Université de Ngaoundéré
Institut Universitaire de Technologie

Base de Données non structurées (NoSQL)

Chapitre 4 Modèles de distribution

Pr. Dayang Paul
Maître de conférence

Année académique
2024/2025

INTRODUCTION

Le principal moteur de l'intérêt pour NoSQL a été sa capacité à exécuter des bases de données sur un grand cluster. À mesure que les volumes de données augmentent, il devient plus difficile et plus coûteux de passer à l'échelle supérieure. Une option plus attrayante est de passer à l'échelle supérieure, c'est-à-dire d'exécuter la base de données sur un cluster de serveurs. L'orientation agrégée s'accorde bien avec la mise à l'échelle supérieure, car l'agrégat est une unité naturelle à utiliser pour la distribution.

En fonction de votre modèle de distribution, vous pouvez obtenir un magasin de données qui vous donnera la possibilité de:

- Traiter de plus grandes quantités de données,
- La capacité de traiter un plus grand trafic de lecture ou d'écriture,
- Une plus grande disponibilité face aux ralentissements ou aux pannes de réseau.

INTRODUCTION

Ces avantages sont souvent importants, mais ils ont un coût. L'exécution sur un cluster introduit de la complexité, ce n'est donc pas quelque chose à faire à moins que les avantages soient convaincants.

De manière générale, il existe **deux méthodes** de distribution des données : **la réplication et le sharding**. La réplication prend les mêmes données et les copie sur plusieurs nœuds. Le sharding place des données différentes sur des nœuds différents.

La réplication et le sharding sont des techniques orthogonales : Vous pouvez utiliser l'une ou l'autre ou les deux La réplication se présente sous deux formes : maître-esclave et pair-à-pair. Nous allons maintenant aborder ces techniques en commençant par les plus simples et en allant vers les plus complexes : d'abord la réplication à un seul serveur, puis la réplication maître-esclave, puis le sharding et enfin la réplication pair-à-pair.

1. Serveur unique

La première option de distribution, la plus simple, est celle que nous recommandons le plus souvent : pas de distribution du tout. Exécutez la base de données sur une seule machine qui gère toutes les lectures et écritures dans le modèle de données. Nous préférons cette option car :

- Elle élimine toutes les complexités introduites par les autres options;
- Elle est facile à gérer pour les personnes chargées des opérations;
- Elle est facile à raisonner pour les développeurs d'applications.

Bien que de nombreuses bases de données NoSQL soient conçues pour fonctionner sur un cluster, il peut être judicieux d'utiliser NoSQL avec un modèle de distribution à serveur unique si le modèle de données du NoSQL est plus adapté à l'application.

1. Serveur unique

Les bases de données graphiques constituent la catégorie la plus évidente dans ce cas : elles fonctionnent mieux dans une configuration à serveur unique.

Si l'utilisation de vos données consiste principalement à traiter des agrégats, un modèle orienté documents ou clés- valeurs à serveur unique peut s'avérer intéressant, car il est plus facile pour les développeurs d'applications.

Dans le reste de ce chapitre, nous allons nous pencher sur les avantages et les complications de schémas de distribution plus sophistiqués.

2. Sharding

Souvent, dans un modèle de données plusieurs personnes accèdent à différentes parties de l'ensemble de données. Dans ces circonstances, nous pouvons prendre en charge l'évolutivité horizontale en plaçant différentes parties des données sur différents serveurs c'est une technique appelée **partage.**

