République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de BOUIRA
Faculté des sciences et des Sciences appliquées Département d'Informatique

Mémoire de fin d'études
Pour l'obtention du diplôme master en informatique

Option : Ingénierie des Systèmes d'information et du Logiciel

Théme

La Cohérence Dans les Systèmes NOSQL

Réalisé par :

- BELLILI Mahmoud
- OUAMRANE Malika

Devant le jury composé de :

- Mr BENNOUAR DJAMEL (Président)
- Mr BAL KAMEL (Examinateur)
- Mr DJOUABRI ABDEREZZAK (Examinateur)
- Mme MAHFOUD ZOHRA (Promotrice)

2021/2022

Dédicace

Je ´dédie ce mémoire
tout d'abord à :

Mes chers parents qui m'ont apportès tout le confort,

À mon épouse CHETTABI EL YAZID qui m'a encouragé `à aller jusqu'au bout.

A tous mes enfants Zakaria, Lina, Idris et Yagoub.

Mes chers frères, mes sœurs, ainsi que leurs petites familles. A toutes ma famille,

A mes amis pour leur encouragement et leurs soutiens.

A tous ceux qui m'ont aidé dans la réalisation de ce travail.

Malika

À mes précieux parents, À mes chers grands parents, À mes frères et mes chères sœurs,

À toute personne qui m'est chère,

Mahmoud

Remerciements

Nous tenons tout d'abord à remercier Dieu le tout puissant et miséricordieux, qui nous a donné la force et la patience et le courage d'accomplir ce Modeste travail.

Et c'est avec une immense reconnaissance que nous tenons à remercier notre encadreur Mme «mahfoud zohra» pour ses précieux conseils et sa disponibilité tout au long de la réalisation de ce travail, ainsi pour l'orientation, la confiance, l'aide et le temps qu'il a bien voulu nous consacrer.

On lui présente donc nos sentiments de gratitude.

Nos vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à notre recherche en acceptant d'examiner notre travail et de l'enrichir par leurs propositions.

Nos sincères remerciements à tous nos professeurs du département d'informatique de la faculté des sciences à bouira

Enfin, nous tenons également à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.

Résumé :

À l'émergence du Big Data, les applications traitent des quantités extrêmement élevées de donnés. Pour assurer leur disponibilité, ces applications utilisent la réplication des données. Cependant, la réplication introduit le problème important de la cohérence des données.

La gestion de la cohérence est primordiale. Les modèles à cohérence forte induisent des coûts importants en termes de performance du fait qu'ils exigent des requêtes de toujours retourner la valeur la plus récente des données. A l'inverse, les modèles à cohérence faible fournissent de meilleures performances ainsi qu'une meilleure disponibilité des données. Toutefois, ces derniers modèles peuvent tolérer des incohérences.

Dans le cadre de ce projet de fin d'études, nous abordons les nombreux mécanismes utilisés par les SGBD Distribués afin de garantir le partitionnement, la réplication et la cohérence des données. Après la présentation de l'état de l'art sur les différents modèles de cohérences, nous nous sommes intéressés à la notion de cohérence faible au forte.

Notre contribution consiste à mettre en œuvre une solution qui permet d'implémenter le modèle de cohérence sur un système d'enchères qui fera office de cas d'utilisation pratique pour tester les performances de ce modèle en termes de latence et débit. Cette solution se compose d'un module de gestion de cohérence qui se charge de vérifier pour chaque requête sur les données, le niveau de cohérence à appliquer selon le besoin ; etd'un SIAD (Système d'Informations d'Aide à la Décision) qui alimente le gestionnaire decohérence en informations sur l'état des enchères. Enfin, nous clôturons ce travail par uneétude comparative des performances d'un modèle de cohérence fort avec le modèle de cohérence faible.

Mots-Clés : BDD distribuées, BDD No-SQL, Big Data, Cohérence, Transactions.

Abstarct :

In the face of the explosion of massive data production, relational DBMSs have been rapidly limited by major web players such as Google, Facebook and Amazon These

limits have led these major web players to develop and adopt new alternative technologies such as NoSQL and the emergence of the term Big Data. These new technologies come to meet the requirements of management, processing, analysis of these gigantic data

To ensure their availability, these applications use data replication. However, replication introduces the important problem of data consistency.

Consistency management is key. Strongly consistent models incur significant performance costs because they require queries to always return the most recent value of the data. Conversely, weak consistency models provide better performance as well as better data availability. However, these latter models can tolerate inconsistencies.

As part of this end-of-studies project, we address the many mechanisms used by Distributed DBMSs to guarantee data partitioning, replication and consistency. After the presentation of the state of the art on the different coherence models, we are interested in the notion of weak to strong coherence.

Keywords : Distributed Databases, No-SQL Databases, Big Data, Consistency, Tran- sactions.

Table des matières

Introduction générale : 1
Contexte : 1
Contribution : 1
Définition du problème : 1
Objectif, exigences et réalisations attendues : 2
Plan de ce mémoire: 2
1.1 Introduction: 4
1.1.1 Réplication: 4
1.1.2 Placement des réplicas: 4
1.2 Gestion des réplicas: 6
1.3 Mise à jour des copies: 6
1.4 Modes de Réplication: 6
1.4.1 Réplication asymétrique: 7
1.4.2 Réplication symétrique: 8
1.5 Convergence des copies: 9
1.6 Cohérence: 10
1.6.1 Modèles de cohérence: 11
1.6.2 Modèles de cohérence forte: 12
1.6.3 Modèles de cohérence faible: 12
1.7 Conclusion 14
2.1 Introduction: 15
2.2 Apache Cassandra: 15
2.2.1 Description : 15
2.2.2 Caractéristiques principales de Cassandra [Girolamo. 2013] 15
2.2.3 Architecture d'un cluster Cassandra: 15
2.2.3.1 clusters Cassandra: 15
2.2.3.2 Data center Cassandra: 15
2.2.3.3 Rack Cassandra: 15
2.2.3.4 Serveur Cassandra: 15
2.2.3.5 Nœuds virtuels 15
2.2.4 Modèle de données pour Cassandra 15
2.2.4.1 Les colonnes: 15
2.2.4.2 Les super colonnes: 15

2.2.4.3 Les familles de colonnes: 15
2.2.4.4 Les super familles de colonnes: 15
2.2.5 Lecture / Ecriture de données: 15
2.2.5.1 Processus d'écriture de Cassandra: 15
2.2.5.2 Processus de lecture de Cassandra: 15
2.2.6 Partitions de données: 15
2.2.7 Réplication des données dans Cassandra: 15
2.2.8 Niveau de cohérence dans Cassandra: 15
2.2.8.1 En cas d'écriture: 15
2.2.8.2 En cas de lecture: 15
2.2.9.1 Types de données CQL : 1
2.3 Conclusion 1
3.1 Introduction 40
3.2 Qu'est-ce-que YCSB ? 40
3.3 Architecture de YCSB 40
3.4 Les Workloads du benchmark YCSB : 40
3.5 Niveaux de références de YCSB 40
3.6 Conclusion 40
4.1 Introduction 44
4.2 Technologies utilisées 44
4.2.1 Benchmark YCSB 44
4.2.3 SGBDs utilisés 44
4.3 Solution détaillée 44
4.3.1 Architecture Technique 44
4.3.2 Phase de prétraitements 44
4.3.3 Phase de traitements (Tests) 44
4.3.4 Phase post-traitements 44
4.4 Configuration de l'environnement 44
4.4.1 Ressources Techniques 44
4.4.2 Configuration de Cassandra 44
4.4.3 Configuration du benchmark 44
4.5 Déploiement 44
4.6 Conclusion 44
5.1 Introduction 54

5.2 Plan de tests 54
5.2.1 Niveau de cohérence 54
5.3 Résultats et discussions 54
5.3.1 Configuration 1 55
5.3.2 Configuration 2 : 56
5.3.3 Configuration 3 : 58
5.3.4 Configuration 4 : 60
5.3.5 Configuration 5 : 62
5.3.6 Configuration 6 : 64
5.3.7 Comparaison : 65
5.4 Conclusion 67
Conclusion générale : 68
Perspectives 69

Liste des figures

Figure 1.1 Réplication asymétrique synchrone 7
Figure 1.2 Réplication asymétrique asynchrone..... 7
Figure 1.3 Réplication symétrique synchrone 8
Figure 1.4 Réplication symétrique asynchrone 8
Figure 1.5 Réalisation d'une transaction 11

Figure 2.1 Architecture d'un cluster Cassandra 16
Figure 2.2 Un cluster Cassandra composé de six Data Centers 17
Figure 2 3 Deux Data Centers composés de deux Racks chacun 17
Figure 2.4 Data Centers composés de deux Racks 18
Figure 2.5 Chaque nœud physique du cluster possède quatre Nœuds vetuels 19
Figure 2.6 Processus d'écriture 26
Figure 2.7 processus de lecture 27
Figure 2.8 Schéma d'un cluster à quatre nœuds 28

Figure 3.1 Architecture de YCSB 40

Figure 4.1 Architecture Technique de la solution 45
Figure 4 2 Vérification de l'environnement d'exécution 46
Figure 4.3 Détail technique sur serveur1 49
Figure 4.4 Détail technique sur serveur2 50
Figure 4.5 Détail technique sur switch 50
Figure 4.6Configuration des nœuds seeds 50
Figure 4.7 Configuration de l'adresse RPC 51
Figure 4.8 Adresse de liaison 51
Figure 4.9 Diagramme de déploiement 52

Figure 5.1 Configuration 1 en latence(Read) 55
Figure 5.2 Configuration 1 en latence (Write) 55
Figure 5.3 Configuration 1 en débit 56
Figure 5.4 Configuration 2 en latence(Read) 57
Figure 5.5 Configuration 2 en latence(Write) 57
Figure 5.6 Configuration 2 en débit 58
Figure 5.7 Configuration 3 en latence(Read) 59
Figure 5.8 Configuration 3 en latence(Write) 59
Figure 5.9 Configuration 3 en débit 60
Figure 5.10 Configuration 4 en latence(Read) 61
Figure 5.11 Configuration 4 en latence(Write) 61
Figure 5.12 Configuration 4 en débit 62
Figure 5.13 Configuration 5 en latence(Read) 62

Figure 5.14 Configuration 5 en latence(Write) 63
Figure 5.15 Configuration 5 en débit 63
Figure 5.16 Configuration 6 en latence(Read) 64
Figure 5.17 Configuration 6 en latence (Write) 64
Figure 5.18 Configuration 6 en débit 65
Figure 5.19 Comparaison en latence (read)..... 65
Figure 5.20 Comparaison en latence (write) 66
Figure 5.21 Comparaison en débit..... 66

Liste des tableaux

Table 2.1 Niveaux de cohérence en écriture sous Cassandra 15
Table 2.2 Niveaux de cohérence en lecture sous Cassandra 15

Table 3.1 Workloads de YCSB 40

Table 4.1 Liste des materiels et logiciels..... 49

Liste des abréviations

ACID	**A**tomicity **C**onsistency **I**solation **D**urability
BASE	**B**asically **A**vailable **S**oft state **E**ventual consistency
BDD	**B**ase **D**e **D**onneées
BDDD	**B**ase **D**e **D**onneées **Distribuées**
CAP	**C**onsistency **A**vailability **P**artition tolerence
CQL	**C**assandra **Q**uery **L**anguage
JSON	**J**ava**S**cript **O**bject **N**otation
NoSQL	**N**ot **O**nly **S**tructured **Q**uery **L**anguage
RAM	**R**andom Acces **A**ccess **M**emory
SGBD	**S**ystème **D**e **G**estion de **B**ase de **D**onnées
SGBDD	**S**ystème **D**e **G**estion de **B**ase de **D**onnées **D**istribués
SGBDR	**S**ystème **D**e **G**estion de **B**ase de **D**onnées
RIAD	**R**elationnel **S**ystème d'**I**nformations d'**A**ide à la **D**écision
SPOF	**S**ingle **P**oint **O**f **F**ailure
SQL	**S**tructured **Q**uery **L**anguage
YCSB	**Y**ahoo **C**loud **S**erving **B**enchmark

Introduction générale :

Contexte :

Les bases de données relationnelles ont été développées comme une technologie pour stocker des données structurées et organisées sous forme de tableaux. Aux cours des années elles sont devenues l'élément essentiel des organisations et le modèle de référence pour la gestion des données des systèmes d'informations, cependant, avec l'augmentation continue des données stockées et analysées, les bases de données relationnelles commencent à présenter une variété de limitations.

C'est dans ce contexte que les bases de données NoSQL étaient développées pour fournir un ensemble de nouvelles fonctionnalités de gestion des données tout en surmontant certaines limites de bases de données relationnelles.

Pour assurer la disponibilité de ces grosses quantités de données, les Big Data utilisent la réplication des données, ce qui requière une gestion des copies multiples afin d'assurer leur cohérence. Différents niveaux de cohérence sont définis dans la littérature. La cohérence forte garantie que toutes les copies soient identiques mais cette solutionreste très coûteuse à mettre en œuvre dans les systèmes à grande échelle. Contrairement à la cohérence forte. La cohérence faible quant à elle, est tolérée par d'autres applications telles que les réseaux sociaux où un retard de quelques secondes dans la visualisation des mises à jour des données ne pose pas de problème. La cohérence faible est souhaitable puisqu'elle offre une meilleure disponibilité contrairementà la cohérence forte.

Contribution :

Ce projet couvrira le sujet de la cohérence et les copies multiples dans un contexte big data et qui pourrait voir son application sur des systèmes tels que les Web Shops, les et les exigences qui doivent Être satisfaits afin d'obtenir de bonnes performances dans l'accès à l'information, la définition du problème qui sera également abordée, Discuter de l'objectif de la recherche, des exigences, de la technologie qui sera utilisée et Également des réalisations attendues.

Définition du problème :

Face à ces grandes quantités de données, les systèmes possédants différentes catégoriesde données pouvant être manipulées parfois en cohérence faible et parfois en cohérence forte font généralement recours à l'utilisation d'un seul niveau de cohérence afin d'arriverà satisfaire les requêtes émanant des clients, cette solution permet en effet de garantir uniquement la cohérence des données ou leur disponibilité (théorème CAP).

Dans cette optique, apporter un moyen d'utiliser des niveaux de cohérences différents sur les données et selon le besoin est devenu une nécessité. C'est pourquoi notre travail consiste à apporter une réponse quant à l'apport que peut apporter l'utilisation d'une cohérence faible ou forte permettant la manipulation des données avec la cohérence nécessaire au moment nécessaire. Nous essayons de répondre donc à la question des performances d'un tel niveau de cohérence sur un système réel dans un contexte Big Data**.**

Objectif, exigences et réalisations attendues :

Dans ce travail, on a examiné différents concepts introduits dans la littérature pour le traitement parallèle du volume de données et effectuerons une comparaison entre eux afin d'identifier quel concept est le meilleur en ce qui concerne le volume différent, le type différent et la vitesse différente de données.

L'exigence est d'avoir un serveur Cassandra avec un stockage suffisant et suffisamment de données.

Nous espérons identifier quelle serait la meilleure architecture pour des problèmes particuliers Selon le type de données, le volume de données ainsi que le temps de réponse requis pour le temps réel analyse.

Pour l'atteindre nous devrions :

- Maîtriser des outils de simulations des systèmes transactionnels pour les Big Data.
- Configurer sur un cluster réel une infrastructure qui supporte un système distribué pour les Big Data.
- Se familiariser avec les concepts liés aux BDDs Distribuées, notamment la cohérence.
- Effectuer des simulations sur des données de l'ordre des Big Data, en effectuant des chargements de données, des tests et des analyses sur les résultats obtenus.

Dans ce chapitre, nous présentons un aperçu de certaines informations de base, y compris définition de Big Data, Architectures du Big Data et le principe et les fonctionnalités des bases de données NoSQL.

Plan de ce mémoire:

Ce mémoire est organisé en deux parties contenant chacune un certain nombre de chapitres, qui se présentent comme suit :

Partie théorique : Est consacrée à la présentation de certaines notions de base relativesaux Big Data ainsi que les systèmes NOSQL.

Chapitre 1 : la cohérence et les copies multiple : les différents concepts de réplication et de gestion de la cohérence des répliques.

Dans la partie expérimentale : cette partie est composée de trois chapitres :

Chapitre 2 : Apache Cassandra : nous allons présenter une fiche descriptive de la solution étudiées.

Chapitre 3 : Framework YCSB : on a présenté un benchmark performant qui sert de moyen pour tester les niveaux de cohérence de ce SGBD.

Chapitre 4 : tests des résultats et configuration **:** Dans ce chapitre, nous effectuons les tests de cohérences sur la BDD distribuée pour enfin aboutir à une étude comparative des performances de chacun des niveaux de cohérence sur la base d'une discussiondes graphes obtenus comme résultat de chaque test.et Enfin nous clôturons notre thèse par une conclusion générale et quelques perspectives à notre travail.

Chapitre 1: la cohérence et les copies multiples

1.1 Introduction:

La réplication est une des importantes solutions utilisées dans les systèmes répartis, dans le but d'optimiser les performances en termes de disponibilité et de temps d'accès aux données. En effet, la disponibilité des données permet d'accéder aux données répliquées même en cas de défaillance d'un site. De plus, en plaçant une copie des données près de leur point d'utilisation, les performances d'accès sont alors optimisées.

Toutefois lorsqu'une copie est modifiée, la modification doit être propagée à toutes les autres copies du système afin de garantir leur cohérence. Cependant, le maintien de la cohérence a un coût. Il faut faire un compromis entre le coût et la qualité de la cohérence.

Ils existent plusieurs modèles de cohérence dans la littérature, nous distinguons deux grandes familles de cohérence: les modèles de cohérence forte et les modèles de cohérence faible ou relâchée.

La cohérence forte garantie que toutes les copies soit identiques mais cette solution reste très coûteuse à mettre en œuvre dans les systèmes à grande échelle. Contrairement à la cohérence forte, les modèles de la cohérence relâchée tolèrent une certaine incohérence entre les réplicas pour une petite durée afin d'alléger le système en terme de performance d'accès.

Nous présentons dans ce chapitre les aspects de base de la réplication puis nous abordons les modèles de cohérence et les travaux connexes qui existent dans la littérature.

1.1.1 Réplication:

La réplication des données [Meylan 2005] [Exbrayat 2007] est très utilisée dans les systèmes distribués pour augmenter la disponibilité des données et en conséquence optimiser les performances d'accès. Cette solution consiste à créer des copies d'une donnée et de les stocker dans des sites différents. Les requêtes de lectures/ écriture sont alors exécutées sur le site disposant de la copie la plus proche du client, ce qui diminue le coût de transfert aux données. Quand à la disponibilité des données, la réplication permet de ne plus dépendre d'un site central unique, et utiliser une copie sur un site lorsqu'elle est indisponible sur un autre.

La mise en place de la réplication des données dans les grilles n'est pas une mince tâche, à cause notamment de la forte dynamicité des nœuds de cet environnement. En effet, afin de pouvoir gérer la réplication des données de manière dynamique, trois questions fondamentales doivent être posées:

1. A quel moment les réplicas doivent être créés ?
2. Quels fichiers doivent être répliqués ?
3. Où seront placés les réplicas?

1.1.2 Placement des réplicas:

Plusieurs stratégies de placement des réplicas ont été proposées dans la littérature [Rasool et al. 2007] [Ranganathan et Foster 2001] [Park et al. 2003] [Liu et Wu 2006] [Rahman et al. 2007].

Les performances de ces stratégies sont souvent évaluées en temps de réponse moyen et en bande passante consommée. Par exemple dans [Ranganathan et Foster 2001] les auteurs étudient plusieurs stratégies dynamiques de réplication dans une grille à topologie hiérarchique. Nous citons dans ce qui suit les stratégies considérées:

- **Aucune** réplicas**:** l'ensemble de données est disponible à la racine de la hiérarchie.
- **Meilleur client:** chaque nœud maintient un historique détaillé pour chaque fichier qu'il détient et les nœuds qui le sollicitent et à quelle fréquence. A un moment donné chaque nœud vérifie si le nombre de requêtes sur l'un des fichiers qu'il détient a dépassé un certain seuil, si oui le meilleur client pour ce fichier est identifié et une copie est créée au niveau de ce nœud.
- **réplicas en cascade:** une fois le seuil, pour un fichier, est dépassé à la racine, une réplicas est créée au prochain niveau, mais sur le chemin au meilleur client. Par conséquent le nouveau emplacement pour la réplicas est un ancêtre du meilleur client. Une fois le seuil pour le fichier est dépassé au niveau 2, il sera alors répliqué à la prochaine rangée inférieure et ainsi de suite.
- **Mise en antémémoire simple:** Étant donné que les fichiers sont volumineux, et un client n'a suffisamment d'espace que pour stocker un seul fichier à la fois, les fichiers sont remplacés rapidement.
- **Mise en antémémoire et réplication en cascade:** Cette stratégie combine les stratégies 3 et 4. Le client sauvegarde les fichiers localement. Le serveur identifie régulièrement les fichiers populaires et les propage vers le bas de la hiérarchie.
- **diffusion rapide:** Dans cette méthode une réplicas d"un dossier est stockée à chaque nœud tout au long de son chemin au meilleur client.

Ces stratégies sont comparées selon trois modes d"accès aux données comme suit:

- **P-Random:** mode d'accès aléatoire.
  - **P1:** les données qui contiennent un degré de localité temporelle, c.à.d. les fichiers récemment consultés sont susceptibles d'être à nouveau consultés.
  - **P2:** les données contenant un degré de localité temporelle et géographique. Cette dernière veut dire que les fichiers récemment consultés par un client sont susceptibles d'être consultés par les clients à proximité.

Bien que très avantageuse, la réplication présente tout de même un inconvénient majeur de cohérence des données où toutes les copies d'une même donnée doivent être identiques. Dans un système à large échelle, tel que sur les cloud computing, il arrive que des copies d'une même donnée ne soient pas identiques. Pour cela des mécanismes doivent être mis en place pour assurer que les copies qui divergent à un moment donné, finissent toutes par converger vers le même état.

De plus, la propagation des mises à jour d'une copie vers toutes les autres copies peut avoir parfois un impact négatif sur les performances du système, celle-ci ne doit pas altérer de façon excessive le temps de réponse aux requêtes des utilisateurs.

1.2 Gestion des réplicas:

Dans ce qui suit nous présenterons quelques caractéristiques concernant la gestion des réplicas dans les systèmes distribués, tel que la mise à jour des copies, les modes de réplication et la convergence des copies répliquées.

1.3 Mise à jour des copies:

Le problème majeur de la réplication est posé lors de la mise à jour des copies. Une diffusion des mises à jour est appliquée d'une copie aux autres copies, cette diffusion est assurée par le système réparti. Plusieurs techniques de diffusion sont possibles, il y a celles basées sur la diffusion de l'opération de la mise à jour, et celles basées sur la diffusion du résultat de l'opération. Diffuser le résultat présente l'avantage de ne pas ré-exécuter l'opération sur les sites de la copie, mais l'inconvénient est de nécessiter un ordonnancement identique des mises à jour sur tous les sites afin d'éviter la perte des mises à jour.

La mise à jour synchrone est le mode de distribution dans lequel toutes les mises à jour locales sont effectuées suite à une mise à jour globale. Dans le contexte des copies, ce mode de distribution est très utile lorsque les mises à jour effectuées sur un site doivent être prises en compte immédiatement sur les autres sites. L'avantage de ce type de mise à jour est de garder toutes les données au dernier niveau de la mise à jour. Le système peut alors garantir de fournir la dernière version des données quel que soit la copie accédée. Cependant ce type de mise à jour reste très coûteux en ressources.

La mise à jour asynchrone représente le mode de distribution dans lequel certaines mises à jour locales effectuées suite à une mise à jour globale sont accomplies dans des transactions indépendantes, en temps différé. Le temps des mises à jour des copies peut être plus ou moins contrôlé. L'avantage de ce type de mise à jour est la possibilité de mettre à jour en temps choisi des données, tout en autorisant l'accès aux versions anciennes avant la mise à jour. L'inconvénient est que l'accès à la dernière version n'est pas garanti.

1.4 Modes de Réplication:

Lors du traitement des données répliquées sur un système distribué, en fonction de la propagation des mises à jour et du traitement des copies des données, on distingue deux modes de réplication: symétrique et asymétrique.

Le premier est définit par la symétrie des comportements des copies d'une donnée répliquée. Chaque copie est traitée d'une façon identique aux autres, c'est le cas de la réplication symétrique. Le deuxième mode distingue deux comportements d'une donnée répliquée ; la copie primaire et les copies secondaires. La copie primaire est la seule à effectuer tous les traitements (lecture/écriture). Les copies secondaires, surveillent la copie primaire. En cas de défaillance de la copie primaire, une copie secondaire devient la nouvelle copie primaire, c'est le cas de la réplication asymétrique.

