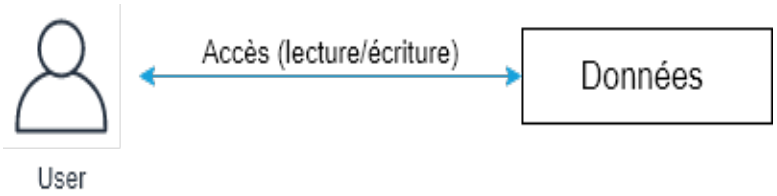


FIGURE 4.6 – Vue idéale des données.

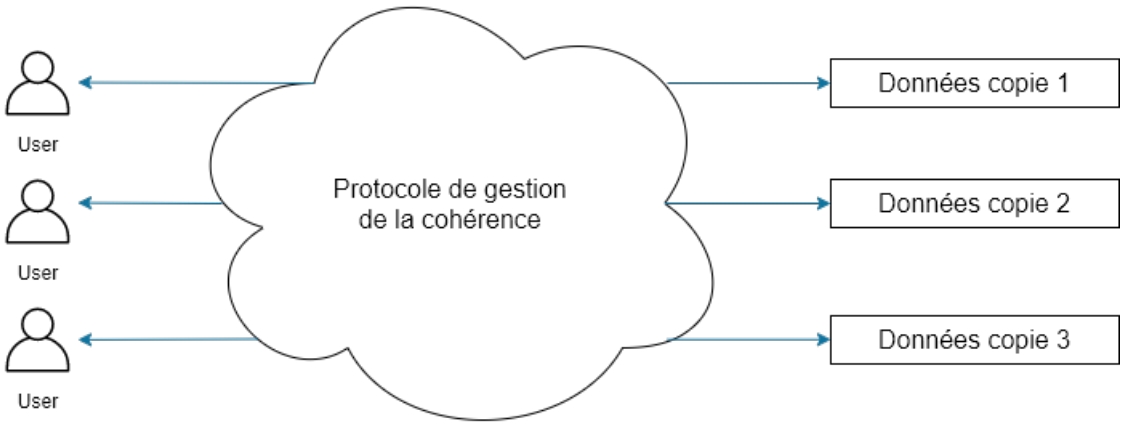


FIGURE 4.7 – Vue réelle des données.

4.3.2 Les protocoles de cohérence

Il existe deux catégories de protocoles de cohérence [44] :

1. Protocoles pessimistes
2. Protocoles optimistes

1. **Les protocoles pessimistes :**
   Le but de cette approche est de faire l'illusion qu'il n'existe qu'une réplique. Elle assure ainsi une cohérence forte, c'est à dire, il n'y a pas de divergence entre les répliques. La réplication pessimiste ne supporte pas la mobilité et le passage à l'échelle .

   Inconvénients :

   - Elle est très mal adaptée à des environnements incertains ou instables, tels que les systèmes mobiles ou les grilles à fort taux de changement.
   - Elle ne peut supporter le coût de mise à jour quand le degré de réplication est très élevée.
   - Elle est inadaptée à des environnements qui exigent un partage de données tels que les environnements collaboratifs.

   Plusieurs protocoles pessimistes ont été définis, parmi lesquels nous pouvons citer les protocoles ROWA, ROWAA et Quorum.

   - **ROWA**(Read One Write All) : consiste à écrire dans toutes les copies et lire une seule copie. Ce protocole garantit que toutes les copies sont identiques à tout moment, mais l'inconvénient de ce protocole est que tout le système va être bloqué si un nœud tombe en panne jusqu'à ce que la panne soit réparée.
   - **ROWAA** (Read Only Write All Available) : Pour remédier au problème précédent, le protocole ROWAA propage les mises à jour uniquement sur les copies qui ne sont pas en panne. Quand une copie reprend sa disponibilité, elle doit d'abord se synchroniser pour effectuer les modifications manquantes. Toutes les copies disponibles sont mise à jour de manière synchrone et les copies indisponibles sont mises à jour de manière asynchrone.
   - **QUORUM** : C'est un protocole de cohérence pessimiste, dans lequel les mises à jour se font de manière synchrone sur un sous-ensemble de copies. Ce sous-ensemble forme ce que l'on appelle un quorum en écriture. Pour les lectures, elles se font sur un autre sous-ensemble, appelé quorum de lecture. Une requête d'écriture n'est validée que lorsque au moins$((N/2)+1)$ copies sont mises à jour. Lors d'une requête de lecture, il est alors nécessaire de consulter au moins $(N/2)$ copies, où N est le nombre de répliques dans le système.

2. **Les protocoles optimistes :**
   A la différence de l'approche pessimiste, cette approche optimiste autorisent l'accès à n'importe quelle réplique et ce à tout instant. Dans ce cas, les copies peuvent diverger et un utilisateur peut donc observer des copies d'un même objet avec des valeurs différentes.

   De cette manière, il est alors possible d'accéder à une réplique qui n'est pas forcément cohérente, ce qui fait que cette approche tolère une certaine divergence entre les répliques. Par contre, quand toutes les opérations auront été propagées à toutes les répliques, alors toutes les copies devront être identiques.
   En présence de divergence, ce type d'approche nécessite une phase de détection de divergence entre répliques puis une phase de correction de cette divergence, pour ramener les répliques vers un état cohérent. Bien qu'elle ne garantisse pas une cohérence forte comme dans le cas pessimiste, l'approche optimiste possède néanmoins un certain nombre d'avantages que nous pouvons résumer comme suit :

- **Disponibilité :** Les protocoles optimistes fonctionnent bien dans les réseaux lents et incertains, parce qu'ils peuvent propager des mises à jour sans bloquer les accès à toutes les répliques.
- **Flexibilité de gestion du réseau :** Les protocoles optimistes fonctionnent également bien dans les réseaux dynamiques. Une telle propriété est essentielle dans les environnements mobiles.
- **Scalabilité :** Les protocoles optimistes peuvent soutenir un plus grand nombre de répliques, parce que la propagation des mises à jour exige moins de coordination entre les copies.

Comparaison des protocoles de cohérence :

Caractéristiques	Protocole Pessimiste	Protocole Optimiste
— Synchronisation.	— Immédiate.	— Retardée.
— Cohérence.	— Forte.	— Faible.
— Disponibilité.	— Faible.	— Forte.
— Temps d'accès.	— Important.	— Réduit.
— Performance (Temps de réponse).	— Faible.	— Bonne.
— Taille des applications.	— Petite.	— Grande échelle.
— Blocage des utilisateurs.	— Oui.	— Non.

Table 4.1 – Comparaison des protocoles de cohérence

La réplication optimiste est utilisée dans plusieurs domaines d'application, y compris la gestion de données à large échelle, les systèmes d'informations et les environnements collaboratifs.

4.4 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 liée à la cohérence de données.

Pour garder la bonne gestion des bases de données dans les systèmes distribuées il faut veiller au respect de certains critères [20] :

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 des 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.

Tolérance aux pannes : Le stockage des 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 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.

Partie B

4.5 Les modèles de cohérence :

Comme discuté précédemment, mettre en place un modèle de cohérence est l'un des problèmes les plus importants dans la conception des systèmes de stockage à grande échelle [45]. En règle générale, la cohérence des données peut être divisée en deux catégories [20] :

- Du point de vue des données, le système de stockage de données distribué synchronise les opérations d'accès aux données de tous les processus pour garantir des résultats corrects.
- Du point de vue du client, le système synchronise uniquement les opérations d'accès aux données du même processus, indépendamment des autres, pour garantir leur cohérence.

Remarque :

Cette perspective est justifiée car il arrive souvent que les mises à jour partagées soient rares et accèdent principalement à des données privées.

Cependant, aujourd'hui, les chercheurs cherchent les moyens de présenter un hybride des modèles de cohérence par rapport aux exigences des applications. L'un des principaux objectifs de cette recherche est d'introduire certains des modèles de cohérence qui, contrairement aux modèles hybrides, sont non seulement capables d'assurer la cohérence des systèmes de stockage distribué, mais sont également capables de couvrir les besoins des applications qui sont généralement répondu par les modèles de cohérence hybride [46].

Les modeles de coherence peuvent etre orienté données (Data-centric) ou orienté clients (Client-centric).

4.5.1 Modèles de cohérence centrés sur les données (Data-Centric) :

Dans cette perspective, les modèles de cohérence cherchent à garantir que les opérations d'accès aux données suivent certaines règles qui garantissent le bon fonctionnement du système de stockage. Ces règles sont basées sur la définition des résultats attendus après les opérations de lecture et d'écriture (même si ces opérations sont effectuées simultanément).

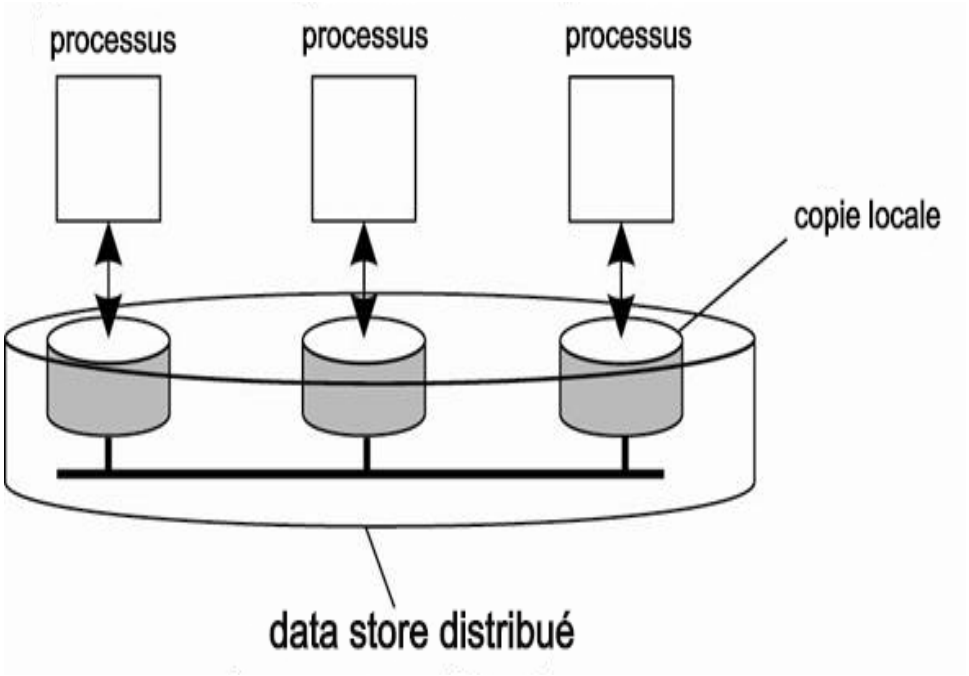


FIGURE 4.8 – Schéma représentant le Modèles de cohérence centrés sur les données.

Cependant, l'absence d'une horloge globale rend l'identification de la dernière opération d'écriture difficile, ce qui nécessite certaines restrictions sur les valeurs de données qui peuvent être renvoyées par une opération de lecture.

Les modèles de cohérence les plus utilisées et qui entrent dans cette catégorie sont :

- La cohérence stricte (strict consistency)
- La cohérence séquentielle (sequential consistency)
- La cohérence causale (causal consistency).
- La cohérence PRAM (PRAM consistency).
- La cohérence faible (weak consistency).

1. **Strict consistency :**

   La cohérence stricte est le modèle de cohérence le plus solide qui nécessite une synchronisation globale permanente. Cette synchronisation se fait en utilisant une heure globale absolue (un timestamp). La création de cette synchronisation par le temps physique entre les serveurs est dans une certaine mesure impossible. En d'autres termes, les répliques doivent être synchronisées globalement et en permanence. Ce modèle est si coûteux, alors que le système n'a pas besoin d'être toujours synchronisé globalement. Cependant, dans les systèmes distribués, cette simplicité impose de nombreuses dépenses. En ce qui concerne le comportement du modèle de cohérence sur les répliques [46].

   Dans un tel système, il y'a un chevauchement entre les sous ensembles de répliques qui sont mis à jours dans le cadre de l'écriture, et les sous ensembles de répliques consultés lors d'une lecture, c'est ainsi qu'un système strictement cohérent peut toujour garantir une lecture des données les plus récentes.

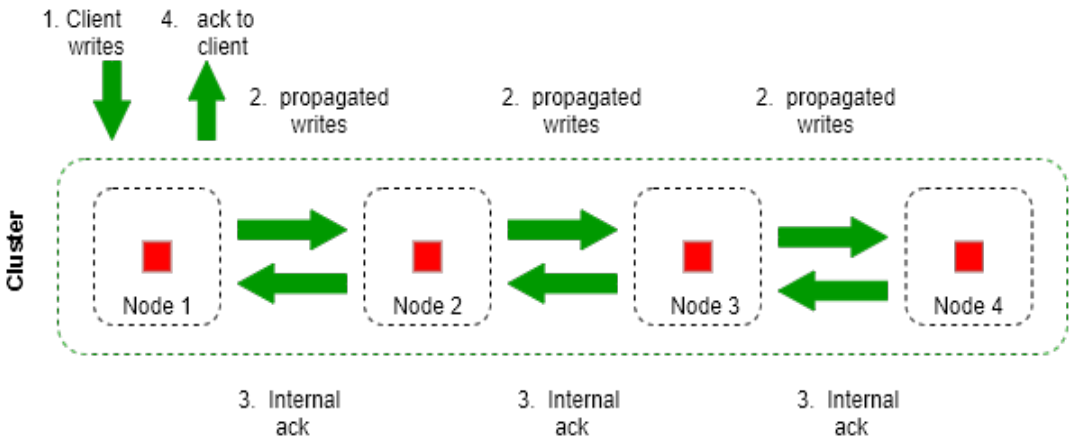


FIGURE 4.9 – Système strictement cohérent.

à travers la **figure 4.9** nous pouvons observer que lors d'une écriture(mise à jour) au niveau d'un système distribuée strictement cohérent au niveau d'un noeud (serveur), cette écriture est synchronisée juste après vers toutes les autres répliques simultanément, ainsi il n'y a pas d'incohérence lors de la lecture des données [47].

Pour résumer un tel système :

- *Renvoie toujours la dernière écriture : pour toute opération d'écriture entrante, une fois qu'une écriture est acquittée au client, la valeur mise à jour est visible lors de la lecture depuis n'importe quel nœud.*
- *Garantie la résilience des données : pour toute opération d'écriture entrante, une fois qu'une écriture est acquittée vers le client, la mise à jour est protégée contre la défaillance du nœud avec redondance.*

Ce modèle de cohérence fournit le niveau de cohérence le plus élevé. Il indique que, si une opération d'écriture est effectuée sur un élément de données, le résultat doit être instantanément visible pour tous les processus, quelle que soit la réplique sur laquelle l'opération d'écriture a été exécutée. Pour y parvenir, un ordre de temps global absolu doit être maintenu.

Inconvénient :
Un tel modèle de cohérence n'est pas toujours utilisé. Principalement parce que la mise en œuvre d'une cohérence stricte peut avoir un impact significatif sur les performances. Plus précisément, la latence et le débit seront affectés. L'ampleur de l'impact dépend du scénario.

> **Exemple :** *Dans le cas d'une application qui surveillerait la charge des processeurs d'un réseau d'ordinateurs afin de choisir sur quelle machine exécuter une tâche, les données qui représentent la charge des processeurs évoluent sans arrêt : il n'est pas nécessaire d'avoir à tout moment accès à la dernière valeur de la charge [48].*

2. **Sequential consistency : (SC)**

Une autre variante des modèles de cohérence est la cohérence séquentielle. Ce modèle a d'abord été défini par Lamport [1] en 1979. Dans la discipline de la mémoire partagée pour les systèmes multiprocesseurs. Ce modèle est une variante plus simple du modèle de cohérence stricte mais reste assez strict car c'est un modèle qui requiert que [20] :

- Toutes les opérations soient sérialisées dans le même ordre dans toutes les répliques.
- Toutes les opérations du même processus soient exécutées dans l'ordre où le système de stockage les a reçues.

Dans ce modèle, l'opération d'écriture est vue entre les processus avec un ordre égal, cependant l'opération de lecture effectuée par les autres processus n'est pas observable.
Afin de bien comprendre ce modèle on va se focaliser sur le cas ou il y'a qu'une seul réplique à laquelle plusieurs processus peuvent accéder [46] :

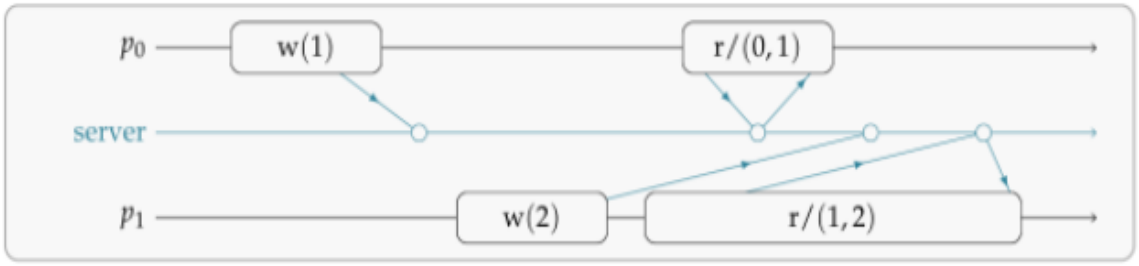


FIGURE 4.10 – Cohérence séquentielle.

Afin de décrire le comportement attendu d'un objet partagé c'est d'exiger que ce dernier imite l'existence d'une seule copie de l'objet dans le réseau, à laquelle tous les processus accèdent séquentiellement. Une façon de comprendre cette propriété est que cette copie partagée est stockée sur un serveur qui trie les opérations selon ses propres ordres séquentiels.

Par exemple, considérons l'historique donné sur la **figure 4.10**, dans lequel deux processus accèdent à un registre se trouvant au niveau d'un serveur qui gère la copie unique de l'objet ; le processus p0 écrit 1 dans le registre à fenêtre glissante (opération w (1)), le deuxième processus écrit 2. puis p0 et p1 veulent tout les deux lire le registre (opération r). le résultat de la lecture de p0 n'est pas mis à jour (plus précisément elle ne va pas avoir lieu avant que la lecture de p1 ne soit terminé), mais une fois l'écriture terminée les deux processus auront accès à la dernière mise à jour de l'objet partagée. En outre pour écrire, un processus envoi simplement un message au serveur et attend sa réponse. Puisque le serveur traite séquentiellement les messages, les opérations sur l'objet partagé apparaîtront totalement ordonnées.

1. Leslie B. Lamport, né le 7 février 1941 à New York, est un chercheur en informatique américain, spécialiste de l'algorithmique répartie. Il a obtenu le prix Turing 2013. Il est le concepteur du logiciel libre de composition de documents LaTeX

Exemple
Pour simplifier si on a une donnée A au niveau d'un serveur et que l'on a 4 processus qui exécute successivement les opérations suivante au niveau du serveur ou il y'a l'objet partagée A :

P1 : Write A' à la place de A
P2 : Load B
P3 : Load C
P4 : Load D

Un modèle de cohérence sequentielle dit que les chargement de B,C et D n'auront pas lieu avant que l'écriture de A' soit terminée alors que la valeur de A n'est même pas consulter au niveau du reste du programme ce qui induit à une perte performance.

Le but de la cohérence séquentielle (SC) est d'imiter ce comportement. Il est à noter que le serveur n'est pas nécessaire pour la mise en œuvre de la cohérence séquentielle,un objet partagé séquentiellement cohérent ne doit se comporter que de la même manière. Autrement dit dans un système distribuée il faut garantir cette manière de fonctionner au niveau de chaque réplique ce qui provoque une perte de performance significative.

3. **Causal consistency :**

Ce modèle a d'abord été proposé pour les systèmes distribués partagés. Il est plus faible que le modèle de cohérence séquentielle. Ce modèle de cohérence fait la distinction entre les événements qui ont une relation de cause à conséquence et ceux qui n'en ont pas. Dans le cas où l'opération de lecture est le résultat de plusieurs opérations d'écriture sur la réplique, l'opération de lecture ne s'exécute pas tant que l'opération d'écriture n'est pas terminée. Ce modèle garantit qu'un client ne lit pas deux opérations d'écriture liées dans un ordre incorrect et si le client a lu la dernière valeur, il ne lit pas la valeur périmée [46].
Plus précisément le modèle de cohérence causale assouplit l'exigence du modèle séquentiel pour une meilleure concurrence. Contrairement au modèle de cohérence séquentielle, dans le modèle de cohérence causale, tous les processus ne voient que les opérations de référence mémoire dans le même ordre (correct) qui sont potentiellement liées de manière causale. Les opérations de référence de mémoire qui ne sont pas potentiellement liées de manière causale peuvent être vues par différents processus dans des ordres différents [45].

Remarque : *Deux requêtes A et B ont une dépendance causale si au moins l'une des deux conditions suivantes est remplie [20] :*

(a) *A et B sont tous deux exécutés sur un seul thread et l'exécution de l'une précède l'autre dans le temps.*
(b) *B lit une valeur qui a été écrite par A.*

De plus, cette dépendance est transitive, en ce sens que si A et B ont une dépendance causale et que B et C ont une dépendance causale, alors A et C ont aussi une dépendance causale.

Ainsi, dans un scénario d'un système de stockage toujours disponible dans lequel les demandes ont des dépendances causales, un niveau de cohérence plus strict que celui fourni par le modèle causal ne peut être atteint en raison des compromis du théorème CAP.

Si un processus exécute une opération d'écriture A, et qu'un processus (le même ou un autre) qui a observé A effectue alors une opération d'écriture B, alors il est possible que A soit la cause de B ; nous disons que A «cause potentiellement» ou «précède causalement» B. La cohérence causale garantit que si A précède causalement B, alors chaque processus du système observe A avant d'observer B. Inversement, deux opérations d'écriture C et D sont dites concurrentes, ou causalement indépendant, si aucun des deux ne précède causalement l'autre. Dans ce cas, un processus peut observer soit C avant D, soit D avant C.

Exemple 1 :
Voici un exemple de cohérence causale.
Les relations causales sont respectées dans la séquence d'événements suivante :

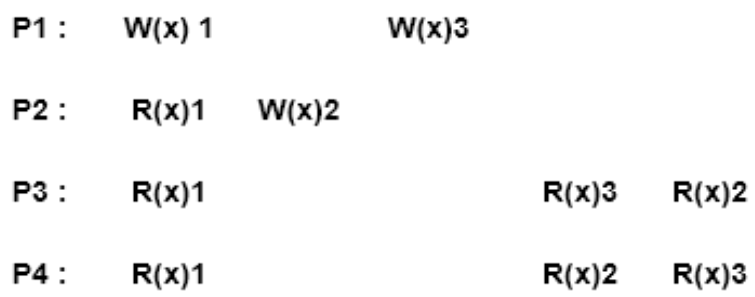


FIGURE 4.11 – Une séquence correcte d'événements qui satisfait la cohérence causal.

Le processus P2 observe, lit, l'écriture antérieure W (x) 1 effectuée par le processus P1. Par conséquent, les deux écritures W (x) 1 et W (x) 2 sont liées de manière causale. Sous cohérence causale, chaque processus observe d'abord W (x) 1, avant d'observer W (x) 2. On remarque que les deux opérations d'écriture W (x) 2 et W (x) 3, sans opérations de lecture intermédiaires, sont simultanées et les processus P3 et P4 les observent (lisent) dans des ordres différents.