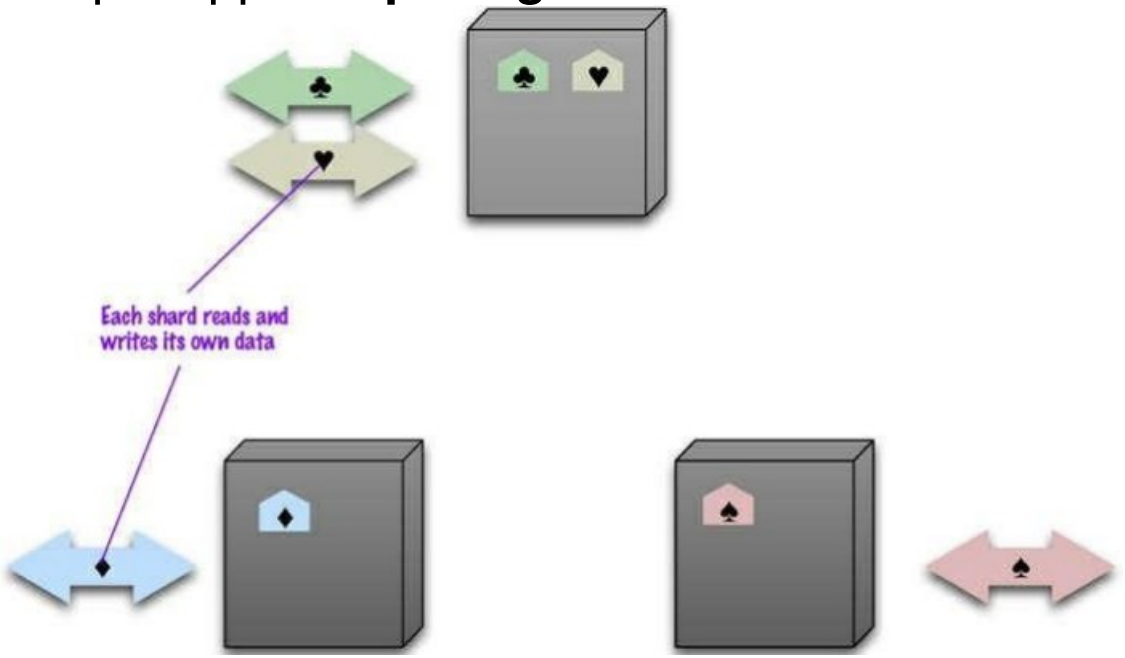

Figure 1. Le sharding place différentes données sur des nœuds distincts, chacun d'entre eux effectuant ses propres lectures. écrit.

2. Sharding

Dans le cas idéal, nous avons différents utilisateurs qui parlent tous à différents nœuds de serveur. Chaque utilisateur ne doit parler qu'à un seul serveur et obtient donc des réponses rapides de ce serveur. La charge est bien répartie entre les serveurs - par exemple, si nous avons dix serveurs, chacun d'eux ne doit gérer que 10 % de la charge.

Bien sûr, le cas idéal est une bête plutôt rare. Pour s'en rapprocher, il faut s'assurer que les données auxquelles on accède ensemble sont regroupées sur le même nœud et que ces regroupements sont disposés sur les nœuds de manière à fournir le meilleur accès possible aux données.

Comment regrouper les données de façon à ce qu'un utilisateur obtienne ses données à partir d'un seul serveur.

2. Sharding

C'est là que l'orientation vers les agrégats s'avère très utile. L'intérêt des agrégats est qu'ils sont conçus pour combiner des données auxquelles on accède couramment, de sorte que l'agrégat apparaît comme une unité de distribution évidente

Lorsqu'il s'agit de disposer les données sur les nœuds, plusieurs facteurs peuvent contribuer à améliorer les performances. Si vous savez que la plupart des accès à certains agrégats sont basés sur un emplacement physique, vous pouvez placer les données à proximité de l'endroit où elles sont consultées.

Si vous avez des commandes pour une personne qui vit à Boston, vous pouvez placer ces données dans votre centre de données de l'est des États-Unis

2. Sharding

Un autre facteur est d'essayer de garder la charge égale. Cela signifie que vous devez essayer d'organiser l'agrégat de façon à ce qu'il soit distribué de manière égale sur les nœuds qui reçoivent tous la même quantité de charge.

Cela peut varier au fil du temps, par exemple si certaines données ont tendance à être consultées certains jours de la semaine - il peut donc y avoir des règles spécifiques au domaine que vous souhaitez utiliser.

Dans certains cas, il est utile de regrouper les agrégats si vous pensez qu'ils peuvent être lus dans l'ordre. De cette façon, les données de plusieurs pages peuvent être consultées ensemble pour améliorer l'efficacité du traitement.

2. Sharding

Le partage est particulièrement utile pour les performances car il peut améliorer les performances en lecture et en écriture. L'utilisation de *la réplication*, en particulier avec la mise en cache, peut améliorer considérablement les performances en lecture mais n'est pas très utile pour les applications qui ont beaucoup d'écritures. Le **sharding** permet d'échelonner horizontalement les écritures.