1.4.1 Réplication asymétrique:

La réplication asymétrique distingue un site maître chargé de centraliser les mises à jour, il est appelé aussi site primaire, il est le seul autorisé à mettre à jour les données et est chargé de diffuser les mises à jour aux autres copies dites secondaires (esclaves). Dans ce mode de réplication, le site primaire effectue les contrôles et garantit l'ordonnancement correct des mises à jour.

Le problème de la gestion de copie asymétrique est la panne du site maître. Dans ce cas, il faut choisir un remplaçant si l'on veut continuer les mises à jour. On aboutit alors à une technique asymétrique mobile dans laquelle le site maître change dynamiquement. Il existe deux types pour la réplication asymétrique: synchrone et asynchrone.

La réplication asymétrique synchrone utilise un site maître qui pousse les mises à jour en temps réel vers un ou plusieurs sites esclaves.

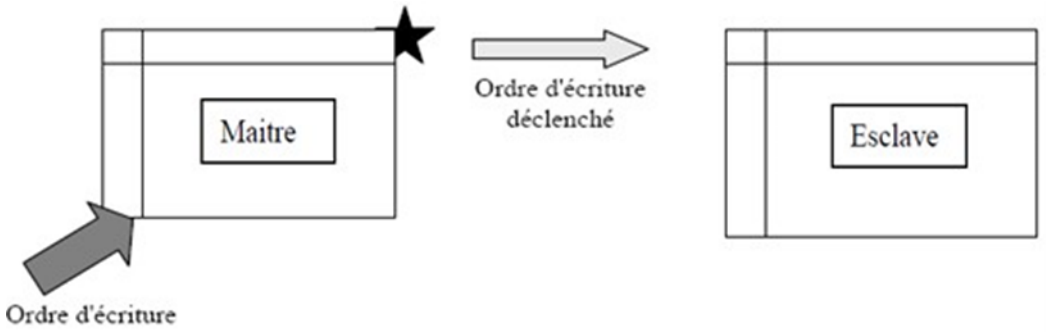


Figure 1.1 Réplication asymétrique synchrone

Dans le cas de la réplication asymétrique asynchrone les mises à jour sont poussées en temps différé via une file d'attente. Les mises à jour seront exécutées ultérieurement, à partir d'un déclencheur externe, une horloge par exemple.

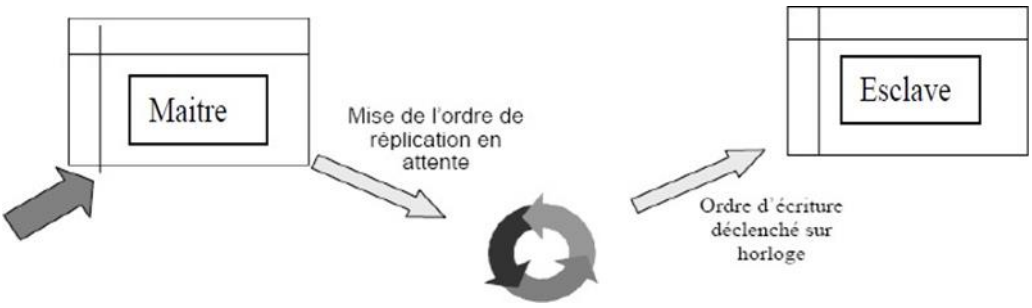


Figure 1.2 Réplication asymétrique asynchrone

1.4.2 Réplication symétrique:

A l' opposé de la réplication asymétrique, la réplication symétrique ne privilégie aucune copie, elle permet des mises à jour simultanées de toutes les copies par des transactions différentes.

Dans ce cas chaque copie peut être mise à jour à tout instant et assure la diffusion des mises à jour aux autres copies. Ce mode pose le problème de concurrence d'accès, en risquant de faire diverger les copies. Un algorithme global de résolution des conflits doit donc être mis en œuvre. Pour la réplication symétrique aussi on a deux modes: la réplication symétrique synchrone et asynchrone.

Dans la réplication symétrique synchrone, il n'y a pas de table maître et les accès aux écritures sont gérés de manière concurrente. Toutes les copies sont accessibles à la fois en lecture et en écriture. Chaque copie possède ses propres triggers qui lors de la réception d'une requête de mise à jour déclenchent le déploiement de cette dernière vers toutes les autres copies.

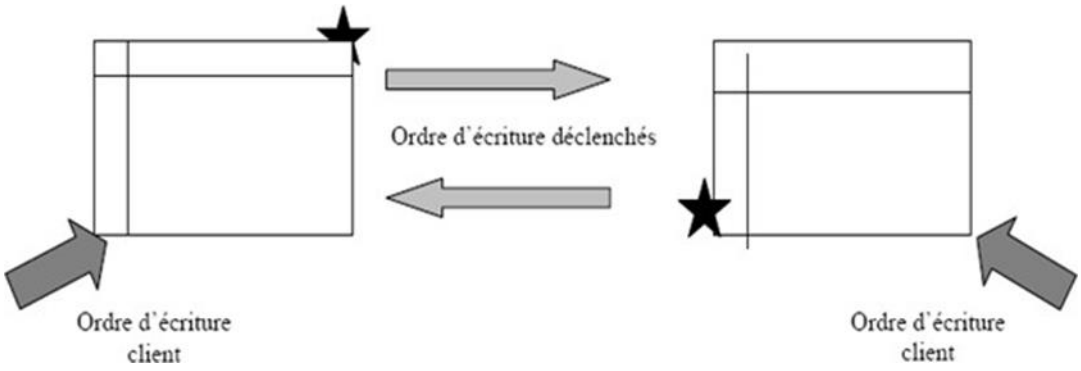


Figure 1.3 Réplication symétrique synchrone

Dans la réplication symétrique asynchrone, les mises à jour des copies sont mises dans une file d'attente pour être déclenchées ultérieurement. Cependant cette technique risque de provoquer des incohérences entre les données.

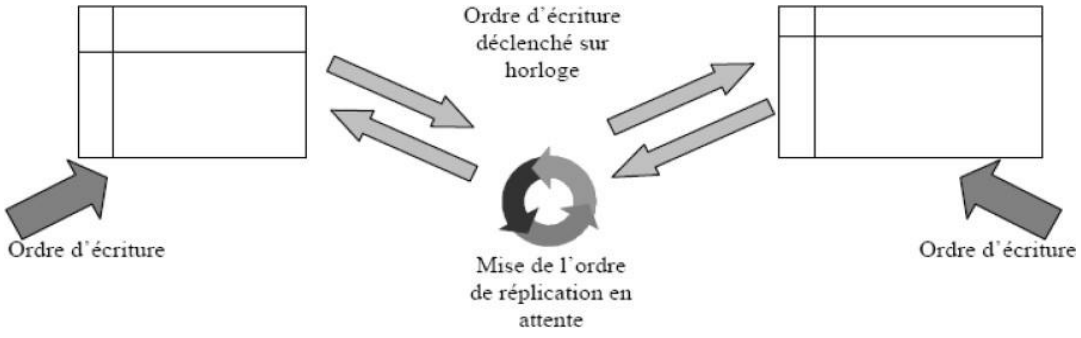


Figure 1.4 Réplication symétrique asynchrone

1.5 Convergence des copies:

La convergence des copies asymétriques est assurée par le site maître. Ce dernier règle les problèmes de concurrence et envoie les mises à jour dans l'ordre où il les applique aux sites secondaires. Encore faudrait-il que ces derniers exécutent les mises à jour dans le bon ordre, ce qui présente quelques difficultés [Moussa 2007].

Dans le cas des copies symétriques, la convergence est plus difficile à assurer, à cause des mises à jour simultanées qui risquent d'être accomplies dans un ordre différent sur une même donnée. On obtient alors des états différents, ce qui signifie que les copies divergent, cela est due au non-respect de la sérialisabilité de l'exécution globale des transactions. Une exécution d'un ensemble de transactions est dit sérialisable si et seulement si elle est équivalente à une exécution séquentielle (ou série) de transactions. Ce problème peut être résolu par des mécanismes de contrôle de concurrence, comme par exemple ceux présentés dans [Meylan 2005] et qui sont:

- ✓ **La prévention des conflits par verrouillage des copies:** Ce mécanisme consiste à demander à toute transaction mettant à jour, d'obtenir un verrou en écriture sur chacune des copies. Les risques de verrous mortels sont alors importants si plusieurs transactions mettent à jour simultanément plusieurs copies. Ces risques peuvent être prévenus en ordonnant l'ordre de visite des sites.
- ✓ **La détection des conflits par ordonnancement des mises à jour:** L'ordonnancement s'effectue par un mécanisme d'estampillage ou de certification. Le système doit alors utiliser des estampilles synchronisées pour ne pas favoriser un site plutôt qu'un autre. Ici les risques de transactions multiples sont importants.

En asymétrique il est impossible de défaire des transactions validées. Les différentes approches se contentent alors de détecter les conflits et de les reporter vers l'utilisateur qui doit mettre en œuvre une technique de résolution de conflits. On peut utiliser le mécanisme des estampilles décrit précédemment ou le mécanisme de vérification des valeurs avant mise à jour, cela consiste à utiliser la valeur avant mise à jour de l'objet afin de vérifier que la valeur de la copie correspond bien à celle sur laquelle est basée la mise à jour. Chaque message de mise à jour doit donc comporter la valeur avant mise à jour et la valeur après. Dans le cas où la valeur avant est différente de celle figurant dans la base, un conflit est détecté, ce qui veut dire qu'une autre transaction a effectué une mise à jour concurrente [Meylan 2005].

Après détection des conflits, il faut les résoudre, il s'agit de réconcilier les copies qui ont divergé. La réconciliation peut être purement syntaxique et ne pas prendre en compte la signification des données. Comme on peut aussi utiliser une solution asymétrique en choisissant un site prioritaire. La réconciliation peut faire appel à signification des données et des mises à jour, c'est le cas d'une procédure de réconciliation sémantique [Meylan 2005].

1.6 Cohérence:

Dans les systèmes distribués la cohérence des réplicas est d'assurer deux propriétés fondamentales, la première est qu'à n'importe quel moment toutes les copies d'une certaine donnée sont identiques, la deuxième est que n'importe quelle opération de lecture sur une donnée (ou une copie de donnée) doit retourner la dernière valeur écrite.

La cohérence est une des propriétés ACID (Atomicité Cohérence Isolation Durabilité) qui assure qu'une transaction fait passer une base de données d'un état cohérent à un autre état cohérent grâce au respect des contraintes d'intégrité. Une transaction est une séquence d'instructions, modifiant des données (incluant lectures, écritures) afin de garantir la cohérence des bases de données.

Le mécanisme de gestion des transactions doit assurer certaines propriétés afin de préserver la cohérence des bases de données lors de l'exécution des différentes transactions sur ces dernières

- **Atomicité d'une transaction:** La gestion de transaction nécessite la propriété d'atomicité d'une transaction c'est à dire d'assurer que les opérations mêmes les plus complexes englobées au sein d'une transaction ne soient perçues que comme une opération unique. De plus il faut que soit toutes les modifications liées à cette opération soient effectuées, soit aucune ne l'est. C'est le système qui est à même d'assurer l'atomicité des transactions. Comme une transaction peut ne pas se terminer normalement (à cause d'une panne par exemple) on définit donc les points: avant et après validation. Une transaction ayant atteint son point de validation sera dite validée (elle ne peut plus être remise en cause). Une transaction n'ayant pas encore atteint son point de validation sera dite vivant.
- **Permanence (durabilité):** Le problème de la permanence est de faire en sorte que lorsqu'une transaction a atteint son point de validation, les effets de la transaction soient conservés sur la base quelles que soient les circonstances. La solution est obtenue par l'intermédiaire de fichiers journaux qui conservent la trace des transactions successives qui ont été effectuées sur la base. Lors d'une transaction de mise à jour, la base de données passe d'un ancien état à un nouvel état et on conserve dans le journal, l'identification de la transaction, l'identification des éléments modifiés, leur ancienne valeur et leur nouvelle valeur (Figure 1.5.).

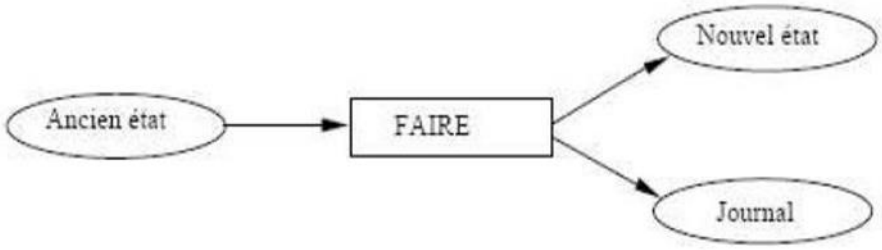


Figure 1.5 Réalisation d'une transaction

- **Isolation**: La base de données est découpée en éléments appelés granules (c'est en général l'administrateur en accord avec le concepteur qui fixe le niveau de granularité (n-uplet, page, table). L'isolation d'un élément dans une transaction est obtenue par le mécanisme des verrous (lock), Ceci peut engendrer deux grands types de problèmes:
  - ✓ Le premier est celui de la famine. Lorsqu'un verrou est relâché sur un élément A, le système choisit parmi les transactions candidates en attente. Si les transactions sont liées au niveau de priorité de l'utilisateur certaines d'entre elles risquent d'attendre longtemps.
  - ✓ Le deuxième type de problème concerne l'inter blocage. Cette situation se présente lorsqu'un ensemble de transactions attendent mutuellement le déverrouillage d'éléments actuellement verrouillés par des transactions de cet ensemble.
  - ✓ Le mécanisme de verrouillage permet donc d'isoler un élément pour une transaction, cependant il faut encore que cette utilisation soit faite à bon escient.
- **Sérialisabilité:** Une exécution d'un ensemble de transactions est sérialisable si et seulement si elle est équivalente à une exécution séquentielle (ou série) de transactions. Quand les transactions sont arbitraires, la sérialisabilité est la seule propriété qui assure le bon entrelacement. La sérialisabilité peut être obtenue en imposant aux transactions que tous les verrouillages précèdent tous les déverrouillages. Les transactions sont alors dites à deux phases: une phase d'acquisition des verrous puis une phase de libération.

1.6.1 Modèles de cohérence:

Les modèles de cohérence sont caractérisés par un ensemble de protocoles de gestion de la cohérence entre les réplicas des données. Nous distinguons deux grandes familles: les modèles à cohérence forte et les modèles à cohérence relâchée (ou faible).

Pour chaque modèle de cohérence des contraintes imposées aux programmeurs sont définies et les modèles de cohérence imposant aux applications l'utilisation d'opérations autres que les opérations d'accès aux données (comme des opérations de synchronisation) sont des modèles de cohérence relâchée, ils sont également appelés modèles de cohérence avec synchronisation, les autres sont des modèles de cohérence forte, ou modèles de cohérence sans synchronisation.

Le modèle de cohérence le plus contraignant est celui qui correspond exactement à la vue unique des données.

La mise en place de protocoles de la cohérence stricte dans un système réparti pose deux problèmes :

I) Définition de “l’évènement le plus récent” et II) la Réalisation “instantanée” des opérations. La définition de l’évènement le plus récent nécessite des horloges parfaitement synchronisées (écart nul entre deux sites quelconques). Une opération qui nécessite un accès distant ne peut être instantanée (plus précisément, une opération locale lancée après une opération distante peut se terminer avant)

La cohérence stricte est un modèle idéal (non réalisable), que l’on essaie d’approcher au moyen de modèles moins contraignants, à savoir des modèles de cohérence relâchée.

1.6.2 Modèles de cohérence forte:

Les modèles de cohérence forte [Monnet 2006] sont caractérisés par des contraintes fortes entre la dernière écriture et la prochaine lecture. Ils utilisent les seules primitives de lecture et d’écriture. Parmi les protocoles de cohérence forte, on peut citer les protocoles suivants:

- **Cohérence atomique:** C’est le modèle de cohérence le plus strict en terme de cohérence. Dans ce modèle, toute lecture retournera la dernière valeur qui a été écrite dans le système. La mise en œuvre d'un tel modèle s'avère cependant très coûteuse voir même irréalisable.
- **Cohérence séquentielle:** Formalisée par Leslie Lamport [Lamport 1979] ce modèle est moins restrictif, il garantit que tous les processus percevront les opérations dans le même ordre mais il ne garantit pas qu'une lecture retourne la dernière valeur affectée par une écriture. Avec ce modèle de cohérence, le résultat de toutes les exécutions est le même que si les opérations de tous les processus avaient été exécutées dans un ordre séquentiel donné dans lequel toutes les opérations de chaque processus suivent l'ordre du programme.

1.6.3 Modèles de cohérence faible:

Les modèles de cohérence faible [Monnet 2006] utilisent, en plus des primitives de lecture et d’écriture, des primitives manipulant des variables de synchronisation. En intégrant ces synchronisations avec les modèles de cohérence, il est possible de relâcher encore plus les contraintes de cohérence afin d’obtenir un protocole de cohérence moins coûteux et plus performant.

Parmi les protocoles de cohérence faible, on peut citer les protocoles suivants:

- **Cohérence causale**: Le modèle de cohérence causale [Hutto et Ahamad 1990] relâche les contraintes par rapport au modèle de cohérence séquentiel. Il n'est en effet pas toujours nécessaire d'obtenir un ordre total unique vu par chacun des processus. Ce modèle se base sur la relation de causalité (happened before) décrite dans [Lamport 1978] pour déterminer l'ordre des écritures. Un modèle de cohérence est dit causal s‟il garantit les deux conditions suivantes:
  - ✓ Les opérations d’écriture potentiellement causalement liées doivent être perçues par tous les processus dans le même ordre ;

- ✓ Les opérations d'écriture non causalement liées peuvent être perçues dans des ordres différents, sur des processus différents.

❖ **Cohérence PRAM:** Un modèle de cohérence est PRAM s'il garantit que les opérations d'écriture effectuées par un même processus sont perçues par tous les processus dans l'ordre où ces opérations ont été effectuées. Les opérations d'écriture effectuées par différents processus peuvent être perçues par chaque processus dans un ordre différent. Il permet un relâchement du modèle de cohérence causale: certaines transitivités dues à la relation de causalité ne sont pas considérées. C'est un modèle implantable sous forme de files d'opérations d'écriture.

❖ **Cohérence à la longue**: Un modèle de cohérence est à la longue s'il garantit que les opérations d'écriture seront propagées ultérieurement. C'est le modèle de cohérence le plus faible. Ce modèle n'impose aucun ordre sur les opérations.

❖ • **Cohérence faible:** Un modèle de cohérence est faible s'il garantit les trois propriétés suivantes:

- ✓ 1. Les opérations sur les variables de synchronisation se font selon un modèle de cohérence séquentielle ;
- ✓ 2. L'accès aux variables de synchronisation ne peut se terminer que si toutes les opérations d'écriture et de lecture sont terminées sur tous les sites;
- ✓ 3. Les opérations de lecture et d'écriture ne peuvent se faire que si toutes les opérations sur les variables de synchronisation sont terminées sur tous les sites.

Ce modèle garantie pour un processus: que toutes les modifications effectuées sont reçues par tous les processus et qu'il a reçu toutes les modifications faites par tous les processus, et il permet le regroupement des propagations.

❖ **Cohérence au relâchement**: Un modèle de cohérence est au relâchement s'il garantit les trois propriétés suivantes:

- ✓ Les opérations de lecture et d'écriture ne peuvent se faire que si toutes les opérations d'acquisition précédentes sont terminées sur tous les sites.
- ✓ L'opération de relâchement ne peut se terminer que si toutes les opérations d'écriture et de lecture sont terminées sur tous les sites.

Quand un processus effectue une opération de relâchement, les modifications qu'il a effectuées seront observées par tous les processus faisant une acquisition ultérieure. Il existe deux protocoles mettant en œuvre ce modèle. Le premier concerne la cohérence au relâchement impatiente où toutes les modifications entre les deux opérations de synchronisation sont propagées au moment du relâchement à tous les processus, même ceux qui n'effectueront pas d'acquisition ultérieure. Le deuxième concerne la Cohérence au relâchement paresseuse où toutes les modifications entre les deux opérations de synchronisation sont propagées au moment d'une acquisition par un autre processus. Le surcoût dû à ce protocole est compensé par le nombre et la taille des messages échangés.

❖ **Cohérence à l'entrée:** Basée sur l'association à chaque variable partagée un objet de synchronisation comme le verrou [Bershad et al. 1993].

Cette stratégie permet de ne transférer que des mises à jour en rapport avec la donnée, et non plus un morceau de mémoire. L'avantage est qu'une écriture interdit uniquement l'accès à la donnée et non pas l'accès à une zone mémoire où d'autres données peuvent être stockées. L'établissement de la relation entre donnée et verrou est cependant laissé à la charge du programmeur. La cohérence à l'entrée fait la distinction entre les accès en lecture et les accès en écriture grâce à l'utilisation de deux verrous. Si l'écrivain reste unique, les lecteurs sont multiples et peuvent donc lire en parallèle. Une lecture ne peut cependant pas avoir lieu en même temps qu'une écriture sur la même donnée.

- **Cohérence de portée:** Le modèle de cohérence à l'entrée [Iftode et al. 1996] est proche du modèle de cohérence à l'entrée. Le modèle de cohérence à l'entrée impose au programmeur d'associer explicitement un verrou à chaque donnée. Le modèle de cohérence de portée quant à lui utilise les opérations de synchronisation déjà présentes dans le programme. Dans ce cas, l'association verrou/donnée est réalisée de manière implicite par le programme.

1.7 Conclusion

Dans ce chapitre nous avons présenté les différents concepts de la réplication gestion de dans les systèmes distribués, ainsi nous avons discutés les deux types fondamentaux de la cohérence des copies multiples en présentant quelque modèle de chaque type.

Chapitre 2: Apache Cassandra

2.1 Introduction:

Pour stocker et traiter les volumes de données massifs du Big Data, les bases de données traditionnelles ne sont pas adaptées. Il est nécessaire d'utiliser de nouveaux outils. Parmi les solutions existantes, on compte Apache Cassandra**.**

Ce chapitre a pour objectif d'introduire un SGBD NoSQL « Apache Cassandra » en mettant l'accent sur ses différents niveaux de cohérences qui sont le cœur de notre travail. Par la suite nous présenterons un benchmark très performant « YCSB » qui se chargera de mesurer les différents niveaux de cohérences d'Apache Cassandra.

2.2 Apache Cassandra:

2.2.1 Description :

La base de données Cassandra est initialement développé par Facebook afin de répondre à des besoins concernant son service de messagerie, puis elle a été libérée en Open Source et a été adoptée par plusieurs autres grands du Web tel que Digg.com ou Twitter. Elle est aujourd'hui l'un des principaux projets de la fondation Apache, une organisation à but non lucratif développant des logiciels Open Source. Cassandra est un système de gestion de base de données NoSQL orientée colonne et écrit en java qui permet la gestion massive de données réparties sur plusieurs serveurs, assurant ainsi unehaute disponibilité des données.

2.2.2 Caractéristiques principales de Cassandra [Girolamo. 2013]

Voici une liste des principales caractéristiques de Cassandra :

Haute tolérance aux pannes : les données d'un nœud sont automatiquement répliquéessur différents nœuds. De cette manière, les données qu'il contient sont également disponibles sur d'autres nœuds si l'un des nœuds venait à être hors service. Conçu sur leprincipe qu'une panne n'est pas une exception mais une normalité, il est simple de remplacer un nœud qui tombe sans rendre le service indisponible **[Girolamo. 2013].**

Modèle de données riche : Cassandra propose un modèle de données basé sur BigTable (Google) de type clé-valeur. Elle permet de développer de nombreux cas d'utilisation dans l'univers Web **[Girolamo. 2013].**

Elastique : Que ce soit en écriture ou en lecture, les performances augmentent de façon linéaire lorsqu'un serveur est ajouté au cluster. Cassandra assure également la disponibilité du système et le non interruption au niveau des applications **[Girolamo. 2013].**

2.2.3 Architecture d'un cluster Cassandra:

Cassandra a été conçu en tenant compte du fait que les défaillances système/matérielles se produisent et se reproduisent. Pour cette raison, Cassandra a été développée comme un système distribué *peer−to−peer* où tous les nœuds remplissent les mêmes fonctions, ce qui signifie qu'il n'y a pas de point de défaillance unique.

L'une des plus grandes forces de Cassandra réside dans sa disponibilité et sa mise à l'échelle qu'elle atteint grâce à un système entièrement distribué dans lequel les données sont répliquées sur plusieurs nœuds en fonction des paramètres de l'utilisateur. **[Lakshman and Malik .2010]**

Cassandra regroupe un certain nombre de composants qui sont organisés et hiérarchisés de la manière suivante.