Exemple 2 : Une séquence d'événements qui viole la cohérence causal

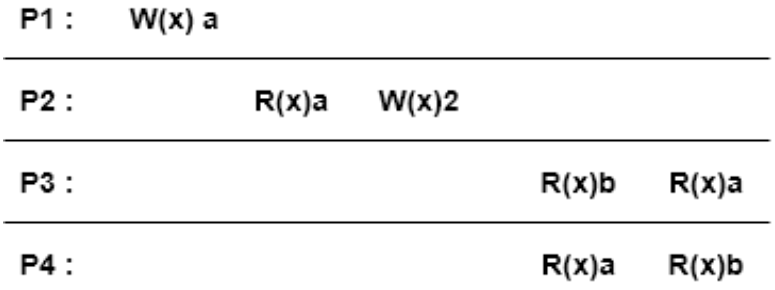


FIGURE 4.12 – Une séquence d'événements qui viole la cohérence causal.

Si un processus exécute une opération d'écriture A, et qu'un processus (le même ou un autre) qui a observé A effectue alors une opération d'écriture B, alors il est possible que A soit la cause de B, donc il faut que chaque processus du système observe A avant d'observer B, or P3 observe B puis observe A et cela ne doit pas se produire dans un système causalement cohérent.

Si on enlève la lecture de P2 de a (R(x)a) alors le système devient causalement cohérent :

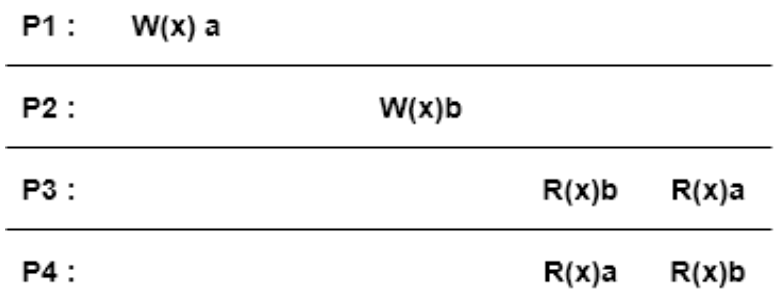


FIGURE 4.13 – Une séquence d'événements cohérente.

4. **PRAM consistency :**

The first-in, first-out (FIFO), également connu sous le nom de pipelined random access memory (PRAM) est un autre type de modèles de cohérence centrée sur les données. Ce modèle a été proposé par Lipton [2] et Sandberg en 1988 pour la mémoire partagée, c'est un modèle qui garantit que la vue de chaque processus est cohérente avec l'ordre dans lequel les écritures ont été effectuées par chaque processus. Chaque processus doit être capable d'expliquer l'histoire par une linéarisation de ses propres connaissances. La cohérence pipelinée est plus faible que la cohérence séquentielle, pour laquelle il est en outre nécessaire que les linéarisations vues par différents processus soient identiques. La cohérence PRAM est locale à chaque processus.
En d'autres termes, il n'y a aucune garantie sur l'ordre dans lequel les écritures sont vues par différents processus, bien que les écritures d'une seule source doivent conserver leur ordre comme si elles étaient dans un pipeline.

Le raisonnement qui a conduit à ce modèle est le suivant : considérons un multiprocesseur où chaque processeur possède une copie locale de la mémoire partagée. Pour que la mémoire soit évolutive, un accès doit être indépendant du temps nécessaire pour accéder aux mémoires des autres processeurs. Ils ont proposé que lors d'une lecture, une PRAM renverrait simplement la valeur stockée dans la copie locale de la mémoire. Lors d'une écriture, il mettrait d'abord à jour la copie locale et diffuserait la nouvelle valeur aux autres processeurs. En supposant un temps constant pour lancer une opération de diffusion, le but de rendre constant le coût d'une lecture ou d'une écriture est ainsi atteint. En termes de contraintes d'ordre, cela équivaut à exiger que tous les processeurs observent les écritures d'un seul processeur dans le même ordre alors qu'ils peuvent être en désaccord sur l'ordre des écritures par différents processeurs. L'historique d'exécution suivant est légal sous PRAM mais pas sous CC (cohérence causale) :

2. Richard J. Lipton, naissance le 6 septembre 1946, est un chercheur anglo-américain en informatique reconnu notamment pour son travail en algorithmique et en cryptographie. Il a reçu le prix Knuth en 2014.

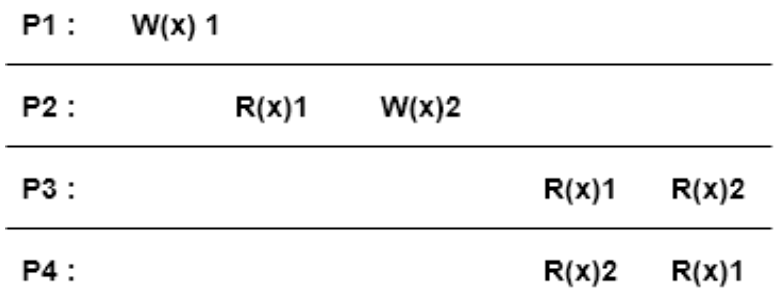


FIGURE 4.14 – Une séquence d'événements qui satisfait la cohérence PRAM.

P3 et P4 observent les écritures de P1 et P2 dans des ordres différents, bien que W (x) 1 et W (x) 2 soient potentiellement liés de manière causale. Ainsi, ce ne serait pas un historique légale pour CC.

L'un des points importants de ce modèle de cohérence est son indépendance du temps dans la hiérarchisation des opérations. D'après ce qui est illustré sur la figure 4.15 suivante, la séquence d'écriture des valeurs b et c par le processus P2 est d'abord l'opération d'écriture de b, puis l'opération sur le serveur Si. Le but est d'observer cette séquence entre l'ensemble des processus. Dans le processus P4, cette séquence est différente avec l'observation des autres processus, alors il y aurait des violations dans le stockage des données en termes de cohérence FIFO [46].

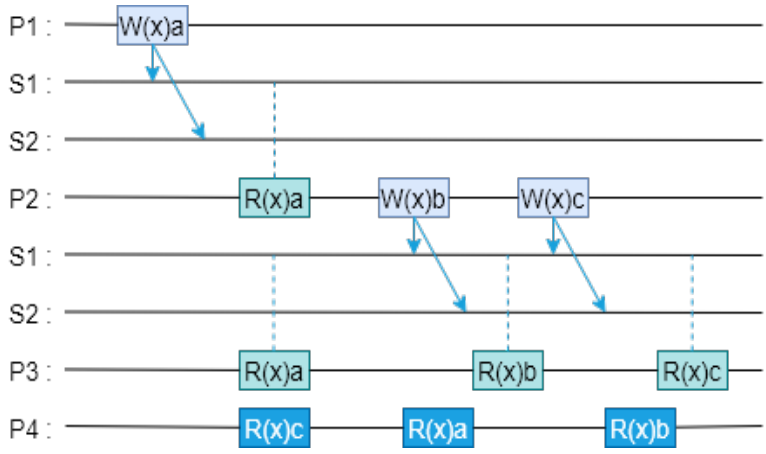


FIGURE 4.15 – Le comportement des processus sur les éléments de données en fonction de la cohérence FIFO.

Dans la **figure 4.15** ci-dessus on remarque que le processus P4 viole cette cohérence PRAM car la lecture de P4 des deux écriture w(x)b ainsi que w(x)c du même processus P2 en été lu différemment que leur ordre d'émission.

5. **Weak consistency :**

Ce modèle de cohérence est une autre variante des modèles data-centric, il a été proposés en 1986 par Micheal Dubois, l'idée de base derrière le modèle de cohérence faible consiste à appliquer la cohérence à un groupe d'opérations plutôt qu'à des opérations individuelles, cela évite la synchronisation globale. Pour ce faire, il définit une variable de synchronisation qui joue le rôle de token (jeton).

Le processus qui possède ce jeton peut effectuer les opérations de lecture ou d'écriture sur la ressource partagée. Dans ce modèle, l'accessibilité à la ressource se fait

par la cohérence séquentielle au sein de la variable de synchronisation. Si le processus ne possède pas ce jeton, aucune des opérations de lecture ou d'écriture n'est privilégiée sur la ressource partagée. Ce modèle est établi dans les conditions suivantes [49] :

- Les accès aux variables de synchronisation sont cohérents de manière séquentielle.
- Aucune opération sur une variable de synchronisation ne peut être effectuée tant que toutes les écritures précédentes n'ont pas été effectuées partout.
- Aucune opération de lecture ou d'écriture sur des éléments de données n'est autorisée tant que toutes les opérations précédentes sur les variables de synchronisation n'ont pas été effectuées.

Il faut notez que le sens de «précédent» est bien défini car il fait référence à l'ordre des programmes. C'est-à-dire qu'un accès précède l'accès B si et seulement si le processeur qui a exécuté l'accès B a précédemment exécuté l'accès A. La synchronisation des accès fonctionne comme des clôtures. Au moment où un accès de synchronisation est effectué, tous les accès antérieurs par ce processeur sont garantis comme ayant été effectués et tous les accès futurs par ce processeur sont garantis de ne pas avoir été effectués. Le modèle de synchronisation correspondant à ces contraintes d'ordre d'accès est relativement simple. Un programme s'exécutant sur un système faiblement cohérent apparaît séquentiellement cohérent si les deux contraintes suivantes sont observées [50] :

- Il n'y a pas d'accès concurrents.
- La synchronisation est visible par le système de mémoire.

Dans ce modèle de cohérence, si plusieurs processus veulent simplement effectuer l'opération de lecture de la mémoire partagée, alors tous peuvent avoir la variable de synchronisation à leur service. Cependant, si un processus veut effectuer l'opération d'écriture sur la mémoire partagée, alors jusqu'à la fin de l'opération d'écriture par le processus, les autres processus ne peuvent pas avoir la variable de synchronisation jusqu'à la fin de l'opération d'écriture.

plus précisément afin d'exécuter tout type d'opération, soit de lecture, soit d'écriture sur la réplique, la possession de la variable de synchronisation est nécessaire. Cette variable n'est accessible qu'après la fin de l'opération d'écriture par le processus ainsi la variable de synchronisation est libre.

Exemple :

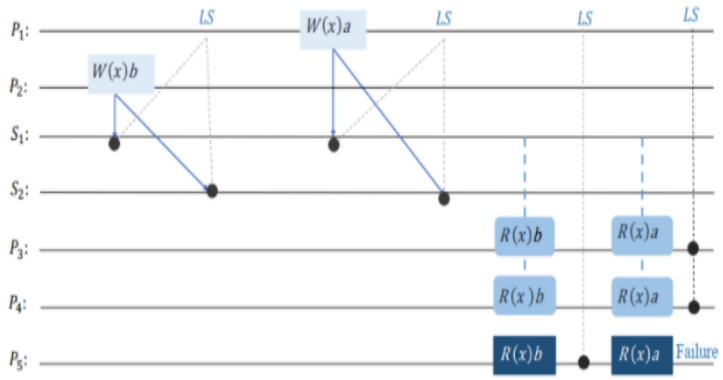


FIGURE 4.16 – Le comportement des processus sur les éléments de données en fonction de la cohérence faible.

La **figure 4.16** nous montre un cas de violation de la cohérence faible, effectivement afin d'effectuer toute opération sur la mémoire, avoir la variable de synchronisation est nécessaire, la violation se produit dans le stockage de données si le processus n'a pas reçu cette dernière. Lorsque le processus demande au serveur Si de lire une valeur sans la variable de synchronisation, la valeur renvoyée peut ne pas être valide.

La **figure 4.16** illustre le comportement de ce modèle de cohérence.Les points rouges sur la figure indiquent la libération de la variable de synchronisation après la fin de l'opération sur la réplique existant dans le serveur Si. Le point sur lequel cette figure se concentre est que la variable de synchronisation pendant l'opération de lecture est partagée entre plusieurs processus simultanément. Cependant, alors qu'un processus utilise la variable de synchronisation pour le processus d'écriture, aucun autre processus ne peut y avoir accès. Si un processus comme P5 libère la variable après l'opération de lecture, alors s'il veut lire a sans recevoir la variable, par la suite la valeur de lecture de a n'est pas valide et il y aurait violation en termes de cohérence faible dans la mémoire partagée.

4.5.2 Modèles de cohérence centrés sur le client (Client-Centric) :

Dans les systèmes distribués, le maintien de la cohérence séquentielle afin de contrôler les opérations simultanées est essentiel. Un magasin de données réparties se caractérise par une relative absence de mises à jour simultanées. Le but est alors de maintenir une vue cohérente des éléments de données pour un processus client individuel qui fonctionne actuellement sur le magasin de données.

Les modèles de cohérence les plus utilisées et qui entrent dans cette catégorie sont :

- La cohérence éventuelle (eventual consistency)
- La cohérence de lecture monotone (monotonic reads consistency)
- La cohérence d'écriture monotone (monotonic writes consistency).
- La cohérence de lecture-écriture (read-your-writes consistency).
- La cohérence écritures après lectures (writes-follow-reads consistency).

Pour mieux comprendre par la suite ces modèles de cohérence :

- considérons un système de dénomination mondial tel que DNS. L'espace de noms DNS est partitionné en domaines, où chaque domaine est attribué à une autorité de dénomination, qui agit en tant que propriétaire de ce domaine. Seule cette autorité est autorisée à mettre à jour sa partie de l'espace de nom. Par conséquent, les conflits résultant de deux opérations qui souhaitent toutes deux effectuer une mise à jour sur les mêmes données (c'est-à-dire des conflits d'écriture / écriture) ne se produisent jamais. La seule situation qui doit être gérée est celle des conflits de lecture-écriture, dans lesquels un processus souhaite mettre à jour un élément de données tandis qu'un autre tente simultanément de lire cet élément. En fin de compte, il est souvent acceptable de propager une mise à jour de manière paresseuse, ce qui signifie qu'un processus de lecture ne verra une mise à jour qu'après un certain temps depuis la mise à jour.
- Le World Wide Web est un autre exemple. Dans pratiquement tous les cas, les pages Web sont mises à jour par une seule autorité, telle qu'un webmaster ou le propriétaire réel de la page. Il n'y a normalement aucun conflit d'écriture / écriture à résoudre. D'autre part, pour améliorer l'efficacité, les navigateurs et les proxys Web sont souvent configurés pour conserver une page récupérée dans un cache local et pour renvoyer cette page à la demande suivante.
- Un aspect important des deux types de caches Web est qu'ils peuvent renvoyer des pages Web obsolètes. En d'autres termes, la page mise en cache qui est renvoyée au client demandeur est une version plus ancienne par rapport à celle disponible sur le serveur Web réel. En fait, de nombreux utilisateurs trouvent cette incohérence acceptable (dans une certaine mesure).

Ces exemples peuvent être considérés comme des cas de bases de données distribuées et répliquées (à grande échelle) qui tolèrent un degré relativement élevé d'incohérence. Ils ont en commun que si aucune mise à jour n'a lieu pendant une longue période, toutes les répliques deviendront progressivement cohérentes. Cette forme de cohérence est appelée cohérence éventuelle.

Les magasins de données qui sont finalement cohérents ont donc la propriété qu'en l'absence de mises à jour, tous les réplicas convergent vers des copies identiques les uns des autres. La cohérence finale exige essentiellement que les mises à jour soient garanties de se propager à tous les réplicas. Les conflits d'écriture-écriture sont souvent relativement faciles à résoudre si l'on suppose que seul un petit groupe de processus peut effectuer des mises à jour. La cohérence finale est donc souvent peu coûteuse à mettre en œuvre.

Les éventuels magasins de données cohérents fonctionnent tant que les clients accèdent toujours à la même réplique. Cependant, des problèmes surviennent lorsque différentes répliques sont accessibles sur une courte période de temps. Ceci est mieux illustré en considérant qu'un utilisateur mobile accède à une base de données distribuée, comme le montre la figure suivante :

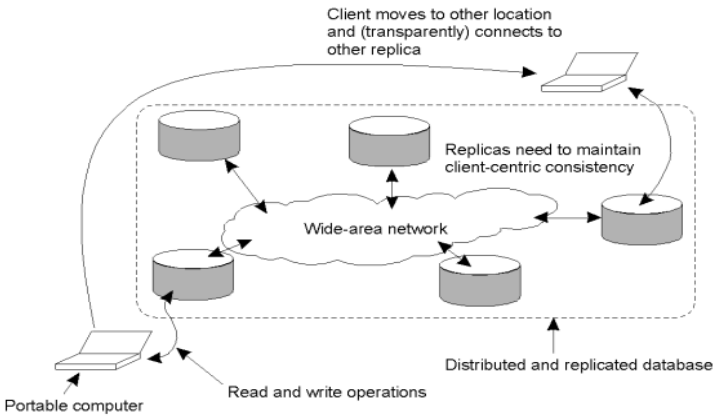


FIGURE 4.17 – Le principe d'un utilisateur mobile accédant à différentes répliques d'une base de données distribuée.

L'utilisateur mobile accède à la base de données en se connectant à l'une des répliques de manière transparente. En d'autres termes, l'application exécutée sur l'ordinateur portable de l'utilisateur ne sait pas sur quelle réplique elle fonctionne réellement. Supposons que l'utilisateur effectue plusieurs opérations de mise à jour, puis se déconnecte à nouveau. Plus tard, il accède à nouveau à la base de données, éventuellement après s'être déplacé vers un autre emplacement ou en utilisant un autre périphérique d'accès. À ce stade, l'utilisateur peut être connecté à une réplique différente de celle d'avant, comme le montre la figure.

Cependant, si les mises à jour effectuées précédemment n'ont pas encore été propagées, l'utilisateur remarquera un comportement incohérent. En particulier, il s'attendrait à voir toutes les modifications apportées précédemment, mais à la place, il semble que rien du tout ne s'est produit.

Cet exemple est typique des magasins de données cohérents à terme et est dû au fait que les utilisateurs peuvent parfois opérer sur des réplicas différents. Le problème peut être atténué en introduisant une cohérence centrée sur le client. En substance, la cohérence centrée sur le client fournit des garanties pour un seul client concernant la cohérence des accès à un magasin de données par ce client. Aucune garantie n'est donnée concernant les accès simultanés par différents clients.

1. **Eventual consistency (Cohérence éventuelle) :**

   La cohérence éventuelle renforce le modèle de cohérence faible (Weak Consistency) . Les réplicas ont tendance à converger vers le même état de données. Pendant l'exécution de ce processus de convergence, il est possible pour les opérations de lecture de récupérer une version plus ancienne au lieu de la dernière. La fenêtre d'incohérence dépendra des délais de communication entre les répliques et ses sources, la charge sur le système et le nombre de répliques impliquées [51].

   Ce modèle est à mi-chemin entre un modèle de cohérence forte et un modèle de cohérence faible. La cohérence éventuelle est une fonctionnalité populaire offerte par de nombreuses bases de données NoSQL. Cassandra est l'un d'entre eux, et il peut offrir une disponibilité et une partition du réseau à un niveau tel qu'il ne compromet pas la convivialité des sites Web les plus consultés au monde qui utilisent Cassandra. L'un d'eux est Facebook, la société qui a initialement développé Cassandra [51].

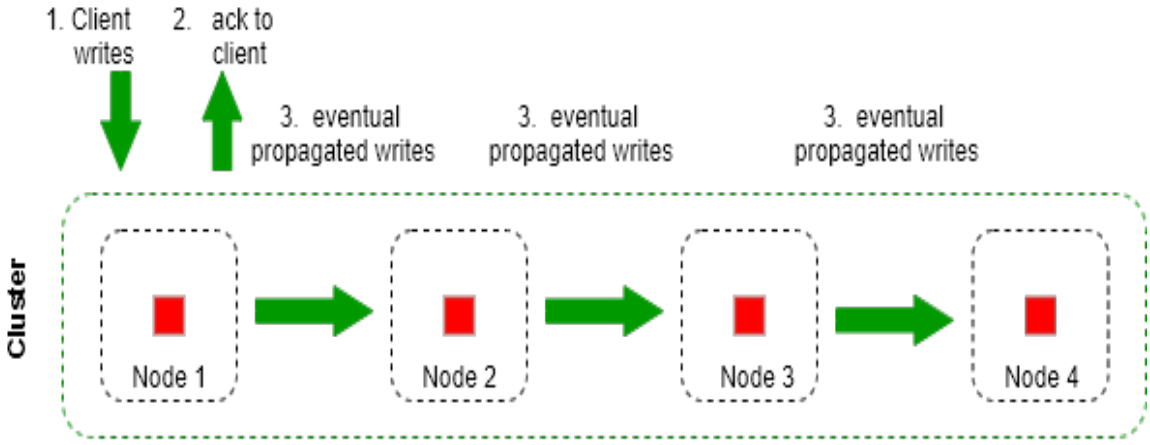


FIGURE 4.18 – Système éventuellement cohérent.

à travers la figure 4.18 nous pouvons observer que lors d'une écriture(mise à jour) au niveau d'un système distribuée éventuellement cohérent au niveau d'un noeud (serveur), cette écriture est synchroniser juste après ou bien après un temps donnée vers toutes les autres répliquent éventuellement.

Pour résumer un tel système :

- *Le système retourne finalement la dernière écriture : affaiblit les conditions de cohérence en ajoutant le mot «éventuellement».*
- *Risque de perte de données en cas de défaillance du nœud : ajoute la condition «à condition qu'il n'y ait pas de panne permanente».*

Pourquoi une cohérence éventuelle et pas une cohérence strict ?

Une cohérence stricte n'est pas toujours requise et une cohérence éventuelle peut suffire dans certains cas d'utilisation. Par exemple, dans un panier, disons qu'un article est ajouté et que le centre de données a échoué. Ce n'est pas un désastre pour le client d'ajouter à nouveau cet article. Dans ce cas, une cohérence éventuelle

serait suffisante.

Cependant, vous ne voudriez pas que cela se produise sur votre compte bancaire avec un dépôt que vous venez d'effectuer. Faire disparaître votre argent parce qu'un nœud a échoué est inacceptable. Une stricte cohérence est requise dans les transactions financières.

Pourquoi une cohérence éventuelle par rapport aux autres modèles :

- Plus d'opportunités d'accès concurrentiel qu'une cohérence stricte, séquentielle ou causale
- La cohérence séquentielle nécessite des connexions hautement disponibles (Beaucoup de bavardages entre clients / serveurs).
- La cohérence séquentielle n'est pas adaptée à certains scénarios

    1- Clients déconnectés (par exemple, votre ordinateur portable se déconnecte, mais vous souhaitez toujours modifier votre document partagé).

    2- Partitionnement du réseau entre les centres de données.

    3- Les applications peuvent préférer une incohérence potentielle à une perte de disponibilité.

2. **Monotonic reads consistency (Cohérence de lecture monotone) :**

Lorsqu'un réplica nécessite la cohérence des données, les dernières modifications apportées à l'élément de données sont envoyées à la copie exigeante par le réplica. Le modèle de cohérence garantit que si un processus observe une valeur dans un certain temps, il ne verra jamais sa valeur précédente.

Par exemple, un utilisateur peut lire le courrier entrant tout en se déplaçant. Chaque fois que l'utilisateur se connecte à un serveur de messagerie différent, ce serveur récupère toutes les mises à jour du serveur que l'utilisateur a précédemment visité. Monotonic Reads garantit que l'utilisateur voit toutes les mises à jour, quel que soit le serveur à partir duquel la lecture automatique a lieu.