Le sharding n'améliore guère la résilience lorsqu'il est utilisé seul. Bien que les données se trouvent sur différents nœuds, la défaillance d'un nœud rend les données de ce shard indisponibles aussi sûrement que dans le cas d'une solution à serveur unique. L'avantage en termes de résilience qu'il procure est que seuls les utilisateurs des données de ce shard en pâtiront ; cependant, il n'est pas bon d'avoir une base de données dont une partie des données est manquante.

2. Sharding

Malgré le fait que le sharding est rendu beaucoup plus facile avec les agrégats, ce n'est toujours pas une étape à prendre à la légère. Certaines bases de données sont conçues dès le départ pour utiliser le sharding, auquel cas il est sage de les faire fonctionner sur un cluster dès le début du développement, et certainement en production.

D'autres bases de données utilisent le sharding comme une étape délibérée à partir d'une configuration à serveur unique. Dans ce cas, il est préférable de commencer par un serveur unique et de n'utiliser le sharding que lorsque vos projections de charge indiquent clairement que vous manquez de marge de manœuvre.

3. Réplication maître-esclave

Avec la distribution maître-esclave, vous répliquez les données sur plusieurs nœuds. Un nœud est désigné comme maître, ou primaire. Ce maître est la source faisant autorité pour les données et est généralement responsable du traitement de toutes les mises à jour de ces données. Les autres nœuds sont des esclaves, ou secondaires. Un processus de réplication synchronise les esclaves avec le maître.

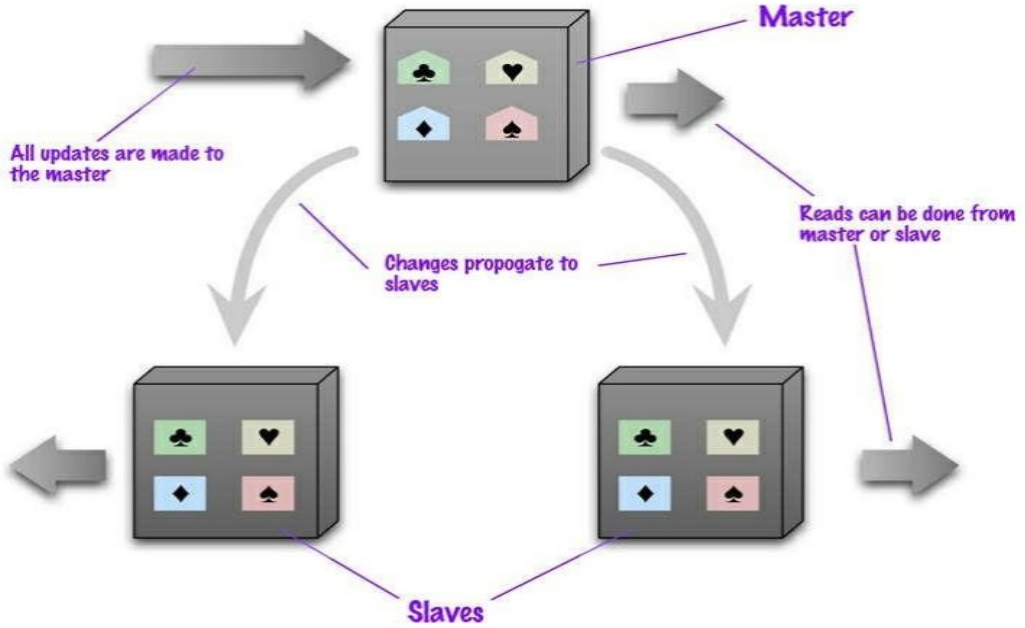

Figure 2. Les données sont répliquées du maître aux esclaves.

3. Réplication maître-esclave

La réplication **maître-esclave** est particulièrement utile pour la mise à l'échelle lorsque vous avez un jeu de données à lecture intensive. Vous pouvez évoluer horizontalement pour traiter davantage de demandes de lecture en ajoutant des **nœuds esclaves** supplémentaires et en veillant à ce que toutes les demandes de lecture soient acheminées vers les esclaves.

Cependant, vous êtes toujours limité par la capacité du maître à traiter les mises à jour et sa capacité à transmettre ces mises à jour. Par conséquent, ce n'est pas un très bon schéma pour les ensembles de données avec un trafic d'écriture important, bien que le fait de charger le trafic de lecture aidera un peu à gérer la charge d'écriture.