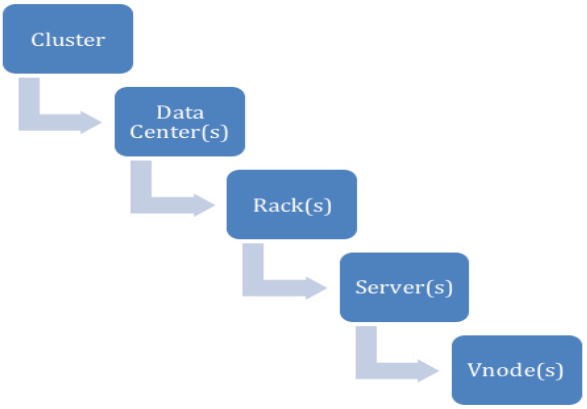


Figure 2.1 Architecture d'un cluster Cassandra

2.2.3.1 clusters Cassandra:

Dans le modèle Cassandra Data, la base de données Cassandra stocke les données via Cassandra Clusters. Les clusters sont essentiellement le conteneur le plus externe de la base de données Cassandra distribuée. La base de données est répartie sur plusieurs machines fonctionnant ensemble. Chaque machine agit comme un nœud et possède sa propre réplique en cas de panne. Ces nœuds sont disposés dans un format en anneau en tant que cluster. On illustre ça par la figure ci-dessous.[1]

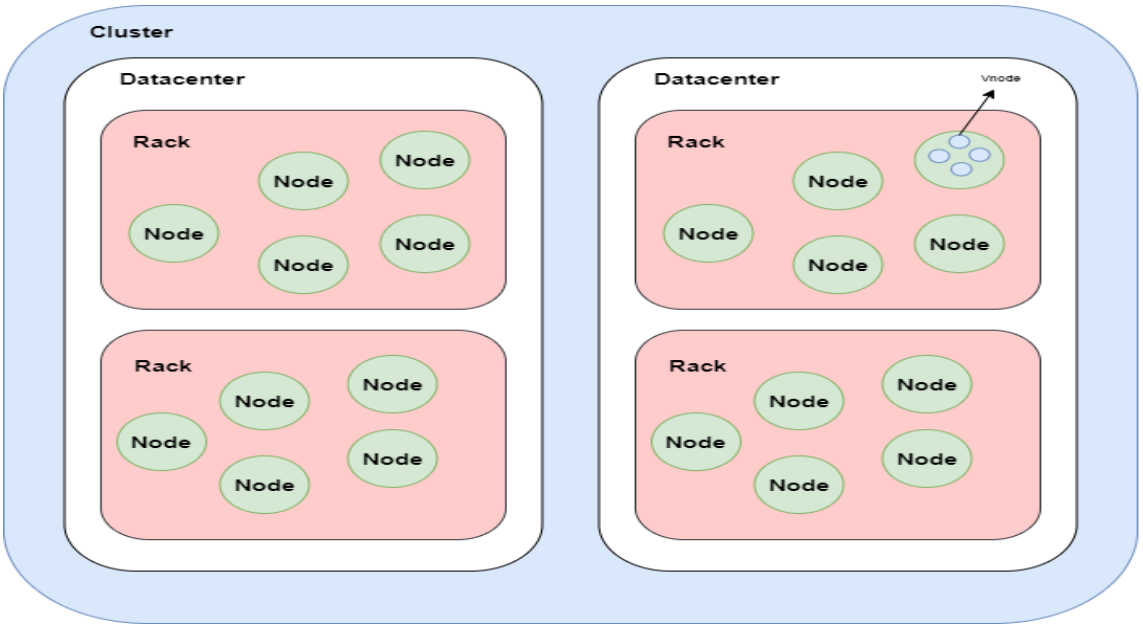


Figure 2.2 Un cluster Cassandra composé de six Data Centers

2.2.3.2 Data center Cassandra:

Un Data center Cassandra est un ensemble de nœuds groupés qui sont appelés Rack, Ci-dessous un schéma représentatif.

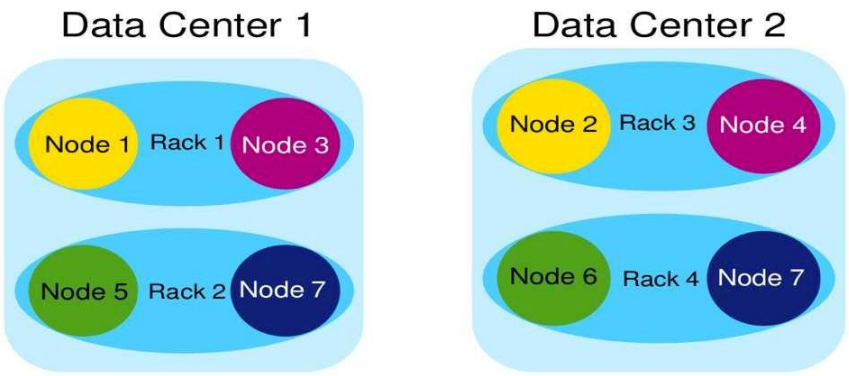


Figure 2 3 Deux Data Centers composés de deux Racks chacun

2.2.3.3 Rack Cassandra:

Le terme « rack » est généralement utilisé pour expliquer la topologie du réseau. Un rack est un ensemble de machines logées dans un même boîtier physique. Chaque machine du rack possède son propre processeur, sa propre mémoire et son propre disque dur. Cependant, le rack n'a pas de processeur, de mémoire ou de disque dur propre.

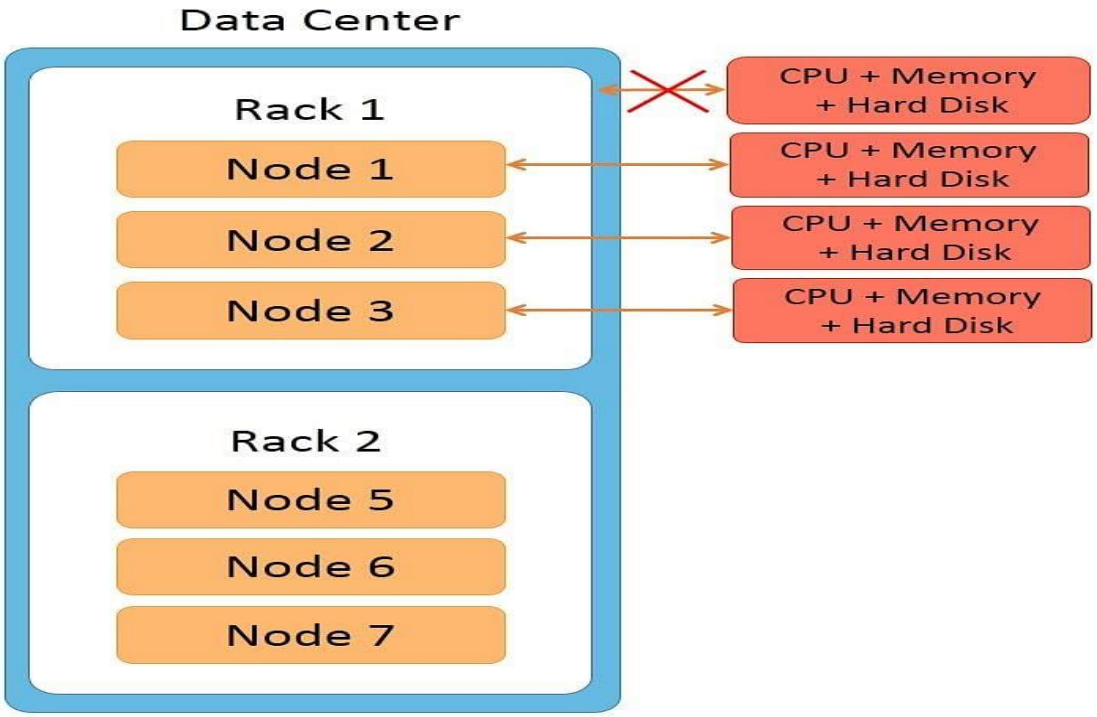


Figure 2.4 Data Centers composés de deux Racks

Les caractéristiques des racks sont:

- ✓Toutes les machines du rack sont connectées au commutateur réseau du rack
- ✓Le commutateur réseau du rack est connecté au cluster.
- ✓Toutes les machines du rack ont une alimentation commune. Il est important de noter qu'un rack peut tomber en panne pour deux raisons: une panne de commutateur réseau ou une panne d'alimentation.
- ✓Si un rack tombe en panne, aucune des machines du rack n'est accessible. Il semblerait donc que tous les nœuds du rack soient en panne[2].

2.2.3.4 Serveur Cassandra:

C'est une instance Cassandra (Nœud) installée sur une machine contenant 256 nœuds virtuels (Vnode) par défaut. Autrement dit, n'importe quel nœud Cassandra peut agir comme un serveur durant le déploiement de la BDD.

2.2.3.5 Nœuds virtuels

Les nœuds virtuels d'un cluster Cassandra sont également appelés vnodes. Les Vnodes peuvent être définis pour chaque nœud physique du cluster. Chaque nœud de l'anneau peut contenir plusieurs nœuds virtuels. Par défaut, chaque nœud a 256 nœuds virtuels.

Les nœuds virtuels permettent d'obtenir une granularité plus fine dans le partitionnement des données, et les données sont partitionnées dans chaque nœud virtuel à l'aide de la valeur de hachage de la clé. Lors de l'ajout d'un nouveau nœud au cluster, les nœuds virtuels qu'il contient obtiennent des parties égales des données existantes. Il n'est donc pas nécessaire d'équilibrer séparément les données en exécutant un équilibreur. L'image représente un cluster avec quatre nœuds physiques[2].

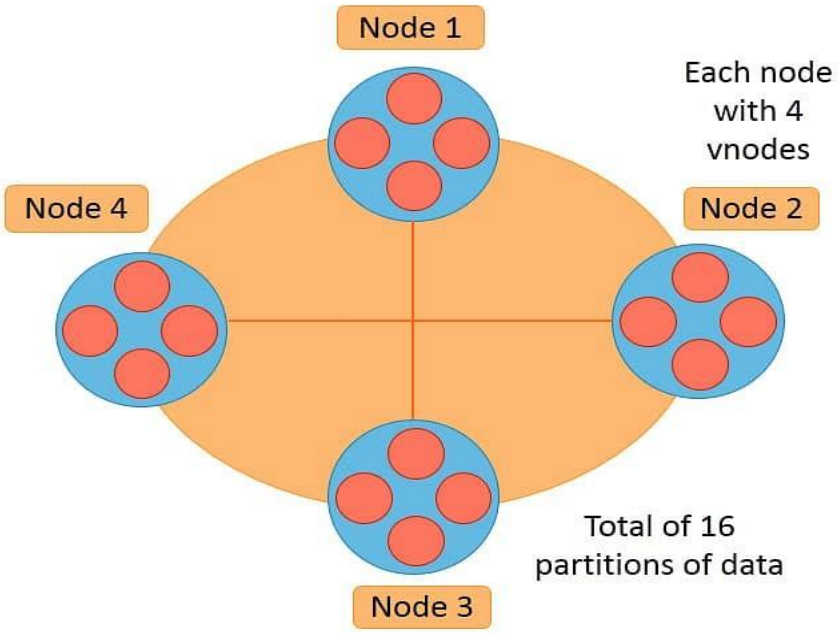


Figure 2.5 Chaque nœud physique du cluster possède quatre Nœuds vetuels

Chaque nœud physique du cluster possède quatre nœuds virtuels. Il y a donc 16 nœuds virtuels dans le cluster.

Si 32 To de données sont stockées sur le cluster, chaque vnode recevra 2 To de données à stocker. Si un autre nœud physique avec 4 nœuds virtuels est ajouté au cluster, les données seront distribuées à 20 vnodes au total, de sorte que chaque vnode disposera désormais de 1,6 To de données[2].

2.2.4 Modèle de données pour Cassandra

Pour bien comprendre le modèle de données utilisé par Cassandra il est important de définir un certain nombre de termes utilisés par la suite. Tout d'abord le:

✓**keyspace:** S'apparente à un namespace c'est en général le nom donné à votre application.

✓**Column Familly:** est-ce que ressemble le plus aux tables en SQL.

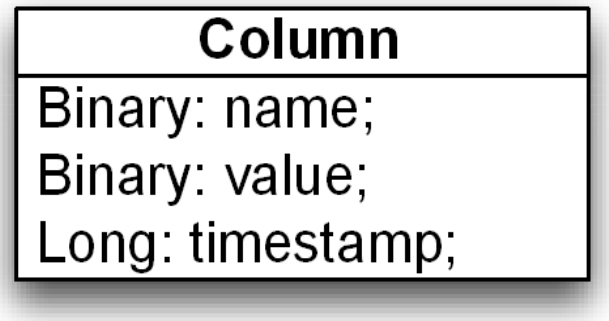
✓ **Key:** est une clé qui va représenter une ligne.

✓ **Column:** représente une valeur, elles disposent de 3 champs: son nom, sa valeur et un timestamp représentant la date à laquelle a été inséré cette valeur. Et pour finir,

✓ **Super Column:** contiennent une liste de colonnes.

2.2.4.1 Les colonnes:

L**es** données sont représentées en colonne comme le montre l'image ci-dessous. Dans une colonne on trouve 3 champs, son nom, sa valeur associée et un entier représenter la date à laquelle la donnée a été insérée dans la colonne.

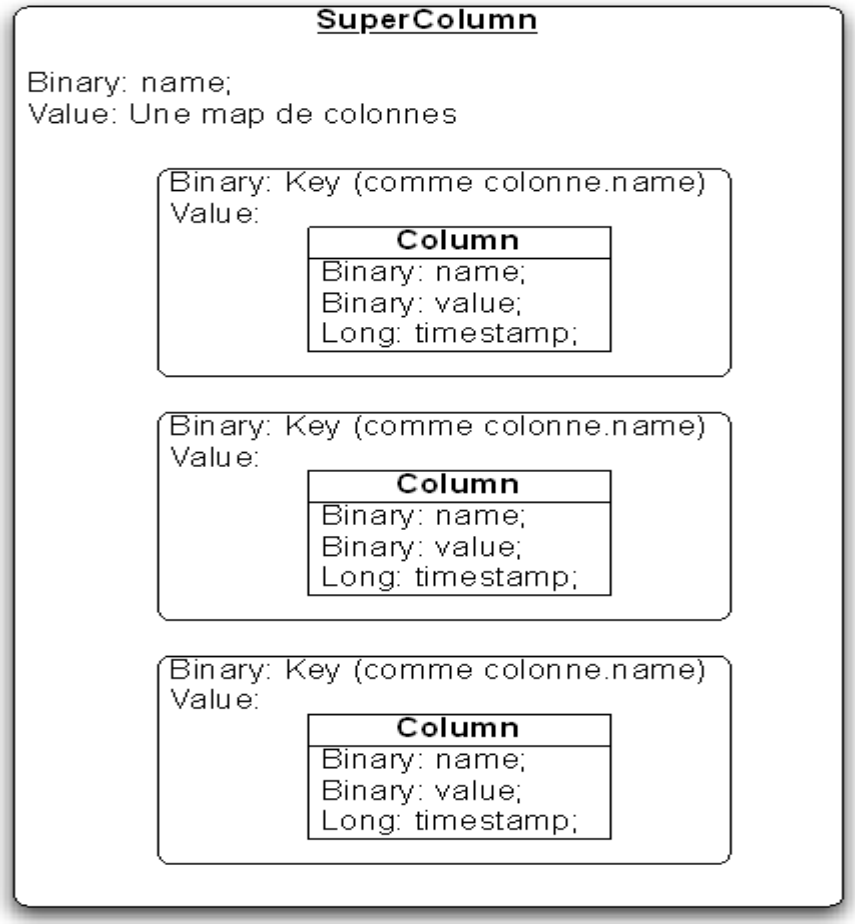


Voici une simple implémentation en JAVA d'une colonne qui va nous servir pour la suite des explications. Comme vous pouvez le constater on retrouve bien les 3 champs composant une colonne.

```
public class Column {
    private String name;
    private String value;
    private Long timestamp;
    public Column(String name, String value) {
            this.name = name;
            this.value = value;
    }
  }
```

2.2.4.2 Les super colonnes:

Les super colonnes sont une liste de colonne, on veut faire le parallèle avec une base SQL cela représente une ligne. On retrouve cette correspondance clé-valeur, la clé permet d'identifier la super colonne tandis que la valeur est la liste des colonnes qui la compose[3].



Pour son implémentation JAVA notre super colonne est composée d'un nom correspondant à sa clé et une Map de colonnes qui sont elle-même identifiées par une clé correspondant au nom de la colonne.

```
public class SuperColumn {
        private String name;
        private Map<String, Column> value;
        public void setName(String bs) {
                this.name = bs;
        }
        public void setValue(String name, Column value) {
                this.value.put(name, value);
        }
}
```

Pour l'insertion de donnée en utilisant ce modèle on doit tout d'abord initialiser notre super colonne et lui attribuer une clé, nous pouvons ensuite commencer à insérer des données.

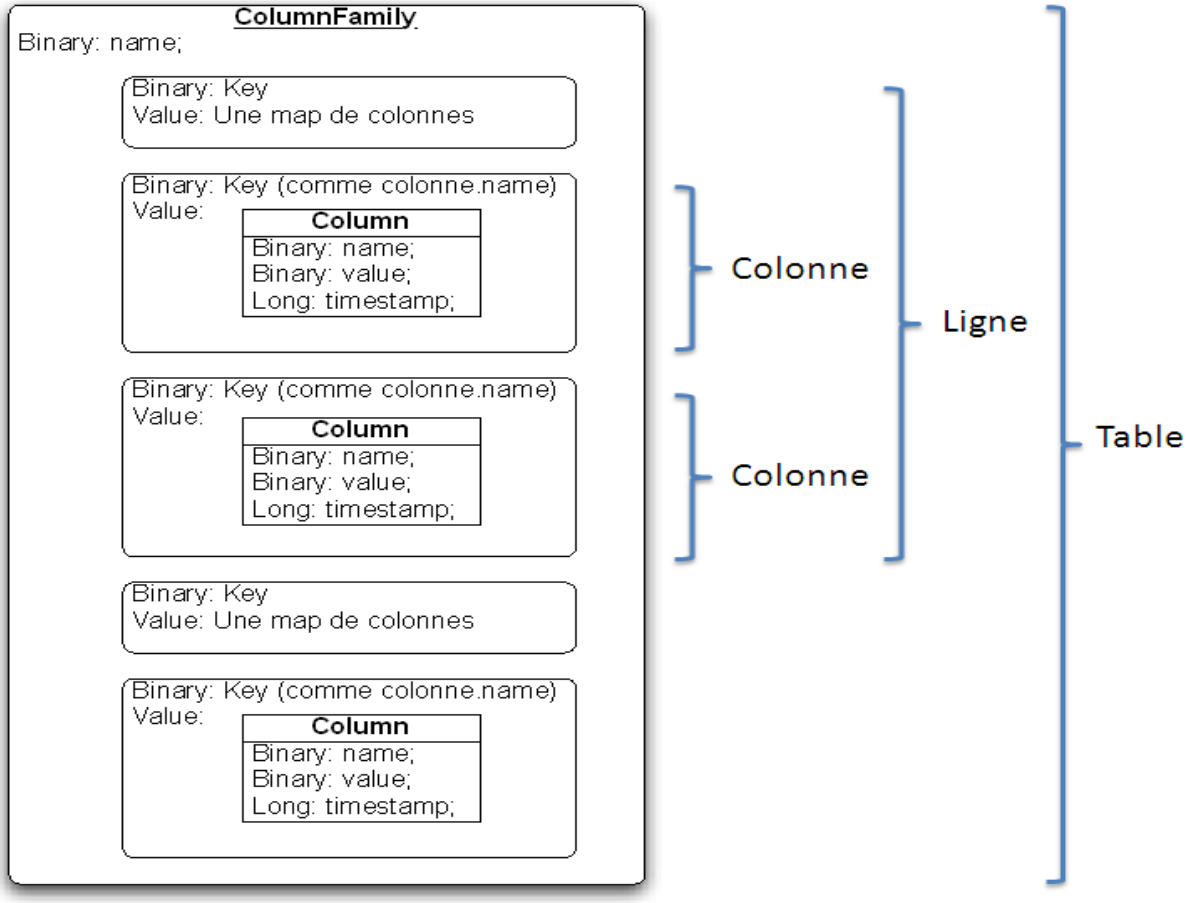
```
SuperColumn sc = new SuperColumn();
sc.setName("person1");
sc.put("firstname", new Column("firstname", "malika"));
sc.put("familyname", new Column("familyname", "ouamrane"));

sc = new SuperColumn();
sc.setName("person2");
sc.put("firstname", new Column("firstname", "mahmoud"));
sc.put("familyname", new Column("familyname", "bellili"));
```

Voilà comment les données sont organisées en gardant à l'esprit la notion de clé-valeur. D'après notre exemple nous pouvons voir que nous avons créé 2 super colonnes (person1 et person2) disposant chacune de 2 colonnes qui décrivent le firstname et le familyname de chaque personne[3].

SuperColumns	
Key	**Value**
person1	Column **Name** / **Value** firstName / John lastName / Calagan
person2	Column **Name** / **Value** firstName / George lastName / Truffe

2.2.4.3 Les familles de colonnes:

Les familles de colonnes sont ce qui ressemble le plus aux tables en SQL. A l'image des super colonnes elles disposent elles aussi d'une Map mais cette fois-ci de super colonnes et non pas de colonnes[3].



On retrouve le même mécanisme que les super colonnes, c'est uniquement la Map qui change pour y insérer les super colonnes.

```
public class ColumnFamily {

	private String name;
	private Map<String, SuperColumn> value;

	public void setName(String bs) {
		this.name = bs;
	}
	public void setValue(String name, SuperColumn value) {
		this.value.put(name, value);
	}
}
```

Pour l'insertion de donnée en utilisant ce modèle, il faut tout d'abord instancier une famille de colonne (ColumnFamily) et lui attribuer un nom qui sera sa clé (dans cet exemple "AddressBook"). On peut ensuite créer autant de super colonne que l'on souhaite et l'ajouter à notre famille de colonne.

```
ColumnFamily cf = new ColumnFamily();
cf.setName("AddressBook");

SuperColumn row= new SuperColumn();
row.setName("person1");
row.put("firstname", new Column("firstname", "John"));
row.put("familyname", new Column("familyname", "Calagan"));
cf.setValue("person1", row);

row = new SuperColumn();
row.setName("person2");
row.put("firstname", new Column("firstname", "George"));
row.put("familyname", new Column("familyname", "Truffe"));
cf.setValue("person2", row);
```

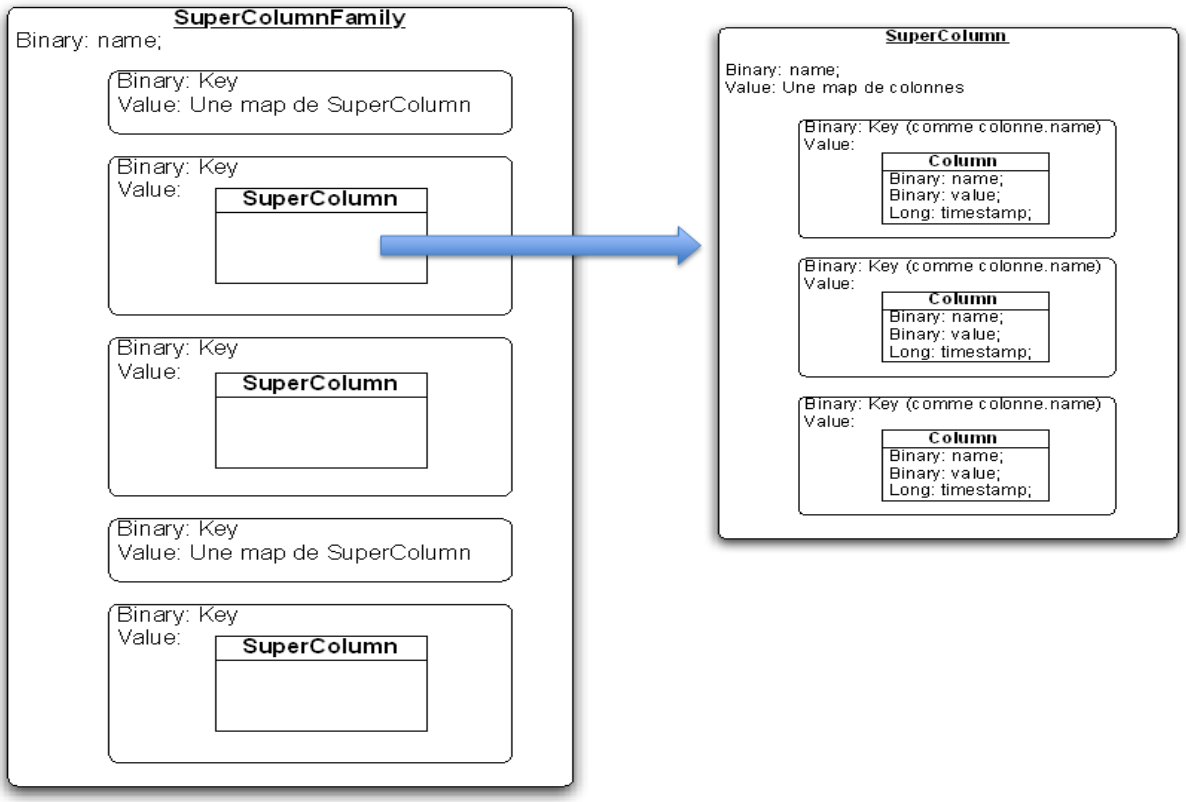
Voilà comment sont organisées les données une fois insérées dans la famille de colonnes.