Ces opérations de lecture séquentielle reflètent un grand nombre d'opérations d'écriture . Si l'opération de lecture séquentielle est effectuée par un client, le contenu mis à jour sera renvoyé de manière significative.
Dans le modèle de cohérence de lecture monotone, si le client a lu une valeur de la mémoire, alors il ne verrait aucune autre valeur qui a été écrite avant cela. En d'autres termes, en effectuant l'opération de lecture par le client, une valeur valide est lue à partir de la réplique sur laquelle toutes les modifications antérieures ont été affectées. Il en résulte que l'opération de lecture est valide lorsque le client en ayant accès à la réplique existant dans le serveur a exécuté toutes les modifications sur la réplique d'objet. Sinon, il fera face à des violations dans la mémoire partagée en utilisant la cohérence de lecture monotone. Pour mieux comprendre ce modèle, son comportement est illustré sur la **figure 4.19** suivante.

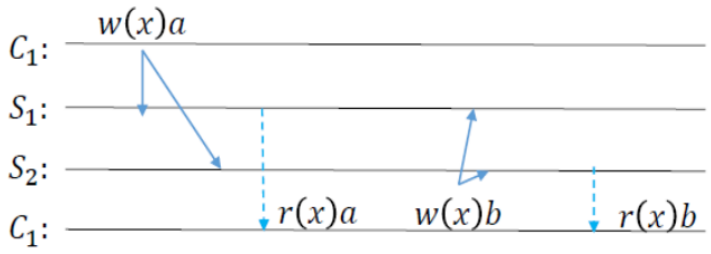


FIGURE 4.19 – Le comportement des processus sur les éléments de données basé sur la cohérence de lecture monotone.

Le comportement du modèle de cohérence de lecture monotone sur la mémoire partagée est introduit comme le client Ci, peut lire la valeur valide b, quand en ayant accès à chacune des répliques dans les serveurs S1 et S2 toutes les opérations antérieures ont été exécutées sur le réplique.

3. **Monotonic writes consistency (Cohérence d'écriture monotone) :**

L'opération d'écriture par le processus sur l'élément de données est acceptable lorsque l'opération d'écriture précédente est exécutée par le même processus sur la réplique. Comme la bibliothèque du logiciel qui doit remplacer une ou plusieurs fonctions pour être mise à jour. Le point important est que dans la cohérence d'écriture monotone, les opérations d'écriture sont effectuées avec la même séquence qui a été lancée. L'opération d'écriture par un client sur une réplique est assurée lorsque l'ensemble des opérations d'écriture précédentes sont enregistrées par le même client sur la même réplique. Ce modèle de cohérence propage l'opération d'écriture par rapport aux priorités entre les opérations. En fin de compte, il faut que chaque opération d'écriture soit visible dans l'ordre de présentation.

Le comportement de la cohérence d'écriture monotone est exprimé comme suit : l'opération d'écriture 1 s'exécute correctement lorsque l'opération d'écriture de 2 est effectuée avant cela. En d'autres termes, lorsque le client écrit la valeur valide 1 sur la réplique que l'opération 2 sur la même réplique dans le serveur est effectuée avant cela .

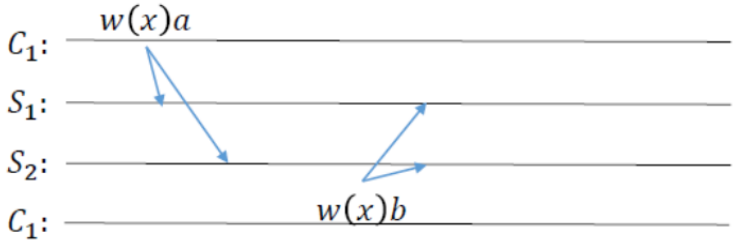


FIGURE 4.20 – Le comportement des processus sur les éléments de données en fonction de la cohérence d'écriture monotone.

Ce modèle de cohérence qui est introduit sur la **figure 4.20** exprime que le client Ci effectue l'opération d'écriture de la valeur valide b sur l'élément de données x si toutes les opérations d'écriture précédentes sur cet élément sont exécutées sur le serveur Si.

4. **Read-your-writes consistency (Cohérence de lecture-écriture) :**

Dans ce modèle, l'opération d'écriture est effectuée par le client sur l'élément de données x qui est toujours visible lors des opérations de lecture successives pour le même client sur l'élément de données x. Cela signifie que les opérations d'écriture sont terminées avant que la prochaine opération de lecture ne soit effectuée par le même client. Ce modèle de cohérence pourrait être spécifié comme suit :

- Le temps d'accès aux données est prolongé (comme le changement de mot de passe).
- Similaire à la cohérence en lecture seule, mais avec la différence que la cohérence de la dernière opération de lecture est déterminée par la dernière opération d'écriture du client.

Dans ce modèle, la nouvelle valeur écrite est lue par le client au lieu de la précédente valeur. Ce modèle nécessite que chaque opération d'écriture soit visible dans l'ordre d'exécution. Toute séquence de transactions telle que la lecture non validée est basée sur la priorité spécifiée par l'observateur global, et l'opération de lecture reflète ses opérations d'écriture précédentes. La mise à jour de la page Web peut être désignée comme l'une des applications de ce modèle, elle garantie que le navigateur Web affiche la version la plus récente au lieu de sa copie en cache. Afin d'avoir une meilleure compréhension de ce modèle, la **figure 4.21** illustre le fonctionnement de ce modèle.

Le modèle de cohérence de lecture-écriture est exprimé comme suit : le client Ci peut lire une valeur valide b lorsque l'opération d'écriture a est préalablement exécutée sur le réplica existant dans le serveur Sj. Où b est la valeur écrite valide par le client Ci. Afin d'avoir une meilleure compréhension de ce modèle, la **figure 4.21** illustre le fonctionnement de ce modèle.

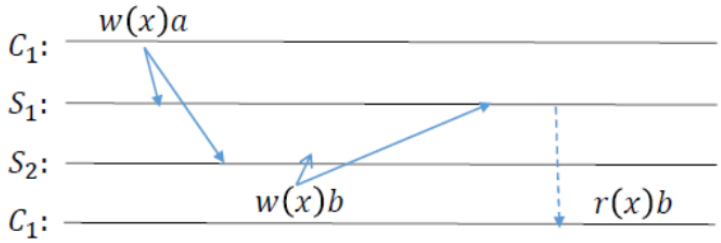


FIGURE 4.21 – Le comportement des processus sur les éléments de données en fonction de la cohérence de lecture de votre écriture.

Le comportement de ce modèle selon la **figure 4.21** est tel que le client peut lire la nouvelle valeur b comme résultat valide, avant l'exécution de l'opération d'écriture b par le client C1 sur la donnée du serveur Si, tous écrivent les opérations sur cet élément de données particulier sont exécutées dans la version existante du serveur. Sinon, il rencontrera une incohérence dans le référentiel de données partagé.

5. **Writes-follow-reads consistency (Cohérence écritures aprés lectures) :**

Dans la cohérence Writes-follow-reads, également connue sous le nom de session causale, les propagations de mise à jour sont basées sur la dernière opération de lecture. Le client peut écrire sur l'élément de données x, lorsque la dernière valeur de l'élément de données x est lue par le même client. Par exemple, sur Twitter, un client peut publier un retweet sur un message qu'il / elle a déjà vu. Avec l'opération de lecture, ce modèle vérifie l'opération d'écriture de dictée précédente du client et assure la nouvelle opération d'écriture. Ce modèle est lié à la relation «happening-before» qui est proposée par Lamport.

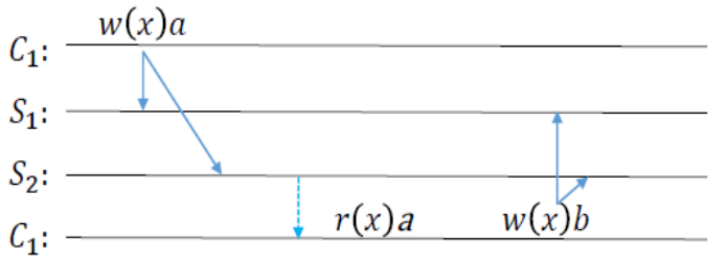


FIGURE 4.22 – Le comportement des processus sur les éléments de données en fonction de l'écriture suit la cohérence de lecture.

Comme on peut le voir sur la **figure 4.22**, si le client Ci lit la valeur a de la donnée x sur le serveur Sj, alors cette valeur est écrite sur le même serveur puis écrit la nouvelle valeur b sur la donnée x sur serveur Sj et par conséquent la lecture de suivi d'écriture fonctionne correctement.

Partie C

4.6 Méthodes de réplication de la cohérence :

Dans cette section, nous présentons un aperçu des méthodes de pointe pour la cohérence des répliques dans les environnements de nuages. Cet aperçu comprend les méthodes que nous avons considérées comme étant parmi les plus représentatives de la littérature. En nous basant sur les similitudes de leurs idées de base, nous avons classé ces méthodes en trois catégories distinctes :

1. Les méthodes de cohérence fixes.
2. Les méthodes de cohérence configurables.
3. Les méthodes de contrôle de la cohérence.

Dans ce qui suit, nous décrivons d'abord les caractéristiques génériques de chaque catégorie, puis nous présentons un aperçu de ses spécifique.

4.6.1 Méthodes de cohérence fixes :

Dans cette catégorie nous allons voir les méthodes qui offrent des garanties de cohérence redéfinies et fixes dans les systèmes de stockage dans le nuage. Les méthodes représentatives dans cette catégorie sont les deux types suivants :

(a) Cohérence basée sur le séquençage des événements
(b) Cohérence stricte basée sur l'horloge.

(a) Cohérence basée sur le séquençage des événements :

Les méthodes de cohérence fondées sur le séquençage des événements visent à masquer la complexité de la réplication, car la sérialisation des transactions est coûteuse et souvent inutile dans les applications web. Ainsi, elles offrent une solution simple, mais dans de nombreuses situations, une garantie de cohérence efficace. Le système le plus représentatif qui met en œuvre ce type de méthode est :

Les développeurs de PNUTS ont observé que les applications web manipulent généralement un enregistrement à la fois, alors que les différents Les dossiers peuvent être situés dans différentes localités géographiques. Par conséquent, une méthode de cohérence basée sur le séquençage des événements établit que toutes les répliques d'un enregistrement donné reçoivent toutes les mises à jour appliquées à cet enregistrement dans le même ordre. C'est mise en ouvre en désignant l'une des répliques en tant que maître de chaque enregistrement, afin que ce maître reçoive tous les écrits envoyés à ce dossier par les autres répliques. Si une La majorité des écrits de ce dossier sont envoyés à une réplique, cette réplique devient le maître de ce dossier [52][53].

(b) Cohérence stricte basée sur l'horloge :

Les méthodes de cohérence stricte basées sur l'horloge sont caractérisées par l'utilisation de mécanismes basés sur l'horloge pour contrôler les horodatages afin d'assurer une cohérence stricte .Elles offrent la garantie que les objets arbitraires dans le magasin de données sont accessibles de manière atomique et isolés des accès concurrents. L'approche qui sous-tend cette méthode de cohérence est basée sur la capacité d'un système à fournir un journal d'horodatage pour suivre l'ordre dans lequel les opérations se produisent. Selon Bravo et al. ce type de technique est mis en œuvre par les systèmes de stockage de données eux-mêmes et peut avoir un impact sur les garanties de cohérence qu'ils fournissent.

Spanner et Clock-SI sont des systèmes représentatifs qui mettre en œuvre ce type de méthode de cohérence. Spanner est un système NewSQL évolutif, distribué à l'échelle mondiale service de base de données conçu, construit et déployai par Google. Spanner combine et étend les idées de deux recherches les communautés : la communauté des bases de données et les systèmes communauté. De ce dernier, l'évolutivité et la tolérance aux pannes sont les caractéristiques les plus représentatives fournies par Clé. Comme la réplication est utilisée pour la disponibilité mondiale et la localité géographique, les applications peuvent utiliser Spanner pour une grande disponibilité, même en cas de catastrophes naturelles de grande ampleur. Spanner permet aux données de différentes applications à répartir sur différents ensembles de serveurs dans le même centre de données. Pour cette raison, la tolérance de partition est une exigence importante dans Spanner. En outre, Spanner fournit des contraintes pour le contrôle de la lecture et de l'écriture latence. Spanner attribue également des horodatages d'engagement d'importance globale qui reflètent l'ordre de sérialisation des qui peuvent être distribués. De plus, Spanner fait en sorte que si une opération T2 commence après la commit d'une transaction T1, puis l'horodatage de commit de T2 doit être supérieure à l'horodatage de T1.
D'autre part, Clock-SI fournit un protocole entièrement distribué pour les magasins de données cloisonnés qui soutient la disponibilité et l'évolutivité, apportant des avantages en termes de performances. Il évite également un "point de défaillance unique" et, par conséquent, un goulot d'étranglement potentiel des performances, améliorant ainsi la latence et le débit des transactions. Clock-SI met en œouvre la réplication d'isolation dite "snapshot", qui est un critère de cohérence pour le stockage de données partitionnées. Dans cette stratégie, les opérations en lecture seule sont lues à partir d'un instantané cohérent et d'autres opérations effectuent un commit si aucun objet écrit par ces transactions n'a été écrit simultanément. L'horloge physique locale est utilisée par chaque transaction pour identifier son horodatage de lecture. Un autre exemple de système qui met en œuvre l'instantané La réplication de l'isolement est Vela , qui est un système permettant de faire fonctionner des bases de données relationnelles standard sur le cloud. Vela fournit un maître primaire et deux répliques secondaires, qui sont répliquées de manière synchrone, offrant ainsi une tolérance aux pannes. De plus, Vela s'appuie sur la virtualisation matérielle pour améliorer les performances en réduisant la complexité avec un coût minimal en ressources. Au lieu d'utiliser la réplication des données pour la durabilité, Vela utilise cette technique pour améliorer la latence, découplant la mise à jour et les charges de travail en lecture seule. De plus, Vela offre une élasticité, en surveillant le pourcentage de temps d'inactivité du processeur, et une involutivité, en ajoutant de nouvelles répliques en fonction de sa charge de travail [54][55][56].

4.6.2 Méthodes de cohérence configurables :

Les méthodes incluses dans cette catégorie mettent en œuvre des mécanismes qui fournissent des niveaux de cohérence configurables, tels que des garanties de cohérence auto-adaptatives ou flexibles, qui permettent de sélectionner ou de spécifier le niveau de cohérence souhaité à un moment donné. Ces méthodes sont de deux types, la cohérence automatisée et auto-adaptative et la cohérence flexible, et sont décrites ci-dessous :

(a) **Cohérence automatisée et auto-adaptative :** Les méthodes de cohérence automatisées et auto-adaptatives visent à appliquer dynamiquement de multiples degrés de cohérence sur des objets de données distincts. Trois approches principales ont été adoptées pour les mettre en œuvre

- **L'estimation de Stale :**
  Cette approche est basée sur une estimation du taux d'opérations de lecture qui reviennent des exemples d'objets de données qui ont déjà été mis à jour à des valeurs plus récentes - le "stale" se lit comme suit . Une fois que le modèle d'estimation est calculé, il est possible d'identifier la clé paramètre qui affecte les lectures périmées et ensuite à l'échelle augmentation/réduction du nombre de répliques. Cette approche est principalement mis en œuvre par ***Harmony*** , qui est cloud storage system qui identifie automatiquement les paramètres cluse affectant les lectures périmées, telles que les états du système et les exigences de l'application. Harmony ajoute une latence minimale tout en réduisant les lectures de données périmées de prés de 80 %. Son objectif est de régler progressivement et dynamiquement le niveau de cohérence au moment de l'exécution en fonction des exigences de cohérence des applications, afin de fournir des compromis adéquats entre la cohérence et les performances et la disponibilité. Un site Le modèle d'estimation intelligent des lectures périmées est l'aspect clé d'Harmony. Son mécanisme de mise à l'échelle élastique du nombre de répliques maintient une fraction minimale tolérable de lectures périmées, ce qui permet d'atteindre le niveau de cohérence requis tout en obtenant de bonnes performances [57][58].

- **La divergence limite l'application :**
  Cette approche permet d'évaluer et de faire respecter les limites de divergence sur les objets de données (tableau/rangée/colonne), fournissant ainsi des niveaux de cohérence qui peuvent être automatiquement ajustés sur la base d'informations statistiques . L'évaluation a lieu sur un vecteur de divergence chaque fois qu'une demande de mise à jour est reçue, bien qu'il soit nécessaire d'identifier les objets de données concernés. Si une limite est dépassée, toutes les mises à jour depuis la dernière réplication sont placées dans une file d'attente de type FIFO pour être propagées et exécutées sur les autres réplications. Le système le plus représentatif qui met en ouvre cette approche est le VFC3 (Versatile Framework for Consistency in Cloud Computing) , qui adopte un modèle de cohérence pour les données répliquées entre les centres de données. Le modèle VFC3 prend en compte les différentes sémantiques des données et ajuste automatiquement les niveaux de cohérence en fonction des informations statistiques. Son principal objectif est d'offrir un contrôle sur la cohérence afin de fournir une haute disponibilité sans compromettre les performances. En outre, le VFC3 cible les nuages des magasins de données tabulaires, offrant une rationalisation des ressources et l'amélioration de la qualité de service (QoS), ce qui réduire la latence [59][60].

- **Allocation dynamique :**
  Cette approche sélectionne de manière dynamique vers quel serveur (ou même un ensemble de serveurs) chaque lecture d'une donnée doit être dirigée, afin que le meilleur service soit fourni compte tenu de la configuration actuelle et des conditions du système. Cette approche est donc adaptable à des configurations distinctes de réplications et d'utilisateurs, ainsi qu'à des conditions changeantes, telles que des variations de la performance du réseau ou de la charge du serveur. Un autre aspect important de cette approche est le fait qu'elle permet aux développeurs d'applications de fournir un accord de niveau de service (SLA) qui spécifie les souhaits de cohérence/retard des applications de manière déclarative. Pileus est un système de stockage à valeur clé qui met en ouvre cette approche en fournissant une diversité d'options de garantie de cohérence pour les environnements de données répliquées et distribuées au niveau mondial. En fait, Pileus permet plusieurs systèmes ou clients d'un système pour obtenir différents degrés de cohérence, même s'ils partagent les mêmes données. En outre, Pileus favorise la disponibilité et les performances en limitant l'ensemble des serveurs appropriés, de sorte que les lectures fortes doivent être dirigées vers le site principal et que toute réplique peut répondre aux lectures éventuelles. Une grande table peut être divisée en une ou plusieurs tables plus petites afin d'atteindre une certaine évolutivité. Dans Pileus, les utilisateurs effectuent des opérations pour accéder aux données qui est partitionné et répliqué entre des serveurs distincts, Elle doit donc soutenir la tolérance et la fiabilité de la partition [61].

(**b**) **Cohérence flexible :**
Les méthodes de cohérence flexibles couvrent des approches distinctes qui s'adaptent de manière flexible à des modèles de cohérence prédéfinis. Il existe quatre approches principales pour mettre en œuvre de telles méthodes.

- **Basé sur des invariants :**
  Cette approche renforce les éventuelles la cohérence, ce qui permet aux demandes de préciser les règles de cohérence, ou invariants, qui doivent être maintenus par le système. Une fois que ces invariants sont définis, il est possible d'identifier les opérations qui sont potentiellement dangereux en cas d'exécution simultanée, ce qui permet choisir entre la prévention des violations et la réparation invasive technique. Indigo est un système de middleware qui met en œuvre cette approche. Il prend en charge une méthode de cohérence alternative construite sur un magasin de données à valeur clé géo-répliqué et partitionné. En outre, Indigo garantit des invariants d'application forts, tout en assurant une faible latence à un système finalement cohérent. Indigo s'appuie sur la tolérance aux pannes du système de stockage sous-jacent, ce qui la défaillance d'une machine à l'intérieur d'un centre de données ne conduit pas à toute perte de données [62].

- **Linéarisable/éventuel :** Cette approche soutient des mises à jour avec un choix de linéarisation et de cohérence éventuelle. GEO est un logiciel libre de géodistribution acteur2 système qui met en œuvre cette approche. Il soutient les protocoles de cohérence, qu'ils soient répliqués ou uniques. La réplication peut fournir des lectures rapides et toujours disponibles mises à jour. GEO améliore également les performances grâce à la mise en cache les états acteurs dans un ou plusieurs centres de données. En outre, le Un système d'acteurs géographiquement repartis peut réduire les temps de latence d'accès en exploitant la localité, puisque la politique de mise en cache pour chaque L'acteur peut être déclaré comme une seule ou plusieurs substances. La mise en cache de plusieurs instances peut réduire la latence d'accès pour les acteurs sans localité. Dans GEO, un acteur peut être déclarés comme volatils ou persistants. Dans le premier cas, la la dernière version réside dans la mémoire et peut être perdue lorsque les serveurs tombent en panne, alors que dans le second cas, la dernière version réside dans la couche de stockage. Si un utilisateur demande une linéarisation, GEO garantit une linéarisation réelle en temps, entre l'appel et le retour [63].

- **Basé sur des exigences :**
  Cette approche répond aux exigences de cohérence des demandes. La discussion qui suit adopte la notion de service (demandé par une application) comme une généralisation des opérations de lecture et d'écriture utilisées jusqu'à présent. Le compromis entre les exigences de cohérence et d'évolutivité est traité en introduisant la notion de régions de cohérence et de politiques de cohérence orientées vers la prestation de services. Une région de cohérence est définie comme une unité logique qui représente les exigences de cohérence et d'extensibilité au niveau de l'état de l'application. Ce concept est utilisé pour définir les limites de cohérence qui séparent chaque groupe de services devant être commandés. Ainsi, les services qui doivent être maintenus en cohérence doivent être associés à une certaine région. La définition de la région à laquelle un service appartient et des services qui peuvent être fournis simultanément est fixée par les administrateurs du système. La réplication évolutive orientée service (SSOR) est un intergiciel qui met en œuvre cette approche. Le SSOR présente un protocole électoral basé sur la région (REP) qui fournit un mécanisme permettant d'équilibrer la charge de travail entre les séquenceurs, améliorant ainsi efficacement l'élasticité dans le nuage. La réplication est utilisée par le SSOR pour fournir une tolérance aux pannes de bout en bout entre les clients finaux et les services dans le nuage. Afin de garantir la fiabilité, le SSOR met en ouvre des solutions qui tolèrent les plantages des séquenceurs et des non séquenceurs, et introduit le concept de synchronisation de la distribution des régions pour gérer les plantages simultanés des nœuds. Une meilleure performance est également obtenue en réduisant la charge sur le séquenceur, en étendant le protocole MSP (Multi-fixed Sequencers Protocol) La SSOR couvre trois types distincts de régions de cohérence : (1) Région de conflit (CR), qui est une région composée de services ayant des exigences contradictoires en matière de cohérence, quelle que soit la session ; (2) Région de conflit de session (SCR), qui est une région comprenant les services d'une session particulière ayant des exigences contradictoires en matière de cohérence ; et (3) Région non conflictuelle (NCR), qui est une région n'imposant aucune contrainte ou exigence en matière de cohérence [64].

- **Adaptable :**
  Cette approche permet de gérer les charges de travail imprévisibles en permettant au système d'être réglé en fonction des capacités de manière élastique et flexible. Grâce à cette caractéristique, elle permet aux applications d'effectuer des lectures éventuellement ou fortement cohérentes selon les besoins. Amazon DynamoDB3 est un service de base de données NoSQL très fiable et rentable qui met en œuvre cette approche. DynamoDB fournit une haute disponibilité et un haut débit avec une latence très faible. Il a été conçu pour être évolutif et pour atteindre des performances élevée même à grande échelle. Il a été construit sur la base de l'expérience acquise avec son prédécesseur Dynamo. DynamoDB adopte la cohérence éventuelle comme modèle par défaut, ce qui ne garantit pas qu'une lecture éventuellement cohérente reflétera toujours le résultat d'une écriture récemment terminée. D'autre part, en adoptant un modèle de cohérence plus fort, il renvoie un résultat qui reflète toutes les écritures qui ont a reçu une réponse positive avant cette lecture [65].

