Chapitre 4 : Bases de Données NOSQL

Section 1 : Cassandra

Aiway

Your way to AI

Architecture (1/2)

Aiway
Your way to AI

- Système distribué P2P
- Composés de plusieurs nœuds identiques
  - Pas de notion de nœud maître
- Les données sont partitionnées par défaut à travers les différents nœud du cluster
- Les données sont répliquées pour assurer une tolérance aux fautes maximale
  - L'utilisateur contrôle le nombre de répliques qu'il désire avoir pour ses données
- Lecture et écriture à partir de n'importe quel nœud, indépendamment de l'emplacement des données

Architecture (2/2)

Aiway
Your way to AI

- Utilisation du protocole Gossip pour la communication entre les différents nœuds du cluster
  - Échange de données entre les nœuds chaque seconde
- Structure orientée colonnes, à l'image de BigTable
- Vocabulaire :
  - Keyspace (équivalent de *database*)
  - Column Family (équivalent de *table*)
    - Schéma plus flexible et dynamique qu'une table
  - Colonne (équivalent à enregistrement)
    - Indexée par une clé
    - D'autres champs peuvent être également indexés, mais à la demande

Partitionnement (1/2)

Aiway
Your way to AI

- Partitionnement facile des données à travers les différents nœuds participants du cluster
- Chaque nœud est responsable d'une partie de la base de données
- Les données sont insérées par l'utilisateur dans une famille de colonnes
- Elles sont ensuite placées sur un nœud, selon sa clé de colonne
- Stratégie de partitionnement spécifiée *cassandra.yaml*
- Si la stratégie d'une base est modifiée, il faut recharger toutes les données

Partitionnement (1/2)

Aiway
Your way to AI

- Stratégies de partitionnement
  - Partitionnement aléatoire
    - Par défaut, recommandé
    - Données partitionnées le plus équitablement possible à travers les différents nœuds
    - Utilisation de MD5 pour le hachage de chaque clé de famille de colonnes
  - Partitionnement ordonné
    - Sauvegarde les clés de familles de colonnes par ordre à travers les nœuds d'un cluster
    - Peut provoquer des problèmes, surtout pour la répartition des charges (des nœuds plus volumineux que d'autres)

Réplication (1/3)

Aiway
Your way to AI

- Pour assurer la tolérance aux fautes et pas de SPOF, il est possible de créer une ou plusieurs copie(s) de chaque colonne à travers les nœuds participants
- L'utilisateur spécifie le facteur de réplication désiré à la création du keyspace
- Les données sont insérées par l'utilisateur dans une famille de colonnes
- La colonne est répliquée dans les nœuds du cluster selon le facteur de réplication

Réplication (2/3)

Aiway
Your way to AI

- Stratégie de réplication
  - Stratégie simple :
    - La colonne originelle est placée sur un nœud déterminé par le partitionneur
    - La réplication est placée sur le nœud suivant de l'anneau (clockwise)
    - Pas de considération pour l'emplacement dans un data-center ou dans une baie (approprié pour les déploiements sur un seul datacenter)
  - Stratégie par topologie réseau
    - Plus de contrôle sur l'emplacement des répliques de colonnes
    - Parcourt le cluster (clockwise) à la recherche d'un nœud dans une baie différente. Si introuvable, stocker dans un nœud de la même baie
    - Un *groupe de réplication* est spécifié pour distinguer les datacenters

Réplication (3/3)

Aiway
Your way to AI

- Mécanisme de réplication
  - Utilisation de SNITCH :
    - Définition de la manière dont les nœuds sont groupés dans un réseau
    - Répartition des nœuds entre baies et datacenters
  - Plusieurs types de SNITCH :
    - Simple SNITCH : utilise la stratégie simple
    - Rack-Inferring SNITCH : détermine la topologie du réseau en analysant les adresses IP : 2nd octet → Data center, 3e octet → Baie
    - Property File SNITCH : se base sur une description de l'utilisateur pour déterminer l'emplacement des nœuds (*cassandra-topology.properties*)
    - EC2 SNITCH : pour déploiement dans Amazon EC2 (basé sur l'API AWS)
  - Défini dans le fichier de configuration *cassandra.yaml*