<table>
<tr><td colspan="2">ColumnFamily</td></tr>
<tr><td>Key</td><td>Value</td></tr>
<tr><td>AddressBook</td><td>
<table>
<tr><td colspan="2">SuperColumns</td></tr>
<tr><td>Key</td><td>Value</td></tr>
<tr><td>person1</td><td>
<table>
<tr><td colspan="2">Column</td></tr>
<tr><td>Name</td><td>Value</td></tr>
<tr><td>firstName</td><td>John</td></tr>
<tr><td>lastName</td><td>Calagan</td></tr>
</table>
</td></tr>
<tr><td>person2</td><td>
<table>
<tr><td colspan="2">Column</td></tr>
<tr><td>Name</td><td>Value</td></tr>
<tr><td>firstName</td><td>George</td></tr>
<tr><td>lastName</td><td>Truffe</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

2.2.4.4 Les super familles de colonnes:

Les super familles de colonnes permettent d'ajouter un niveau d'imbrication supplémentaire. Cela permet de représenter l'ensemble d'une base de données. Dans ce cas, au lieu d'avoir une map de super colonne comme précédemment on dispose d'une map de famille de colonne[3].

```
public class SuperColumnFamily {
```




```
        String name;
  <Map<String, Map<String, SuperColumn>> value;
}
```

2.2.5 Lecture / Ecriture de données:

2.2.5.1 Processus d'écriture de Cassandra:

Le processus d'écriture Cassandra garantit des écritures rapides. Les étapes du processus d'écriture de Cassandra sont:

- ✓ Les données sont écrites dans un commitlog sur disque.
- ✓ Les données sont envoyées à un nœud responsable en fonction de la valeur de hachage.
- ✓ Les nœuds écrivent des données dans une table en mémoire appelée memtable.
- ✓ À partir de la memtable, les données sont écrites dans une sstable en mémoire. Sstable signifie table de chaînes triées. Cela a une donnée consolidée de toutes les mises à jour de la table.
- ✓ À partir de sstable, les données sont mises à jour dans la table réelle.

Si le nœud responsable est en panne, les données seront écrites sur un autre nœud identifié comme tempnode. Le tempnode conservera temporairement les données jusqu'à ce que le nœud responsable devienne actif. Le schéma ci-dessous décrit le processus d'écriture lorsque les données sont écrites dans la table A.

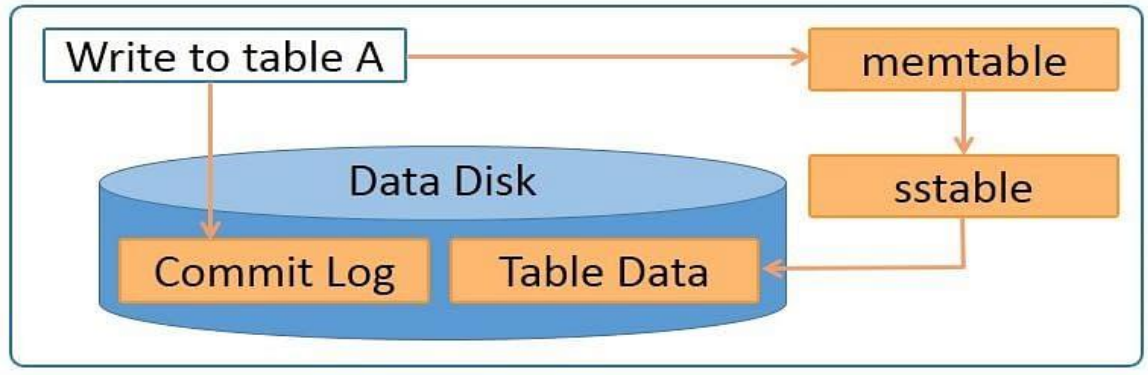


Figure 2.6 Processus d'écriture

Les données sont écrites dans un commitlog sur disque pour la persistance. Il est également écrit dans une table mémoire en mémoire. Les données Memtable sont écrites dans sstable qui est utilisé pour mettre à jour la table réelle.

2.2.5.2 Processus de lecture de Cassandra:

Le processus de lecture Cassandra garantit des lectures rapides. La lecture se produit sur tous les nœuds en parallèle. Si un nœud est en panne, les données sont lues à partir de la réplique des données. La priorité de la réplique est attribuée en fonction de la distance. Les caractéristiques du processus de lecture Cassandra sont:

✓ Les données sur le même nœud ont la priorité et sont considérées comme des données locales.

✓ Les données sur le même rack ont la deuxième préférence et sont considérées comme étant locales au rack.

✓ Les données sur le même centre de données se voient attribuer la troisième préférence et sont considérées comme étant locales au centre de données.

✓ Les données d'un centre de données différent sont celles qui ont le moins de préférence.

✓ Les données dans memtable et sstable sont vérifiées en premier afin que les données puissent être.

✓ récupérées plus rapidement si elles sont déjà en mémoire. Le schéma ci-dessous représente un cluster Cassandra.

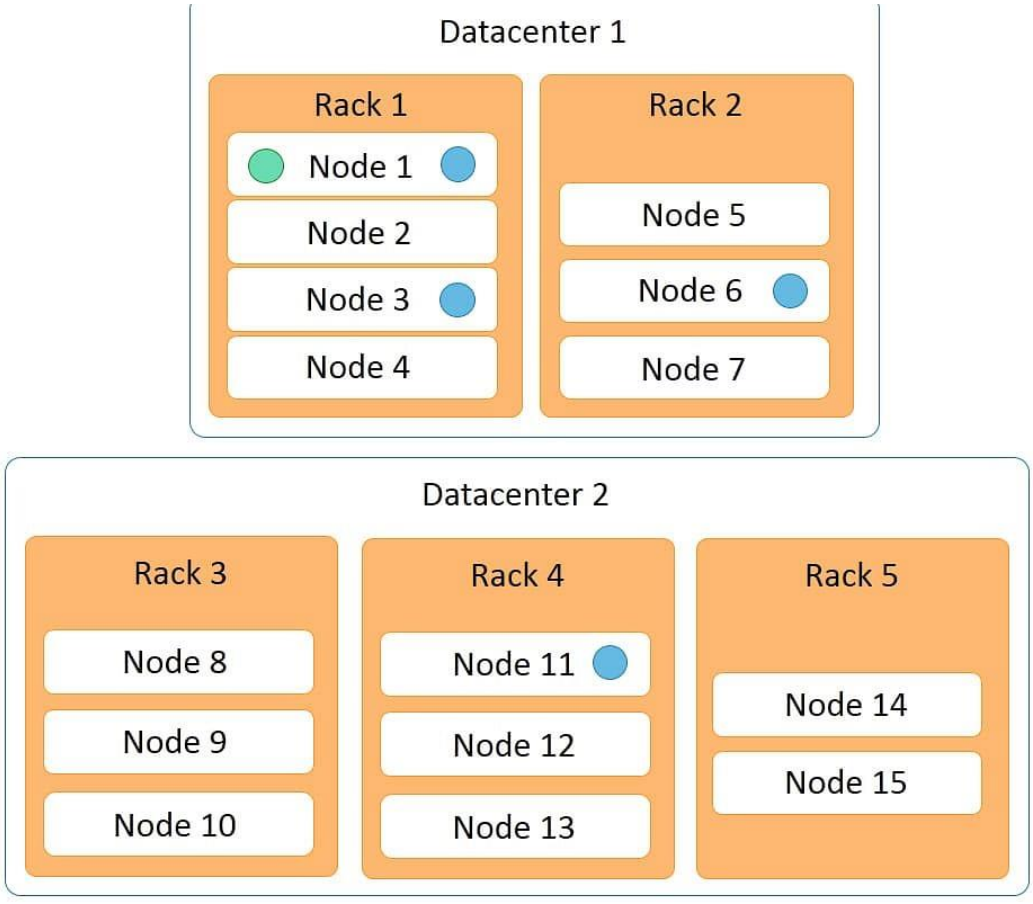


Figure 2.7 processus de lecture

Il dispose de deux centres de données:

- ✓ centre de données 1
- ✓ centre de données 2

Le centre de données 1 a deux racks, tandis que le centre de données 2 a trois racks. Quinze nœuds sont répartis sur ce cluster avec les nœuds 1 à 4 sur le rack 1, les nœuds 5 à 7 sur le rack 2, etc

2.2.6 Partitions de données:

Cassandra effectue une distribution transparente des données en partitionnant horizontalement les données de la manière suivante:

- ✓ Une valeur de hachage est calculée en fonction de la clé primaire des données.
- ✓ La valeur de hachage de la clé est mappée à un nœud du cluster
- ✓ La première copie des données est stockée sur ce nœud.
- ✓ La distribution est transparente car vous pouvez à la fois calculer la valeur de hachage et déterminer où une ligne particulière sera stockée.

Le schéma suivant illustre un cluster à quatre nœuds avec des valeurs de jeton de 0, 25, 50 et 75.

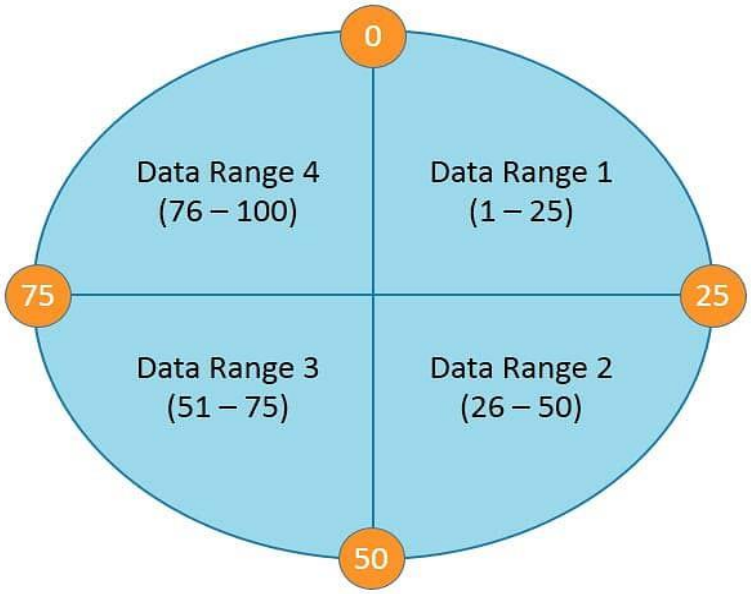


Figure 2.8 Schéma d'un cluster à quatre nœuds

Pour une clé donnée, une valeur de hachage est générée dans la plage de 1 à 100. Les clés avec des valeurs de hachage dans la plage de 1 à 25 sont stockées sur le premier nœud, 26 à 50 sont stockées sur le deuxième nœud, 51 à 75 sont stockées sur le troisième nœud, et 76 à 100 sont stockés sur le quatrième nœud. Veuillez noter que les jetons réels et les valeurs de hachage dans Cassandra sont des entiers positifs de 127 bits.

2.2.7 Réplication des données dans Cassandra:

La réplication fait référence au nombre de répliques conservées pour chaque ligne. La réplication fournit la redondance des données pour la tolérance aux pannes. Un facteur de réplication de 3 signifie que 3 copies de données sont conservées dans le système.

Dans ce cas, même si 2 machines sont en panne, vous pouvez accéder à vos données à partir de la troisième copie. Le facteur de réplication par défaut est 1. Un facteur de réplication de 1 signifie qu'une seule copie des données est conservée, donc si le nœud qui contient les données tombe en panne, vous perdrez les données.

Cassandra permet une réplication basée sur des nœuds, des racks et des centres de données, contrairement à HDFS qui permet une réplication basée uniquement sur des nœuds et des racks. La réplication entre les centres de données garantit la disponibilité des données même lorsqu'un centre de données est en panne.

2.2.8 Niveau de cohérence dans Cassandra:

Cassandra offre une cohérence des données configurable sur un cluster de base de données. Cela signifie qu'un développeur ou un administrateur peut décider exactement d'une cohérence forte (par exemple, tous les nœuds doivent répondre) ou d'une cohérence éventuelle (par exemple, un seul nœud répond, avec d'autres étant éventuellement mis à jour). La configuration de la cohérence des données est prise en charge sur un ou plusieurs data centers, et un développeur ou un administrateur dispose de nombreuses options de cohérence parmi lesquelles choisir[4].

Le réglage s'effectue en choisissant le niveau de cohérence de chaque requête. Au moment de l'écriture, le niveau de cohérence est le nombre de nœuds qui ont acquitté l'écriture. Au moment de la lecture, c'est le nombre de nœuds qui ont répondu. Lorsque deux nœuds fournissent une valeur différente, la plus récente est conservée et un mécanisme de mise à jour se déclenche. Les principaux niveaux de cohérence offerts par Cassandra sont ONE, QUORUM et ALL. Souvent utilisés en ONE-ONE pour une

cohérence éventuelle (faible) ou QUORUM-QUORUM pour une cohérence forte. D'autres combinaisons sont possibles, mais elles ne doivent être utilisées que si l'on en maitrise les conséquences **[Chiky .2016]**

Nous décrirons dans ce qui suit les différents niveaux de cohérence offerts par Cassandra que ce soit pour l'opération de lecture ou d'écriture.

2.2.8.1 En cas d'écriture:

Le niveau de cohérence en cas d'écriture détermine le nombre de réplicas sur lesquelles l'écriture doit être faite avant de renvoyer un accusé de réception à l'application cliente. Ci-dessous les différents niveaux de cohérence en cas d'écriture[4].

Niveau de cohérence	Description	Usage
ALL	Une écriture doit être écrite sur leCommit Log, la Memtable R et sur tous les nœuds des réplicas dans le cluster.	Fournit la plus forte cohérence et la plus faible disponibilité de tout autre niveau.
EACH_QUORUM	Cohérence forte. Une écriture doitêtre faite sur le Commit Log, sur la Memtable et sur la majorité des nœuds des réplicas dans tout le Data Center.	Utilisé dans plusieurs Data Cen- ters du cluster pour maintenir strictement la cohérence au même niveau dans chaque Data Cen- ter. Par exemple, choisir ce niveau si nous souhaitons qu'une lecture échoue lorsqu'un Data Center esthors service.
QUORUM	Une écriture doit être faite sur le Commit Log, sur la Memtable et sur la majorité des réplicas.	Fournit une forte cohérence si nous voulons tolérer un certain ni- veau de défaillance
LOCAL_QUORUM	Cohérence forte. Une écriture doitêtre faite sur le Commit Log, la Memtable et sur la majorité des nœuds des réplicas dans le même Data Center que le nœud coordi- nateur. Évite la latence de la com- munication entre les Data Cen-ters.	Utilisé pour maintenir la cohé- rence localement (dans un Data Center unique). Peut être utilisé avec SimpleStrategy.

ONE	Une écriture doit être faite sur le Commit Log, sur la Memtable et sur au moins un nœud de réplicas.	Ici les exigences de cohérence ne sont pas strictes
TWO	Une écriture doit être faite sur le Commit Log, sur la Memtable et sur au moins deux nœuds de ré- plicas.	Similaire à ONE.
THREE	Une écriture doit être faite sur le Commit Log, sur la Memtable et sur au moins trois nœuds de ré- plicas.	
LOCAL_ONE	Une écriture doit être envoyée à au moins un nœud de réplication dans le Data Center local et être reconnue par ce nœud	Dans plusieurs Data Center des clusters, un niveau de cohérence de ONE est souvent souhaitable, mais le trafic entre les Data Cen- ters ne l'est pas. LOCAL_ONE accomplit ceci. Pour des raisons de sécurité et de qualité, nous pouvons utiliser ce niveau de co- hérence dans un Data Center hors connexion pour empêcher la connexion automatique aux nœuds en connectés dans d'autres Data Centers si un nœud hors connexion tombe en panne.
any	Une écriture doit être faite sur au moins un nœud. Si tous les répli- cas sont hors service, une écriture sera réalisée sur n'importe quel nœud (ce nœud n'a pas une copie de cette donnée) et sera propagée sur le reste des réplicas lorsqu'ils seront disponibles	Fournit une faible latence et ga- rantit que l'écriture n'échoue ja- mais. Offre la plus faible cohérence et la plus haute disponibi lité.
serial	Atteint la cohérence linéarisable pour les transactions légères en empêchant les mises à jour incon- ditionnelles.	Ce niveau de cohérence est uni- quement utilisé avec une transac- tion légère. Equivalent à QUO- RUM.

Local- serial	Identique à SERIAL mais limité au Data Center. Une écriture doit être faite de manière condition- nelle dans le Commit Log, dans la Memtable et sur la majorité des nœuds de réplicas dans le même Data Center.	Identique à SERIAL. Utilisé pour la récupération après défaillance

Table 2.1 ***Niveaux de cohérence en écriture sous Cassandra***

2.2.8.2 En cas de lecture:

Le niveau de cohérence en cas de lecture spécifie le nombre de réplicas qui doivent répondre à une demande de lecture avant de renvoyer des données à l'application cliente[4].
Ci-dessous les différents niveaux de cohérence en cas de lecture.

Niveau de cohérence	**Description**	**Usage**
ALL	Renvoie la réponse à l'application cliente après que tous les réplicas ont répondu. L'opération de lec- ture échouera si au moins un ré- plica n'a pas répondu.	Fournit la plus grande cohérence de tous les niveaux et la plus faible disponibilité de tous les ni- veaux.
QUORUM	Renvoie la réponse à l'application cliente après que la majorité des réplicas ont répondu à partir de n'importe quel Data Center.	Assure une forte cohérence si nous pouvons tolérer un certain niveau de défaillance.
LOCAL-QUORUM	Renvoie la réponse à l'application cliente après que la majorité des réplicas appartenant au mêmeData Center que le nœud coordinateur aient répondu. Évite la latence de la communication entre les Data Centers.	Utilisé dans plusieurs Dat Cen- ters du cluster avec une straté- gie de positionnement de réplicas compatible avec le rack (Network- TopologyStrategy) et un snitch correctement configuré. Non com- patible avec SimpleStrategy
	Renvoie une réponse du	Fournit la plus haute

ONE	premier réplica ayant répondu. Une mise à jour en lecture s'exécute éven- tuellement en arrière-plan pour rendre les autres réplicas cohé- rents.	disponibilité de tous les niveaux si nous pouvons tolérer une probabilité relativement élevée de lire des données incohérentes. Les réplicas contactés pour les lectures peuvent ne pas toujours avoir la donnée la plus récente.
TWO	Renvoie la réponse la plus récente des deux premiers réplicas ayant répondu.	Similaire à ONE.
THREE	Renvoie la réponse la plus récente des trois premiers réplicas ayant répondu.	Similaire à ONE.
LOCAL_ONE	Renvoie la réponse du premier réplica ayant répondu du même Data Center que le nœud coordinateur.	Même chose en cas d'écriture.
SERIAL	Permet de lire l'état actuel (et peut-être non validé) des données sans proposer une nouvelle addition ou mise à jour. Si une lecture SERIAL trouve une transaction non validée en cours, elle validera la transaction dans le cadre de la lecture. Semblable à QUORUM.	Utilisé pour lire la dernière valeur d'une colonne après qu'un utilisateur a sollicité une transaction lé- gère pour écrire dans la colonne.
LOCAL_SERIAL	Similaire à SERIAL mais se res- treint à un même Data Center. Similaire à LOCAL_QUORUM	Utilisé pour obtenir une cohérence linéarisable pour les transactions légères.

Table 2.2 Niveaux de cohérence en lecture sous Cassandra

2.2.9 Cassandra Query language CQL :

Les langages de requête sont des langages informatiques utilisés pour effectuer des requêtes dans des bases de données et des systèmes d'information. Cassandra Query Language est le principal langage de requête pour communiquer avec la base de données Apache Cassandra.

Contrairement au SQL (Structured Query Language) qui est un langage de requête relationnelle basé sur des tables, ou les bases de données évoluent verticalement, avec un schéma fixe qui gèrent des volumes de données modérés. Les langages de requête CQL peuvent effectuer des requêtes sur des clusters de serveurs distribués horizontalement, sont extrêmement évolutifs, gèrent des données non structurées, ne nécessitent pas de schéma fixe, évitent les jointures et sont faciles à mettre à l'échelle.

Cassandra CQL est un langage déclaratif développé pour fournir une abstraction lors de l'accès à Apache Cassandra.

Les constructions de base de Cassandra (CQL) comprennent:

- **Espace** de clés : Similaire à une base de données RDBMS, un espace de clés est un conteneur pour les données d'application qui doit avoir un nom et un ensemble d'attributs associés. L'espace de clés Cassandra est une base de données SQL.

- **Familles de colonnes/tables** : Un espace de clés se compose d'un certain nombre de familles/tables de colonnes. Une famille de colonnes Cassandra est une table SQL.

- **Clé primaire/Tables** : Une clé primaire se compose d'une clé de ligne/partition et d'une clé de cluster, et fonctionne pour permettre aux utilisateurs d'identifier de manière unique les lignes internes de données. Une clé de ligne/partition détermine le nœud sur lequel les données sont stockées. Une clé de cluster détermine l'ordre de tri des données dans une ligne particulière.

Exemple de langage de requête Cassandra:

- ✓ **CQL Create Keyspace** : Pour créer un keyspace appelé « fruit », exécutez l'instruction ci-dessous:
- ✓ **CREATE KEYSPACE** fruit WITH REPLICATION = { 'class': 'SimpleStrategy' , 'replication_factor': 3};
- ✓ Exécutez la commande "DESC KEYSPACES" une fois de plus pour voir si l'espace de touches "fruit" est créé.
- ✓ Langage de requête Cassandra vs SQL
- ✓ Le langage de requête CQL est une interface NoSQL intentionnellement similaire à SQL, offrant aux utilisateurs qui sont à l'aise avec les bases de données relationnelles un langage familier qui abaisse finalement la barrière d'entrée à Apache Cassandra.

Certaines caractéristiques associées à SQL mais non disponibles dans Cassandra (CQL) incluent:

- ❖ **Clauses WHERE arbitraires :** Un prédicat ne peut contenir que des colonnes spécifiées dans la clé primaire.

- ❖ **Clauses ORDER BY arbitraires** : Ordre by ne peut être appliqué qu'à une colonne de cluster.

- ❖ **GROUP BY** : Les données identiques ne peuvent pas être regroupées dans CQL.

- ❖ **Cassandra CQL JOIN** : Les données des familles de colonnes ne peuvent pas être jointes dans CQL sans outils tiers.

- ❖ **Transactions complexes** : Là où SQL prend en charge les transactions complexes, CQL prend en charge les transactions simples.

CQL est utilisé dans les bases de données non relationnelles, est compatible avec Apache Cassandra, utilise une syntaxe de requête de type SQL, permet aux termes "table" et "columnfamily" d'être interchangeables et utilise son propre protocole binaire CQL. La requête CQL peut également renvoyer des objets. Les objets d'une collection sont accessibles par itération via le champ de collection dans l'objet de résultat. Contrairement aux colonnes

SQL hautement structurées, les champs de résultat peuvent être eux-mêmes des collections ou des objets, plutôt que de simples valeurs de type de données.

2.2.9.1 Types de données CQL :

Apache Cassandra CQL prend en charge un ensemble complet de types de données, notamment des types de collection, des types personnalisés, des types natifs, des types de tuple et des types définis par l'utilisateur:

- **type de collection** s : destiné à stocker et/ou dénormaliser de petites quantités de données ; prend en charge les collections Maps, Sets et Lists
- **types personnalisés** : une chaîne qui peut être chargée par Cassandra et contient le nom de la classe Java qui étend la classe AbstractType côté serveur
- **types natifs** : inclut Bigint, Blob, Boolean, Counter, Date, Decimal, Double, Duration, Float, Inet, Int, Smallint, Text, Time, Timestamp, Timeuuid, Tinyint, Uuid, Varchar, Varint
- **types de tuples** : un ensemble ordonné de valeurs qui peuvent être considérées fonctionnellement comme des UDT anonymes avec des champs anonymes et qui ne peuvent être mis à jour que dans leur ensemble
- **types définis par l'utilisateur** : un ensemble nommé composé de champs nommés et typés qui peuvent être créés, modifiés et supprimés à l'aide de l'instruction create_type_statement.

2.3 Conclusion

Dans ce chapitre nous avons présenté le système NOSQL « apache Cassandra », en décrivant son l'architecture, le modèle de donnés utilisé, la réplication ainsi que les niveaux de cohérence avec ce système. Nous avons aussi présenté CQL le langage de requête d'apache Cassandra.