4.6.3 Méthodes de contrôle de la cohérence :

Par ailleurs, au lieu de traiter directement les problèmes de cohérence des données, certaines méthodes se concentrent sur la mise en place de mécanismes permettant aux propriétaires de données de détecter les violations de cohérence dans le stockage dans le nuage. Cela signifie que les clients peuvent vérifier leurs propres données et prendre des décisions en fonction de la manière dont le Cloud Service Provider (CSP ou fournisseur de services dans le cloud en français) stocke et gère leurs répliques selon le niveau de cohérence qui a été convenu dans le contrat de niveau de service. Les méthodes de cette catégorie sont de deux types, Cohérence vérification et l'audit de cohérence, et sont décrits suivant.

(**a**) **Vérification de cohérence :** Les méthodes de vérification de la cohérence reposent sur deux approches, à savoir une approche basée sur les protocoles et une approche basée sur les contrats.

- **Basé sur un protocole :**
  Cette approche est basée sur un protocole qui permet à un groupe de clients se faisant mutuellement confiance détecter les violations de cohérence sur un stockage dans le cloud. Il est adoptée par VICOS (Verification of Integrity and Consistency for Cloud Object Storage) . VICOS soutient le concept de linéarisation de la fourchette, qui saisit la notion de cohérence la plus forte possible dans les modèles multi-clients. La méthode peut garantir cette notion en enregistrant l'évolution causale des points de vue de l'utilisateur dans leur l'interaction avec le serveur. Lorsque ce dernier crée seulement une seule divergence entre les points de vue de deux clients, il est garanti que ces clients n'observeront jamais d'autres opérations par la suite. En d'autres termes, si ces utilisateurs, plus tard communiquer et le serveur leur ment, la violation sera immédiatement découverte. Ainsi, les utilisateurs peuvent vérifier un grand nombre de transactions passées en effectuant une seule vérification [66].

- **Basé sur un contrat :**
  Cette approche fournit un schéma de vérification qui permet aux propriétaires de données de s'assurer que le CSP respecte l'accord de niveau de service (SLA) pour le stockage des données dans des répliques multiples. Elle est mise en ?ouvre par DMR-PDP (Dynamic Multi-Replica Provable Data Possession) [51]. Le contexte dans lequel s'inscrit ce système est le suivant : chaque fois que les propriétaires de données demandent au CSP de répliquer des données sur différents serveurs, ils doivent payer pour cela. Par conséquent, les propriétaires de données doivent être fermement persuadés que le CSP stocke toutes les copies de données convenues dans le contrat de niveau de service, et que toutes les copies stockées à distance exécutent correctement les mises à jour demandées par les utilisateurs. Cette approche permet de résoudre ces problèmes en empêchant le CSP de tricher sur le stockage des données, par exemple en conservant moins de copies que ce qui a été payé. Ce système est basé sur une technique appelée "Possession de données probables" , qui est utilisée pour vérifier et valider l'intégrité et la cohérence des données stockées sur les serveurs distants [67].

(**b**) **Audit de cohérence :**
Ce type de méthode est basé sur une architecture qui consiste en un vaste nuage de données géré par un CSP et plusieurs petits nuages d'audit composés d'un groupe d'utilisateurs qui coopèrent à un travail spécifique (par exemple, la révision d'un document ou la rédaction d'un programme). Le niveau de cohérence requis qui devrait être fourni par le nuage de données est stipulé par un SLA impliquant le nuage d'audit et la nuage de données. Une fois le SLA défini, le nuage de données d'audit peut vérifier si le nuage de données le viole, ce qui permet de quantifier, en termes monétaires ou autres, la gravité de la violation. La mise en œuvre de la cohérence en tant que service (CaaS) . Elle s'appuie sur une structure d'audit à deux niveaux, à savoir : l'audit local et l'audit global .L'audit local permet à chaque utilisateur d'effectuer indépendamment des opérations de traçage locales, axées sur la lecture monotone et des cohérences de lecture-écriture. L'audit global, d'autre part, exige qu'un auditeur soit périodiquement élu à partir du cloud d'audit pour effectuer des les opérations de traçage, en se concentrant sur la cohérence causale. Ce est soutenue par la construction d'un graphique dirigé des opérations, appelé le graphe de priorité. Si il est acyclique, le niveau de cohérence requis est préservée [20][68].

Partie D

Synthèse

Dans ce chapitre, nous avons examiné plusieurs méthodes et modèles proposées dans la littérature pour assurer la cohérence des répliques dans les systèmes de stockage de données en cloud distribués.

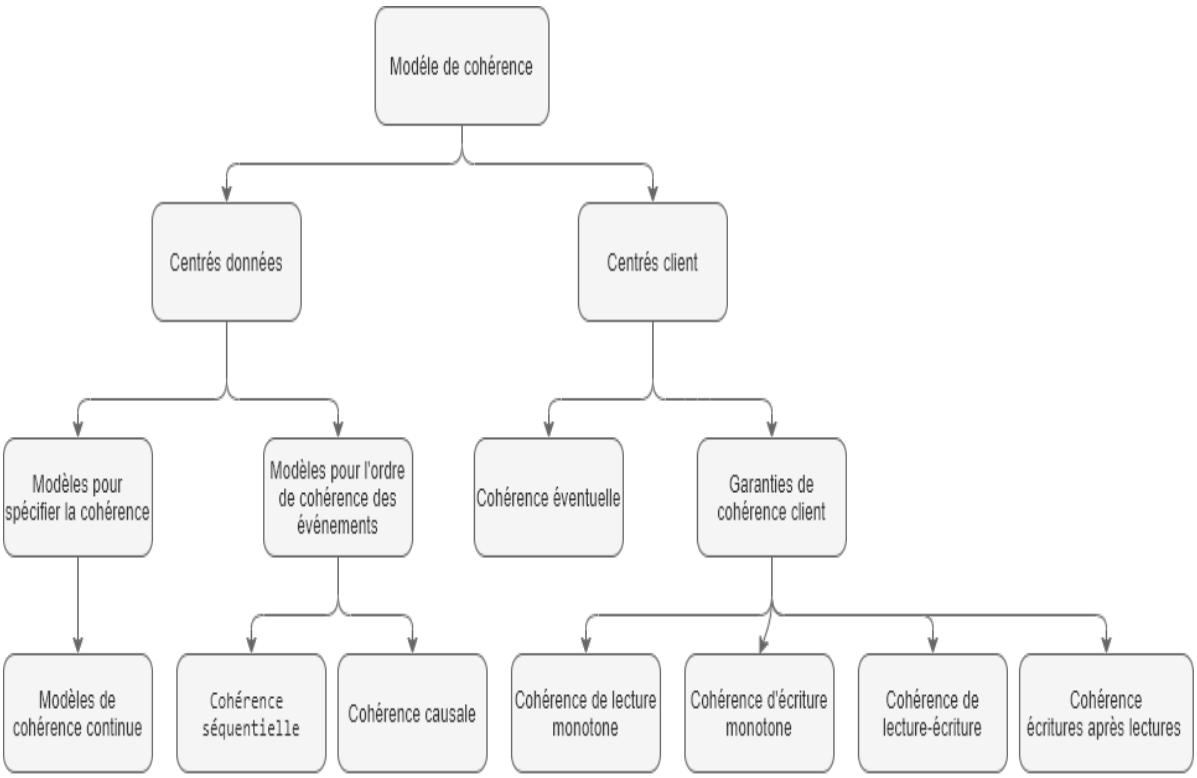


FIGURE 4.23 – Classification des différents modèles de cohérence.

Assurer la cohérence des bases de données répliquées est un sujet de recherche important qui présente de nombreux défis en ce sens que ces systèmes doivent fournir un état cohérent de toutes les réplications, malgré l'existence de transactions simultanées.En d'autres termes, ces systèmes doivent fournir une série de stratégies pour la mise à jour et la propagation des données afin de garantir que, si une copie est mise à jour, toutes les autres doivent l'être également.

Ce chapitre, par nécessité, n'épuise pas tous les sujets liés à la cohérence des réplications dans les systèmes de stockage de données en cloud distribués.

Comme nous l'avons vu dans la section 4.6, les méthodes de réplication de la cohérence peuvent être regroupées en trois catégories : méthodes de cohérence fixes, méthodes de cohérence configurables et méthodes de contrôle de la cohérence. Les méthodes de cohérence fixe sont principalement basées sur le versionnage des événements. Elles capturent l'idée de l'ordonnancement des événements au moyen de stratégies de contrôle telles qu'un numéro de séquence qui représente une version d'un objet de données ou des mécanismes basés sur des horloges qui sont des concepts bien compris dans les systèmes distribués .L'idée qu'un événement se produise avant un autre représente une relation de cause à effet et l'ordonnancement total des événements parmi les répliques s'est révélé très utile pour résoudre les problèmes de synchronisation liés à la cohérence des données. Ainsi, les méthodes de cohérence dans cette catégorie étendent ce concept sur des

scénarios spécifiques.

Les méthodes de cohérence configurables, quant à elles, visent généralement à fournir des mécanismes qui ajustent automatiquement le degré de cohérence. C'est une caractéristique importante pour les applications qui ont des caractéristiques temporelles, ainsi que pour les systèmes de stockage dans les nuages en temps réel. Plus précisément, les méthodes de cohérence configurables sont adaptées pour répondre aux exigences de cohérence des applications qui doivent s'adapter à des modèles de cohérence prédéfinis.

D'autre part, les méthodes de contrôle de la cohérence ne fournissent pas de garanties spécifiques, mais se concentrent sur la détection de l'apparition de violations de la cohérence dans le stockage des données dans le Cloud. Malgré cela, ces méthodes offrent des contributions significatives qui conviennent aux scénarios où plusieurs clients coopèrent sur des données stockées à distance dans un service potentiellement malhonnête et doivent s'en remettre au CSP pour garantir leur exactitude. En outre, ces clients doivent vérifier si les mises à jour des données demandées ont été correctement exécutées sur toutes les copies stockées à distance, tout en maintenant le niveau de cohérence requis.

Le tableau 4.2 résume les méthodes utilisées dans le cadre de cette recherche (enquête) sur les modèles et méthodes de cohérence . En particulier, la colonne "description" établit un bref lien entre les modèles de cohérence de la section 4.5 "Les modèles de cohérence" et les méthodes abordées dans la section 4.6 "Méthodes de réplication de la cohérence" . Les relations ne sont cependant pas tout à fait nettes, car certaines méthodes sont souples en ce qui concerne le modèle de cohérence qu'elles suivent, tandis que d'autres dépendent de l'application, sur la base d'un contrat ou d'un accord de niveau de service entre l'application et le système.

Catégorie	Méthode	Brève description
Cohérence fixe	Basé sur le séquençage des événements Cohérence	Établit que toutes les répliques d'un enregistrement donné appliquent toutes les mises à jour à un enregistrement dans le même ordre et sont, par conséquent, liées à la **cohérence séquentielle**.
	Cohérence stricte basée sur l'horloge	Utilise des mécanismes basés sur l'horloge pour contrôler les horodatages afin de garantir **une cohérence stricte**.
Cohérence configurable	Cohérence automatisée et auto-adaptative	Fournit une **cohérence** progressivement et dynamiquement **ajustable** au moment de l'exécution en fonction des exigences de cohérence des applications. Applique **des degrés de** cohérence **croissants** pour différents types de données, en fonction de leur sémantique.
	Garanties de cohérence flexibles	Permet aux applications de spécifier **des règles de cohérence**, ou **invariants**, qui doivent être gérées par le système. Prend en charge les mises à jour avec un choix entre **la cohérence linéarisable** et **la cohérence éventuelle**. Prend en charge **les exigences de cohérence des applications** et s'adapte de manière flexible aux **modèles de cohérence prédéfinis**. Permet aux applications d'effectuer des **lectures éventuellement ou fortement cohérentes** selon les besoins.
Surveillance de la cohérence	Vérification de la cohérence	Permet à un groupe de clients se faisant mutuellement confiance de détecter les **violations d'intégrité** et de **cohérence des données**. Permet au propriétaire des données de s'assurer que le fournisseur de services cloud stocke toutes les copies de données convenues dans le **contrat de niveau de service**.
	Audit de cohérence	Implémente une structure d'audit locale et globale pour permettre à un groupe de clients de détecter les **violations de cohérence**.

TABLE 4.2 – Récapitulative des méthodes de cohérence des répliques étudiées

Le tableau 4.3 résume les besoins de stockage soutenus par les systèmes que nous avons abordés afin de souligner quels sont les principaux compromis de cohérence qu'ils considèrent, dans la perspective générale du théorème de CAP. Notez que dans le tableau 4.3, nous n'abordons que les systèmes qui mettent en œuvre une méthode de cohérence spécifique, puisque les méthodes de contrôle de la cohérence ne visent qu'à détecter les violations de la cohérence.

Catégorie	Systèmes représentatifs	Exigences de stockage									
		Automatisation	Disponibilité	Élasticité	Tolérance aux pannes	Faible latence	Tolérance de partition	Performance	Fiabilité	Évolutivité
Méthodes de cohérence fixe	PNUTS		✓		✓	✓	✓	✓	✓	✓
	Spanner		✓		✓	✓	✓			✓
	Clock-SI		✓			✓	✓	✓		✓
	Vela			✓	✓	✓		✓		✓
Méthodes de cohérence configurables	Indigo				✓	✓	✓			
	GEO		✓		✓	✓		✓		✓
	SSOR			✓	✓			✓	✓	✓
	Harmony	✓	✓	✓		✓		✓		
	VFC	✓	✓			✓		✓		
	DynamoDB		✓			✓	✓	✓	✓	✓
	Pileus	✓	✓				✓	✓	✓	✓

TABLE 4.3 – Exigences de stockage des systèmes représentatifs des méthodes de cohérence des répliques étudiées

	Modèle de cohérence centré sur les données					Modèle de cohérence centré sur le client				
Catégorie	Cohérence strict	Cohérence séquentielle	Cohérence causale	Cohérence PRAM	Cohérence faible	Cohérence éventuelle	Cohérence lecture monotone	Cohérence écriture monotone	Cohérence de lecture-écriture	Cohérence écritures aprés lectures
Cohérence fixe	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
Cohérence configurables	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
Surveillance de la cohérence	✗	✗	✓	✗	✗	✗	✓	✓	✓	✓

TABLE 4.4 – Classification des modèles de cohérence

Bien que le théorème de la CAP aborde une question importante dans les systèmes distribués, il existe d'autres compromis liés à la cohérence qui ont un impact direct sur les systèmes modernes de gestion de bases de données distribuées . Ces compromis sont particulièrement liés à la performance, à l'extensibilité et à la latence, comme décrit ci-après.

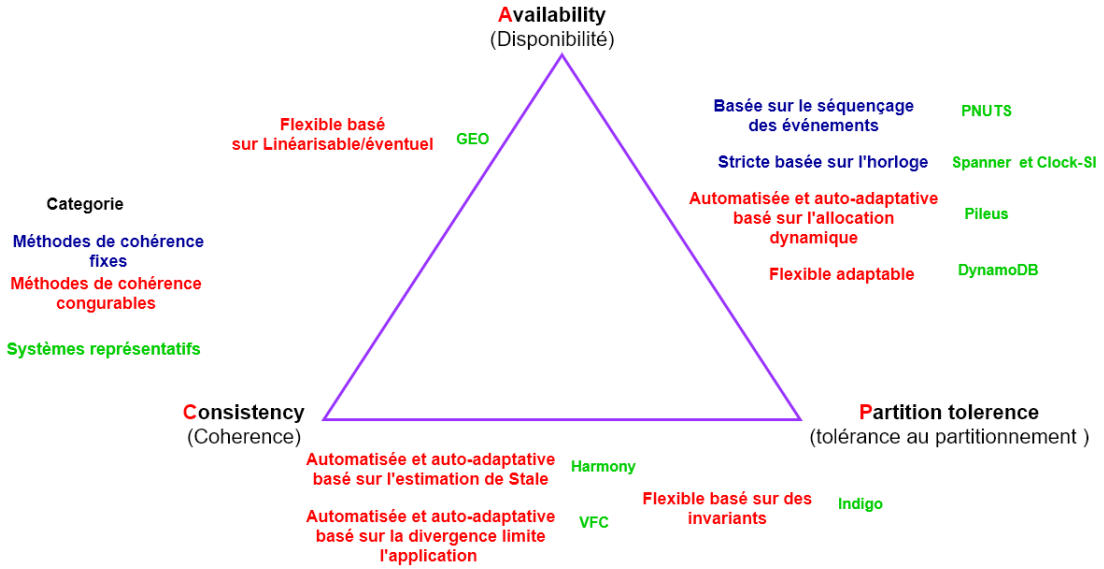
Autre compromis

Cohérence et performance : Comme le souligne Brewer, douze ans après avoir proposé son théorème de CAP , le compromis entre cohérence et performance est encore plus pertinent. Il soutient que les partitions sont rares, de sorte qu'un SGBDD ne devrait considérer le compromis entre la cohérence et la disponibilité que lorsque la tolérance de partition est requise. Toutefois, le compromis entre cohérence et performance est permanent.

Cohérence et évolutivité : Un système évolutif, à son tour, est obtenu de manière non triviale lorsque la cohérence est nécessaire, car cela peut être très coûteux. Ainsi, afin d'améliorer l'évolutivité, certains systèmes fournissent souvent un état de cohérence détendu. Cependant, le prix à payer est que l'état de chaque réplique peut ne pas être toujours le même

Cohérence et latence : Selon Abadi [69] , il existe un lien entre la latence (comprise comme le temps nécessaire pour lancer une opération) et la disponibilité. Un système devient indisponible en présence d'une latence élevée. Sur D'autre part, si la latence diminue, le système devient plus disponible. Cependant, malgré cette implication apparemment évidente, un système peut être disponible, mais peut présenter des taux de latence élevés. Par conséquent, il affirme que les compromis entre cohérence et latence et cohérence et disponibilité sont liés et existent au-delà du théorème de CAP.

Dans ce contexte, le tableau 4.2 montre également les systèmes étudiés dans la perspective des compromis susmentionnés. Comme nous pouvons le voir, la performance, l'extensibilité et la faible latence sont des exigences importantes soutenues par les méthodes de cohérence fixe et configurable. Toutefois, il existe des différences importantes sur la manière dont les méthodes respectives traitent ces exigences.



Dans le cadre de ce mémoire et dans un but pédagogique nous avons choisie d'explorer un des modèle de **cohérence causale** celui-ci fera l'objet du chapitre 5 pour ce faire essayer de réfléchir sur un scénario a concevoir et implémenter pour mettre en évidence la prise en charge de la cohérence par ce modèle et ce a travers une technologie NoSQL a savoir "MongoDB" .

Chapitre 5

Mise en œuvre du modèle de Cohérence Causale

Introduction

La cohérence causale [70] est l'un des critères de cohérence qui peuvent être utilisés sur les bases de données distribuées comme critères de cohérence.

La base de données distribuée fournit une cohérence causale si les opérations de lecture et d'écriture qui sont liées de manière causale sont vues par chaque nœud du système distribué dans le même ordre. Les écritures simultanées peuvent être vues dans un ordre différent dans différents nœuds. La cohérence causale est plus éveillée que la cohérence séquentielle mais plus forte que la cohérence éventuelle.

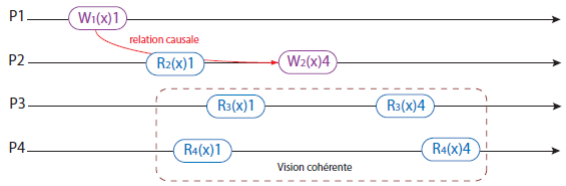


FIGURE 5.1 – Cohérence causal

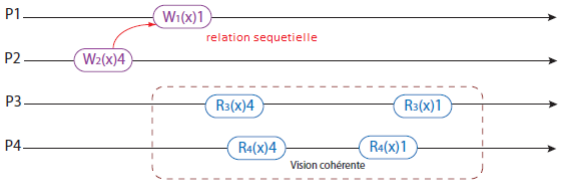
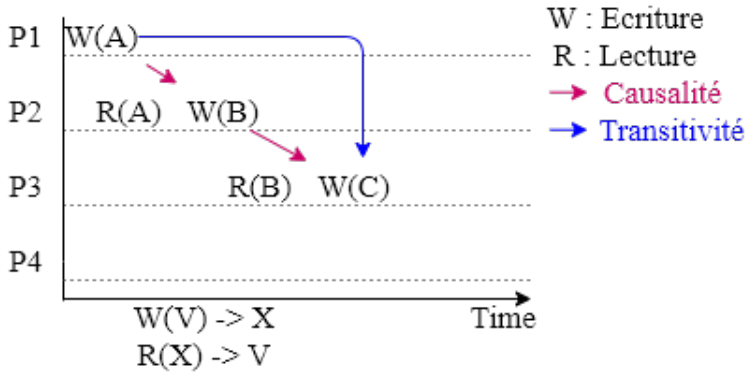


FIGURE 5.2 – Cohérence séquentielle

Lorsqu'une transaction effectue une opération de lecture suivie plus tard d'une opération d'écriture, même sur un objet différent, la première lecture est dite ordonnée de manière causale avant l'écriture. Cela est dû au fait que la valeur créée par l'écriture peut avoir un lien avec le résultat de l'opération de lecture. De même, une opération de lecture est ordonnée de manière causale après l'écriture précédente sur le même objet qui stockait les données récupérées par la lecture. En outre, même deux opérations d'écriture effectuées par le même nœud sont définies comme étant ordonnées de manière causale, dans l'ordre dans lequel elles ont été effectuées. Intuitivement, après avoir écrit la valeur v dans l'objet x, un nœud sait qu'une lecture de x donnerait v, donc une écriture ultérieure pourrait être considérée comme (potentiellement) causalement liée à la précédente. Enfin, l'ordre causal est transitif : c'est-à-dire que si l'opération A est (causalement) ordonnée avant B, et B est ordonnée avant C, implique que A ordonnée avant C.



Les opérations qui ne sont pas liées de manière causale, même par le biais d'autres opérations, sont dites concurrentes. Prenons l'exemple suivant :

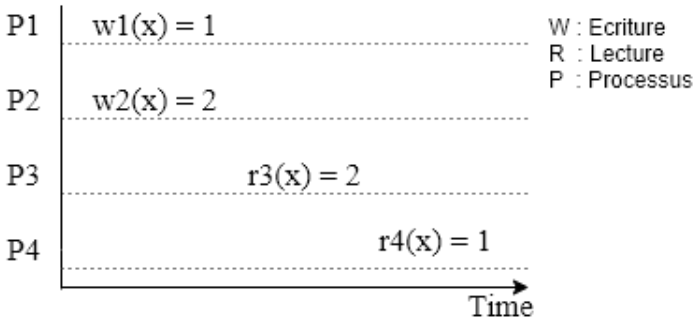


FIGURE 5.3 – Exemple

Cette exécution est causalement cohérente car les opérations de lecture r3 [x] dépendent causalement de l'écriture w2 [x] et l'opération de lecture r4 [x] est causalement dépendante de l'écriture w1 [x].
Remarque : Les opérations d'écriture simultanées peuvent être vues dans un ordre différent sur différentes transactions ou nœuds.