Consistance (1/3)

Aiway
Your way to AI

- Architecture Read and Write Anywhere
- L'utilisateur peut se connecter à n'importe quel nœud, dans n'importe quel datacenter, et lire/écrire les données qu'il veut
- Les données sont automatiquement partitionnées et répliquées à travers le cluster

Consistance (2/3)

Aiway
Your way to AI

- Écriture :
  - Données écrites d'abord dans un *commit log* pour la durabilité
  - Ensuite, écriture en mémoire dans une *MemTable*
  - Une fois la MemTable pleine, les données sont sauvegardées dans le disque, dans une *SSTable* (**S**orted **S**trings **T**able)
  - Même si les transactions relationnelles (commits et rollbacks) ne sont pas supportées, les écritures sont atomiques au niveau des colonnes
    - Soit toutes les colonnes sont modifiées, soit aucune ne l'est
- Cassandra est la base de données NOSQL la plus rapide en écriture

Consistance (3/3)

Aiway
Your way to AI

- Consistance : à quel point est-ce qu'une donnée est à jour et synchronisée sur toutes ses répliques
- Extension du concept de consistance éventuelle à une **consistance ajustable**
- Choix possible entre une consistance forte ou éventuelle selon les besoins
- Ce choix est fait par opération, ce n'est pas une stratégie globale pour la base de données
  - SELECT * FROM users USING CONSISTANCY QUORUM WHERE state='TX';
- Consistance gérée à travers plusieurs data centers

Stratégies d'écriture (1/2)

Aiway
Your way to AI

- Niveau de consistance :
  - **Any** : au moins une écriture doit réussir sur n'importe quel nœud
  - **One** (par défaut) : une écriture doit réussir sur le commit log et la MemTable d'au moins une réplique
  - **Quorum** : une écriture doit réussir sur un certain pourcentage de répliques (% = (facteur de réplication / 2) +1)
    - Meilleure alternative en terme de consistance et de disponibilité
  - **Local-Quorum** : une écriture doit réussir sur un certain pourcentage de nœuds répliques sur le même datacenter que le nœud coordinateur
  - **Each-Quorum** : une écriture doit réussir sur un certain pourcentage de nœuds répliques sur tous les datacenters
  - **All** : une écriture doit réussir sur tous les nœuds répliques

Stratégies d'écriture (1/2)

Aiway
Your way to AI

- Cassandra utilise les **Hinted Handoffs**
  - Elle tente de modifier une colonne sur toutes les répliques

- Si certains des noeuds répliques ne sont pas disponibles, un indice (*hint*) est sauvegardé sur l'un des nœuds en marche pour mettre à jour tous les nœuds en panne une fois rétablis
- Si aucun des noeud réplique n'est disponible, l'utilisation de la stratégie **Any** permettra au nœud coordinateur de stocker cet indice. Mais la donnée ne sera lisible que quand l'un des nœuds concerné sera de nouveau disponible

Stratégie de lecture (1/2)

Aiway
Your way to AI

- Niveau de consistance :
  - **One** (par défaut) : obtention d'une réponse à partir de la réplique la plus proche selon le SNITCH
  - **Quorum** : obtention du résultat le plus récent à partir d'un certain % de nœuds répliques (% = (facteur de réplication / 2) + 1)
  - **Local-Quorum** : obtention du résultat le plus récent à partir d'un certain % de nœuds répliques sur le même datacenter que le nœud coordinateur
  - **Each-Quorum** : obtention du résultat le plus récent à partir d'un certain % de nœuds répliques sur tous les datacenters
  - **All** : obtention du résultat le plus récent à partir de tous les nœuds répliques

Stratégie de lecture (2/2)