Chapitre 3: Framework YCSB

3.1 Introduction

YCSB a été développé pour évaluer les performances des systèmes de gestion des bases de données relationnels et NoSQL tel que Oracle , MongoDB, DocumentDB, DynamoDB, RethinkDB, Couchbase et cassandra.

3.2 Qu'est-ce-que YCSB ?

YCSB est un bunchmark conçu et développé pour évaluer et effectuer des comparaisons entre les systèmes de gestion de base de données,

3.3 Architecture de YCSB

YCSB fournit aux utilisateurs deux parties **[BUGBEE.1917]** :

*** Le client YCSB lui-même :**

C'est un générateur de l'environnement de travail (workload). Autrement dit, il s'occupe de charger des données nécessaires sur les- quelles des tests seront exécutés.

*** Un ensemble d'exécuteur des workloads de base :**

Ils sont conçus pour fournir un résumé complet des métriques de performance d'une plateforme, cependant, les utilisateurs peuvent définir leurs propres types d'exécution dans le butde couvrir d'autre aspect du système. Le type d'exécution par défaut fournit des statistiques de latence et de débit. Ci-dessous l'architecture du benchmark YCSB.

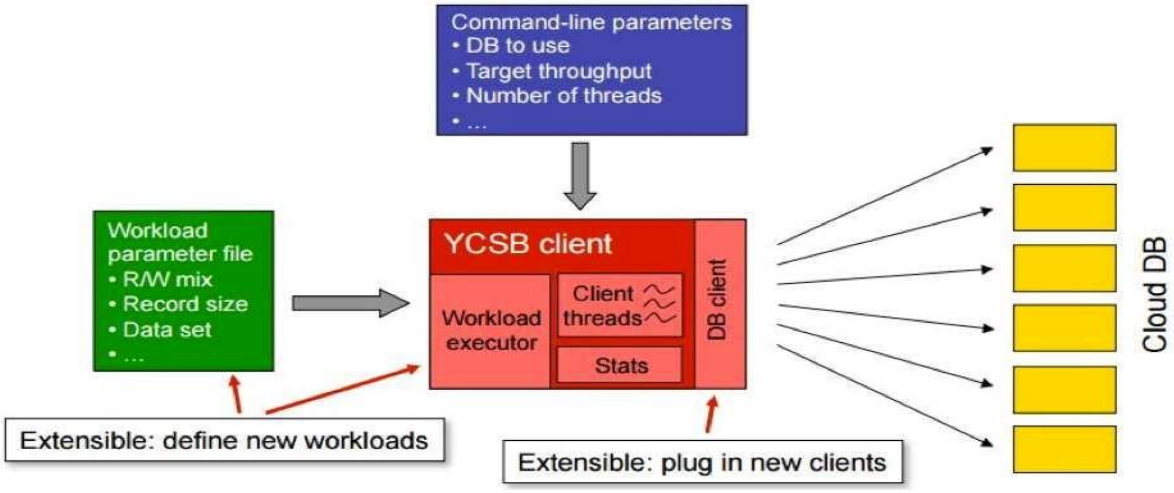


Figure 3.1 Architecture de YCSB

3.4 Les Workloads du benchmark YCSB :

Un ensemble de workloads de base ont été développés pour évaluer différents aspects de la performance d'un système, ces workloads se trouvent dans le package

YCSB Core. En termes de structuration, un package est une collection de workloads connexes. Chaque workload représente un nombre d'opérations de lecture/écriture, pour une taille de don- nées précise, de requêtes spécifiques, etc.

Les utilisateurs de YCSB peuvent développer leurs propres paquets soit en définissant un nouvel ensemble de paramètres pour le workload, et si nécessaire en écrivant du code Java. Les différentes opérations utilisées choisies au hasard peuvent l'une des opérations suivantes :

Insert : Insère un nouvel enregistrement.

Update : Met à jour un enregistrement en remplaçant la valeur d'un champ.

Read : Lit un enregistrement, soit un champ au hasard ou bien tous les champs.

Scan : Lit les enregistrements dans l'ordre, en commençant par une clé d'un enregistrement choisie au hasard. Le nombre d'enregistrements à lire est choisi au hasard.

Le package Core de YCSB contient 5 workloads qui sont décrits dans le tableau ci-dessous :

Workload	**Opérations**
AWorkload A – Update heavy	Read : 50% - Update : 50%
Workload B – Read heavy	Read: 95% - Update : 5%
Workload C – Read only	Read: 100%
AWorkload **D** – **Read latest**	Read : **95**% - Update : **5**%
AWorkload **E** – **short ranges**	Read : **95**% - Update : **5**%

Table 3.1 Workloads de YCSB

Afin d'évaluer un système, on effectue un chargement initial de données (Load) sur lesquelles des workloads (5 types) peuvent être exécutés, Puis on lance l'exécution d'un workloads, en autre de nouveaux workloads peuvent être créés.

D'un autre côté, le package open- source, le Client YCSB peut être utilisé et étendu par les développeurs de manière à ce qu'il accomplisse une tâche (évaluation et/ou comparaison) bien précise d'un système biendéfini.

3.5 Niveaux de références de YCSB

YCSB se contente d'évaluer les systèmes en prenant en compte les deux aspects performances et évolutivité du SGBD. **[Coo+10]**

- **Niveau 1 : performance :**

Le niveau performance concerne la latence de la base de données lorsqu'elle reçoitdes requêtes parvenant de plusieurs sources. La latence d'un système de service est très importante, car derrière il y a toujours un client impatient d'attendre le résultat d'une requête. Cependant il existe un compromis inhérent entre la latence et le débit : lorsque la charge sur le système augmente, la latence des requêtes client augmente également car il y a plus de conflits pour le disque, le processeur et le réseau. Généralement, les concepteurs d'applications doivent décider d'une latence acceptable et mettre en place une infrastructure adéquate pour atteindre le débit souhaité tout en préservant une latence acceptable. Pour réaliser ce niveau de référence, YCSB utilise un générateur de workload qui a pour objectif de définir l'ensemble de données et les charger à la base de données pour ensuite exécuter des opérations sur l'ensemble de données tout en mesurant les performances. Le client YCSB a été implémenté de telle sorte qu'il puisse atteindre les deux objectifs, il permet à l'utilisateur de définir le débit comme paramètre de la ligne de commande Shell et il s'en charge de rapporter la latence qui en résulte.

Niveau 2 : évolutivité :

Un autre aspect clé référençant les services Cloud c'est leur capacité d'évoluer au fil du temps de manière élastique, de telle sorte qu'ils peuvent gérer plus de charge car des mises à jour ne cessent de s'effectuer dans le but d'ajouter de nouvelles fonctionnalités, régler certains bugs, etc. Ce niveau de référence examine l'impact sur les performances au fur et à mesure que l'infrastructure évolue (ajout de nouveaux serveurs). Il existe deux critères de mesure pour ce niveau :

- **Scale-up :**

Comment la base de données réagit lorsque le nombre de machines augmente ?

Dans ce cas, nous mettons en place un nombre donné de serveurs, chargeons des données et exécutons un workload. Ensuite, nous supprimons toutes les données, nous ajoutons d'autres serveurs et chargeons une plus grande quantité de données sur le cluster et ré-exécutons le workload. Si le système de base de données a de bonnes propriétés scale-up, les performances (par exemple, la latence) doivent rester constantes, étant donné que le nombre de serveurs, la quantité de données et le débit offert sont proportionnels.

- **Elastic speed-up :**

Comment la base de données réagit lorsque le nombre de machines augmente pendant l'exécution du système ?

Dans ce cas, nous mettons en place un nombre donné de serveurs, chargeons des données et exécutons un workload. Lorsque le workload est en cours d'exécution, nous ajoutons unou plusieurs serveurs et observons l'impact sur les performances. Un système qui offre unebonne élasticité devrait montrer une amélioration des performances lorsque les nouveaux serveurs sont ajoutés, avec une période d'interruption courte ou inexistante pendant quele système se reconfigure pour intégrer les nouveaux serveurs.

3.6 Conclusion

Dans ce chapitre, Nous avons présenté le benchmark YCSB, utilisé pour évaluer les performances des systèmes de gestion des bases de données, des différents types, l'architecture de YCSB, les Niveaux de références de YCSB.

Chapitre 4: Réalisation et Déploiement

4.1 Introduction

Nous avons jusqu'à présent pu présenter notre solution sur un point de vue conceptuel, il est temps maintenant de concrétiser toutes les études effectuées précédemment afin d'aboutir à un produit final représentant notre solution.

Ce chapitre parlera donc de la réalisation et du déploiement de la solution conçue, touten mettant l'accent sur les ressources technologiques logicielles et matérielles auxquelles nous avons eu recours.

4.2 Technologies utilisées

4.2.1 Benchmark YCSB

Le cœur de notre travail est basé sur l'utilisation de ce benchmark open source du fait qu'il soit compatible avec Apache Cassandra, en effet il permet d'effectuer des tests avec les niveaux de cohérences standards seulement (Forte, Faible), nous l'avons donc adapté à notre besoin en réécrivant certaines parties pour y intégrer un nouveau niveau de cohérence qui est adaptable.

4.2.2 Langages utilisés

4.2.2.1 Java :

Cette solution offre des temps d'accès record aux données stockées dufait que ces listes se situent en RAM.

Java est un langage de programmation et une plate-forme informatique qui ont été créés par Sun Microsystems en 1995. Beaucoup d'applications et de sites Web ne fonctionnent pas si Java n'est pas installé et leur nombre ne cesse de croître chaque jour.

6.2.2.2 Python :

Python est un langage de programmation objet, multi paradigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet.

Python est également utilisable en tant que langage d'extension pour les applications écrites dans d'autres langages nécessitant des interfaces de script ou d'automatisation faciles à utiliser. **[Pyt]** C'est pourquoi nous

l'avons utilisé pour faire le lien entre des commandes Shell en entrée et l'application java de traitements.

4.2.3 SGBDs utilisés

4.2.3.1 Apache Cassandra :

Nous avons utilisé Apache Cassandra comme BDD distribuée afin de mettre en œuvre nos tests de cohérences, ce choix se justifie par la possi- bilité et la facilité que ce SGBD offre aux développeurs afin de configurer la cohérence à l'exécution (Cohérence paramétrable).

4.3 Solution détaillée

4.3.1 Architecture Technique

Dans la figure ci-dessous, nous présentons l'architecture technique de notre solution qui passe essentiellement par trois phases : Prétraitements, Traitements, Post-traitements.

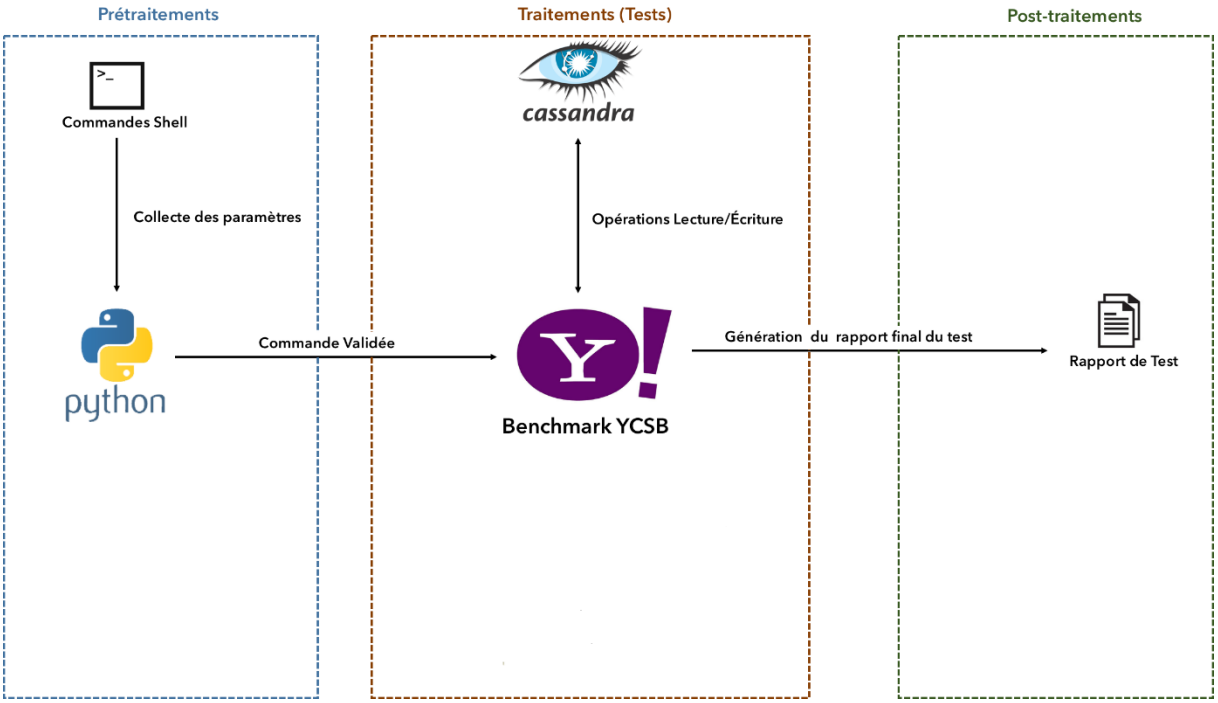


Figure 4.1 Architecture Technique de la solution

4.3.2 Phase de prétraitements

```
try:
    mod = __import__('argparse')
    import argparse
except ImportError:
    print >> sys.stderr, '[ERROR] argparse not found. Try installing it via "pip".'
    exit(1)
```

Figure 4 2 Vérification de l'environnement d'exécution

Dans cette phase on récupère la commande insérée par l'utilisateur, on vérifie que l'environnement d'exécution est bien installé (le package *argparse* doit être disponible), puis on analyse la commande pour décider de sa validité et enfin on informe le client YCSB de l'action à effectuer (Test de cohérence ou Chargement de données).

Les commandes à vérifier et à transmettre au client YCSB sont de type run ou load, etportent la syntaxe suivante :

4.3.2.1 Commande Load

La commande suivante effectue un load (chargement de données) sur la BDD suivantles paramètres introduits :

```
sudo ./bin/ycsb load cassandra-cql
-p hosts=param1
-P workloads/param2
-p consistency=param3
-p paramjconfiguration=param4
-p measurementtype=param5
```

- **Param 1 :** Les adresses IP des nœuds du cluster.
- **Param 2 :** Le type de workload.
- **Param 3 :** Le mode de cohérence (strong, weak).
- **Param 4 :** Les configurations de la cohérence citées dans la partie **5.4.2**.
- **Param 5 :** Type de mesure à inclure (histogram, timeseries).

- **Param 6 :** Le nombre d'enregistrements à insérer.
- **Param 7 :** Le nombre de threads sur lesquels exécuter le test.

4.3.2.2 Commande Run

La commande suivante effectue un run sur la BDD suivant les paramètres introduits :

```
sudo ./bin/ycsb run cassandra-cql
-p hosts=param1
-P workloads/param2
-p consistency=paramj
-p paramjconfiguration=param4
-p measurementtype=param5
-p recordcount=param6
-p operationcount=param7
-p maxexecutiontime=param8
-threads param9
-p exportfile= nom_fichier.txt -s > nom_fichier_infos.txt
```

- **Param 1 :** Les adresses IP des nœuds du cluster.
- **Param 2 :** Le type de workload.
- **Param 3 :** Le mode de cohérence (strong, weak).
- **Param 4 :** Les configurations de la cohérence citées dans la partie **5.4.2**.
- **Param 5 :** Type de mesure à inclure (histogram, timeseries).
- **Param 6 :** Le nombre de lignes de la BDD à prendre en considération.
- **Param 7 :** Le nombre d'opérations à effectuer.
- **Param 8 :** Le temps d'exécution maximum du test.
- **Param 9 :** Le nombre de threads sur lesquels exécuter le test.

4.3.3 Phase de traitements (Tests)

Lors de cette phase, le client YCSB connaît l'opération à exécuter et avec quels para-mètres l'exécuter.

Nous portons un plus grand intérêt à la commande Run car c'est grâce à elle que nostests seront effectués et c'est autour d'elle que la majeure partie de notre travail se base, par ailleurs dans ce qui suit nous omettrons la commande Load pour les chargements de données.

- **Validation syntaxique et sémantique de la commande :** Le client YCSB effectue une dernière validation et cette fois sur les paramètres obtenus depuis la précédente phase afin de vérifier leur aspect syntaxique et sémantique.

- **Initialisation du Workload :** Le client YCSB prépare le chargement qui serasollicité pour les opérations de lecture/écriture lors des tests.

- **Lancement des threads :** Le client YCSB lance les threads sur lesquels les traite-ments seront exécutés, et leur nombre dépend du paramètre entré par l'utilisateur.

Après que les threads soient lancés, le client procède à l'exécution des tests en prenant en considération le nombre d'opérations, le mode de cohérence et le temps d'exécution maximum. Durant toute la période d'exécution, on fait des appels successifs aux méthodes *Read* et *Update* situées dans la classe CassandraCQLClient qui ont pour objectif de lire et écrire des données de la BDDD Cassandra afin de simuler des accès distribués de plusieursclients.

4.3.4 Phase post-traitements

Cette phase consiste à générer un rapport de test exporté dans un fichier .txt, il contient les informations relatives au test (latence, débit, nombre d'opération de lecture et écriture, etc) qui nous servirons lors de la phase de tests pour générer des graphiques de comparaison.

4.4 Configuration de l'environnement

Dans ce qui suit nous allons présenter les différentes étapes suivies pour déployer notre solution.

4.4.1 Ressources Techniques

Table de liste des matériels et logiciels utilisés dans notre configuration

Matériels	**Détails**
2 x host servers	2.4 GHz CPU 12 GB RAM
1 x host servers	3.3 GHz CPU 08 GB RAM
1 x host servers	2.4 GHz CPU 08 GB RAM
Network	SWTCH D-LINK
Logiciel	**Détails**
System d'exploitation	Windows server 2016 / Lubuntu 22.04 /PARROT 18
Base de données	Apache Cassandra 4.0.4
Benchmarking tool	Yahoo! Cloud Serving Benchmark

Table 4.1 Liste des materiels et logiciels

Figure 4.3 Détail technique sur serveur1

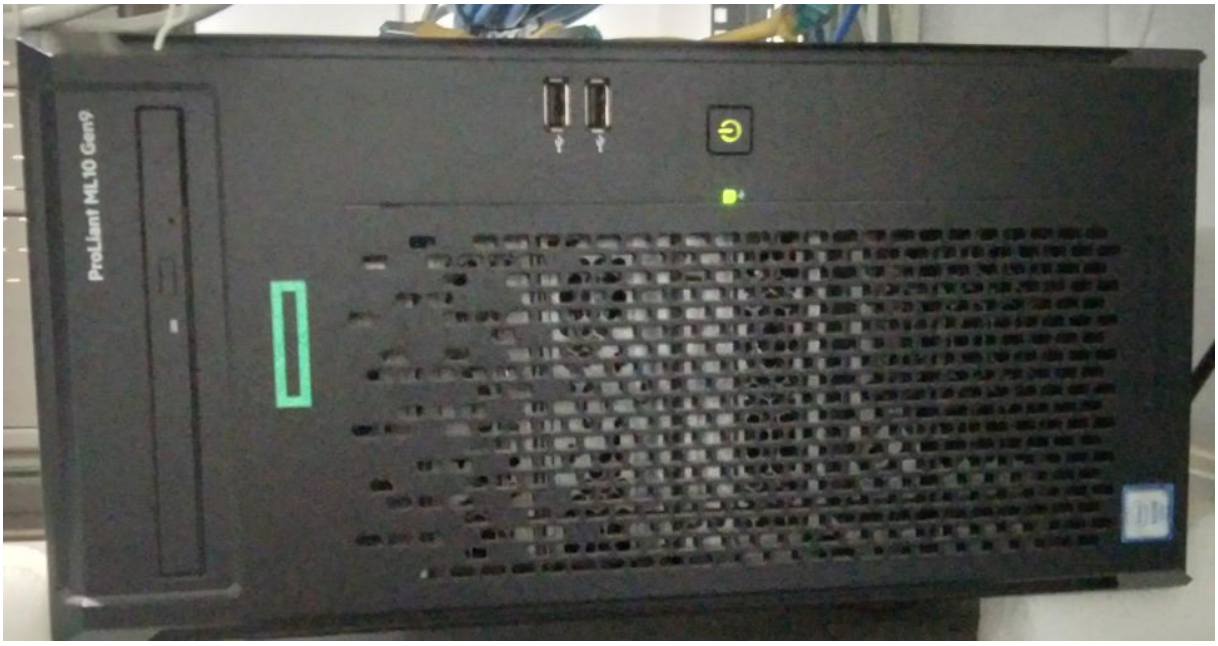


Figure 4.4 Détail technique sur serveur2

Figure 4.5 Détail technique sur switch

4.4.2 Configuration de Cassandra

Pour permettre la configuration d'un cluster Cassandra, nous aurons besoin d'avoir au préalable installés sur chaque nœud du cluster :

- Java 8+
- Python 2.6+

Pour notre solution, nous avons configuré un cluster Cassandra de 3 nœuds, et pour y arriver, nous avons tout d'abord déterminé les nœuds seeds et les nœuds non-seed avec leurs adresses IP, ensuite fixé les adresses RPC et d'écoute dans le fichier Cassandra.yaml comme le montrent les figures qui vont suivre :

- **Nœuds seeds :** On renseigne tout d'abord les nœuds seeds.

```
parameters:
    # seeds is actually a comma-delimited list of addresses.
    # Ex: "<ip1>,<ip2>,<ip3>"
    #- seeds: "127.0.0.1:7000"
      - seeds: "192.168.1.2,192.168.1.5"
```

Figure 4.6Configuration des nœuds seeds

- **Adresse RPC :** Par la suite on renseigne l'adresse d'écoute pour les connexions client sur chacun des 05 nœuds.

```
rpc_address: 192.168.1.4
```

Figure 4.7 Configuration de l'adresse RPC

- **Adresse d'écoute :** Enfin on renseigne l'adresse IP ou le nom d'hôte à laquelle Cassandra se lie pour se connecter à d'autres nœuds.

```
listen_address: 192.168.1.4
```

Figure 4.8 Adresse de liaison

4.4.3 Configuration du benchmark

Pour pouvoir déployer le benchmark YCSB, nous aurons besoin d'avoir au préalable installés :

- Java 8+
- Maven 3.5+
- Python 2.6+

Et afin d'indiquer au benchmark avec quels nœuds du cluster Cassandra communiquer, nous devons lui fournir une adresses IP d'un des nœuds du cluster au minimum avant la phase de prétraitements dans la commande shell.

4.5 Déploiement

Nous pouvons résumer l'architecture de notre solution d'un point de vue infrastructure dans un digramme de déploiement expliquant son déploiement sur le cluster comme suit :

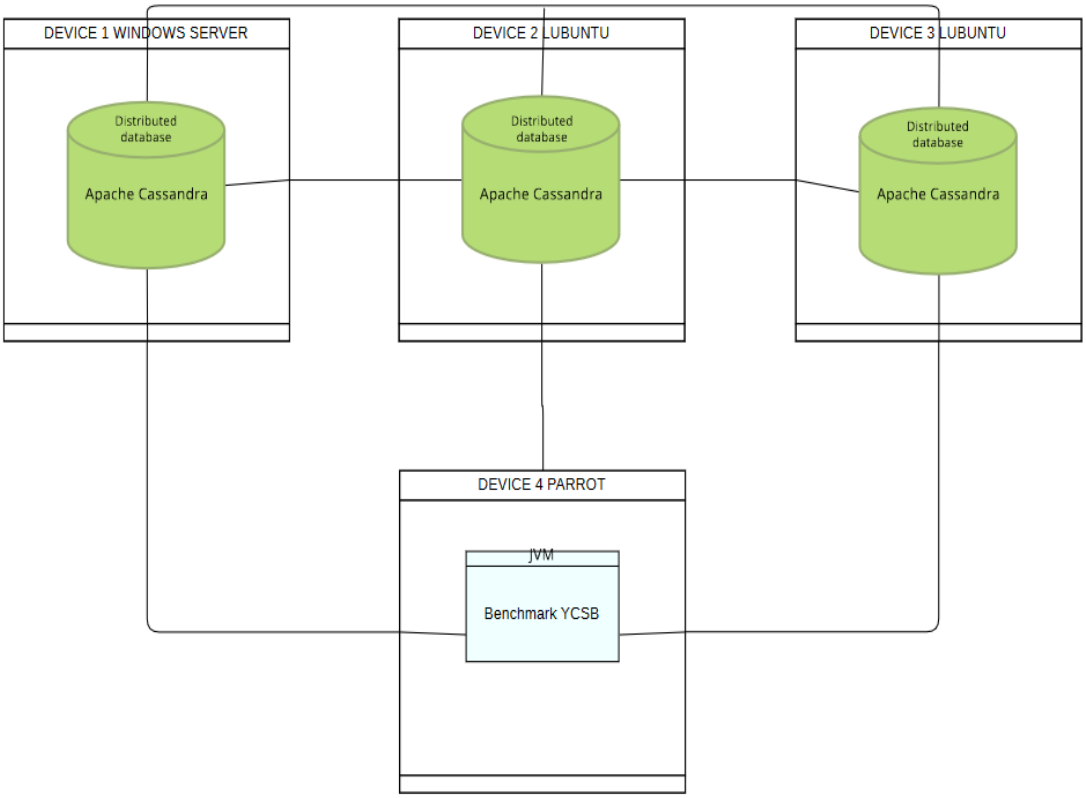


Figure 4.9 Diagramme de déploiement

4.6 Conclusion

Nous avons au cours de ce chapitre présenté l'architecture détaillée de notre solution qui servira de moyen d'évaluation des performances des différents niveaux de cohérences afin de mener notre étude comparative, cela s'effectuera en simulant un nombre élevé de requêtes et une quantité de données proche du Big Data.