Les bases de données qui garantissent la cohérence causale incluent MongoDB et AntidoteDB.

5.1 Comment mettre en œuvre la cohérence causale

La cohérence causale peut être atteinte en utilisant **des horloges de Lamport** [71] ou **des vecteurs de version** [72]. Le modèle de cohérence causale est implémenté à l'aide d'horodatages en plusieurs parties, qui sont attribués à chaque objet. Ces horodatages sont stockés sur un vecteur contenant le numéro de version de l'objet à chaque réplica. Ce vecteur doit être inclus (sous forme de dépendances) dans toutes les requêtes de mise à jour et d'interrogation pour que les opérations respectent l'ordre causal : une opération A ne peut être traitée à un nœud donné que si toutes les opérations dont dépend causalement l'opération A ont déjà appliqué à ce nœud .

Chaque élément de l'horloge vectorielle correspond à un hôte et la valeur de l'élément indique le nombre de messages envoyés depuis cet hôte. Les informations d'horloge vectorielle sont utilisées pour ordonner ou retarder la livraison des messages si nécessaire, maintenant ainsi la cohérence requise. Cependant, pour maintenir la cohérence des éléments de données, nous avons besoin d'informations sur les écritures sur chaque élément de données, et le maintien d'une horloge par élément de données peut aider. Par conséquent, au lieu d'une horloge vectorielle de taille N (nombre d'hôtes), nous maintenons un vecteur de taille M (nombre d'objets). La valeur de v [i] dans le vecteur v contient le nombre d'écritures sur la donnée i.

Chaque hôte gère un vecteur de taille M. Chaque fois qu'il met à jour la valeur d'un élément, il incrémente l'élément correspondant et envoie le vecteur avec le message de l'élément de données et la nouvelle valeur à chaque site, qui a une copie de la réplique. Lorsqu'un hôte reçoit un message de mise à jour, il retarde la livraison d'un message jusqu'à ce que chaque élément de son vecteur soit supérieur ou égal à celui qui est superposé. Après cela, les mises à jour des éléments de données sont appliquées. Dans ce cas, le surdébit de message est O (M) et est donc indépendant du nombre d'hôtes dans le système.

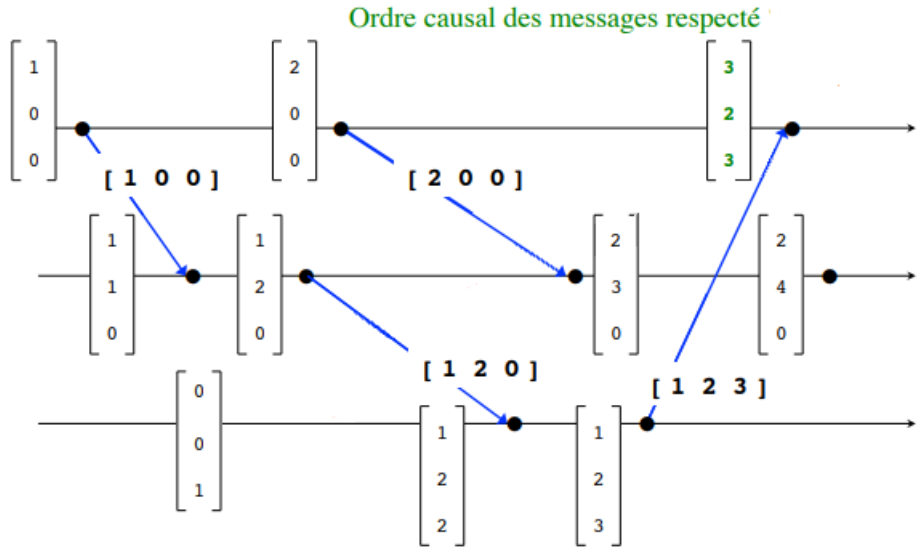


FIGURE 5.4 – Ordonnancement causale a l'aide de l'horloge vectorielle

5.1.1 Horloge de Lamport

L'algorithme d'horodatage de Lamport est un algorithme d'horloge logique simple utilisé pour déterminer l'ordre des événements dans un système informatique distribué. Comme les différents nœuds ou processus ne seront généralement pas parfaitement synchronisés, cet algorithme est utilisé pour fournir un ordre partiel des événements avec une surcharge minimale, et fournit conceptuellement un point de départ pour la méthode d'horloge vectorielle plus avancée. L'horloge de Lamport peut être considérée comme une horloge qui n'a de sens que par rapport aux messages circulant entre les processus. Lorsqu'un processus reçoit un message, il resynchronise son horloge logique avec celle de l'expéditeur.

Principe :

- Un processus incrémente son compteur avant chaque événement.
- Lorsqu'un processus envoie un message, il inclut sa valeur de compteur avec le message.
- A la réception d'un message, le compteur du destinataire est mis à jour, si nécessaire, au plus grand de son compteur courant et de l'horodatage du message reçu. Le compteur est ensuite incrémenté de 1 avant que le message ne soit considéré comme reçu.

5.1.2 Vecteur de version

Un vecteur de version est un mécanisme de suivi des modifications apportées aux données dans un système distribué, dans lequel plusieurs agents peuvent mettre à jour les données à des moments différents. Le vecteur de version permet aux participants de déterminer si une mise à jour a précédé une autre (survenue avant), l'a suivie, ou si les deux mises à jour se sont produites simultanément (et pourraient donc entrer en conflit l'une avec l'autre). De cette manière, les vecteurs de version permettent le suivi de causalité parmi les répliques de données et constituent un mécanisme de base pour une réplication optimiste.

5.2 La cohérence causale dans Facebook

Facebook est un service de réseautage social en ligne. Après s'être inscrit pour utiliser le site, les utilisateurs peuvent créer un profil utilisateur, ajouter d'autres utilisateurs comme amis, échanger des messages, publier des mises à jour de statut et des photos, partager des vidéos et recevoir des notifications lorsque d'autres mettent à jour leurs profils.

Lorsque on se connecte à notre compte sur Facebook, le serveur affiche vos propres messages d'état et les messages d'état de nos amis à ce moment-là. Les messages d'état sur Facebook peuvent contenir des images, des liens et des histoires partagés ou vos propres messages. Naturellement, les données de votre compte nécessitent une forte cohérence, mais pour les données d'état, les modèles de cohérence les plus faibles sont acceptables. Pendant le temps que l'utilisateur est en ligne, les mises à jour de statut des amis d'un utilisateur et de l'utilisateur n'ont pas besoin d'être strictement commandées, et la commande causale est suffisante.

Ainsi, lorsqu'un utilisateur A envoie une mise à jour de statut et qu'un utilisateur B répond à cette mise à jour, il y a un ordre causal sur les deux mises à jour. Cependant, lorsque les utilisateurs C et D effectuent une mise à jour totalement indépendante, l'ordre dans lequel ces mises à jour apparaissent aux utilisateurs A et B n'est pas pertinent. En effet, les utilisateurs A et B ne savent pas dans quel ordre les mises à jour sont effectuées.

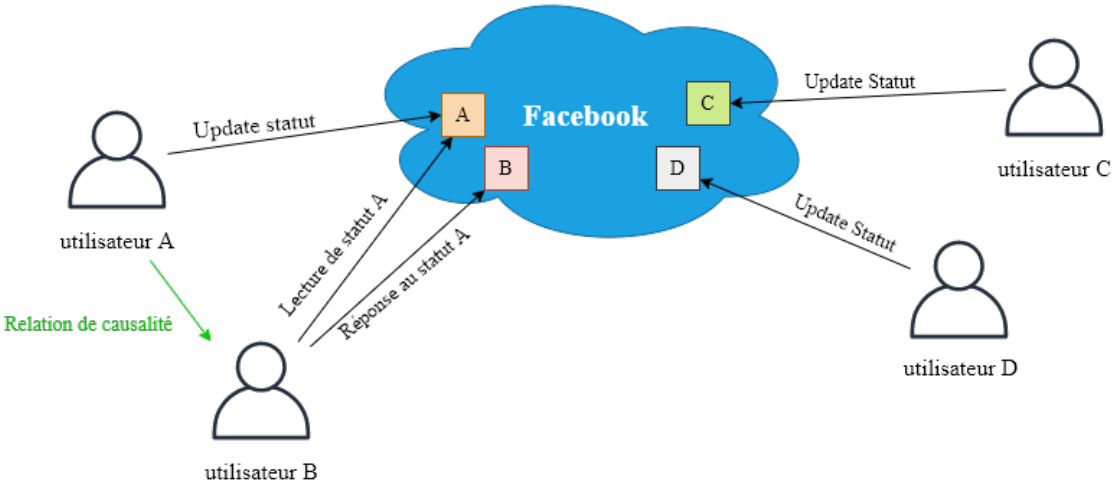


FIGURE 5.5 – exemple

Bien que la cohérence causale soit le modèle de cohérence possible pour les mises à jour de statut Facebook et plusieurs services distribués similaires contenant des mises à jour de statut comme LinkedIn, Twitter et Yahoo, à la connaissance de l'auteur, il n'y a pas de littérature scientifique ou autre qui montrerait la cohérence causale réellement utilisée.

- **Scénario :**

  Considérez le scénario hypothétique suivant sur un site de réseau social comme Facebook [73] :

  1. Sally ne trouve pas son fils Billy. Elle publie la mise à jour S à ses amis : "Je pense que Billy a disparu !"

  2. Un instant après que Sally ait publié S, Billy appelle sa mère pour lui faire savoir qu'il est chez un ami. Sally édite S, ce qui donne S * : "Fausse alarme ! Billy est allé jouer".

  3. James, l'ami de Sally, observe S * et publie le statut J en réponse : "Quel soulagement !"

  Ce scénario aura deux issus selon la mise en œuvre de la cohérence causale ou non.

Cas 1 : Scénario face à l'absence de cohérence causale

Si la causalité n'est pas respectée, un troisième utilisateur, Henry, pourrait percevoir les effets avant leurs causes ; si Henry observe S et J mais pas S*, il pourrait penser que James est heureux d'apprendre la disparition potentielle de Billy !

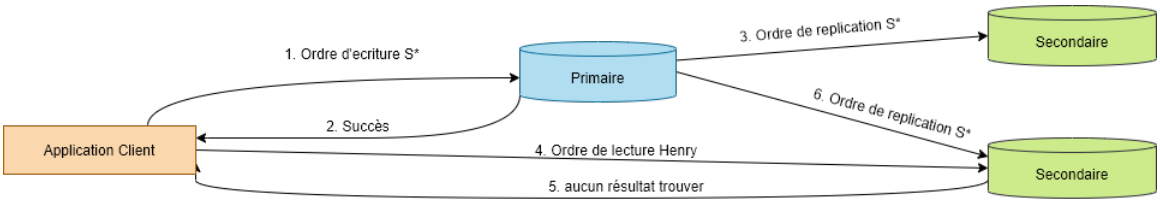


Figure 5.6 – Flux des opérations dans un jeu de réplicas sans cohérence causale appliquée

1. L'application client écrit le message de Sally S* sur le serveur principal
2. Le principal répond qu'il a appliqué avec succès l'écriture
3. Le message est répliqué du primaire vers l'un des secondaires
4. L'application client lit la collecte des commandes sur un serveur secondaire
5. Le secondaire ciblé n'a pas reçu le message S*, il répond donc sans résultat
6. Le message est répliqué du primaire vers l'autre secondaire

Si la causalité est respectée, un troisième utilisateur, Henry, ne pourra pas percevoir les effets J sans S*.

Cas 2 : Application de la cohérence causale.

Nous allons voir se qui se passe quand la coherence causale est appliquer nous allons obtenir se résultat grâce a des solution comme MongoDB qui pour établir un ordre global partiel des événements pour les ensembles de répliques et des clusters partitionnés, MongoDB a implémenté une horloge logique hybride basée sur l' horloge logique Lamport. Chaque écriture ou événement qui change d'état dans le système se voit attribuer une heure à laquelle il est appliqué au primaire. Cette heure peut être comparée entre tous les membres du déploiement. Chaque participant d'un cluster fragmenté, des pilotes aux routeurs de requêtes en passant par les nœuds porteurs de données, doit suivre et envoyer la valeur de la dernière heure dans chaque message, permettant à chaque nœud à travers les fragments de converger dans leur notion de la dernière heure. Les primaires utilisent la dernière heure logique pour attribuer de nouvelles heures aux écritures suivantes. Cela crée un ordre causal pour toute série d'opérations associées. Un nœud peut utiliser l'ordre causal pour attendre avant d'effectuer une lecture ou une écriture nécessaire, en s'assurant que cela se produit après une autre opération.

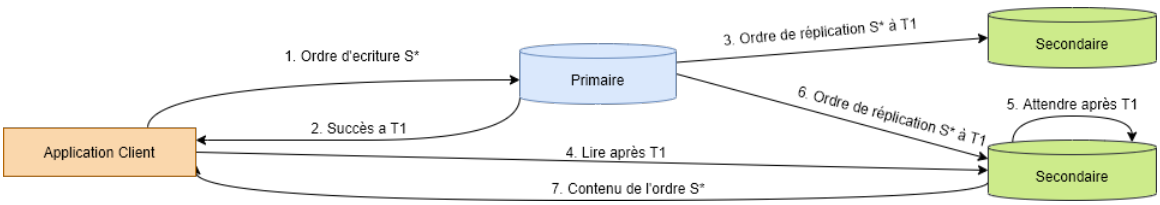


Figure 5.7 – Flux des opérations dans un jeu de réplicas avec cohérence causale appliquée

1. L'application client écrit le message de Sally S* sur le serveur principal
2. Le primaire répond qu'il a enregistré avec succès l'écriture au moment T1
3. Le message est répliqué du primaire vers l'un des secondaires
4. L'application client lit après l'heure T1 sur un secondaire
5. **Le secondaire ciblé n'a pas reçu la réplique du message S* à l'heure T1, il doit donc attendre pour répondre.**
6. Le message est répliqué du primaire vers l'autre secondaire
7. **Le secondaire est capable de répondre avec le contenu du message de S***

Conclusion

Le maintien des relations causales est important pour les services en ligne. La perception humaine de la causalité entre les événements est bien étudiée dans les domaines de la psychologie et de la philosophie ; tout comme l'espace et le temps imposent une causalité aux événements du monde réel, le maintien des relations causales entre les événements virtuels est une considération importante lors de la création de services destinés à l'homme. Parfois, le maintien de la causalité est insuffisant et des modèles plus solides sont nécessaires . Cependant, en particulier compte tenu de sa disponibilité et des avantages de latence, la cohérence causale est souvent un choix raisonnable .

Troisième partie

Implémentation

Chapitre 6

Choix des outils technologiques

Pour la mise en œuvre de notre travail nous avons utilisé l'outil technologique MongoDB comme SGBD pour nos bases de données NoSQL. Tout au long de cette partie nous présenterons les étapes d'installation de ces technologies ainsi que les outils de développement utilisé pour développer un programme qui assure la cohérence et la disponibilité des données

6.1 Présentation de MongoDB

MongoDB est un système de base de donnée orienté objet, dynamique, stable, scalable et sans SQL. Les objets sont stockés sous format JSON dans des documents séparés. Au lieu de stocker les données sous format de tables avec des valeurs, on utilise une hierarchie et un système d'objet JSON pour créer un système plus adaptatif.

L'intérêt de MongoDB, c'est d'implémenter un système à hautes performances avec une scalabitlité parfaite. Le système est extrêmement simple à installer, il fonctionne sous tout les système d'exploitation et il existe des librairies pour le manipuler dans tout les langages[74].

6.1.1 Historique :

- Le développement de MongoDB a commencé en 2007, lorsque la société (alors appelée 10gen) se développait une plate-forme en tant que service comme windows Azure ou Google App Engine. En 2009, MongoDB a été open source en tant que produit autonome avec une licence AGPL.
- Depuis la version 1.4 (Mars 2010), MongoDB a été considéré comme prêt pour la production.
- La version stable 3.0 a été publié le 3 Février 2015 [75].

6.1.2 Fonctionnalité principales :

1. **Réplication dans MongoDB :** un ensemble de repliques est un groupe d'instance mongod qui conservent le meme ensemble de données.Un ensemble de repliques contient plusieurs nœuds porteur de donnnées et en option , un nœud d'arbitrage.Parmi les nœuds porteurs de données,un seul membre est considéré comme le nœud primaire, tandis que les autres nœuds sont considérer comme des nœuds secondaires[76] .

   le nœds primaire reçois toutes les opération d'écriture. Un ensemble de répliques ne peut avoir qu'un seul primaire capable de confirmer les écritures avec la préoccupation d'écriture w : "majorité" ; bien que dans certaines circonstances, une autre

instance mongod puisse se croire transitoirement aussi primaire. La primaire enregistre tous les changements apportés à ses ensembles de données dans son journal des opérations.

Les secondaires réplique l'oplog de la primaire et appliquent les opérations à leurs

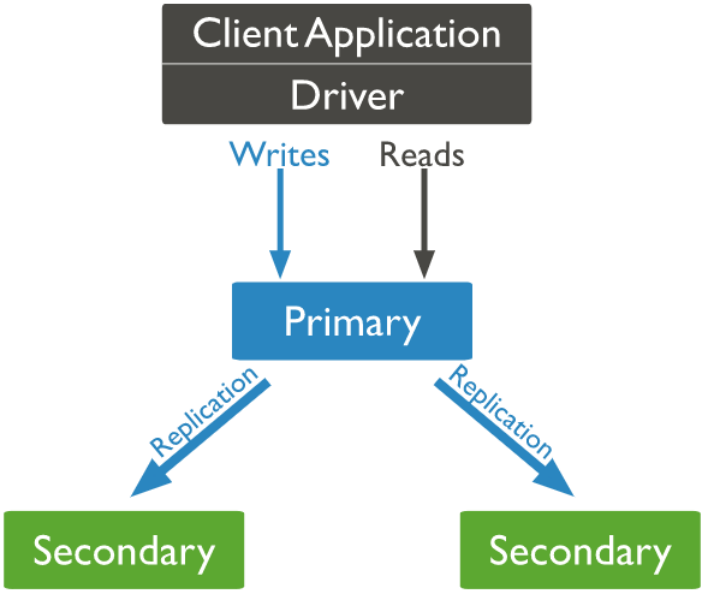


FIGURE 6.1 – Réplique de l'ensemble des opérations de lecture et d'écriture primaires

ensembles de données de telle sorte que les ensembles de données des secondaires reflètent l'ensemble de données de la primaire. Si la réplique primaire n'est pas disponible, une réplique secondaire éligible organisera une élection pour choisir elle-même la nouvelle primaire.

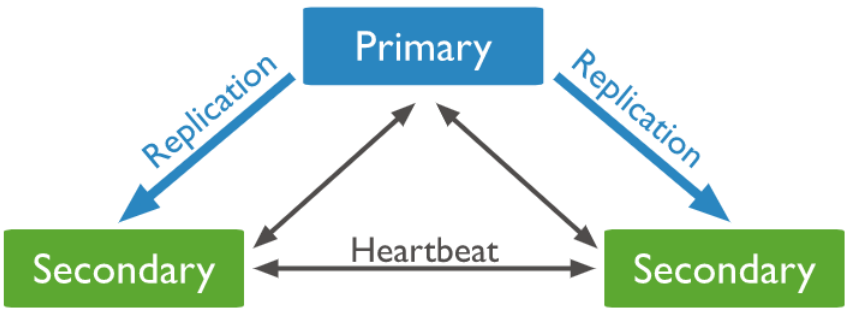


FIGURE 6.2 – Réplique d'un ensemble primaire avec deux secondaires.

Dans certaines circonstances (par exemple, nous avons un primaire et un secondaire mais les contraintes de coût interdisent d'ajouter un autre secondaire), on peut choisir d'ajouter une instance de mongod à un ensemble de répliques en tant qu'arbitre. Un arbitre participe aux élections mais ne détient pas de données (c'est-à-dire qu'il ne fournit pas de redondance de données).

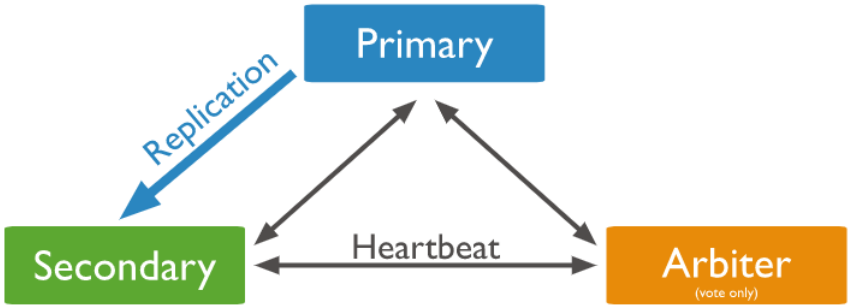


FIGURE 6.3 – Exemple de diagramme de l'environnement VDI d'une entreprise.

Un arbitre sera toujours un arbitre alors qu'un primaire peut se retirer et devenir un secondaire et qu'un secondaire peut devenir la primaire lors d'une élection.

2. **Sharding :** Le sharding est une méthode de distribution des données sur plusieurs machines. MongoDB utilise le sharding pour soutenir des déploiements avec de très grands ensembles de données et des opérations à haut débit et les systèmes de bases de données comportant de grands ensembles de données ou des applications à haut débit peuvent mettre à l'épreuve la capacité d'un seul serveur[77].
   - **Échelle verticale :** consiste à augmenter la capacité d'un seul serveur, par exemple en utilisant une unité centrale plus puissante, en ajoutant de la mémoire vive ou en augmentant l'espace de stockage. Les limites de la technologie disponible peuvent empêcher une seule machine d'être suffisamment puissante pour une charge de travail donnée. De plus, les fournisseurs de services en nuage ont des plafonds de capacité basés sur les configurations matérielles disponibles. Par conséquent, il existe un maximum pratique pour la mise à l'échelle verticale.
   - **Échelle horizontale :**consiste à répartir l'ensemble de données et la charge du système sur plusieurs serveurs, en ajoutant des serveurs supplémentaires pour augmenter la capacité selon les besoins. Bien que la vitesse ou la capacité globale d'une seule machine puisse ne pas être élevée, chaque machine gère un sous-ensemble de la charge de travail globale, ce qui peut offrir une meilleure efficacité qu'un seul serveur à grande vitesse et à grande capacité. L'augmentation de la capacité du déploiement ne nécessite que l'ajout de serveurs supplémentaires selon les besoins, ce qui peut représenter un coût global inférieur à celui du matériel haut de gamme pour une seule machine. La contrepartie est une complexité accrue de l'infrastructure et de la maintenance du déploiement.

   (a) **Cluster fragmenté :**Un cluster fragmenté sur MongoDB se compose des éléments suivants :
      - **Fragment (Shard) :** Chaque fragment contient un sous-ensemble des données fragmentées qui peut être déployé comme un ensemble de répliques.

- **Mongos :** Le mongos agit comme un routeur de requêtes, fournissant une interface entre les applications clientes et le cluster fragmenté.
- **serveurs configurer :** Les serveurs de configuration stockent les métadonnées et les paramètres de configuration pour le cluster.