Aiway
Your way to AI

- Read Repair
  - Cassandra assure que les données fréquemment lues soient consistantes
  - À la lecture d'une donnée, le nœud coordinateur compare toutes ses répliques en arrière plan
  - Si ces données ne sont pas consistantes, envoie une demande d'écriture aux nœuds réplicas pour mettre à jour leur donnée et afficher la donnée la plus récente
  - *Read Repair* peut être configuré par famille de colonnes et est activé par défaut

Gestion des données et objets (1/2

Aiway
Your way to AI

- Deux interfaces pour gérer les objets / données
  - Cassandra CLI (Command Line Interface)
  - CQL (Cassandra Query Language)

- CLI : Interface originelle conçue pour créer des objets ou entrées et manipuler les données

- CQL : Utilisée pour créer / manipuler des données en utilisant un langage proche de SQL

Gestion des données et objets (2/2)

Aiway
Your way to AI

- CQL en détails:
  - Objets tels que les keyspaces, familles de colonnes et index sont créés, modifiés et supprimés avec les requêtes usuelles : *CREATE, ALTER, DROP*
  - Données insérées, modifiées et supprimées avec *INSERT, UPDATE et DELETE*
  - Données lues avec *SELECT*
  - MAIS ne supporte pas des opérations telles que *GROUP BY, ORDER BY* (sauf pour les clés composées et ordonnées seulement selon la deuxième clé primaire)
  - Utiliser la clause *USING CONSISTANCY* pour déterminer le type de consistance forte pour chaque opération (lecture / écriture)

Récapitulatif (1/2)

Aiway
Your way to AI

- Grande scalabilité (Gigaoctet/Petaoctet) → approprié aux BD
  - Gain de performances linéaire grâce à l'ajout des nœuds
- Pas de SPOF
- Réplication et distribution facile à travers les datacenters
- Pas besoin de couche de cache séparée
  - Utilisation des mémoires vives de l'ensemble des nœuds
- Consistance des données ajustable
  - Possibilité de choisir pour chaque opération si une donnée est fortement consistante (tous les nœuds disposent de la même donnée à tout moment) ou éventuellement consistante
- Schéma de données flexible

Récapitulatif (2/2)

Aiway
Your way to AI

- Compression des données
  - Utilisation de l'algorithme de compression Snappy de Google
  - Compression des données pour chaque famille de colonnes
  - Pas de pénalité pour la performance, et même une amélioration notable, grâce à la diminution du nombre de I/O physiques

- Langage CQL très proche de SQL
- Support des langages et plateformes clés
- Pas besoin de matériel ou logiciel particulier

Chapitre 4 : Bases de Données NOSQL

Section 2 : MongoDB

Aiway

Your way to AI

Introduction

Aiway
Your way to AI

- Le nom MongoDB vient de Humongous, qui signifie *énorme*
- SGBD NOSQL (écrit en C++) orienté *documents* à schéma flexible et distribuable
- Développé en 2007 par la firme MongoDB et distribué sous la licence AGPL (open-source)

Structure de données

Aiway
Your way to AI

- Données représentées dans un schéma flexible
- Données stockées sous forme de documents (paire clé/valeur)
- Les documents sont organisés sous forme de collections
  - Groupe de documents reliés par des indexes en commun
- La structure du document n’est pas fixée à sa création
  - Mais en général, les documents d’une collection ont une structure similaire

Terminologie

Aiway
Your way to AI

SQL	MongoDB
Base de données	Base de données
Table	Collection
Ligne	Document
Colonne	Champ
Index	Index
Jointure	Imbrication ou référence
Clé primaire	Clé primaire

Document (1/3)

Aiway
Your way to AI

- Structure de données JSON-like, composée de paires clé/valeur

- Stocké sur le disque sous forme de document BSON
  - Documents BSON (Binary JSON) : représentation binaire sérialisée d'un document JSON
  - Supporte plus de types de données que JSON
  - La partie *valeur* peut avoir n'importe quel type supporté par BSON, dont d'autres documents, des tableaux et des tableaux de documents

```
{"_id": 1, "name": "Peter Parker", "superhero_name": "Spider-Man"}
```