3. Réplication maître-esclave

Un deuxième avantage de la réplication maître-esclave est la résilience en lecture : En cas de défaillance du maître, les esclaves peuvent toujours traiter les demandes de lecture. Encore une fois, ceci est utile si la plupart de vos accès aux données sont des lectures. La défaillance du maître élimine la possibilité de gérer les écritures jusqu'à ce que le maître soit restauré ou qu'un nouveau maître soit nommé.

Cependant, le fait d'avoir des esclaves comme répliques du maître accélère la récupération après une défaillance du maître puisqu'un esclave peut être nommé nouveau maître très rapidement.

La possibilité de nommer un esclave pour remplacer un maître défaillant signifie que la réplication maître-esclave est utile même si vous n'avez pas besoin d'étendre vos activités. Tout le trafic de lecture et d'écriture peut aller vers le maître tandis que l'esclave agit comme une sauvegarde à chaud.

3. Réplication maître-esclave

Les maîtres peuvent être nommés manuellement ou automatiquement. La nomination manuelle signifie généralement que lorsque vous configurez votre cluster, vous configurez un nœud comme maître. Avec la nomination automatique, vous créer un cluster de nœuds et ils élisent l'un d'entre eux pour être le maître.

Outre une configuration plus simple, la nomination automatique signifie que le cluster peut désigner automatiquement un nouveau maître lorsqu'un maître tombe en panne, ce qui réduit les temps d'arrêt.

Pour obtenir la résilience de lecture, vous devez vous assurer que les chemins de lecture et d'écriture dans votre application sont différents, de sorte que vous puissiez gérer une défaillance dans le chemin d'écriture et continuer à lire. Pour ce faire, vous devez notamment faire passer les lectures et les écritures par des connexions distinctes à la base de données.

3. Réplication maître-esclave

La réplication présente des avantages séduisants, mais elle s'accompagne aussi d'un côté sombre inévitable : l'**incohérence**. Vous courez le risque que différents clients, lisant différents esclaves, voient des valeurs différentes parce que les changements n'ont pas tous été propagés aux esclaves.

Dans le pire des cas, cela peut signifier qu'un client ne peut pas lire une écriture qu'il vient de faire. Même si vous utilisez la réplication maître-esclave uniquement pour la sauvegarde à chaud, cela peut être un problème, car si le maître tombe en panne, toutes les mises à jour non transmises à la sauvegarde sont perdues.

4. Réplication de pair à pair

La **réplication maître-esclave** contribue à l'extensibilité de la lecture mais pas à celle de l'écriture. Elle offre une résilience contre la défaillance d'un esclave, mais pas d'un maître. Essentiellement, le maître reste un goulot d'étranglement et un point de défaillance unique.

La **réplication pair-à-pair** s'attaque à ces problèmes en n'ayant pas de maître. Toutes les répliques ont le même poids, elles peuvent toutes accepter des écritures et la perte de l'une d'entre elles n'empêche pas l'accès aux données.