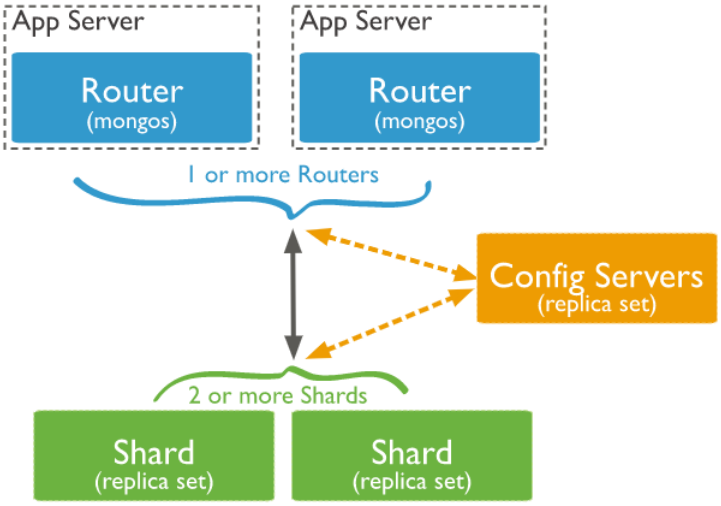


FIGURE 6.4 – Réplique de l'ensemble des opérations de lecture et d'écriture primaires

(b) **Avantages du sharding :**

- **Lecture / Écriture :** MongoDB distribue la charge de travail en lecture et écriture entre les fragments du cluster fragmenté, permettant à chaque fragment de traiter un sous-ensemble d'opérations du cluster.Les charges de travail en lecture et en écriture peuvent être réparties horizontalement sur l'ensemble du cluster en ajoutant des fragments supplémentaires.
- **Capacité de stockage :** Les données sont réparties entre les fragments du cluster, ce qui permet à chaque fragment de contenir un sous-ensemble des données totales du cluster. À mesure que l'ensemble de données augmente, des fragments supplémentaires augmentent la capacité de stockage du cluster.
- **Haute disponibilité :** Le déploiement de serveurs de configuration et de fragments en tant que répliques permet d'accroître la disponibilité. Même si une ou plusieurs répliques de fragments deviennent totalement indisponibles, le cluster fragmenté peut continuer à effectuer des lectures et des écritures partielles. En d'autres termes, bien que les données sur le ou les fragments non disponibles ne soient pas accessibles, les lectures ou les écritures sur les fragments disponibles peuvent toujours réussir.

MongoDB prend en charge la mise à l'échelle horizontale à travers le sharding.

3. **GridFS :** c' est une spécification pour le stockage et la récupération de fichiers dont la taille dépasse la limite de 16 MB du BSON-document. Au lieu de stocker un fichier dans un seul document, GridFS divise le fichier en parties, ou morceaux , et stocke chaque morceau comme un document séparé. Par défaut, GridFS utilise une taille de morceau par défaut de 255 kB ; c'est-à-dire que GridFS divise un fichier en morceaux de 255 kB à l'exception du dernier morceau. Le dernier morceau n'est pas plus gros que nécessaire. De même, les fichiers qui ne dépassent pas la taille du morceau n'ont qu'un dernier morceau, n'utilisant que l'espace nécessaire et quelques métadonnées supplémentaires.

   GridFS utilise deux collections pour stocker les fichiers. Une collection stocke les morceaux de fichiers, et l'autre stocke les métadonnées des fichiers.

   La section Collections GridFS décrit chaque collection en détail.Lorsque vous interrogez GridFS pour un fichier, le pilote réassemble les morceaux selon les besoins. Vous pouvez effectuer des requêtes de plage sur des fichiers stockés via GridFS. Vous pouvez également accéder à des informations provenant de sections arbitraires de fichiers, comme par exemple "sauter" au milieu d'un fichier vidéo ou audio[78].

   - **Quand utiliser les GridFS :** Dans certaines situations, le stockage de gros fichiers peut être plus efficace dans une base de données MongoDB que dans un système de fichiers au niveau du système.
     - Si votre système de fichiers limite le nombre de fichiers dans un répertoire, vous pouvez utiliser GridFS pour stocker autant de fichiers que nécessaire.
     - Lorsque vous souhaitez accéder à des informations provenant de parties de fichiers volumineux sans avoir à charger des fichiers entiers en mémoire, vous pouvez utiliser GridFS pour rappeler des sections de fichiers sans avoir à lire le fichier entier en mémoire.
     - Si vous souhaitez que vos fichiers et métadonnées soient automatiquement synchronisés et déployés sur plusieurs systèmes et installations, vous pouvez utiliser GridFS. Lorsque vous utilisez des ensembles de répliques géographiquement distribués, MongoDB peut distribuer automatiquement les fichiers et leurs métadonnées à un certain nombre d'instances et d'installations mongod.

6.1.3 Caractéristiques de MongoDB :

- **Requête ad hoc :** MongoDB soutient les champs de recherche, des gammes et des expressions régulières. Les requêtes peuvent retourner des champs spécifiques de documents et inclure des fonctions définies par l'utilisateur en JavaScript.
- **Indexage :**Tous les champs dans MongoDB peut être indexé (les index dans MongoDB sont conceptuellement similaires à ceux des SGBDR traditionnels). Il existe également des indices secondaires, des index uniques, des index, des index dispersés géospatiales et index en texte intégral.
- **Fiabilité élevée :** MongoDB fournit une haute disponibilité et la gestion de la charge grâce à une augmentation de la réplication ensemble. Un jeu de répliques est constitué de deux ou plusieurs copies des données. Chaque réplique peut avoir le rôle de la copie primaire ou secondaire à tout moment. La réplique primaire exécute toutes les écritures et les lectures. Les répliques secondaires conservent une copie de la réplication de données primaire à travers un mécanisme de replication inclus avec le produit. En cas d'échec réplique primaire, la réplication de jeu démarre automatiquement un processus électoral pour déterminer quelle réplique secondaire doit devenir primaire. copies secondaires peuvent également prendre des lectures avec des données cohérentes par défaut si nécessaire.

- **Stockage de fichiers :**
  - MongoDB peut également être utilisé comme un système de fichiers, en tirant parti des fonctionnalités de réplication et d'équilibrage sur plusieurs serveurs pour stocker des fichiers, même grand.
  - La fonction, appelée GridFS, est inclus dans les pilotes MongoDB et facilement accessibles en plusieurs langues de développement. MongoDB expose des fonctions de manipulation de fichiers. GridFS est utilisé, par exemple, dans Nginx et plugin lighttpd. Au lieu de stocker le fichier dans un seul document, GridFS divise le fichier en plusieurs parties plus petites, appelées morceaux, et les magasins chacun de ces morceaux dans un document distinct.
  - Ai fichiers peuvent être des métadonnées associées, sur lequel il est également possible de créer des index de texte intégral.
- **Agrégation :**MongoDB prend en charge deux modes d'agrégation de données : MapReduce et cadre Aggregration. Ce dernier fonctionne comme un pipeline et permet d'obtenir des résultats beaucoup plus rapidement que MapReduce grâce à la mise en œuvre en C ++.
- **Collection Capped :**MongoDB prend en charge un appel de collecte de collecte de taille fixe plafonnée. Ce type de collection maintenir l'ordre d'insertion et une fois qu'il atteint la taille définie, se comportent comme des files d'attente circulaires [78].

6.1.4 Architecture de déploiement distribuée

Une architecture MongoDB distribuée utilise les notions suivantes[79] :

- **Replica Set (RS) :**
  - Les Replica Set assurent la haute disponibilité de Mongodb.
  - Un Replica Set est composé d'un nœud primaire et de deux nœuds secondaires. (Règles Mongodb de production)
  - L'écriture se fait obligatoirement sur le nœud primaire
- **Replica Set de config :**
  - Un Replica Set est dédié pour le stockage de la configuration du cluster.
  - Comme tous les autres Replica Set, il est recommandé de le peupler d'au moins 3 nœuds.

- **Sharding :**
  - Mongodb utilise la sharding pour scaler la base de données (scalabilité horizontale)
  - Le sharding distribue les données à travers les n partitions physiques (shards) dont le cluster est composé
  - Bien choisir la clé de sharding est primordial pour une répartition égale des documents insérés dans les différents shards
  - Chaque shard est composé d'un Replica Set
- **Routeur de requêtes :**
  - Le routeur mongos permet de rediriger une requête sur le ou les shards requis, en fonction de la clé de sharding ; il agit comme coordinateur de requête.

Une architecture MongoDB distribuée comprend 3 types de nœuds différents :

- **mongod :**stockent les données des replica set métier ;
- **mongos :** routent les requêtes ;
- **mongoc :** stockent les données d'état et de configuration du cluster (ces nœuds utilisent en fait un moteur mongod, mais pour un replica set particulier : le replica set de configuration).

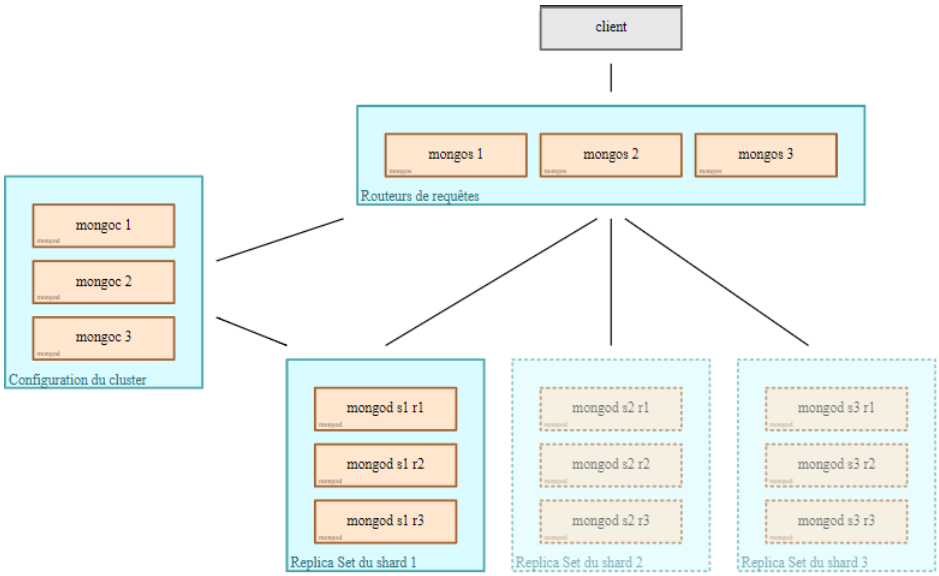


FIGURE 6.5 – Déploiement d'un cluster Mongo DB avec sharding.

Pour la mise en œuvre de notre travail nous avons utilisé l'outil technologique **MongoDB** comme SGBD pour nos bases de données **NoSQL**. Tout au long de cette partie nous présenterons les étapes d'installation de ces technologies ainsi que les outils de développement utilisé pour développer un programme qui assure la cohérence et la disponibilité des données

6.2 Visual Studio Code

Visual Studio Code est un éditeur de code source léger mais puissant qui fonctionne sur votre bureau et est disponible pour Windows,macOS et Linux. Il prend en charge JavaScript, Typescript et Node.js et dispose d'un riche ecosysteme d'extensions pour d'autre langages(tels que C++, C#, Java, Python ,PHP) et environnements d'execution(tels que net et Unity)[80].

FIGURE 6.6 – icon de Visual Code Studio

6.3 Node.js :

Node.js est une plateforme basée sur le moteur d'exécution JavaScript de Chrome pour construire facilement des applications réseau rapides et évolutives. Elle utilise un modèle d'entrée/sortie non bloquant, piloté par les événements, qui la rend légère et efficace, parfaite pour les applications en temps réel gourmandes en données qui s'exécutent sur des périphériques distribués. Il s'agit d'un environnement d'exécution multiplateforme et open source pour le développement d'applications côté serveur et en réseau. Les applications Node.js sont écrites en JavaScript et peuvent être exécutées dans le runtime Node.js sous OS X, Microsoft Windows et Linux[81].

Figure 6.7 – icon de Node.js

6.3.1 Caractéristiques de Node.js

- **Asynchrone et événementiel :** Toutes les API de la bibliothèque Node.js sont asynchrones, c'est-à-dire non bloquantes. Cela signifie essentiellement qu'un serveur basé sur Node.js n'attend jamais qu'une API renvoie des données. Le serveur passe à l'API suivante après l'avoir appelée et un mécanisme de notification des événements de Node.js aide le serveur à obtenir une réponse à l'appel de l'API précédente.

- **Très rapide :** Construite sur le moteur JavaScript V8 de Google Chrome, la bibliothèque Node.js est très rapide dans l'exécution du code.
- **Simple mais très évolutif :** Node.js utilise un modèle à fil unique avec bouclage d'événements. Le mécanisme d'événement aide le serveur à répondre de manière non bloquante et rend le serveur très évolutif, contrairement aux serveurs traditionnels qui créent des flux limités pour traiter les requêtes. Node.js utilise un seul programme d'exécution et le même programme peut fournir un service à un nombre beaucoup plus important de demandes que les serveurs traditionnels comme le serveur HTTP Apache.

- **Pas de mise en mémoire tampon :** Les applications Node.js ne stockent jamais de données en mémoire tampon. Ces applications se contentent de fournir les données par morceaux.

6.3.2 Où utiliser Node.js :

Voici les domaines dans lesquels Node.js s'avère être un partenaire technologique parfait.

- Applications liées aux E/S.
- Applications de diffusion de données en continu.
- Applications intensives de données en temps réel .
- Applications basées sur les API JSON.
- Single Page Applications.

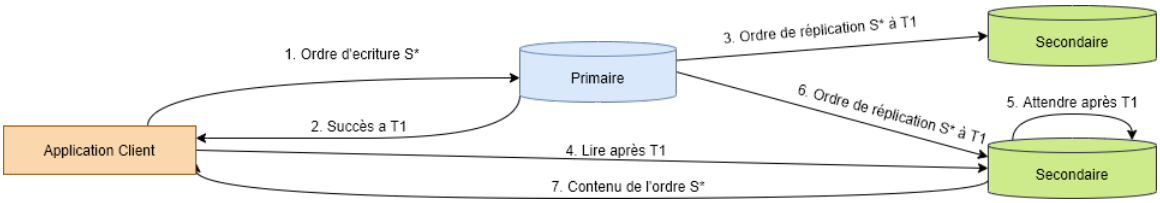
Chapitre 7

Implémentation du scénario avec cohérence causale sous MongoDB

Dans ce chapitre on vas voir les différents étapes à suivre pour implémenté notre scénario qui serons composé comme suit :

- Création des dossier pour accueillir les donnée des 2 serveur secondaire.
- Configuration de la réplication sur MongoDB.
- Simulation de scénario.

Reprenons le schéma de la figure 5.7 qui illustre notre scénario



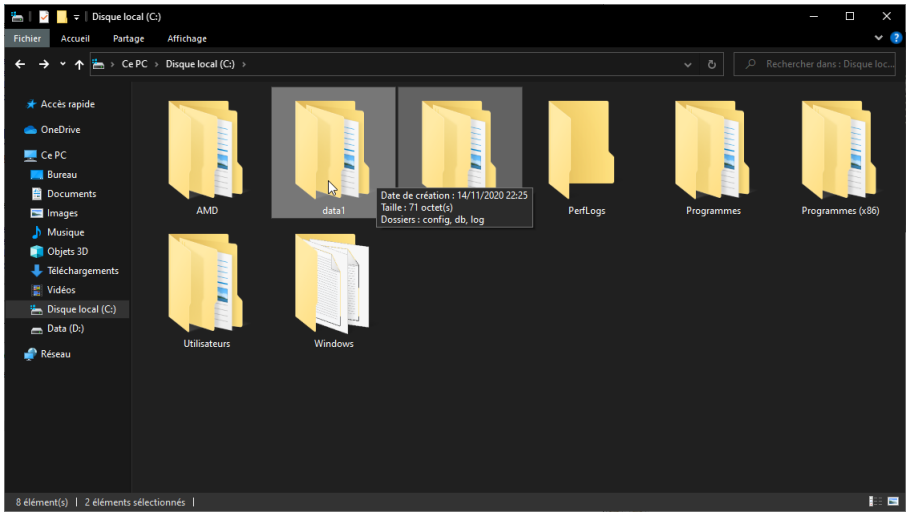
7.1 Implémentation des serveurs

Dans cette partie nous allons montrer comment configurer les 3 serveur que nous allons utiliser pour l'implémentation.

Etape 1 : Création des dossier pour accueillir les donnée des 2 serveur secondaire

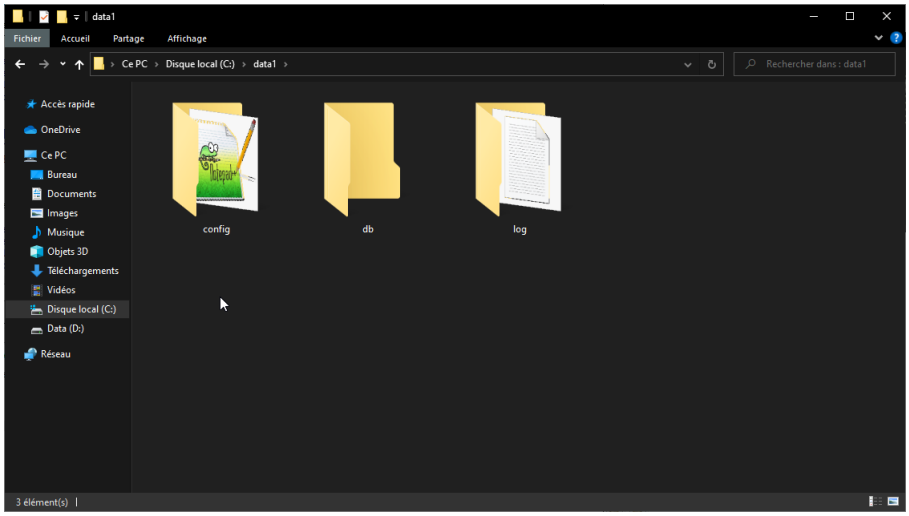
On commence par crées les 2 serveurs secondaire Data Center 1 et Data Center 2, techniquement nous procèderons comme suit :

1. Aller dans la partition système **C :** et créer 2 dossier un pour le premier Data Center que l'on va nommé **data1** et un autre pour le deuxième Data Center que l'on va nommé **data2**.

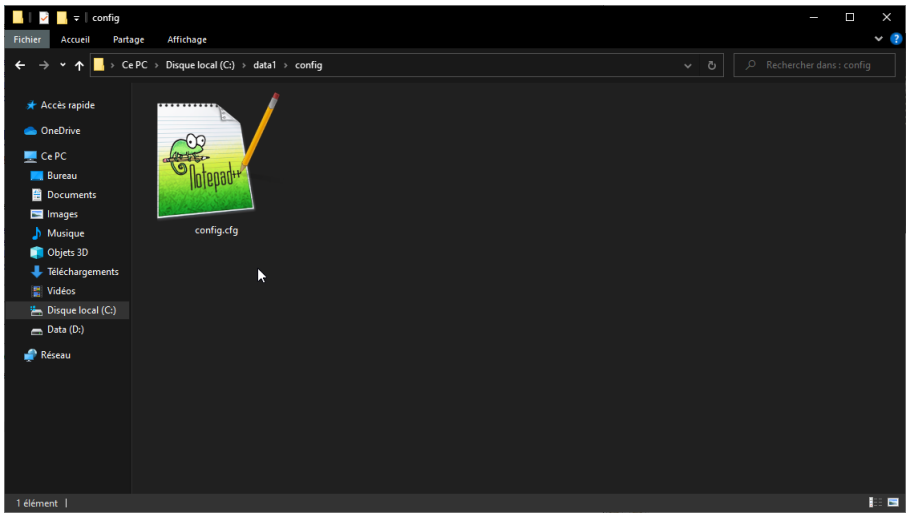


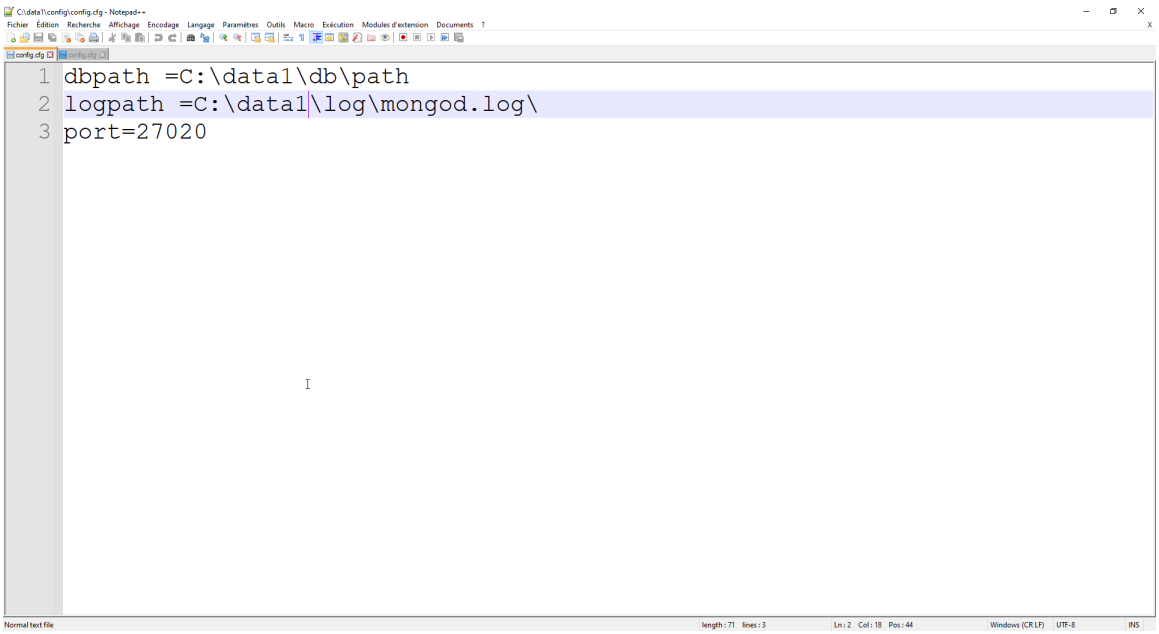
2. Dans ces même dossier nous allons créer 3 autre dossier :

   (a) Un pour stocker les fichier de configuration que l'on va nommé **config**.
   (b) Un autre pour stocker les fichier de la base de donnée que l'on nommera **db**.
   (c) Et le dernier pour stocker le fichier de connexion qui sera nommé **log**.

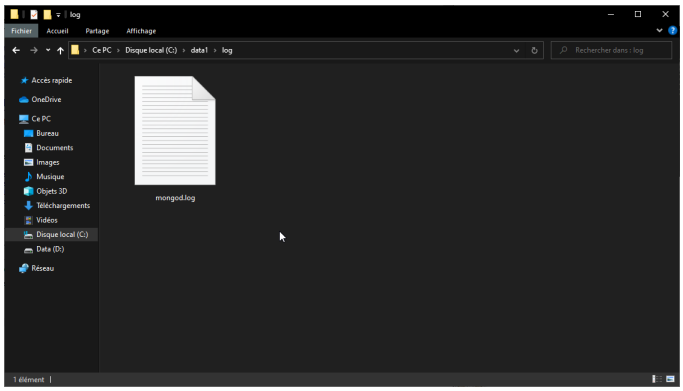


3. Dans le dossier **config** du **Data Center 1** nous allons créer un fichier config.cfg dans lequel nous allons entrer les information comme le chemin d'accès vers la base de donnés qui sera **dbpath** et le chemin d'accès au log qui sera **logpath** et enfin le numéro de port de connexion qui est **27020** .