Document (2/3)

Aiway
Your way to AI

- Le champ *_id*
  - Seul champ obligatoire, utilisé comme clé primaire dans une collection
  - Peut être de tout type autre que Tableau

- Les champs indexés ont une taille limite (1Mo)

- Les noms des champs ne peuvent pas commencer par ‘$’, contenir le caractère ‘.’ (dot) ou le caractère ‘null’

Document (3/3)

Aiway
Your way to AI

- Limites
  - Taille maximale d'un document : 16Mo
    - Utilisation de l'API GridFS pour stocker des documents plus larges que la taille autorisée
    - GridFS permet de diviser les documents en chunks de même taille, et les stocke sous forme de documents séparés
  - MongoDB préserve l'ordre des champs du document, comme défini à leur création, toutefois:
    - Le champ _id doit toujours figurer en premier
    - Les opérations de *update* qui incluent le renommage d'un champ peut entraîner un changement de l'ordre des champs
  - Champ _id → création d'un index unique à la création d'une collect°
    - Utilisation conseillé de ObjectId : objets petits, uniques et rapides

Structure de documents (1/2)

Aiway
Your way to AI

- Deux manières de créer des relations entre les données : *références* et *données imbriquées*
- **Références**
  - Inclusion de liens ou références d'un document à un autre
  - L'application résout ces références pour accéder aux données
  - On dit qu'on utilise des *Modèles de Données Normalisés*
- **Données imbriquées (Embedded Data)**
  - Sauvegarde des données associées dans la même structure
  - Il est possible d'inclure des documents dans un champ/tableau
  - Permet aux application d'extraire et manipuler plusieurs niveaux de hiérarchie en une seule instruction
  - On parle de *Modèles de Données Dénormalisés*

Structure de documents (2/2)

Aiway
Your way to AI

- Comment choisir entre références et données imbriquées ?
- On choisit les références quand :
  - L'imbrication va produire des données dupliquées sans grands avantages en terme de performance de lecture
  - On veut représenter des relations many-to-many complexes
  - On veut modéliser de larges ensembles de données hiérarchiques
- On choisit les données imbriquées quand :
  - On a des relations *contains* entre les éléments (modèles one-to-one) → exemple : personne et adresse
  - On a dess relations *one-to-many*, où les documents enfants (many) apparaissent toujours dans le contexte des documents parents (one) → personne et emails

Opération de lecture (1/3)

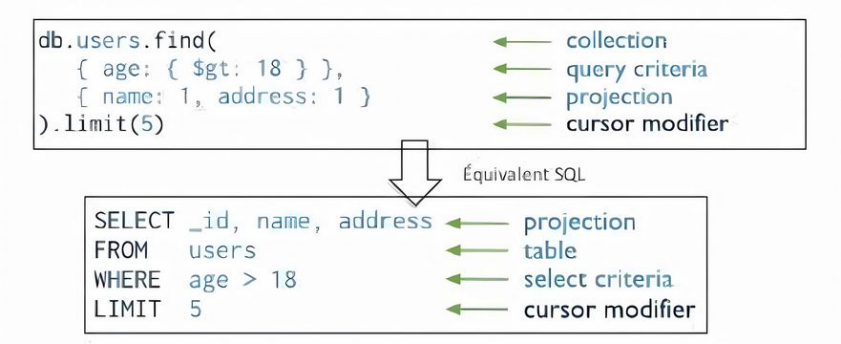
Aiway
Your way to AI

- Cible une *unique* collection spécifique de documents
- Cible un critère ou des conditions spécifiques pour identifier le document à retourner
- Peut inclure une projection sur les champs du document à retourner
- Peut définir des modificateurs pour imposer des limites, un ordre, un filtre...

```
db.users.find( { age: { $gt: 18 } } ).sort( {age: 1 })
```

Opération de lecture (2/3)

- MongoDB fournit une méthode db.*collection*.find() pour l'extraction de données
  - Accepte les critères de la requête, ainsi que les projections
  - Retourne un curseur vers les documents correspondants
- Fournit également une méthode db.*collection*.findOne() retournant un seul document