Cette architecture présentait trois différentes phases, que nous avons détaillé en expliquantl'implémentation des algorithmes de gestions de la cohérence adaptable, puis ensuite en énonçant les principaux prérequis et étapes suivies afin de configurer l'environnement nécessaire pour le déploiement de la solution.

Enfin, nous avons clôturé le chapitre en présentant un diagramme de déploiement qui a donné une idée sur l'architecture de notre solution d'un point de vue infrastructure.

Après l'étape d'implémentation et de déploiement, nous présenterons dans le chapitre suivant l'étape de tests et discussion des résultats qui apportera une réponse aux performances de la cohérence adaptable.

Chapitre 5: Tests et Discussion Des Résultats

5.1 Introduction

Après avoir présenté la mise en œuvre de notre travail, nous arrivons maintenait au point décisif de notre comparaison où l'on pourra observer l'impact de la cohérence sur un système et se référant aux autres niveaux de cohérence.

Dans ce qui suit nous présenterons les tests effectués sur les différents niveaux de cohérence ainsi que les différentes configurations utilisées.

5.2 Plan de tests

Afin d'entamer nos tests et obtenir des résultats, nous avons tenu compte de certains paramètres essentiels qui seront présentés dans ce qui suit.

5.2.1 Niveau de cohérence

En premier lieu nous avons fixé les différentes combinaisons de cohérence possibles citées dans la partie conception en se basant sur les niveaux de cohérence qu'offre Apache Cassandra :

- **Cohérence Faible :**
  - ONE-ONE
  - QUORUM-ONE
  - ONE-QUORUM
  - ONE-ANY
- **Cohérence Forte :**
  - ONE-ALL
  - QUORUM-QUORUM
  - QUORUM-ALL
  - ALL-QUORUM
  - ALL-ALL

5.3 Résultats et discussions

Nous allons dans cette partie présenter les différents graphes obtenus des tests effectués. Nous rappelons que nos tests sont effectués sur un cluster Cassandra de 3 nœuds.

5.3.1 Configuration 1

La 1ere configuration testée consistait à appliquer une cohérence ONE-ONE pour la phase faible et une cohérence ONE-ALL pour la phase forte après le basculement. Les résultats obtenus en latence (lecture et écriture) et en débit dans les figures suivantes :

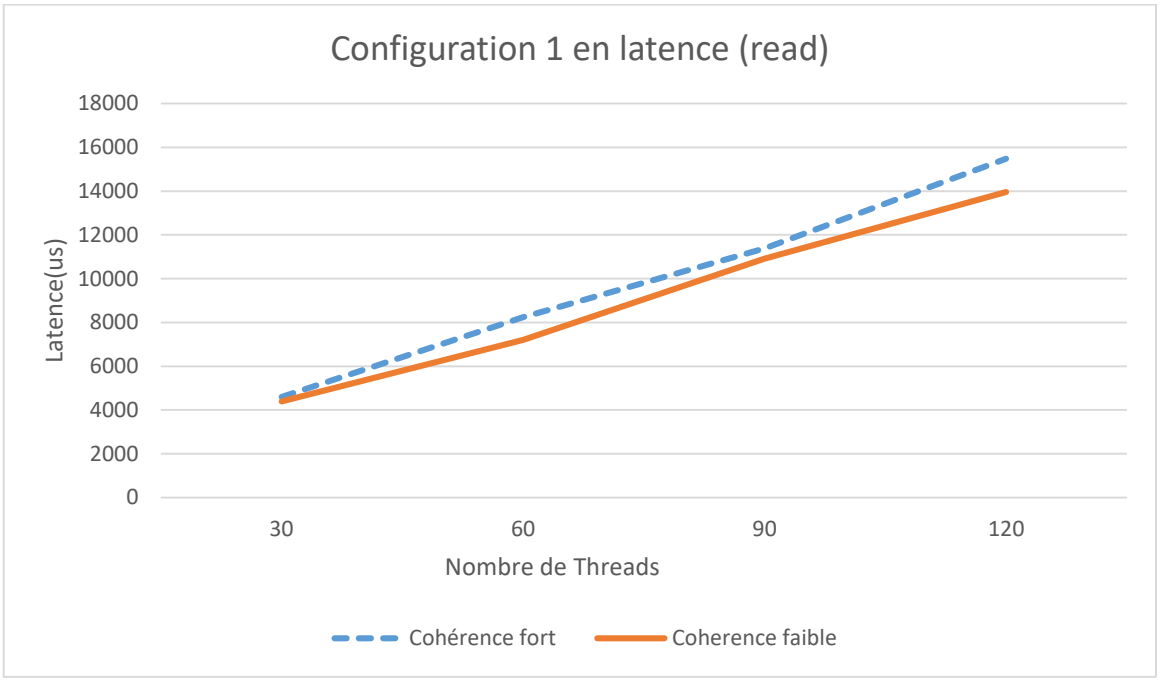


Figure 5.1 Configuration 1 en latence(Read)

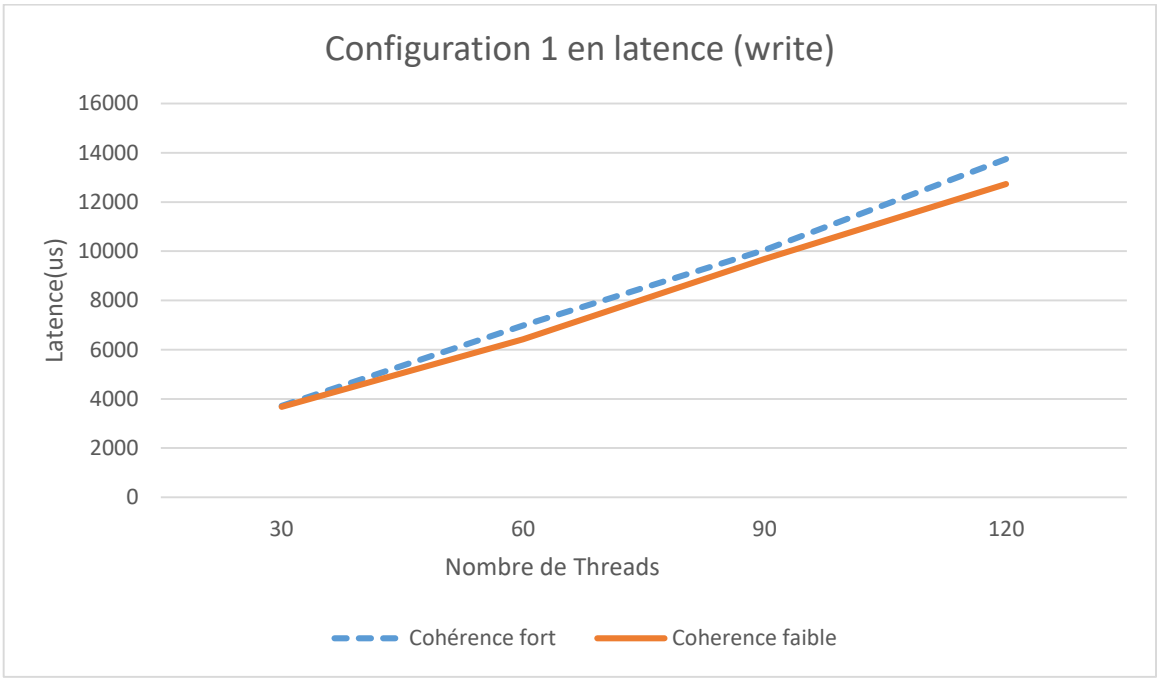


Figure 5.2 Configuration 1 en latence (Write)

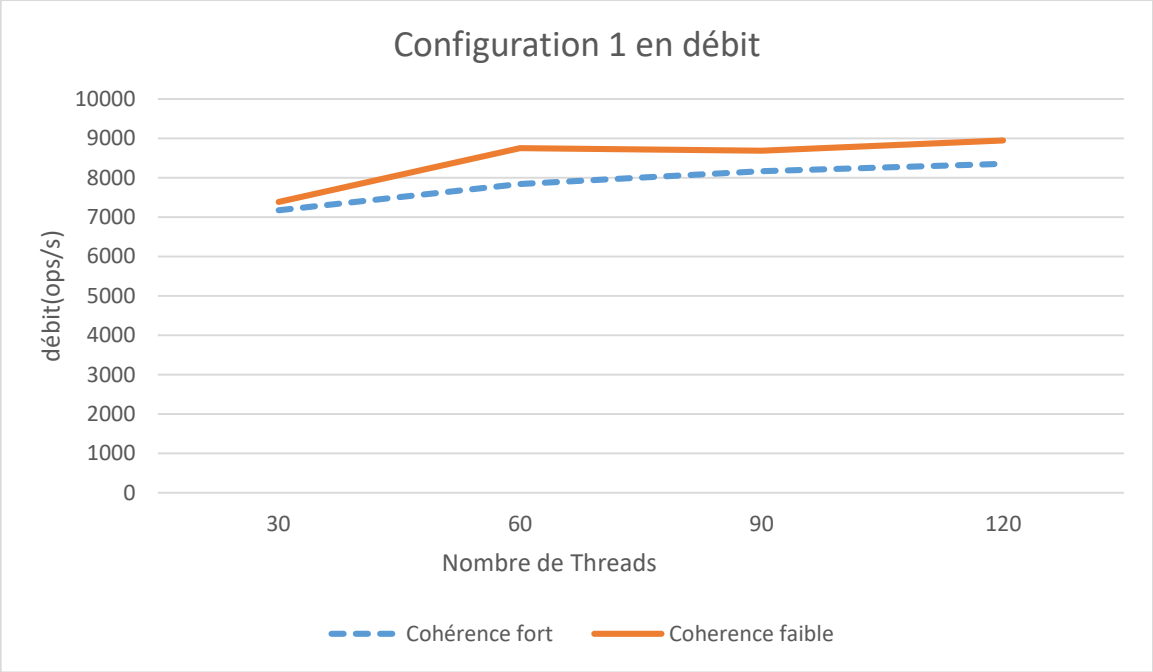


Figure 5.3 Configuration 1 en débit

Nous constatons que la cohérence faible offre de meilleurs résultats en termes de latence pour la lecture et écriture, nous remarquons que l'écart de latence à augmenter avec l'augmentation de nombre de Threads donc l'écart de latence entre la cohérence faible et forte étant important, cela a permis de consacrer une portion de ce temps au gestionnaire de cohérence afin de gérer les basculements entre cohérences.

Pour le débit, nous remarquons que la cohérence faible présente un débit plus élevé que la cohérence forte, c'est dû aux opérations de mises à jour ignorées après l'arrivée des enchères à leur fin.

5.3.2 Configuration 2 :

La 2eme configuration testée consistait à appliquer une cohérence QUORUM-ONE pour la phase faible et une cohérence ONE-ALL pour la phase forte après le basculement. Les résultats obtenus en latence (lecture et écriture) et en débit dans les figures suivantes :

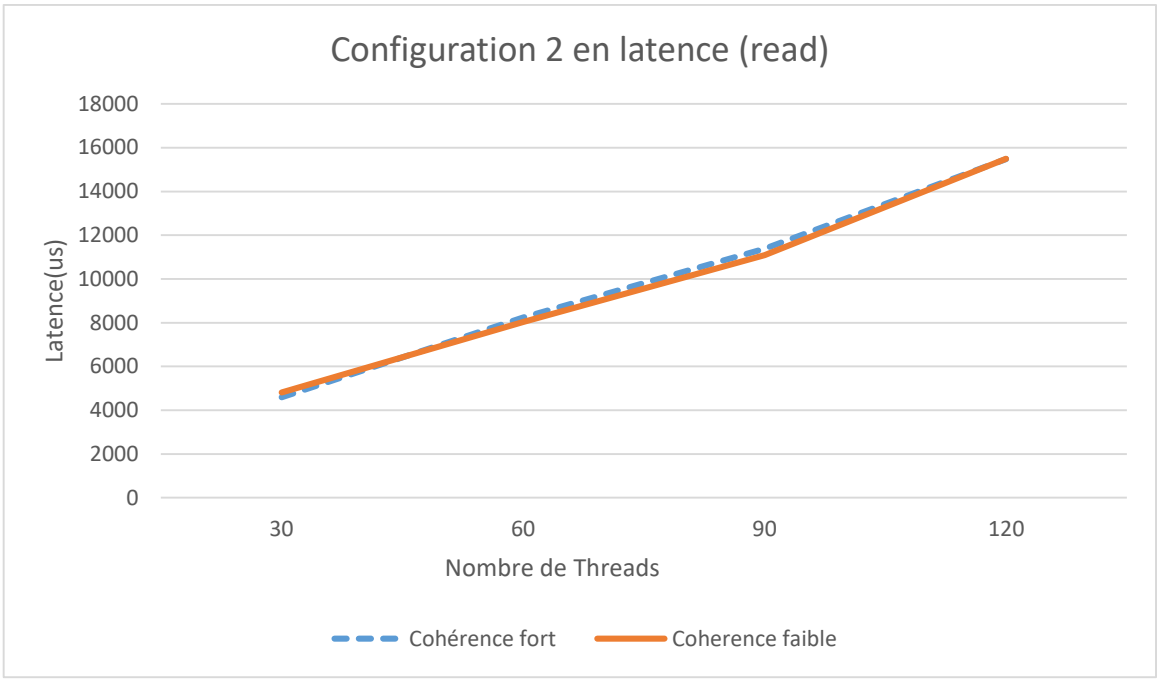


Figure 5.4 Configuration 2 en latence(Read)

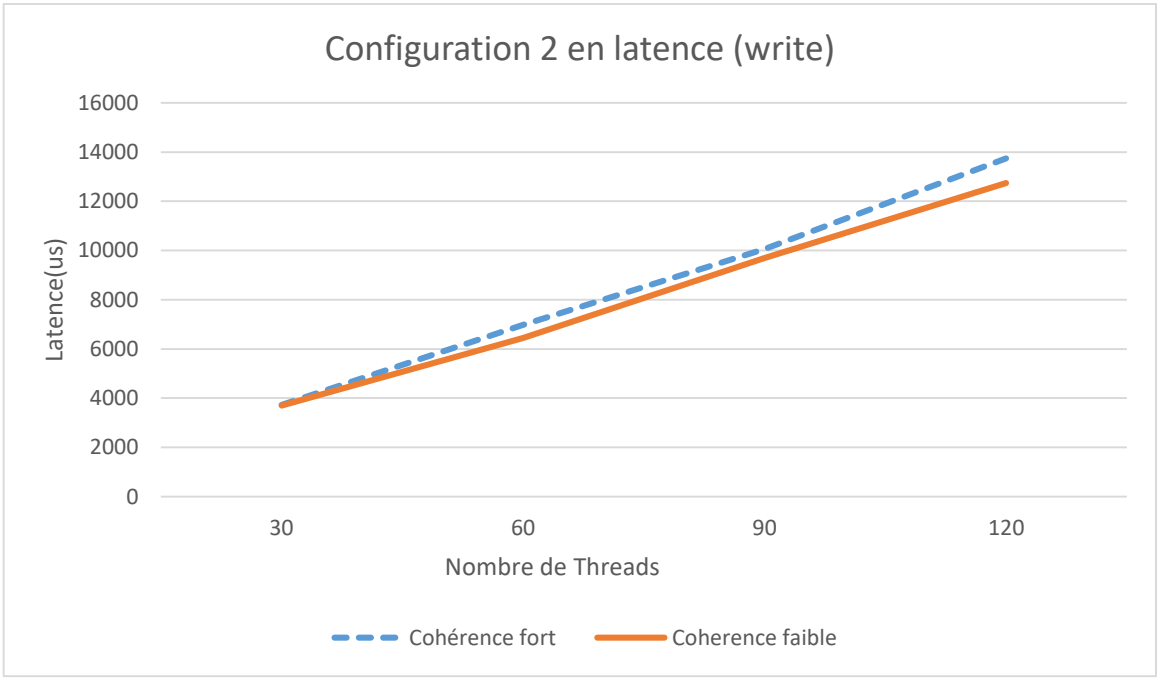


Figure 5.5 Configuration 2 en latence(Write)

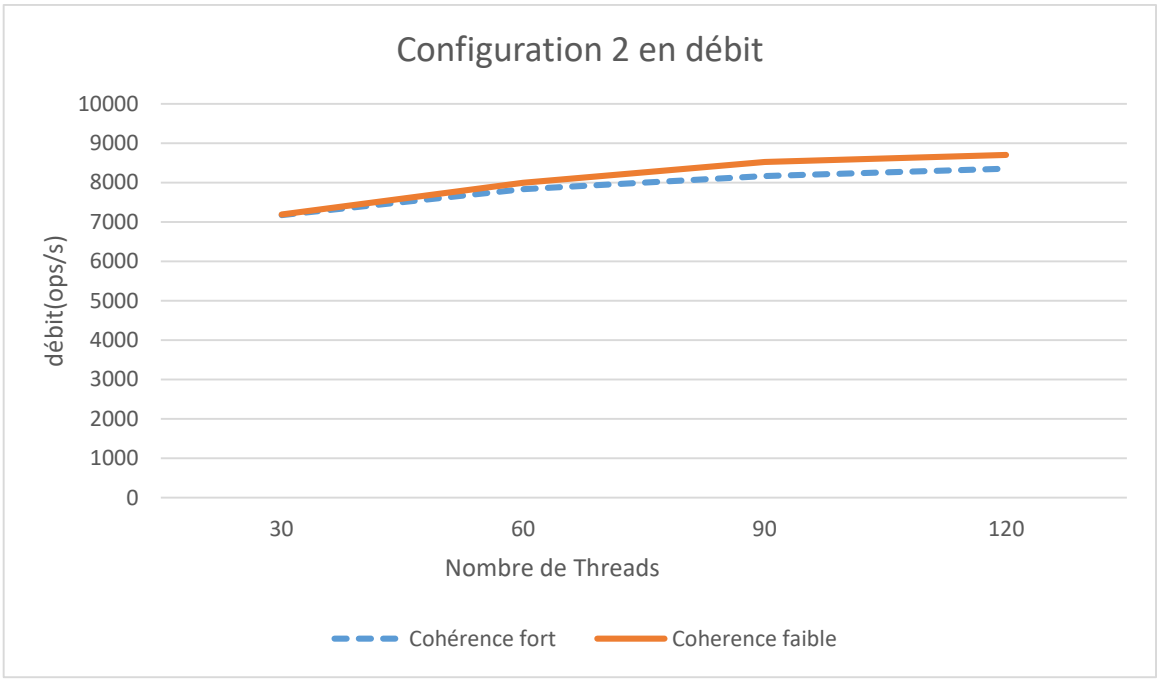


Figure 5.6 Configuration 2 en débit

Dans cette configuration nous remarquons que la latence est très proche en lecture avec un peu d'écart dans le cas d'écriture car on a 3 nœuds, Cela est dû au rapprochement de latence des opérations d'écriture sous Cassandra (QUORUM et ONE) car les opérations d'écriture sous Cassandra sont largement moins coûteuses et donc la différence de latence entre une écriture en ONE et une écriture en QUORUM est insignifiante. Cela n'a donc pas empêché d'avoir un écart de latence entre la cohérence faible et forte qui a offert une opportunité de sacrifier une portion minime pour le gestionnaire de cohérences afin de gérer les basculements entre cohérences.

De même pour le débit, il est meilleur en cohérence faible qu'en cohérence forte, et pour la même raison citée dans la configuration précédente, l'écart de débit reste toujours plus important.

5.3.3 Configuration 3 :

La 3eme configuration testée consistait à appliquer une cohérence ONE-ANY pour la phase faible et une cohérence QUORUM-QUORUM pour la phase forte après le basculement. Les résultats obtenus en latence (lecture et écriture) et en débit dans les figures suivantes :

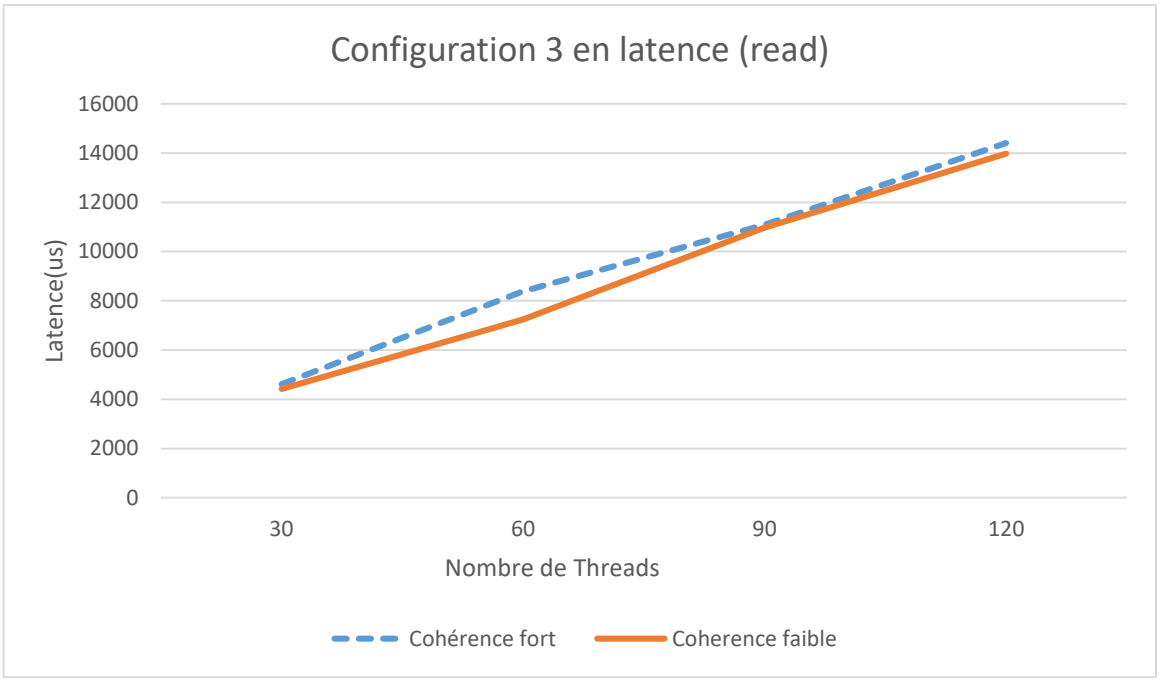


Figure 5.7 Configuration 3 en latence(Read)

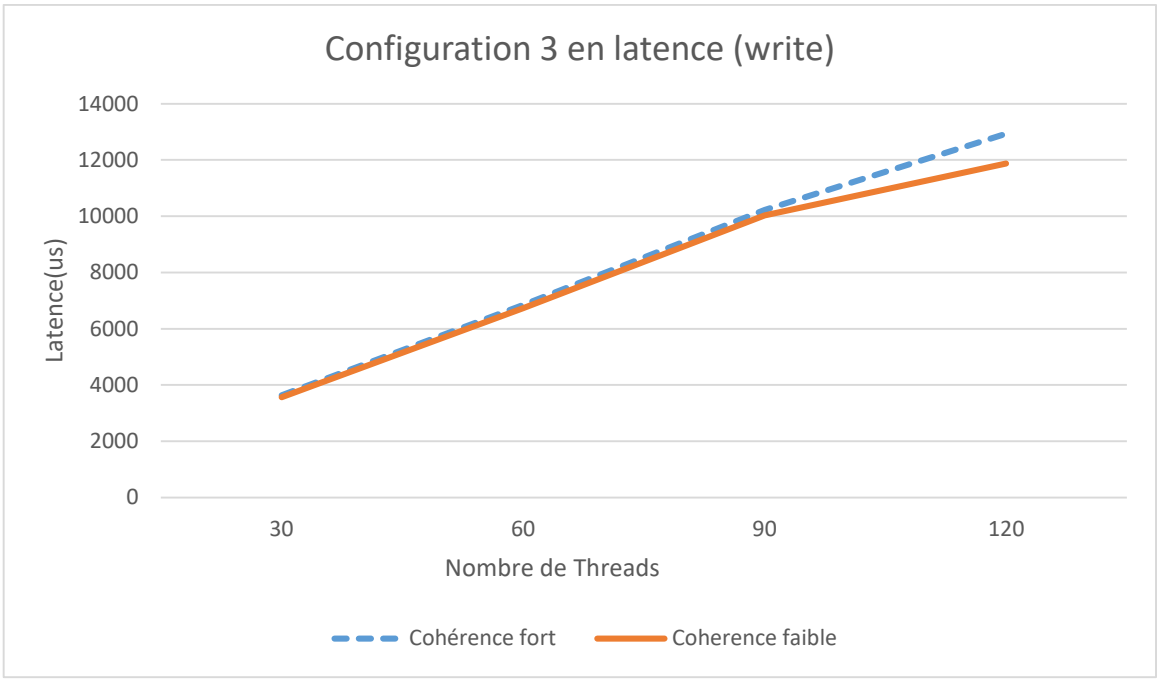


Figure 5.8 Configuration 3 en latence(Write)

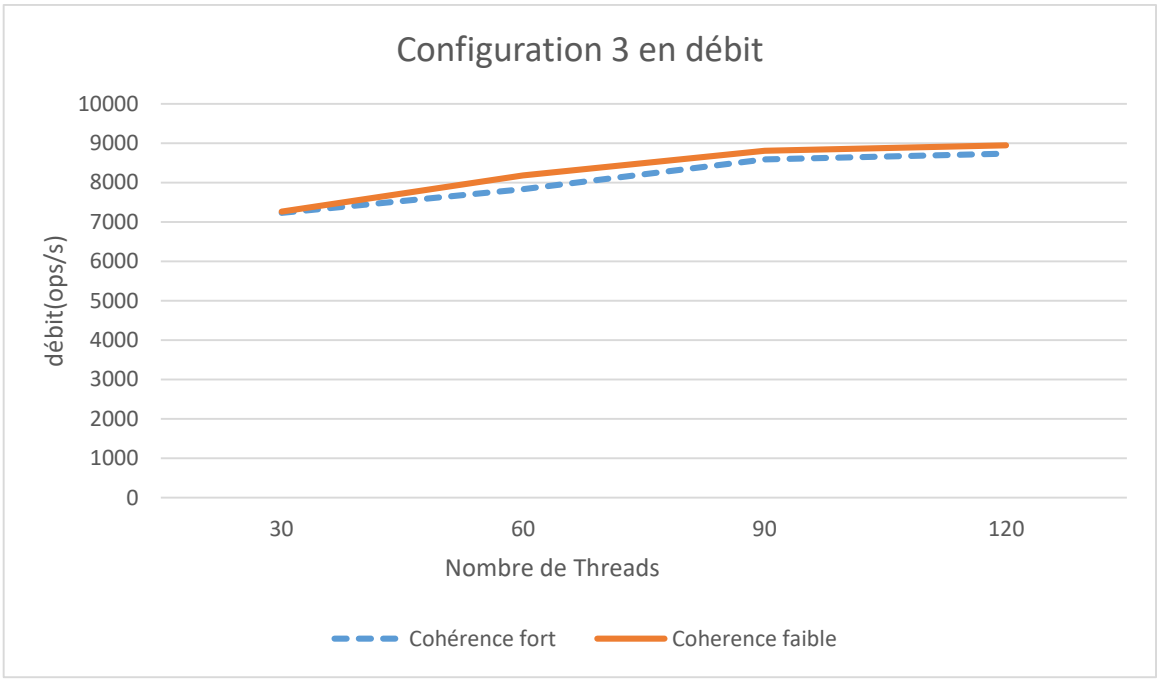


Figure 5.9 Configuration 3 en débit

Cette configuration présente des résultats presque semblables à ceux de la 1ère configuration.