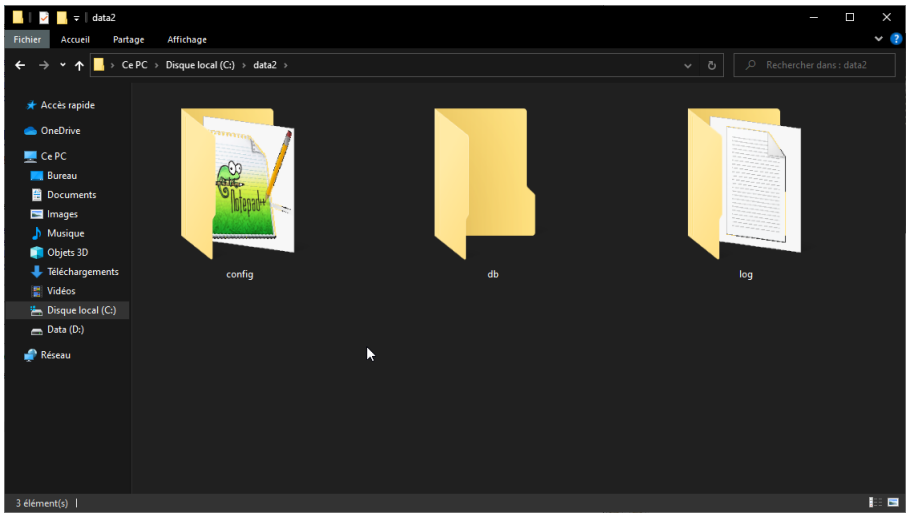


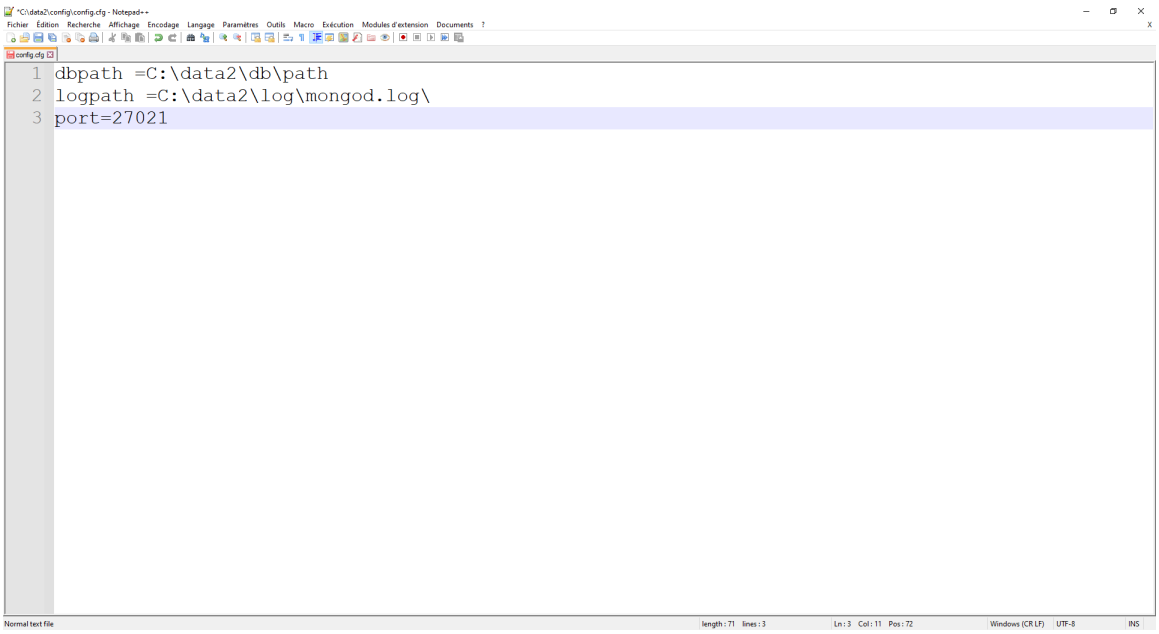


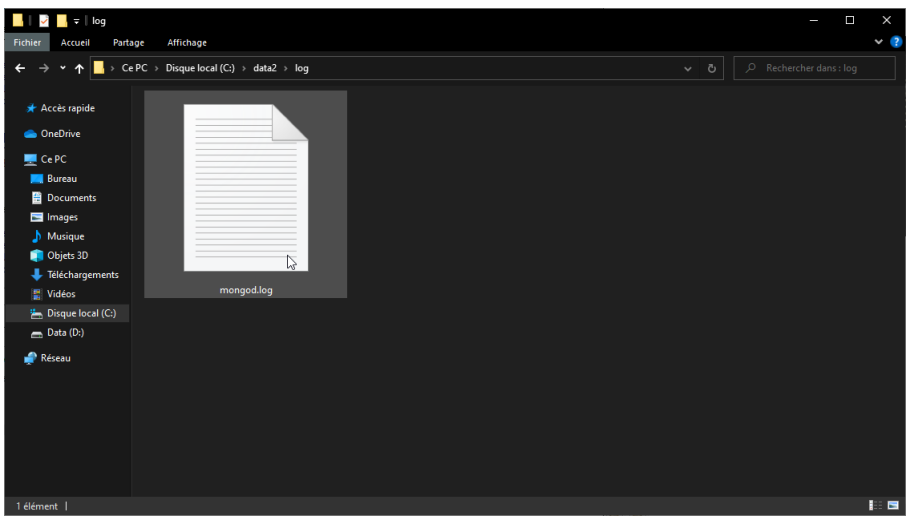
4. Ensuite nous allons créer un fichier mongo.log dans le dossier log du même dossier **Data Center 1** .



5. Nous allons faire la même chose avec le dossier **Data Center 2** mais nous allons changer le numéro de port en 27021 comme si-dessous :







Etape 2 : Configuration de la réplication sur MongoDB.

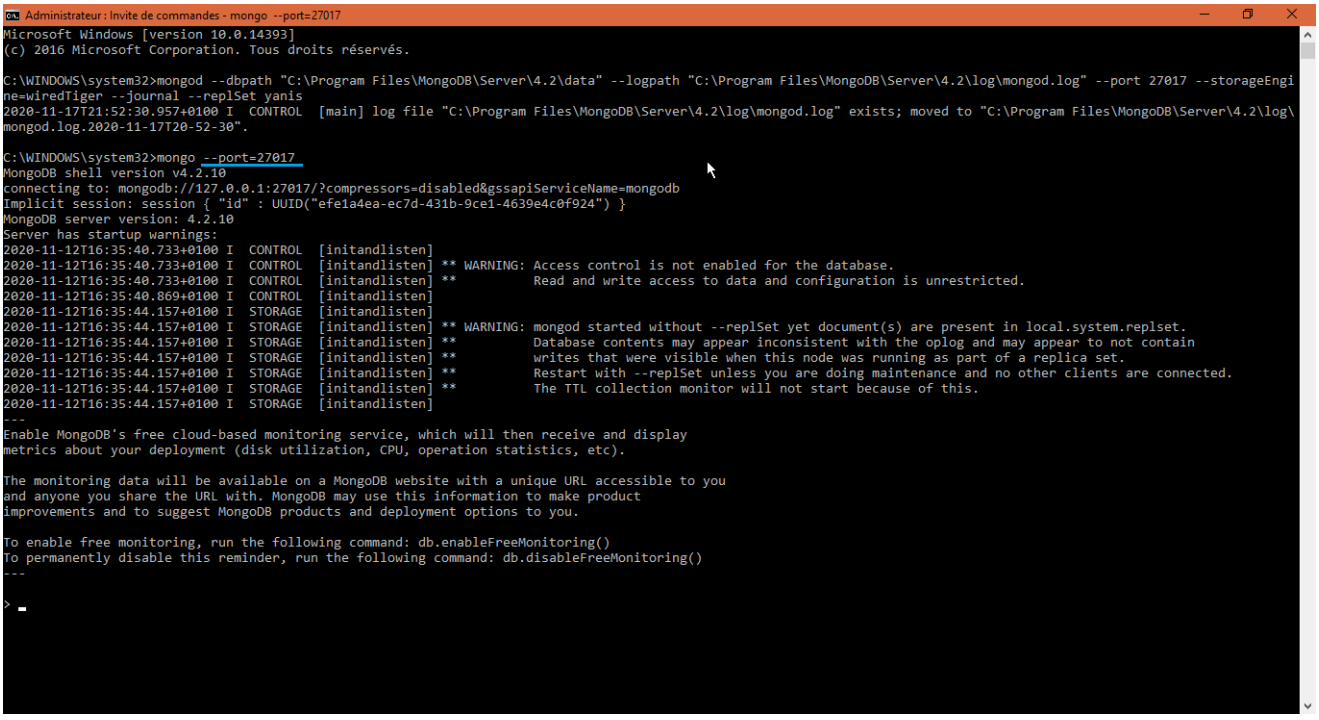
1. Démarrez le serveur autonome avec le commande :

```
mongod --dbpath "C:\Program Files\MongoDB\Server\4.2\data" --logpath
"C:\Program Files\MongoDB\Server\4.2\log\mongod.log" --port 27017
--storageEngine=wiredTiger --journal --replSet yanis
```

2. On se connecte au serveur avec le numéro de port 27017 .

```
mongo  --port = 27017
```

aprés la confirmation de cette commande la connexion est effectuée avec succès au serveur primaire



3. une fois connecter on doit créer la variable rsconf.

```
rsconf={_id:"yanis",members:[{_id:0,host:"localhost:27017"}]}
rs.initiate(rsconf)
```

rs.initiate(configuration) : Lance un jeu de réplicas. Facultativement, la méthode peut prendre un argument sous la forme d'un document contenant la configuration d'un jeu de réplicas.

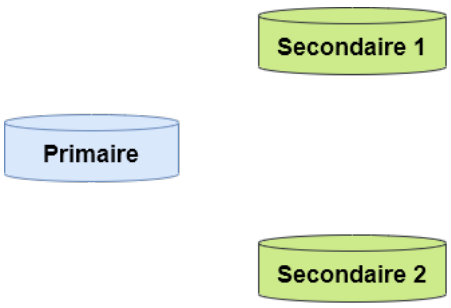


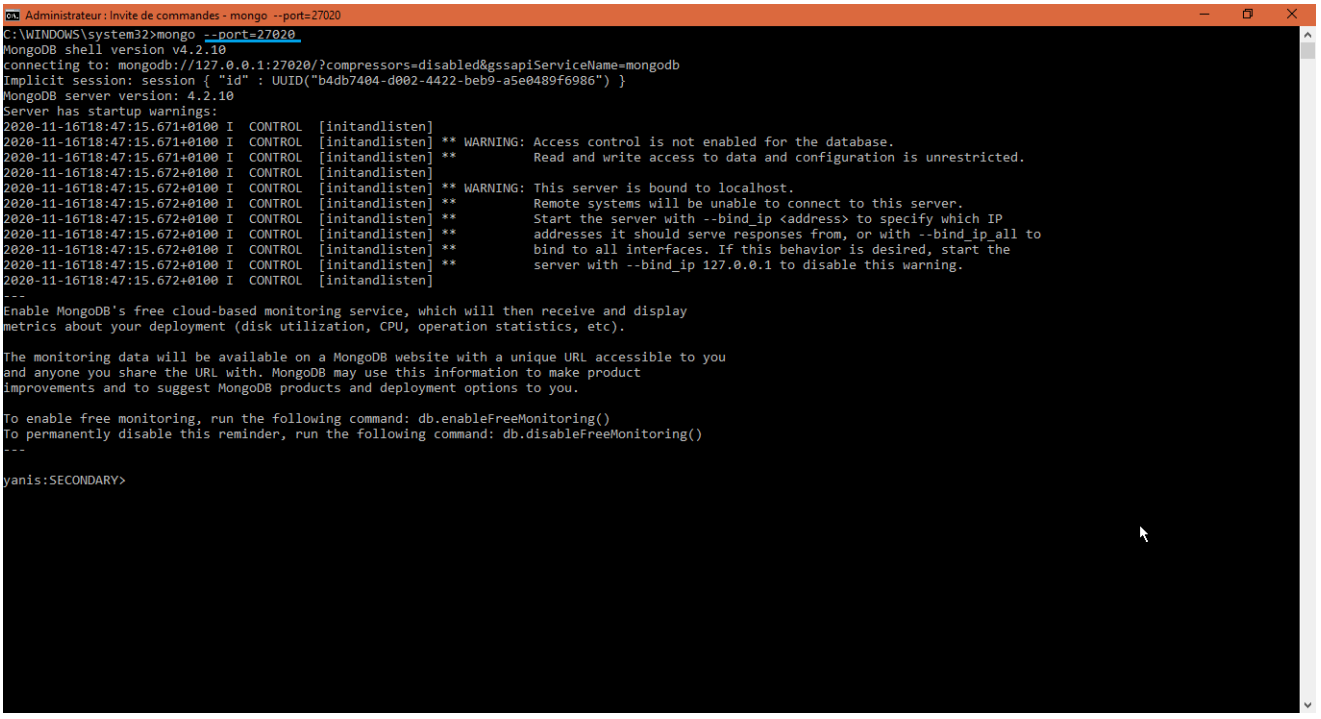
FIGURE 7.1 – Initialisation du jeu de réplicas

4. On démarre le serveur secondaire 1 sur le port 27020.

```
mongod --dbpath "C:\data1\db" --logpath "C:\data1\log\mongod.log"
--port 27020 --storageEngine=wiredTiger --journal --replSet yanis
```

5. On se connecte au premier serveur secondaire avec la commande

```
mongo  --port = 27020
```

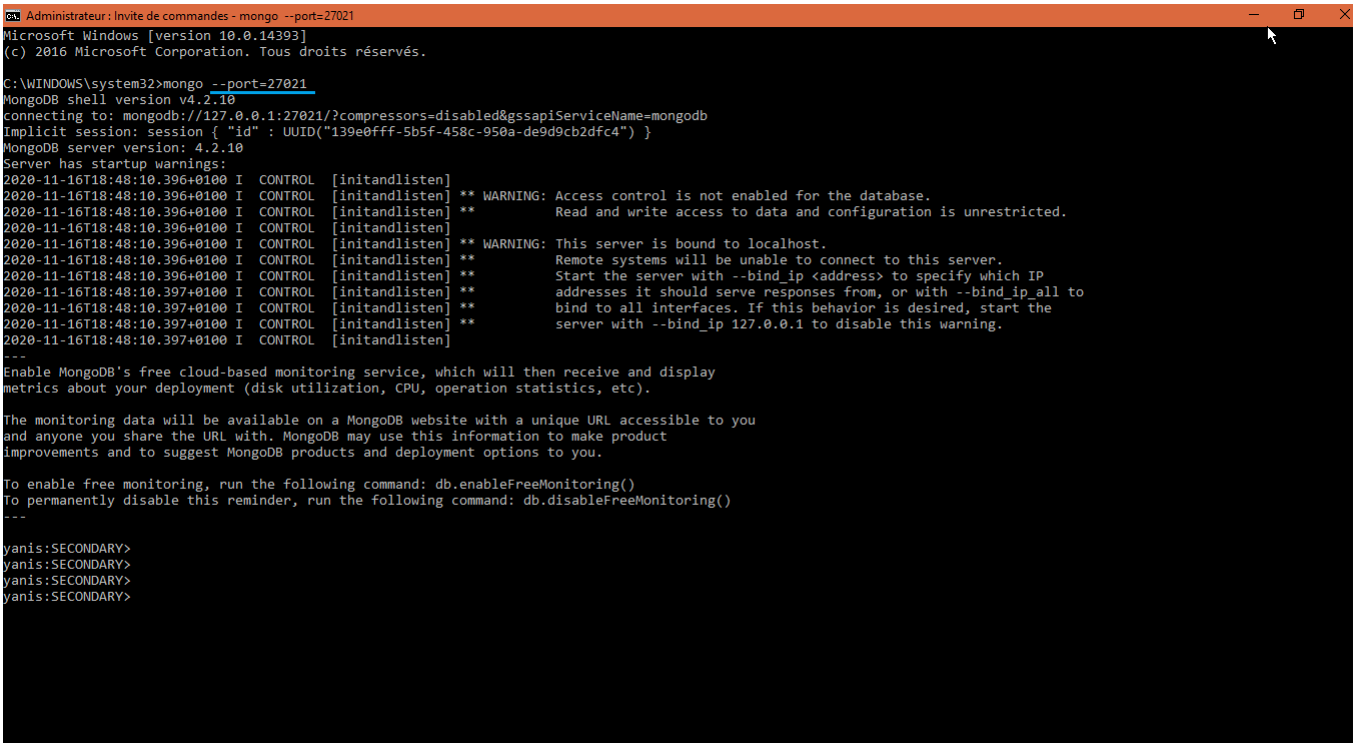


6. Ensuite on démarre le deuxième serveur secondaire sur le port 27021.

```
mongod --dbpath "C:\data2\db" --logpath "C:\data2\log\mongod.log"
--port 27021 --storageEngine=wiredTiger --journal --replSet yanis
```

7. On se connecte au deuxième serveur secondaire.

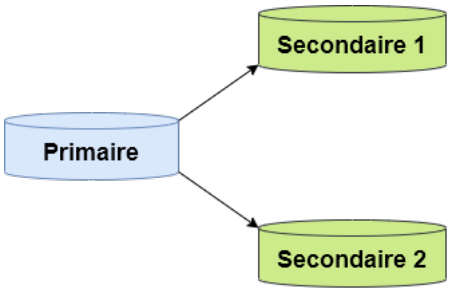
```
mongo  --port = 27021
```



8. On exécute les commandes suivantes sur le serveur principal.

```
rs.add("localhost:27020")
rs.add("localhost:27021")
```

rs.add(hôte) : Ajoute un membre à un jeu de réplicas



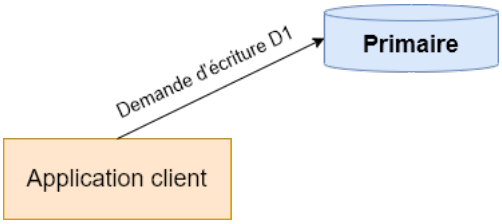
9. Maintenant, nous allons sur les serveurs secondaires et exécutons cette commande sur les deux serveurs secondaires.

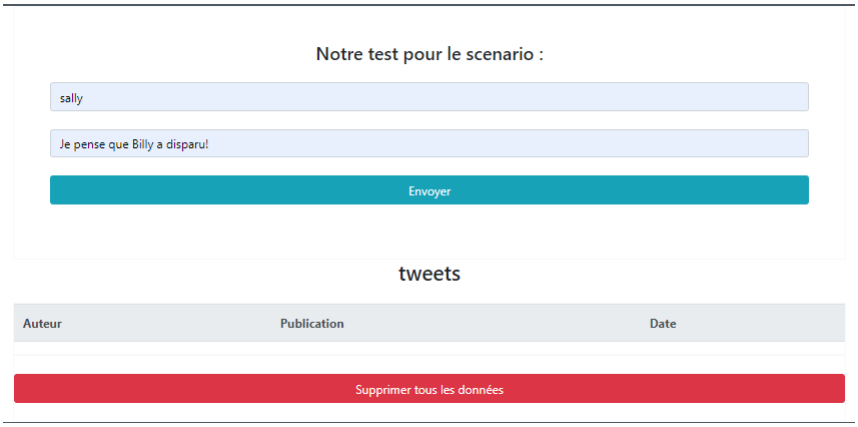
```
rs.slaveOk()
```

Donc dans cette partie nous avons expliqué comment configurer la réplication de MongoDB sur Windows.

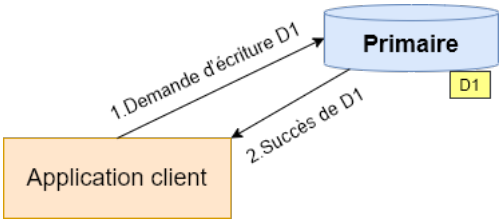
Etape 3 : Simulation de scénario

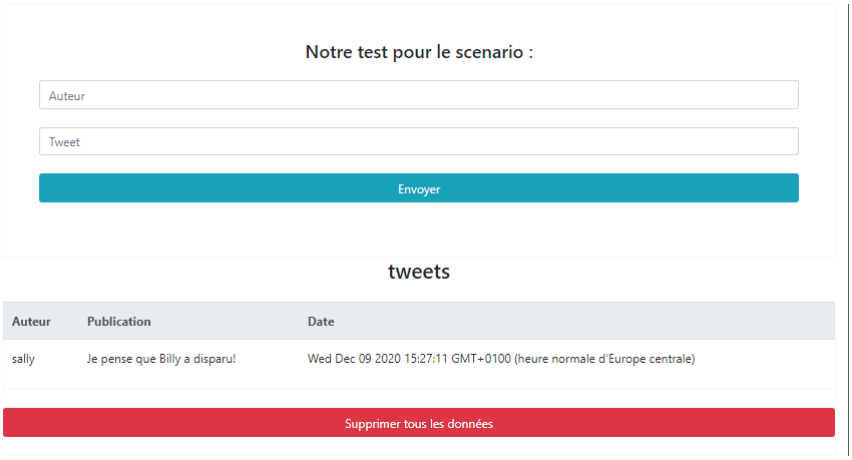
1. Le client fait une demande d'écriture sur le serveur principale.



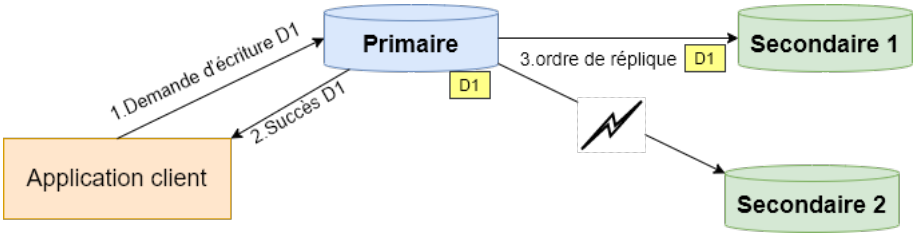


2. Le serveur principale répond qu'il appliqué avec succès l'écriture.



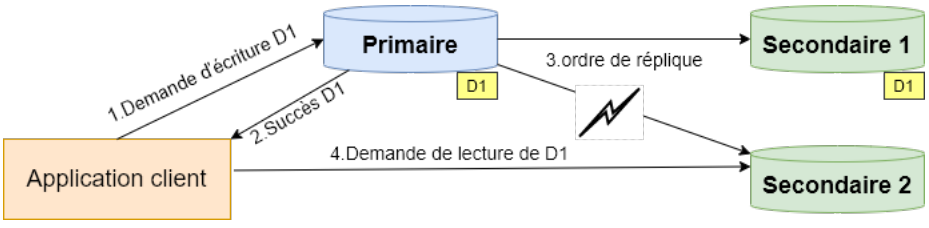


3. Le serveur principale fais un ordre de réplication pour les deux serveurs secondaire au même moment il y eut une coupure de réseaux avec le serveur secondaire 2.