Opération de lecture (3/3)

Aiway
Your way to AI

- Projections
  - Deuxième argument de la méthode *find()*
  - Peuvent spécifier la liste des champs à afficher ou à exclure
  - Mis à part pour exclure le champ *_id* (qui est affiché par défaut), il est interdit de mixer les projections inclusives et exclusives

Opération d'écriture (1/5)

Aiway
Your way to AI

- Modification
  - Création, mise à jour ou suppression de données
  - Ces opérations modifient les données d'une seule collection
  - Définition de critères pour sélectionner les documents à modifier ou supprimer

- Insertion de document
  - Si vous ajoutez un document sans champ *_id*, le système l'ajout lui-même en générant un champ de type *ObjectId*

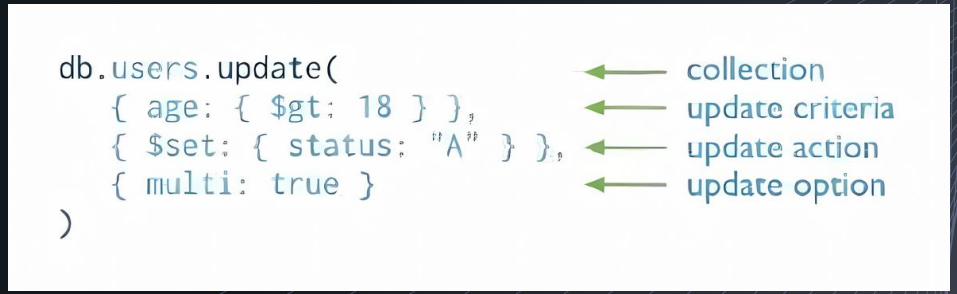
```
db.users.insert(
    {
        name: "John Doe",
        age: 29,
        status: "A",
        group: ["developer", "teacher"]
    }

)
```

Opération d'écriture (2/5)

Aiway
Your way to AI

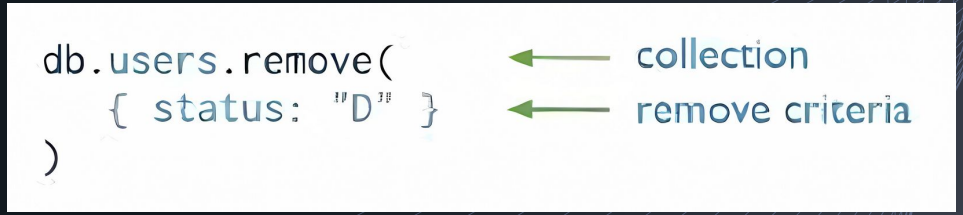
- Mise à jour de documents
  - Méthode *update* peut accepter des critères de sélection des doc à modifier, et des options qui affectent son comportement
  - Une opération *update* modifie un seul document par défaut (plusieurs documents peuvent être modifié avec *multi: true*)
  - Existence de l'option *upsert* : si activée et le document à modifier n'existe pas dans la collection, il est automatiquement inséré



Opération d'écriture (3/5)

Aiway
Your way to AI

- Suppression de documents
  - La méthode remove supprime des documents d'une collection
  - Accepte des critères de sélection des documents à supprimer
  - Supprime par défaut tous les documents correspondant aux critères de sélection (sauf indication du contraire dans un flag approprié)



Opération d'écriture (4/5)

Aiway
Your way to AI

- Les opérations d'écriture sont atomiques au niveau document
  - Aucune opération d'écriture ne peut affecter **atomiquement** plus qu'un seul document ou collection
  - L'utilisation des données imbriquées facilite l'écriture
  - La normalisation des données (référence vers données dans d'autres collections) implique plusieurs opérations d'écriture qui ne sont pas atomiques

Opération d'écriture (5/5)