4. Réplication de pair à pair

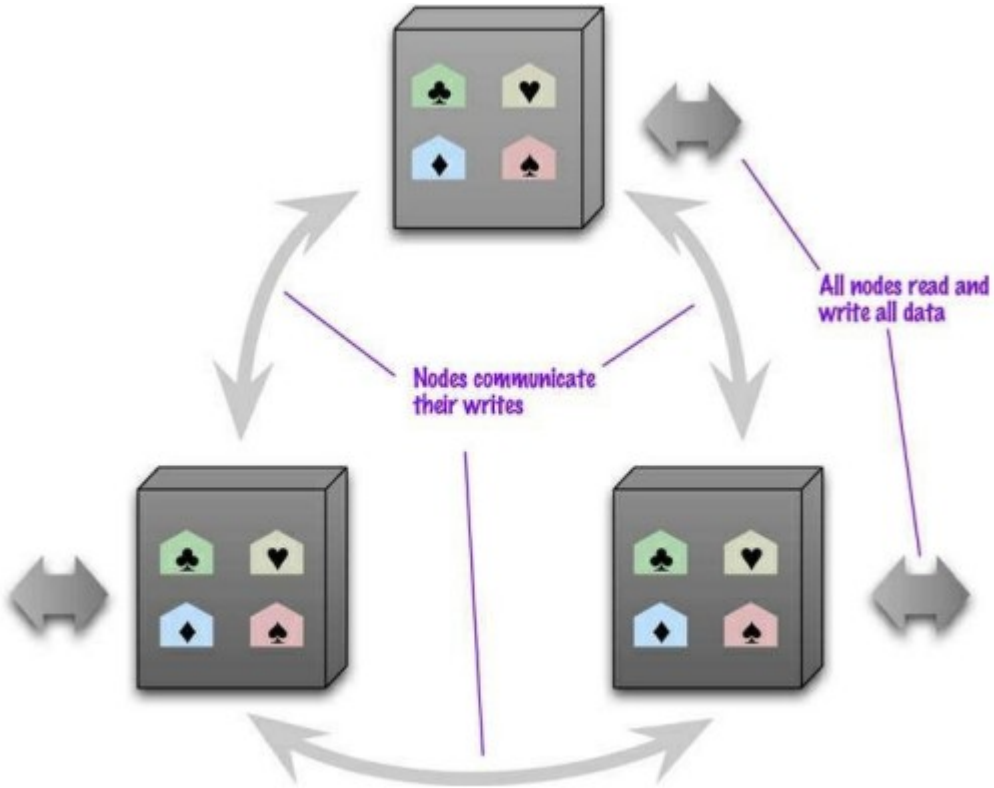

Figure 3. Dans le cas de la réplication poste à poste, tous les nœuds effectuent des lectures et des écritures sur toutes les données.

4. Réplication de pair à pair

Les perspectives sont très prometteuses. Avec un cluster de réplication de pair à pair, vous pouvez surmonter les défaillances de nœuds sans perdre l'accès aux données. De plus, vous pouvez facilement ajouter des nœuds pour améliorer vos performances. Il y a beaucoup à aimer ici, mais il y a des complications.

La plus grande complication est, encore une fois, **la cohérence**. Lorsque vous pouvez écrire à deux endroits différents, vous courez le risque que deux personnes tentent de mettre à jour le même enregistrement en même temps - un conflit d'écriture. Les incohérences en lecture entraînent des problèmes, mais au moins ils sont relativement transitoires.

4. Réplication de pair à pair

Nous reviendrons plus tard sur la façon de gérer les incohérences d'écriture, mais pour l'instant nous allons noter quelques grandes options. D'un côté, nous pouvons:

- ***Nous assurer que chaque fois que nous écrivons des données, les répliques se coordonnent pour éviter tout conflit.*** Cela peut nous donner une garantie aussi forte qu'un maître, bien qu'au prix d'un trafic réseau pour coordonner les écritures.
- Nous n'avons pas besoin que toutes les répliques soient d'accord sur l'écriture, juste une majorité, donc nous pouvons toujours survivre à la perte d'une minorité des nœuds de réplique.

À l'autre extrême, nous pouvons décider de faire face à une écriture incohérente. Dans certains cas, il est possible de mettre en place une politique permettant de fusionner les écritures incohérentes.

5. Combinaison du Sharding et de la Réplication

La réplication et le sharding sont des stratégies qui peuvent être combinées. Si nous utilisons à la fois la réplication maître-esclave et le sharding cela signifie que nous avons plusieurs maîtres, mais que chaque élément de données n'a qu'un seul maître.

En fonction de votre configuration, vous pouvez choisir un nœud comme maître pour certaines données et comme esclave pour d'autres, ou vous pouvez dédier des nœuds aux fonctions de maître ou d'esclave.