De même pour le débit, il est meilleur en cohérence faible qu'en cohérence forte, et pour la même raison citée dans la configuration précédente.

Nous remarquons que ce soit en débit ou en latence, l'augmentation du nombre de threads avait toujours pour effet d'augmenter la latence et de réduire le débit.

5.3.4 Configuration 4 :

La 4eme configuration testée consistait à appliquer une cohérence ONE-QUORUM pour la phase faible et une cohérence ALL-QUORUM pour la phase forte après le basculement. Les résultats obtenus en latence (lecture et écriture) et en débit dans les figures suivantes :

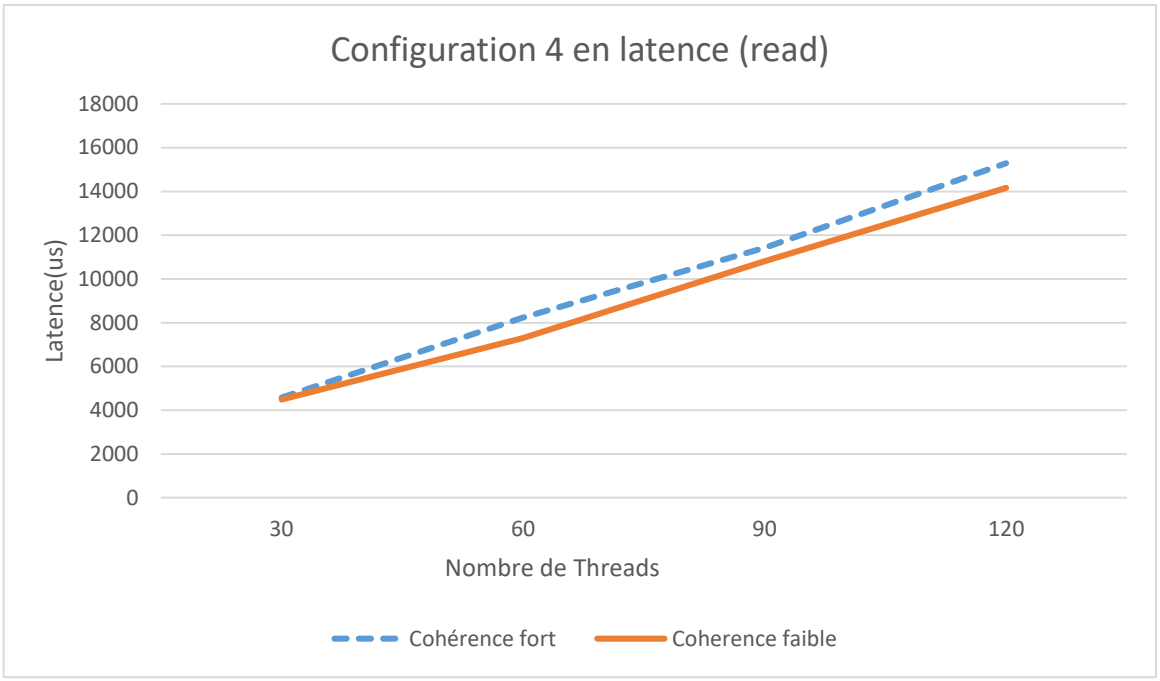


Figure 5.10 Configuration 4 en latence(Read)

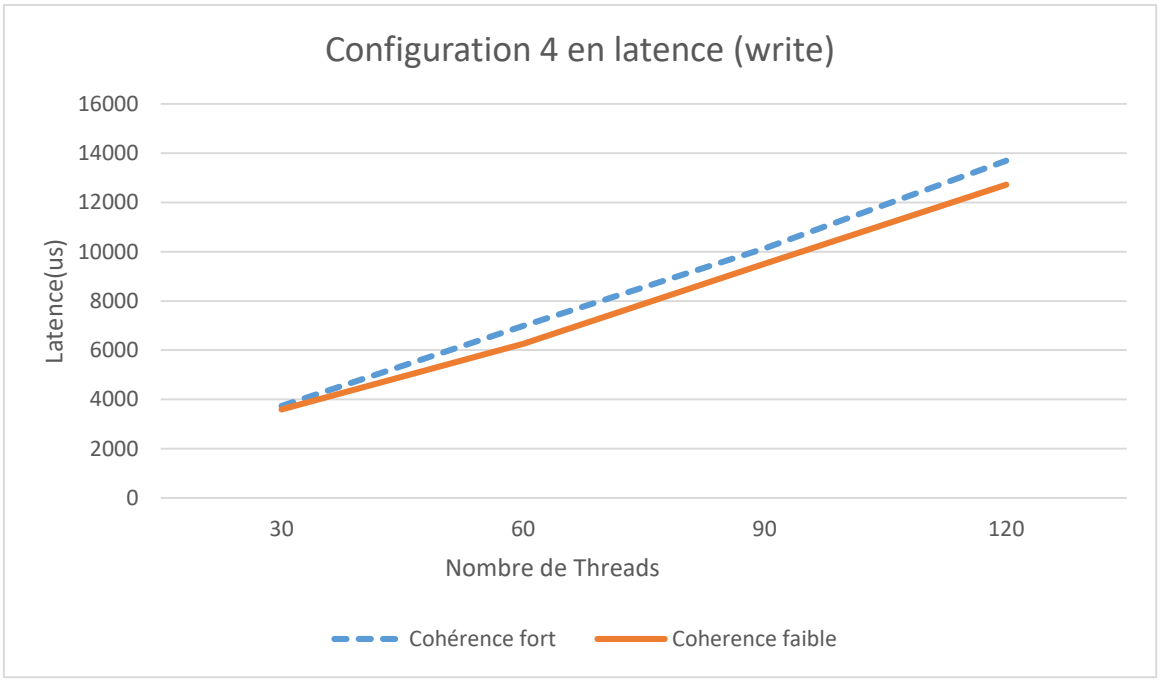


Figure 5.11 Configuration 4 en latence(Write)

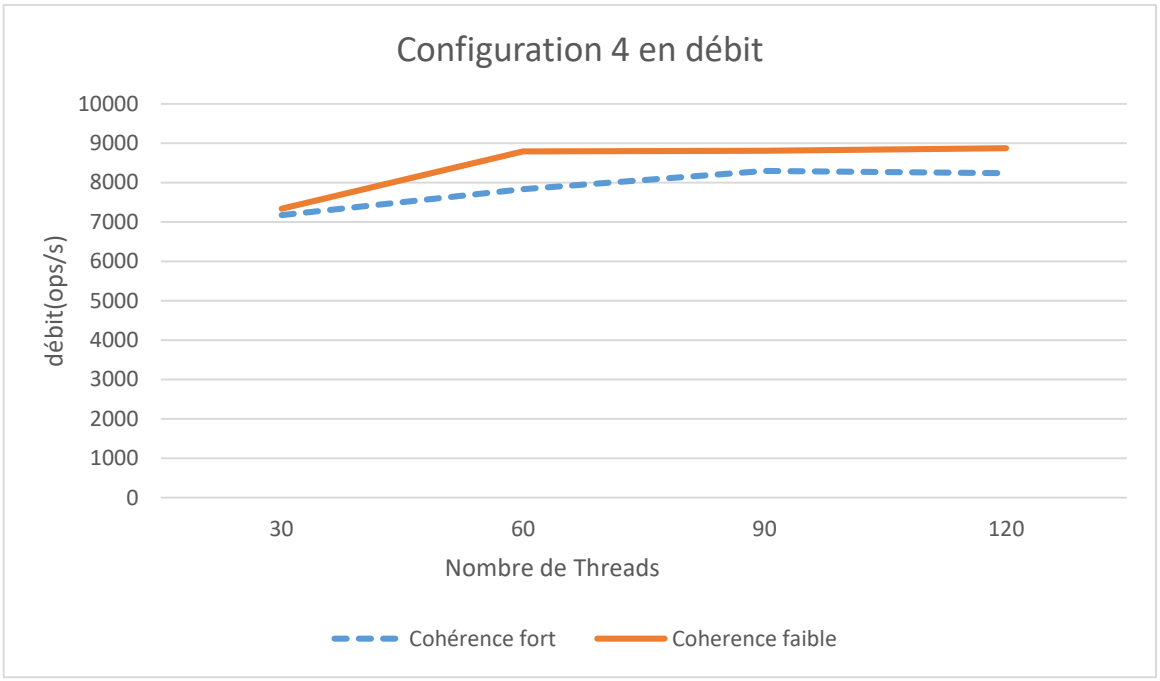


Figure 5.12 Configuration 4 en débit

5.3.5 Configuration 5 :

La 5eme configuration testée consistait à appliquer une cohérence ONE-QUORUM pour la phase faible et une cohérence QUORUM -ALL pour la phase forte après le basculement. Les résultats obtenus en latence (lecture et écriture) et en débit dans les figures suivantes :

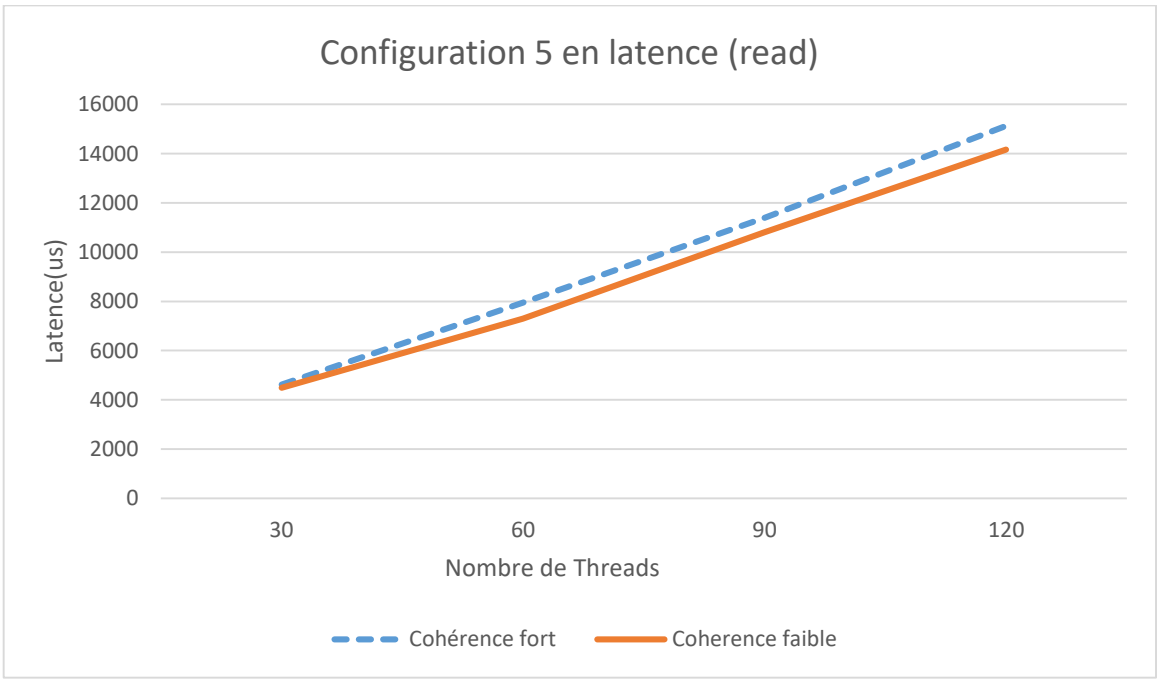


Figure 5.13 Configuration 5 en latence(Read)

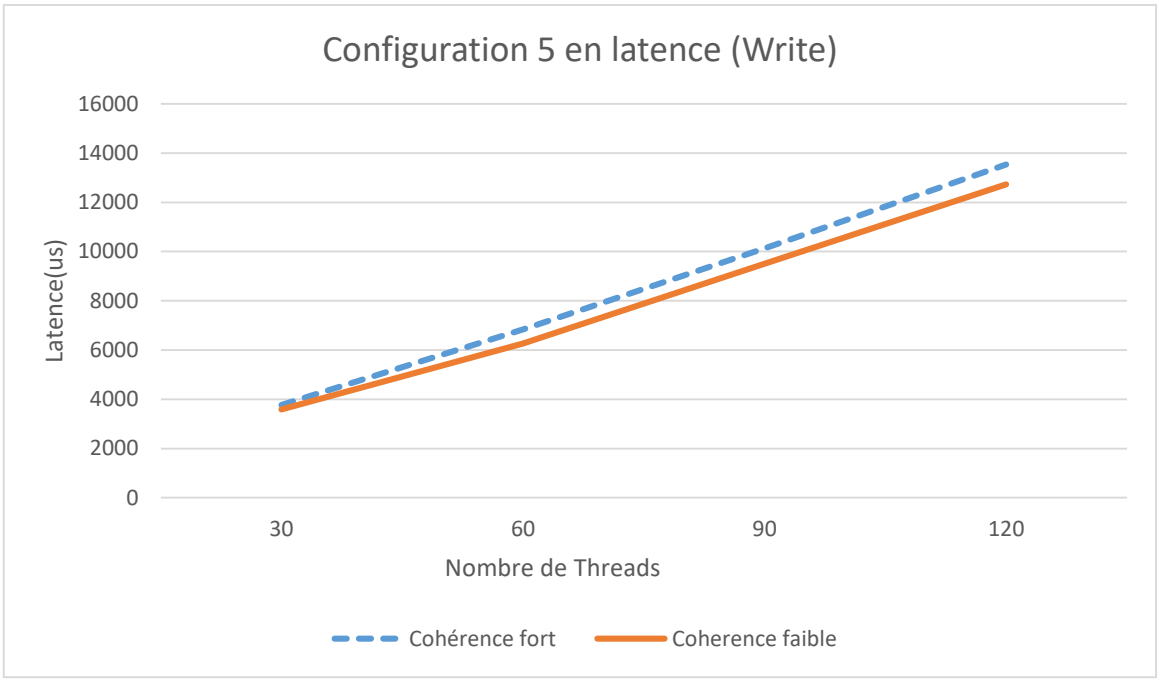


Figure 5.14 Configuration 5 en latence(Write)

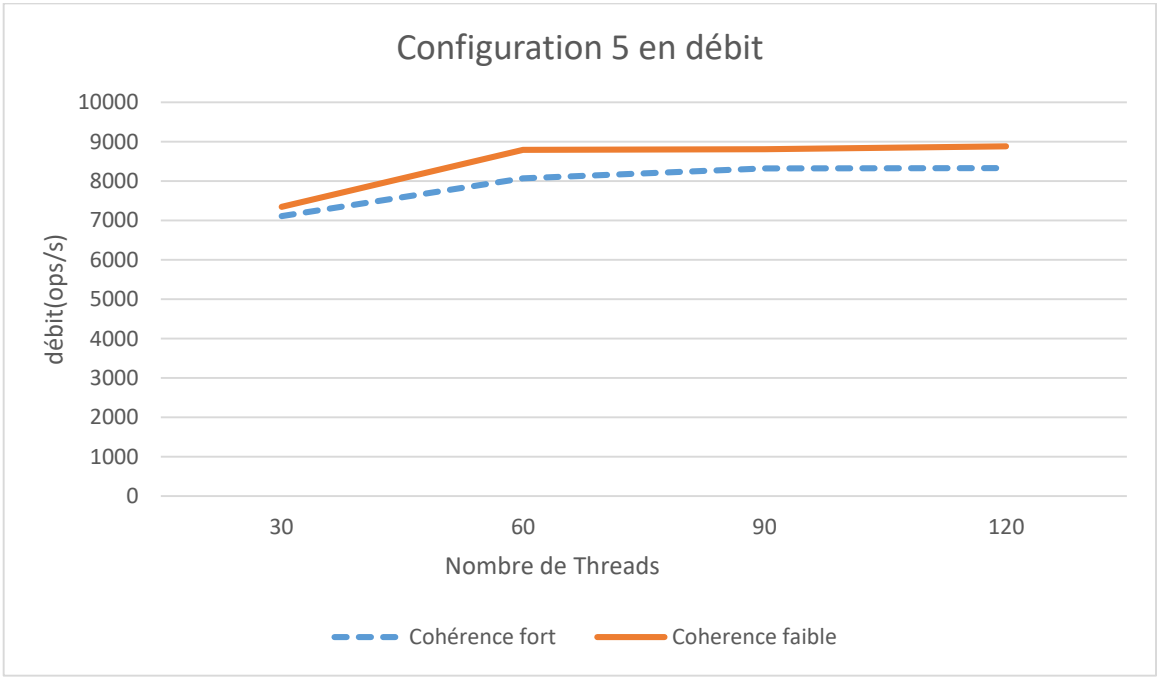


Figure 5.15 Configuration 5 en débit

5.3.6 Configuration 6 :

La 6eme configuration testée consistait à appliquer une cohérence ONE-ONE pour la phase faible et une cohérence ALL -ALL pour la phase forte après le basculement. Les résultats obtenus en latence (lecture et écriture) et en débit dans les figures suivantes :

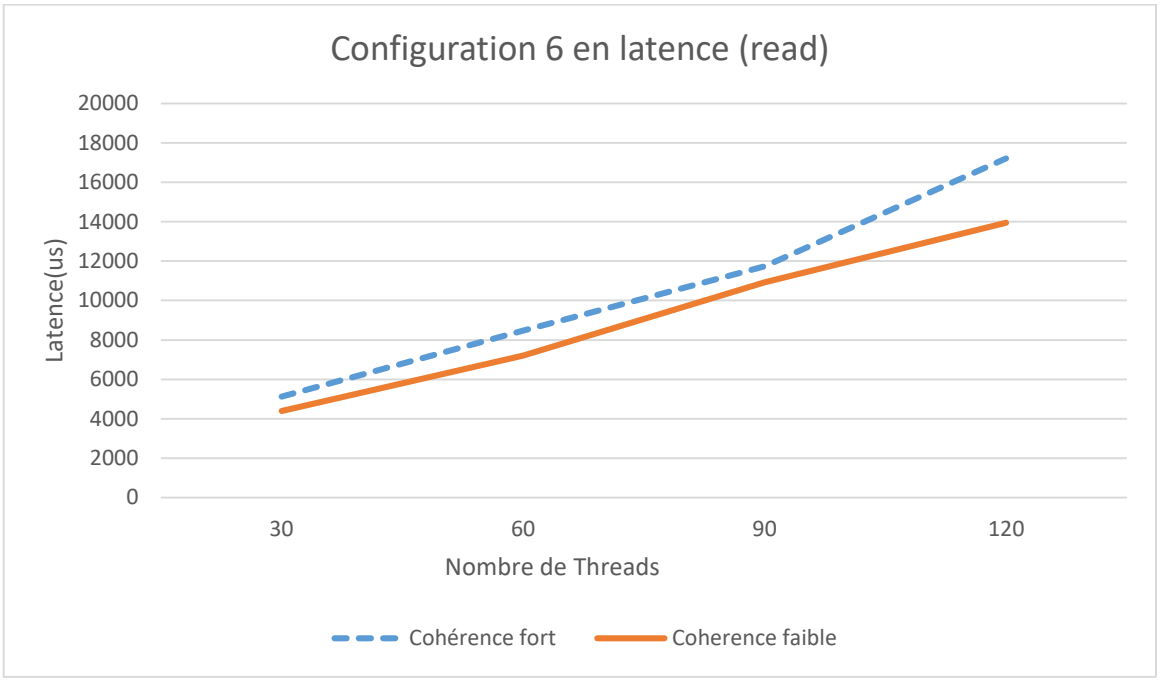


Figure 5.16 Configuration 6 en latence(Read)

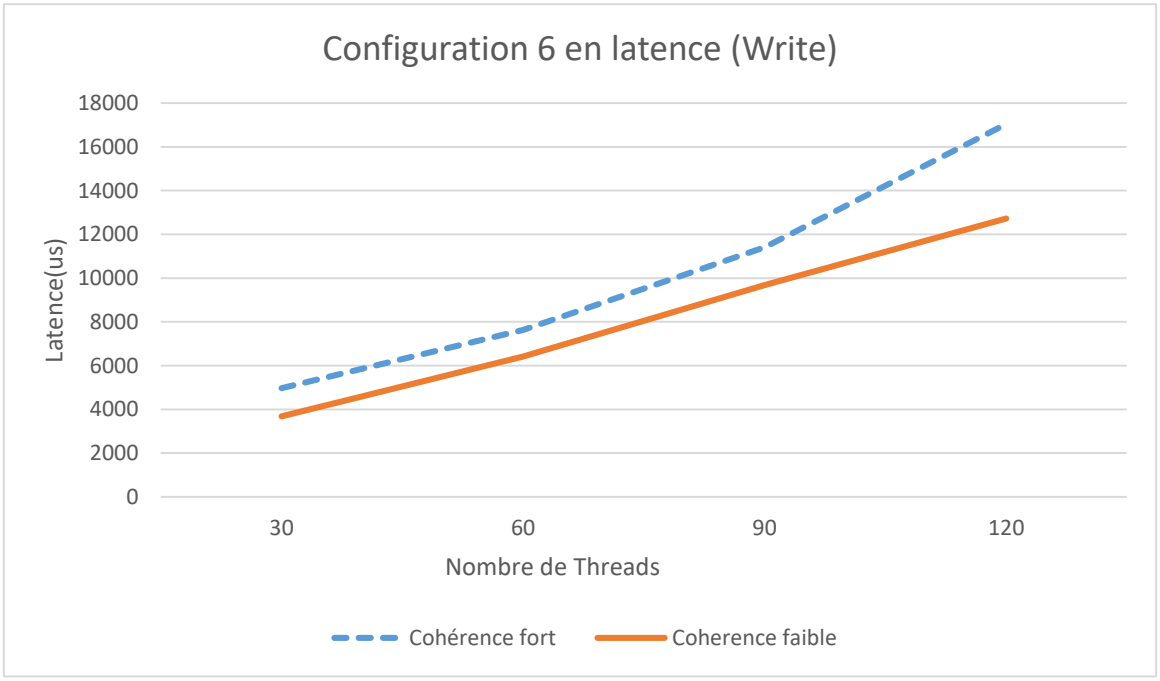


Figure 5.17 Configuration 6 en latence (Write)

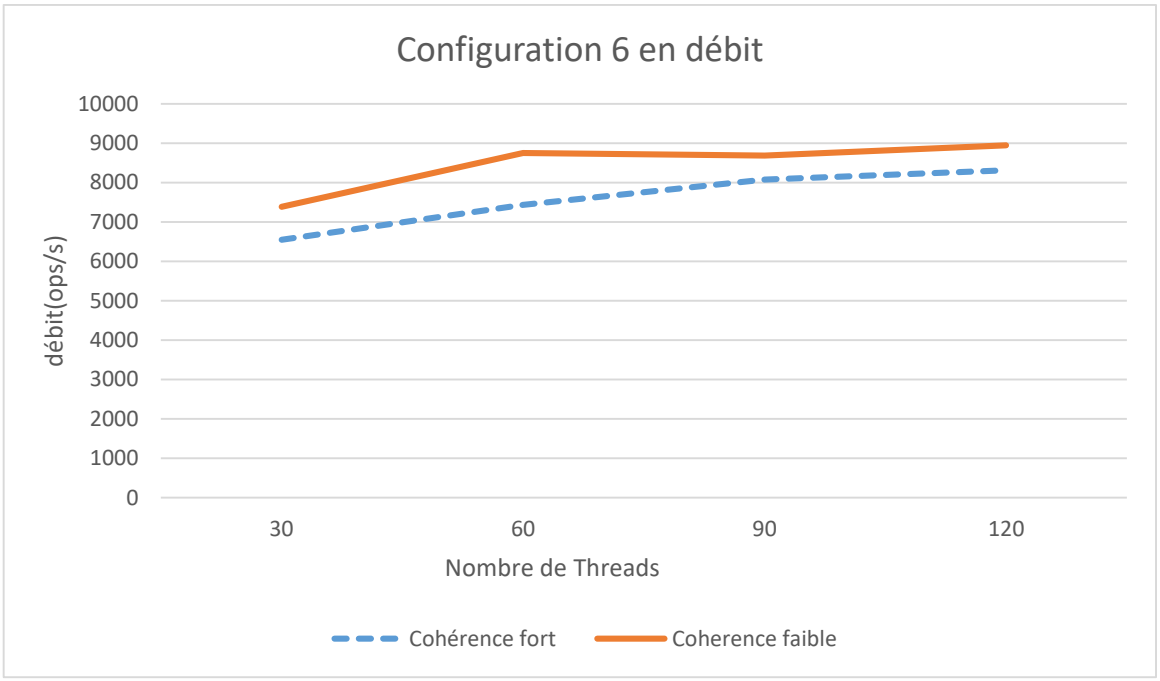


Figure 5.18 Configuration 6 en débit

5.3.7 Comparaison :

Après avoir récolté les résultats de chaque test en variant les niveaux de cohérences qu'offre Apache Cassandra, et après avoir présenté ces résultats sous formes de graphes, nous allons à présent établir une comparaison entre les quatre configurations afin de décider de celle qui présente de meilleurs performances en latence et en débit.