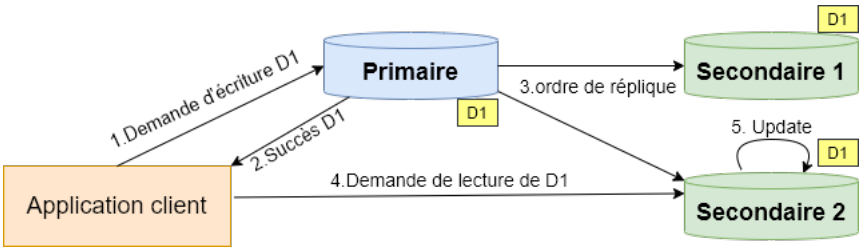

```
C:\Users\21356>mongo --port 27021
MongoDB shell version v4.2.3
connecting to: mongodb://127.0.0.1:27021/?compressors=disabled&gssapiServiceName=mongodb
2020-12-09T15:23:39.096+0100 E  QUERY    [js] Error: couldn't connect to server 127.0.0.1:27021, connection attempt fail
ed: SocketException: Error connecting to 127.0.0.1:27021 :: caused by :: Aucune connexion n□a pu □tre □tablie car l□ordi
nateur cible l□a express□ment refus□e. :
connect@src/mongo/shell/mongo.js:341:17
@(connect):2:6
2020-12-09T15:23:39.108+0100 F  -        [main] exception: connect failed
2020-12-09T15:23:39.108+0100 E  -        [main] exiting with code 1

C:\Users\21356>
```

4. Le client envoi une demande de lecture de la donnée saisi eu serveur secondaire 2.



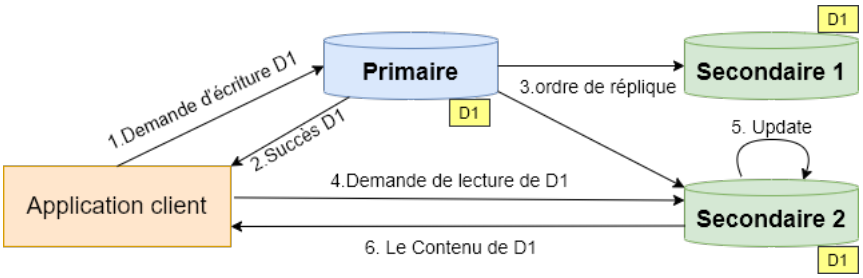

```
const url = 'mongodb://localhost:27021/?replicaSet=madjid'
```

5. Le client devra attendre jusqu'à ce que la connexion vers le serveur secondaire 2 sera rétabli.
6. Une fois la connexion rétabli, le serveur secondaire 2 vas faire une mise à jours de tous les changement des données on ce basent sur celle de primaire et de serveur secondaire 1.




```
Invite de commandes - mongo  --port 27021
C:\Users\21356>mongo  --port 27021
MongoDB shell version v4.2.3
connecting to: mongodb://127.0.0.1:27021/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("4e20139f-c7ec-44a2-b0d8-9e01eee1d5a3") }
MongoDB server version: 4.2.3
Server has startup warnings:
2020-12-09T15:24:26.097+0100 I  CONTROL  [initandlisten]
2020-12-09T15:24:26.098+0100 I  CONTROL  [initandlisten] ** WARNING: Access control is not enabled for
2020-12-09T15:24:26.098+0100 I  CONTROL  [initandlisten] **          Read and write access to data and
unrestricted.
2020-12-09T15:24:26.098+0100 I  CONTROL  [initandlisten]
2020-12-09T15:24:26.098+0100 I  CONTROL  [initandlisten] ** WARNING: This server is bound to localhost
2020-12-09T15:24:26.098+0100 I  CONTROL  [initandlisten] **          Remote systems will be unable to
rver.
2020-12-09T15:24:26.099+0100 I  CONTROL  [initandlisten] **          Start the server with --bind_ip <
y which IP
2020-12-09T15:24:26.099+0100 I  CONTROL  [initandlisten] **          addresses it should serve respons
-bind_ip_all to
2020-12-09T15:24:26.099+0100 I  CONTROL  [initandlisten] **          bind to all interfaces. If this b
, start the
2020-12-09T15:24:26.099+0100 I  CONTROL  [initandlisten] **          server with --bind_ip 127.0.0.1 t
ning.
2020-12-09T15:24:26.099+0100 I  CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.
```

7. Une fois que le serveur est à jours il peut rependre à la demande de client.




```
Invite de commandes - mongo  --port 27021
madjid:SECONDARY> use tweet
switched to db tweet
madjid:SECONDARY> db.publication.find().pretty()
{
        "_id" : ObjectId("5fd0ddba075f905cd8eeeb3b"),
        "nom" : "sally",
        "message" : "Je pense que Billy a disparu!",
        "poster" : ISODate("2020-12-09T14:22:50.655Z")
}
madjid:SECONDARY>
```

7.2 Interface d'application

7.2.1 Interface d'application sans données

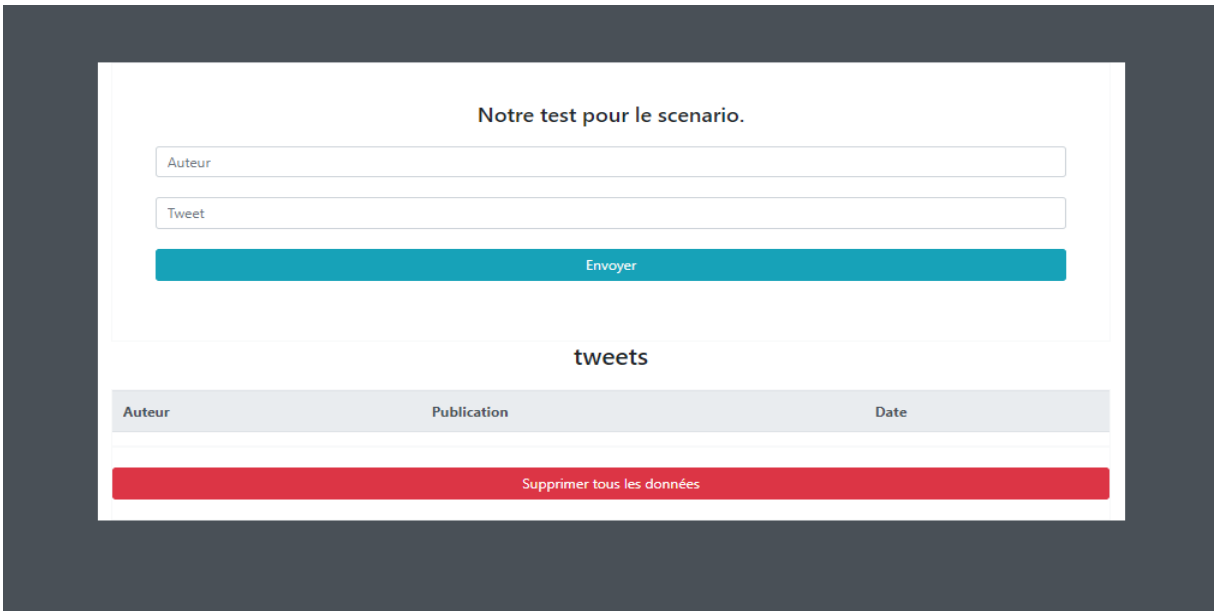


FIGURE 7.2 – Application sans implémentation

7.2.2 Interface d'application avec des données

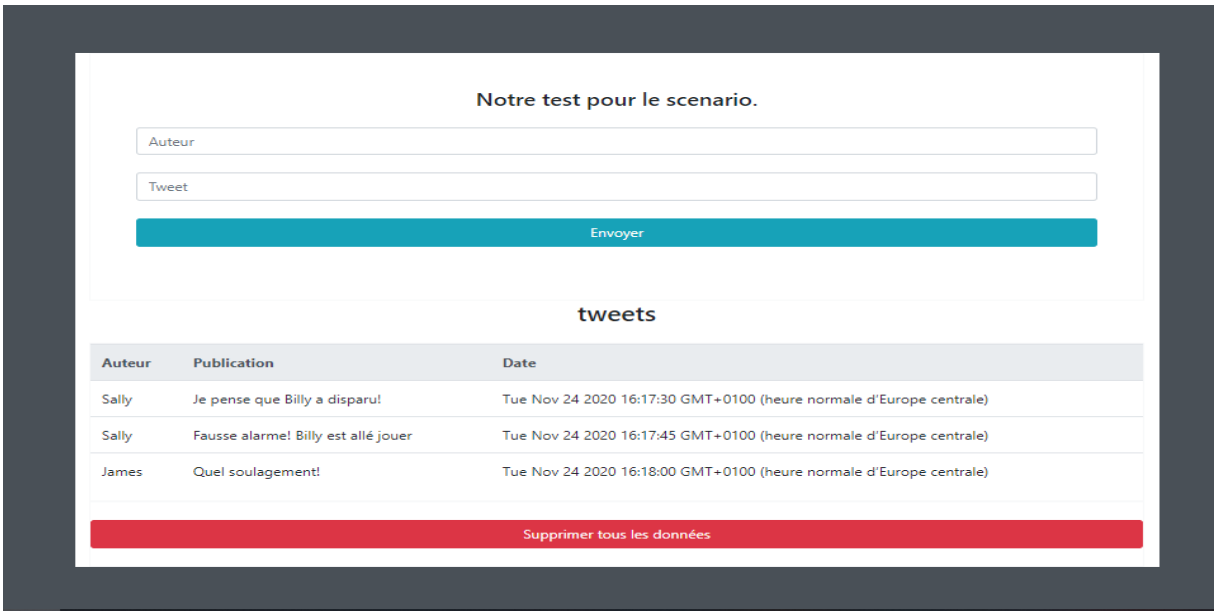


FIGURE 7.3 – Application avec implémentation

Conclusion générale

Au terme de ce travail de recherche bibliographique avec son aspect pratique, nous pouvons dire que nous en avons tiré plusieurs conclusions.

En effet, sur le plan théorique, notre formation a été extrêmement enrichie par un flot de connaissances d'une part globales et approfondies sur le BIG DATA, Cloud, bases de données NoSQL et d'autre parts pointues sur le théorème de CAP et ses conséquences et en particulier la problématique disponibilité et cohérence des données dans les bases de données NoSQL.

Dans un objectif pédagogique d'étude et d'analyse, ce travail nous a conduit à faire une récolte des différents modèles de cohérences auxquels sont arrivés les récentes recherches pour assurer l'équilibre disponibilité et cohérence des données dans les bases de données NoSQL. A l'issue de cette recherche, nous avons tenté de synthétiser ces différentes et nombreuses et connaissances acquises afin d'en dresser une vue panoramique et une vision globale de cette thématique.

Sur le plan technique, deux éléments principaux sont à retenir. Le premier d'entre eux est le modèle de données de Mango DB lui-même. Habitués aux bases de données relationnelles, on a trouvé intéressant de voir les différences qu'apporte Mango DB dans la structure du modèle de données. Ensuite, il était intéressant de voir l'architecture de déploiement distribuées Réplica Set qui nous assure la disponibilité et la cohérence dans les différents serveurs répliqués. En effet, la façon dont il manipule la base de données est plus en phase avec les diverses théories enseignées. Le dernier point technique est la modélisation par la dénormalisation. Etant donné que Mango DB est un système de base de données qui rejette la notion de jointure, l'approche utilisée pour les bases de données relationnelles n'était tout simplement pas possible. Habitué aux systèmes relationnels et à la normalisation des données dont le but est d'éviter la redondance au maximum, cette approche a été au départ déstabilisant mais très instructive, et nous savons dorénavant qu'il n'y pas qu'une seule façon de modéliser le schéma d'une base de données. L'une des questions à laquelle il fallait répondre dans le cadre de ce travail pratique était de savoir si une base de données NoSQL, telle que Mango DB, était adaptée au genre d'application que nous voulions réaliser pour garantir la cohérence. Mango DB (ainsi que les autres bases de données NoSQL), tire son principal avantage de son architecture distribuée, permettant de répartir la charge de travail entre plusieurs machines. Ici, nous avons tiré un grand profit de cet atout, car l'application est destinée à ce type d'architecture.

La deuxième raison est l'aspect réplicatif et cohérence des données. En effet, l'utilisation de Mango DB requiert la duplication des données dans les différentes répliques, ce qui augmente les risques d'incohérences entre les répliques. Par exemple, imaginons qu'un client ajoute un produit a son panier. Dans un système relationnel, la modification ne s'appliquera que sur la table « panier », alors que, dans le système Mango DB, il faudra appliquer le changement sur la collection client et ajouter le produit dans la colonne d'achat et la répliquer dans tous les serveurs. Toutes ces duplications de données entraînent aussi une consommation supplémentaire de l'espace en disque.

Dans un but de concrétisation, nous avons réussi à travers un scénario à mettre en évidence l'impact de la cohérence dans les bases de données NoSQL. Nous avons opté pour le modèle de cohérence causale qui correspond aux intuitions des programmeurs sur le temps et qui est plus disponible que les modèles de cohérence forte, puisque le temps et l'ordre sont si fondamentaux à notre intuition qu'il est difficile de raisonner sur un système qui n'impose pas la cohérence causale.

Nous concluons aussi que les bases de données NoSQL ne sont pas forcément meilleures que les bases relationnelles, et vice-versa et le choix des modèles de cohérence à mettre en place pour faire face à ce problème de cohérence-disponibilité est donc très important et doit être fait très soigneusement.

Bibliographie et Webographie

[1] Greg Ridgeway. Policing in the era of big data. 2018.

[2] Seagate. https://www.seagate.com/fr/fr/our-story/data-age-2025/, November 2018.

[3] FuturaSciences. https://www.futura-sciences.com/tech/definitions/informatique-big-data-15028/, August 2019.

[4] MongoDB. https://www.mongodb.com/big-data-explained, June 2020.

[5] Google Cloud. https://cloud.google.com/what-is-big-data, July 2020.

[6] Alexandros Labrinidis and Hosagrahar V Jagadish. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5(12) :2032–2033, 2012.

[7] Seagate. https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf, November 2018.

[8] Omnicore. https://www.omnicoreagency.com/facebook-statistics/, October 2020.

[9] Dan Noyes. https://zephoria.com/top-15-valuable-facebook-statistics/, October 2020.

[10] Le datascientist. https://le-datascientist.fr/les-10-v-du-big-data, November 2019.

[11] NAVEEN JOSHI. Top 5 sources of big data, November 2017.

[12] FUTURA TECH. https://www.futura-sciences.com/tech/definitions/internet-internet-objets-15158/.

[13] Saagie. https://www.saagie.com/fr/blog/qu-est-ce-que-le-big-data-definition/, December 2016.

[14] Salvador García, Sergio Ramírez-Gallego, Julián Luengo, José Manuel Benítez, and Francisco Herrera. Big data preprocessing: methods and prospects. *Big Data Analytics*, 1(1) :9, 2016.

[15] Alain Fernandez. Business Intelligence, le décisionnel d'entreprise, February 2020.

[16] Bastien L. MapReduce : tout savoir sur le framework Hadoop de traitement Big Data, October 2020.

[17] Bastien L. Cloud Computing – Définition, Avantages et Exemples d'utilisation, February 2017.

[18] M^{elle} BERKANI Nassima M^{elle} MOUSSAOUI Salima. Mémoire de Master professionelle en Informatique Option Administration et sécurité des réseaux La sécurité des données dans le Cloud Computing. Master's thesis, Université A/Mira de Béjaïa Faculté des Sciences Exactes Département d'Informatique, 2015/2016.

[19] Helene Touzet. Moyens de calcul - Cloud bilille, March 2020.

[20] Robson A Campêlo, Marco A Casanova, Dorgival O Guedes, and Alberto HF Laender. A brief survey on replica consistency in cloud environments. *Journal of Internet Services and Applications*, 11(1) :1–13, 2020.

[21] Ivision. https://www.ivision.fr/cloud-et-virtualisation-les-differences/, June 2014.

[22] Bastien L. Définition Data Center : qu'est-ce qu'un centre de données ?, April 2017.

[23] RedHat. https://www.redhat.com/fr/topics/data-storage/file-block-object-storage.

[24] Oracle. https://blog.bronto.com/ch-fr/database/what-is-a-relational-database/.

[25] Mickaël Martin-Nevot. Utilisation des bases de donnees, 2018.

[26] Wassila Guendouzi. introduction aux bases de données, December 2019.

[27] IONOS. https://www.hebergementwebs.com/base-de-donnees-relationnelle-l-essentiel-a-connaitre/article/68347/, May 2019.

[28] IONOS. https://www.ionos.fr/digitalguide/hebergement/aspects-techniques/base-de-donnees-relationnelle/#c165472, May 2019.

[29] IONOS. https://www.ionos.fr/digitalguide/hebergement/aspects-techniques/nosql/, April 2020.

[30] Lionel HEINRICH. Architecture NoSQL et réponse au Théorème CAP. Master's thesis, Haute École de Gestion de Genève, February 2014.

[31] Jing Han, Ee Haihong, Guan Le, and Jian Du. Survey on NoSQL database. In *2011 6th international conference on pervasive computing and applications*, pages 363–366. IEEE, 2011.

[32] Nicolas Travers Régis Behmo. Choisissez votre famille NoSQL, November 2019.

[33] Arnaud COSTES. Une base NoSQL: Cassandra, 2019.

[34] IllustraData. https://www.illustradata.com/bases-nosql-documents-quest-cest/, April 2017.

[35] Vishnu Gopy. TOP 10 – MEILLEURES BASES DE DONNÉES NOSQL 2021, October 2020.

[36] Hadi Hashem. Modélisation intégratrice du traitement BigData, October 2016.

[37] Dirk deRoos. ACID versus BASE Data Stores, July 2009.

[38] Chaimae Asaad, Karim Baïna, and Mounir Ghogho. NoSQL Databases: Yearning for Disambiguation. *arXiv preprint arXiv :2003.04074*, 2020.

[39] Sébastien Monnet. *Contributions à la réplication de données dans les systèmes distribués à grande échelle.* PhD thesis, 2015.

[40] Harshita Pandey. Data Replication in DBMS, August 2019.

[41] Roselin BILEY. Mise en place d'un système de réplication de base de données entre sites distants. Master's thesis, Université de Dschang, 2009.

[42] Tehreem Naeem. https://www.astera.com/fr/type/blog/data-replication/, November 2020.

[43] Loïc Cudennec. Modèles et protocoles de cohérencedes données en environnement volatil. Master's thesis, Universite de Renne 1, June 2005.

[44] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Computing Surveys (CSUR)*, 37(1) :42–81, 2005.

[45] colostate. https://www.cs.colostate.edu/ cs551/CourseNotes/Consistency/TypesConsistency.html, January 2003.

[46] Hesam Nejati Sharif Aldin, Hossein Deldari, Mohammad Hossein Moattar, and Mostafa Razavi Ghods. Consistency models in distributed systems: a survey on definitions, disciplines, challenges and applications. *arXiv preprint arXiv :1902.03305*, 2019.

[47] COHESITY. https://www.cohesity.com/blogs/strict-vs-eventual-consistency/, October 2017.

[48] Safae DAHMANI. *Modèles et protocoles de cohérence de données, décisionet optimisation à la compilation pour des architecturesmassivement parallèles.* PhD thesis, École doctorale SICMA, May 2016.

[49] GMU. https://cs.gmu.edu/ setia/cs707/slides/consistency.pdf.

[50] David Mosberger. Memory consistency models. *ACM SIGOPS Operating Systems Review*, 27(1) :18–26, 1993.

[51] Miguel Diogo, Bruno Cabral, and Jorge Bernardino. Consistency Models of NoSQL Databases. *Future Internet*, 11(2) :43, 2019.

[52] Peter Bailis, Shivaram Venkataraman, Michael J Franklin, Joseph M Hellerstein, and Ion Stoica. Probabilistically bounded staleness for practical partial quorums. *arXiv preprint arXiv :1204.6082*, 2012.

[53] Brian F Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. PNUTS: Yahoo!'s hosted data serving platform. *Proceedings of the VLDB Endowment*, 1(2) :1277–1288, 2008.

[54] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3) :1–22, 2013.

[55] Jiaqing Du, Sameh Elnikety, and Willy Zwaenepoel. Clock-SI: Snapshot isolation for partitioned data stores using loosely synchronized clocks. In *2013 IEEE 32nd International Symposium on Reliable Distributed Systems*, pages 173–184. IEEE, 2013.

[56] Tudor-Ioan Salomie. *Cloud-ready scalable and elastic data processing using off-the-shelf databases, replication and virtualization*. PhD thesis, ETH Zurich, 2014.

[57] Haonan Lu, Kaushik Veeraraghavan, Philippe Ajoux, Jim Hunt, Yee Jiun Song, Wendy Tobagus, Sanjeev Kumar, and Wyatt Lloyd. Existential consistency: measuring and understanding consistency at Facebook. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 295–310, 2015.

[58] Houssem-Eddine Chihoub, Shadi Ibrahim, Gabriel Antoniu, and Maria S Perez. Harmony: Towards automated self-adaptive consistency in cloud storage. In *2012 IEEE International Conference on Cluster Computing*, pages 293–301. IEEE, 2012.

[59] Shraddha P Phansalkar and Ajay R Dani. Tunable consistency guarantees of selective data consistency model. *Journal of Cloud Computing*, 4(1) :1–12, 2015.

[60] Sérgio Esteves, Joao Silva, and Luís Veiga. Quality-of-service for consistency of data geo-replication in cloud computing. In *European Conference on Parallel Processing*, pages 285–297. Springer, 2012.

[61] Douglas B Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K Aguilera, and Hussam Abu-Libdeh. Consistency-based service level agreements for cloud storage. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 309–324, 2013.

[62] Valter Balegas, Sérgio Duarte, Carla Ferreira, Rodrigo Rodrigues, Nuno Preguiça, Mahsa Najafzadeh, and Marc Shapiro. Putting consistency back into eventual consistency. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 1–16, 2015.

[63] Diego Didona, Rachid Guerraoui, Jingjing Wang, and Willy Zwaenepoel. Causal consistency and latency optimality: friend or foe? *arXiv preprint arXiv :1803.04237*, 2018.

[64] Tao Chen, Rami Bahsoon, and Abdel-Rahman H Tawil. Scalable service-oriented replication with flexible consistency guarantee in the cloud. *Information Sciences*, 264 :349–370, 2014.

[65] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall,

and Werner Vogels. Dynamo: amazon's highly available key-value store. *ACM SIGOPS operating systems review*, 41(6) :205–220, 2007.

[66] Marcus Brandenburger, Christian Cachin, and Nikola Knežević. Don't trust the cloud, verify: Integrity and consistency for cloud object stores. *ACM Transactions on Privacy and Security (TOPS)*, 20(3) :1–30, 2017.

[67] Raghul Mukundan, Sanjay Madria, Mark Linderman, and NY Rome. Replicated Data Integrity Verification in Cloud. *IEEE Data Eng. Bull.*, 35(4) :55–64, 2012.

[68] Qin Liu, Guojun Wang, and Jie Wu. Consistency as a service: Auditing cloud consistency. *IEEE Transactions on Network and Service Management*, 11(1) :25–35, 2014.

[69] Daniel Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *Computer*, 45(2) :37–42, 2012.

[70] Mustaque Ahamad, Gil Neiger, James E Burns, Prince Kohli, and Phillip W Hutto. Causal memory: Definitions, implementation, and programming. *Distributed Computing*, 9(1) :37–49, 1995.

[71] Leslie Lamport and Clocks Time. the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7) :558, 1976.

[72] Colin J Fidge. Timestamps in message-passing systems that preserve the partial ordering. *Communications of the ACM*, 1987.

[73] Wyatt Lloyd, Michael J Freedman, Michael Kaminsky, and David G Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 401–416, 2011.

[74] MongoDB. https://www.mongodb.com/fr/what-is-mongodb, 2020.

[75] Quick Programming Tips. MongoDB History, 2020.

[76] MongoDB. https://docs.mongodb.com/manual/replication/.

[77] MongoDB. https://docs.mongodb.com/manual/sharding/.

[78] MongoDB. https://docs.mongodb.com/manual/core/gridfs/.

[79] Vitam. https://www.programmevitam.fr/ressources/DocCourante, August 2020.

[80] Visual Code Studio. https://code.visualstudio.com/docs.

[81] OpenJS Foundation. https://nodejs.org/en/about/.