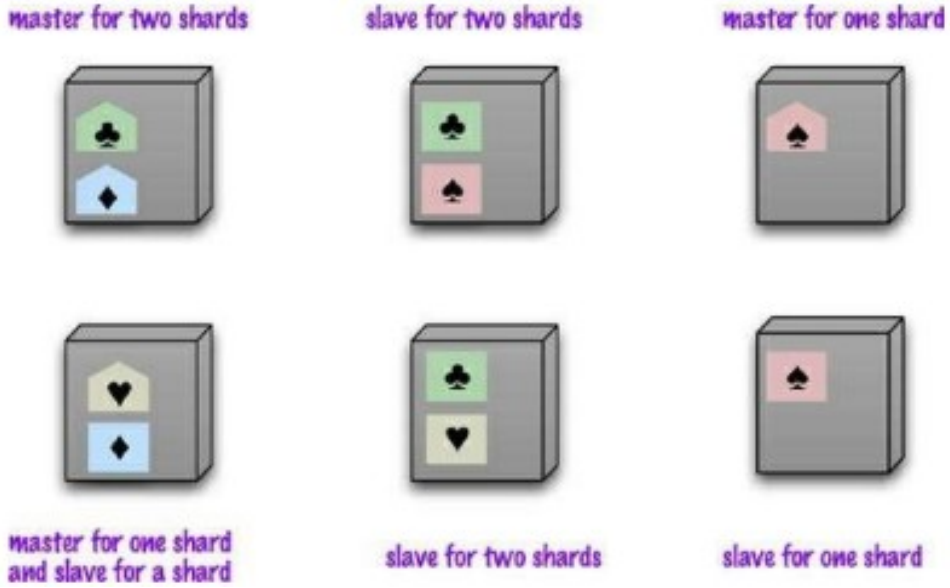

Figure 4. Utilisation de la réplication maître-esclave avec le sharding

5. Combinaison du Sharding et de la Réplication

L'utilisation de la réplication pair-à-pair et du sharding est une stratégie courante pour les bases de données à base de colonnes. Un bon point de départ pour la réplication poste à poste est d'avoir un facteur de réplication de 3, de sorte que chaque shard soit présent sur trois nœuds. Si un nœud tombe en panne, les shards de ce nœud seront construits sur les autres nœuds.

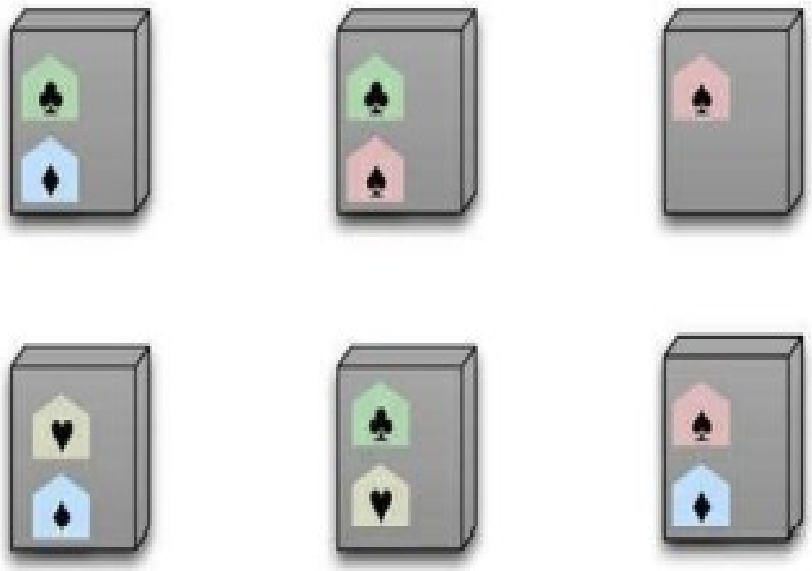

Figure 5. Utilisation de la réplication poste à poste avec le sharding

Points clés

Il existe deux styles de distribution des données :

- Le partage distribue différentes données sur plusieurs serveurs, de sorte que chaque serveur agit comme une source unique pour un sous-ensemble de données.
- La réplication copie les données sur plusieurs serveurs, de sorte que chaque bit de données peut se trouver à plusieurs endroits.

Points clés

Un système peut utiliser l'une ou l'autre de ces techniques, ou les deux.

La réplication se présente sous deux formes :

- **La réplication maître-esclave** fait d'un nœud la copie faisant autorité qui gère les écritures tandis que les esclaves se synchronisent avec le maître et peuvent gérer les lectures.
- **La réplication de pair à pair** permet d'écrire sur n'importe quel nœud ; les nœuds se coordonnent pour synchroniser leurs copies des données.

La réplication maître-esclave réduit les risques de conflits de mise à jour, mais la réplication poste-à-poste évite de charger toutes les écritures sur un seul point de défaillance.

Comment résoudre les problèmes liés à l'incohérence que se produit.