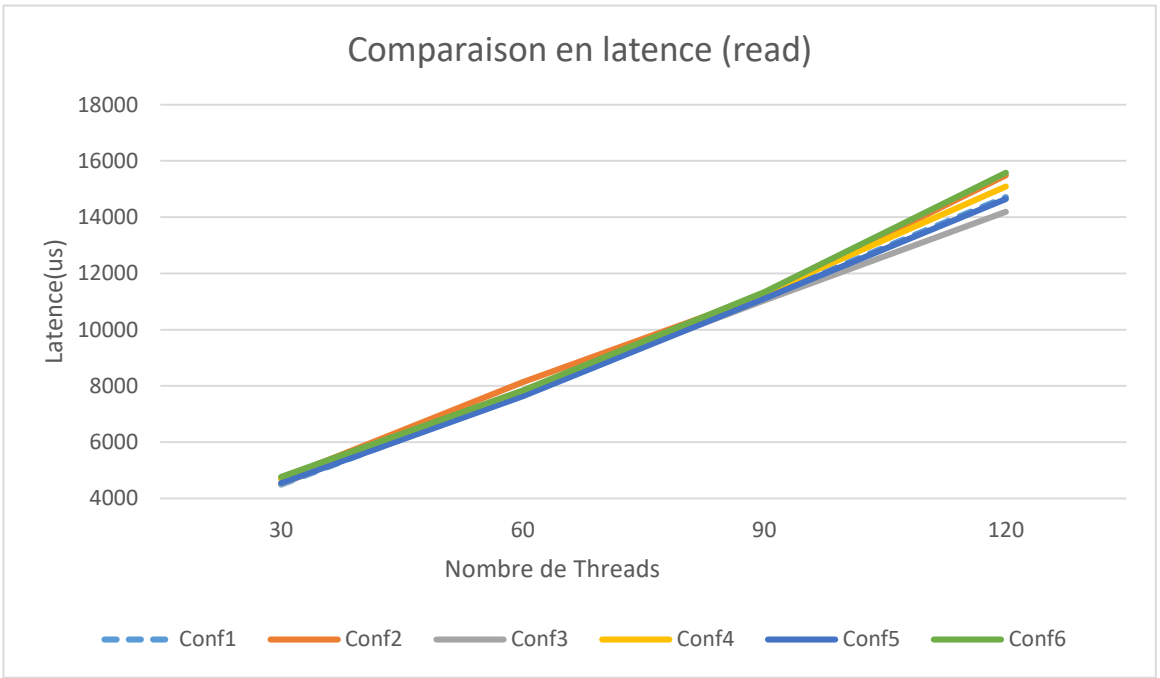


Figure 5.19 Comparaison en latence (read)

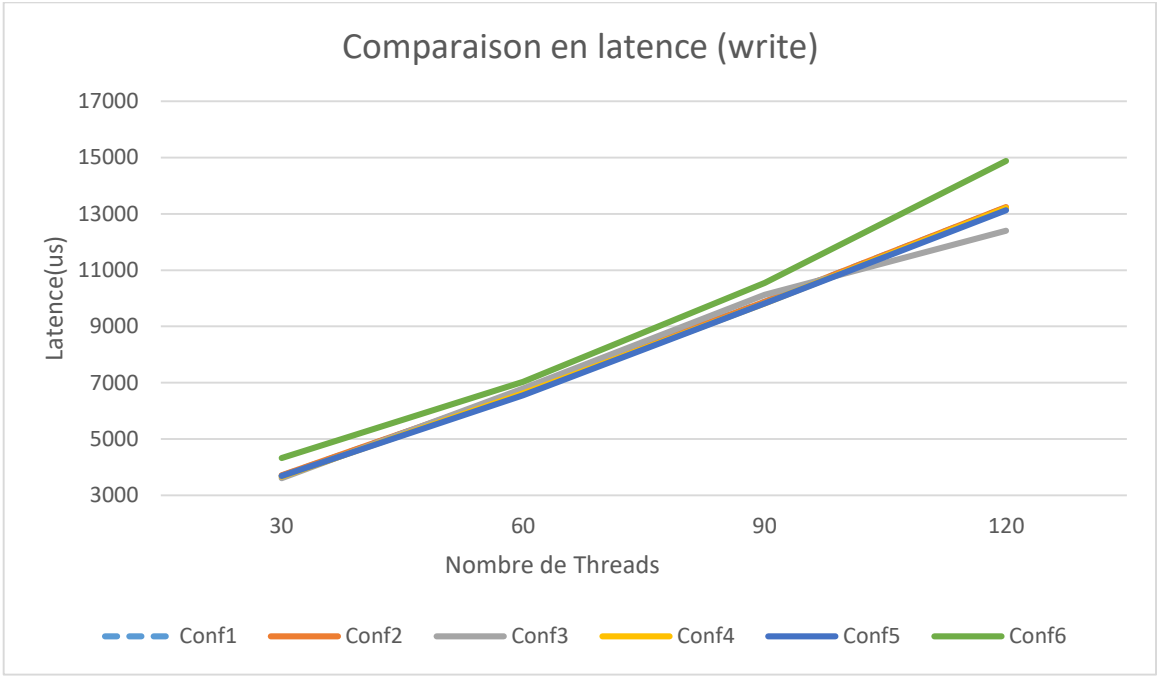


Figure 5.20 Comparaison en latence (write)

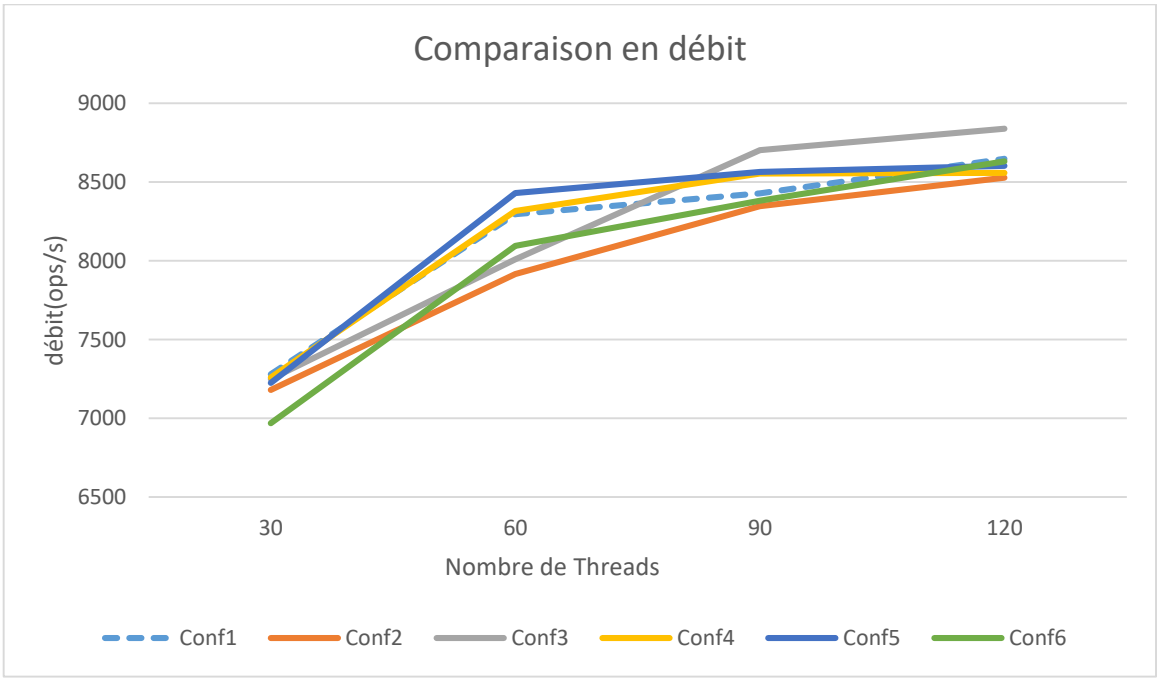


Figure 5.21 Comparaison en débit

D’après les figures ci-dessus, nous remarquons que la meilleure configuration aussi bien en latence qu’en débit, est la troisième configuration (Conf3) ONE - ANY / QUORUM - QUORUM qui offre une latence moyenne approchant les 14000us en lecture et 12000 en écriture, et un débit approchant les 8700 opérations/sec. Cette configuration présente de meilleurs résultats car les lectures et écritures lors de la phase de cohérence faible s’effectuent sur un seul réplica (ONE), de ce fait elle présente des temps de réponse réduits qui vont compenser les accès effectué à Cassandra afin de renseigner le gestionnaire de cohérences sur l’état des enchères.

5.4 Conclusion

Nous avons pu à travers ce chapitre, voir les performances des différents modèles de cohérence, nous avons pu effectuer à travers les graphes présentés une comparaison des différents modèles en discutant chacun des résultats obtenus.

Conclusion générale :

Dans les prochaines années, nous estimons que les technologies des Big Data seront de plus en plus utilisées pour répondre à de nouvelles problématiques pour la gestion de données dans des environnements à grande échelle. Les systèmes NoSQL se placent comme les solutions idéales pour gérer les grands volumes de données, la variété de leurs types caractérisant principalement le concept du Big Data.

Plusieurs solutions NoSQL basées sur des architectures différentes sont développées dans tous les secteurs. La tendance vers une favorisation d'une solution NoSQL précise est loin d'être réalisée à cause du nombre important des solutions existantes et l'absence de normalisation. Actuellement, plusieurs solutions open-source et payantes sont présentées aux acteurs des différents secteurs liés au Big Data, qui mettent les utilisateurs devant un embarras dans le choix du modèle approprié par rapport à leur environnement d'exploitation.

L'objectif ciblé de ce memoire consistait à comparer les niveaux de cohérence dans les bases de données NoSQL.

Après l'étude menée, nous pouvons affirmer que le choix d'utilisation d'un modèle ou un autre, parmi les solutions disponibles dans le marché, dépend de l'environnement dans lequel les données sont exploitées. En effet le type de données et leur traitement sont des indicateurs importants pour définir la ou les solutions utilisées. La fréquence estimée de lecture, d'écriture et de mise à jour ainsi que la taille des données sont les facteurs essentiels pour la sélection de la solution NoSQL appropriée.

Perspectives

Plusieurs travaux de recherche se focalisent actuellement sur le sujet qui se voit très vaste et très ouvert, d'autres comparaisons entre les différentes familles des solutions NoSQL, orientée document, orientée colonne, clé valeur et orientée graphe, dans les environnements Cloud, sont toujours très sollicitées et recommandées pour apporter l'assistance et l'aide nécessaire aux intéressés de Big Data et de Cloud Computing, pour des éventuelles prises de décision sur les solutions convenables.

Perspectives

Enfin, on peut estimer que l'objectif tracé au préalable a été largement atteint, néanmoins ce travail pourrait être complété et prolongé sur plusieurs aspects, ainsi on peut souligner un ensemble de perspectives et de piste de recherches à explorer, citons :

- Pour obtenir les meilleurs résultats il faut utiliser serveur puissant.

- La cohérence elle a la relation avec le matériel et le débit de connexion.

- Etaler notre étude à d'autres solutions NoSQL à savoir : Elasticsearch, Memcached, Amazon DynamoDB, CouchDB, Accumulo et autres.

- L'influence de l'utilisation d'autres modèles de cohérence peut jouer un rôle dans les performances de la cohérence.

En effet cette perspective peut être largement étudiée afin de d'étudier le compromis entre la cohérence et les performances d'accès.

Bibliographie

- ✓ [Brian F COOPER et al. 2010] Brian F COOPER et al. « Benchmarking cloud serving systems with YCSB ». In : *Proceedings of the 1st ACM symposium on Cloud computing*. ACM. 2010,p. 143–154.
- ✓ [A. Tanembaum] Réseaux et systèmes distribués.
- ✓ [Angeline, 2013] KONE. Angeline. Big data rapport de stage, 2013.
- ✓ [Aurélien FAUCRET] Aurélien FAUCRET, NoSQL, une nouvelle approche de stockage et de manipulation des données, Octer A., En savoir plus sur les bases de données NoSQL.
- ✓ [BERTHE.2015] ibrahim BERTHE. A la d´découverte de nosql, 31/10/2015.
- ✓ [Bloor, 2004] Robin Bloor, L'échec des Bases de Données Relationnelles, L'essor de la Technologie Objet et le Recours aux Bases de Données Hybrides. Baroudi Bloor International Inc., 2003, 2004.
- ✓ [BUGBEE.1917] Kurt BUGBEE. « Benchmarking Big Data Cloud-Based Infrastructures ». Thèsede doct. WORCESTER POLYTECHNIC INSTITUTE, 1917.
- ✓ [Chiky .2016] R. Chiky. Cours de Bases de données NoSZL. Telecom Paris Tech. 2016[.
- ✓ [Cornuéjols.2009-2010] Antoine Cornuéjols, BASES DE DONNÉES CONCEPTS ET PROGRAMMATION, AgroParisTech, Spécialité Informatique (2009-2010).
- ✓ [Di Maglie.2012] Matteo Di Maglie, Adoption d'une solution NoSQL dans l'entreprise, Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES, Carouge, 12 septembre 2012 Haute École de Gestion de Genève (HEG-GE.
- ✓ [Di Maglie2012] Matteo Di Maglie, Adoption d'une solution NoSQL dans l'entreprise, Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES, Carouge, 12 septembre 2012 Haute École de Gestion de Genève (HEG-GE).
- ✓ [Djebbara et Belbachir 2010] Réda M. Djebbara et Hafida Belbachir: Dynamic threshold for replicas placement strategy; International Conference on Machine and Web Intelligence (ICMWI), pp. 394 – 401, Alger 2010
- ✓ [Emmanuel 2012] Kouedi Emmanuel. Approche de migration d'une base de données relationnelle vers une base de données nosql orientée colonne. Mémoire master informatique, Université de Yaoundé I, 2012.

- ✓ [Exbrayat 2007] Matthieu Exbrayat ; Bases de Données Réparties: Concepts et Techniques, notes de cours, ULP Strasbourg, Décembre 2007.
- ✓ [Foster 2002] Ian Foster; What is the Grid? A Three Point Checklist; Grid Today, Vol. 1, No. 6, juillet 2002
- ✓ [Furrer 2015] Hadrien Furrer. Sql, nosql, newsql stratégie de choix. PhD thesis, Haute ´école de gestion de Genève, 2015.
- ✓ [Furrer. 2015] Hadrien Furrer. Sql, nosql, newsql stratégie de choix. PhD thesis, Haute école De gestion de Geneve, 2015.
- ✓ [Gabillaud 2009] Jérôme Gabillaud. Oracle 11g: SQL, PL/SQL, SQL* Plus. Editions ENI, 2009.
- ✓ [Gabillaud 2009] J´erome Gabillaud. Oracle 11g : SQL, PL/SQL, SQL* Plus. Editions ENI, 2009.
- ✓ [Grin 2006] Richard Grin. Langage sql. 2006.
- ✓ [Hamel and Marguerit 2013] Marie-Pierre Hamel and David Marguerit. Quelles possibilités offertes par l'analyse des big data pour améliorer les téléservices publics ? Revue française d'administration publique, 2013.
- ✓ [Hutto et Ahamad 1990] Phillip W. Hutto et Mustaque Ahamad; Slow memory: Weakening consistency to enhance concurrency in distributed shared memories; 10th International Conference on Distributed Computing Systems (ICDCS '90), pp. 302-311, Paris, Mai 1990.
- ✓ [Iftode et al. 1996] Liviu Iftode, Jaswinder Pal Singh et Kai Li; Scope consistency: A bridge between release consistency and entry consistency; 8th ACM Annual Symposium on Parallel Algorithms and Architectures (SPAA '96), pp. 277-287, Italie, Juin 1996.
- ✓ [Katal and al.2013] Avita Katal, Mohammad Wazid, and RH Goudar. Big data : issues, challenges, tools and good practices. In Contemporary Computing (IC3), 2013 Sixth International Conference on, pages 404–409. IEEE, 2013.
- ✓ [Lakshman and Malik .2010] Avinash Lakshman et Prashant Malik. « Cassandra : a decentralized struc- tured storage system ». In : ACM SIEOPS Operating Systems Review 44.2 (2010), p. 35–40.
- ✓ [LAKSHMAN et MALIK 2010] Avinash LAKSHMAN et Prashant MALIK. « Cassandra : a decentralized struc- tured storage system ». In : ACM SIEOPS Operating Systems Review 44.2 (2010), p. 35–40.

- ✓ [Lamehamedi et al. 2002] Houda Lamehamedi, Boleslaw Szymanski, Zujun Shentu et Ewa Deelman; Data replication strategies in Grid environments; 5th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP‟02), pp. 378–383, Chine, 2002.
- ✓ [Lamport 1978] Leslie Lamport; Time, clocks, and the ordering of events in a distributed system; Communications of the ACM (CACM), vol. 21, No. 7, pp. 558-565, USA, 1978
- ✓ [Laurent Audibert] Laurent Audibert, Base de Données et Langage SQL, Cours-BD, Institut Universitaire de Technologie de Villetaneuse, Département Informatique.
- ✓ [Léonard.2014] Meyer Léonard, L'AVENIR DU NoSQL Quel avenir pour le NoSQL ?, 2014.
- ✓ [MALETRAS 2012] Xavier MALETRAS. Le nosql- cassandra, thèse professionnelle). Université paris 13, 27/05/2012.
- ✓ [Marie-France 2006] Marie-France La salle, Cours Du Relationnel a l'objet: Limites du Relationnel 24/1/2006.
- ✓ [Mathieu Roger] Mathieu Roger, Bases NoSQL, synthèse d'étude et projets d'inter logiciels, octera [AT] octera [DOT] info.
- ✓ [Meier2006] Andreas Meier, Introduction pratique aux bases de données relationnelles, Ed 2, (Collection IRIS), Springer-Verlag France 2002, 2006..
- ✓ [Meylan 2005] Eddy Meylan; Bases de données: Réplication des données; Support de cours, Haute Ecole spécialisée de Suisse Occidentale, Informatique de Gestion, Février 2005.
- ✓ [Monnet 2006] Sébastien Monnet ; Gestion des données dans les grilles de calcul: support pour la tolérance aux fautes et la cohérence des données; Thèse de doctorat, université de rennes 1, Novembre 2006
- ✓ [Moussa 2007] Rim Moussa; Systèmes de Gestion de Bases de Données Réparties & Mécanismes de Répartition avec Oracle; Rapport de recherche, Ecole Supérieure de Technologie et d'Informatique à Carthage 2007
- ✓ [Niobet 2017] Wilfrid Niobet. Informatique documentaire et les bases de données. 2017.
- ✓ [Olivier Losson] Introduction aux systèmes de gestion de bases de données relationnelles, cours master sciences et technologies. Université Lille1.

- ✓ [OZSU ET VALDURIEZ.2011] M. Tamer OZSU et Patrick VALDURIEZ. *Principles of distributed database systems*. Springer Science Business Media. 2011.
- ✓ [pallaw.2010] Vijay Karishna PALLAW. *Database management systems*. New Delhi : AsianBooks Private Ltd. 2010.
- ✓ [Polo. 2015] O Polo. Big data : Une révision. RESUM ´ E´, 2015.
- ✓ [Ranganathan et Foster 2001] Kavitha Ranganathan et Ian T. Foster; Identifying Dynamic Replication, Strategies for a High Performance Data Grid; Second International Workshop on Grid Computing (GRID' 01), pp. 75-86, UK, 2001
- ✓ [Rasool et al. 2007] Qaisar Rasool, Jianzhong Li, George S. Oreku, Ehsan Ullah Munir et Donghua Yang; A Comparative Study of Replica Placement Strategies in Data Grids; Advances in Web and Network Technologies, and Information Management, Lecture Notes in Computer Science (LNCS), Vol. 4537, pp. 135-143, 2007.
- ✓ [Rudi Bruchez] Rudi Bruchez, Les bases de données NOSQL et le BIGDATA comprendre et mettre en œuvre ,2iéme édition: ÉDITIONS EYROLLES, www.editions-eyrolles.com.
- ✓ [Shorfuzzaman et al. 2008] Mohammad Shorfuzzaman, Peter Graham et Rasit Eskicioglu; Popularity - Driven Dynamic Replica Placement in Hierarchical Data Grids; 9th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT‟‟2008), pp. 524-531, Otago, 2008
- ✓ [Tudorica and Bucur.2011] Bogdan George Tudorica and Cristian Bucur. A comparison between several nosql databases with comments and notes. In Roedunet International Conference (RoEduNet), 2011 10th. IEEE, 2011.
- ✓ [Tutb] TUTORIALSPOINT. *Distributed DBMS*. URL : https://www.tutorialspoint.com/distributedW3C_dbms/index.htm
- ✓ [W2012] http://davidmasclet.gisgraphy.com/post/2010/06/09/10-minutes-pour-comprendre...NoSQL, Avril 2012].
- ✓ 1 https://www.infoq.com/fr/articles/modele-stockage-physique-cassandra/
- ✓ 2 https://www.simplilearn.com/tutorials/big-data-tutorial/cassandra-architecture
- ✓ 3 http://www-igm.univ-mlv.fr/dr/XPOSE2010/Cassandra/modele.html
- ✓ 4 Datastax. (2012). Apache Cassandra™ Documentation. From http://www.odbms.org/wp-content/uploads/2013/11/cassandra10.pdf.
- ✓ HBase Apache HBase Team. (2017). Apache HBase ™ Reference Guide, version 3(0). From https://hbase.apache.org/book.html