Aiway
Your way to AI

- Certaines modifications (un push dans des tableaux, ou bien un ajout d'un champ) augmentent la taille des documents
  - Pour le moteur de stockage MMAP v1, si la taille d'un document dépasse la taille autorisée, MongoDB le réalloue sur le disque
  - Si votre application nécessite plusieurs modifications qui augmentent la taille des documents, considérer plutôt l'utilisation des références

Architecture: Sharding (1/2)

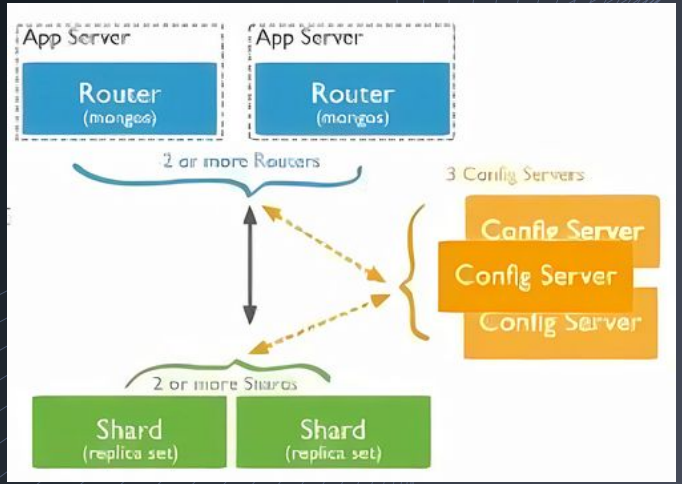
Aiway
Your way to AI

- MongoDB utilise la notion de **Sharding** (ou *horizontal scaling* ou *scale out*) pour stocker ses données dans le cluster
  - Division des données et leur distribution entre plusieurs serveurs ou *shards*
  - Chaque *shard* est une base indépendante, et mis ensemble, forment une seule base de données logique
- Réduction du nombre d'opérations que chaque machine gère
  - Chaque machine gère les données qui y sont stockés
- Réduction de la quantité de données que le serveur a besoin de stocker
  - Plus le cluster grandit, moins un serveur contient de données

Architecture: Sharding (2/2)

Aiway
Your way to AI

- Shards
  - Stockent les données
  - Les données sont distribuées et répliquées sur les shards
- Query Routers
  - Instances *mongos*
  - Interfaçage avec les applications clientes
  - Redirige les opérations vers le shard approprié et retourne le résultat au client
  - Plusieurs QR pour la répartition des tâches
- Config Servers
  - Stockent les méta-données du cluster
  - Définissent le mapping entre les data et les shards
  - Dans les env. de prod. trois CS doivent être définis



Partitionnement des données (1/4)

Aiway
Your way to AI

- Partitionnement des données au niveau des collections, via la ***shard key***
  - Champ simple ou composé indexé qui existe dans chaque document de la collection
- MongoDB divise les valeurs de la clé en morceaux (chunks) et les distribuent de manière équitable entre les shards
- La répartition des clés suit l'une de ces deux méthodes de partitionnement :
  - Basée sur le rang
  - Basée sur le hash
  - Basée sur les tags

Partitionnement des données (2/4)

- Partitionnement basé sur le rang
  - Définition d'intervalles qui ne se chevauchent pas, dans lesquelles les valeurs de la *shard key* peuvent se trouver
  - Permet aux documents avec des clés proches de se trouver dans le même shard
  - Plus facile de retrouver le shard pour une donnée
  - Risque de distribution inéquitable des données (par exemple si la clé est le temps, alors toutes les requêtes dans un même intervalle de temps sont sur le même serveur, d'où une grande différence selon les heures de grande ou de faible activité

Partitionnement des données (3/4)

Aiway
Your way to AI

- Partitionnement basé sur le hash
  - Calcul de la valeur du hash d'un champ, puis utilise ces hash pour créer des partitions
  - Deux documents ayant des clés proches ont peu de chance de se trouver dans le même shard
  - Distribution aléatoire d'une collection dans le cluster, d'où une distribution plus équitable
  - Moins efficace, car le temps de recherche de la donnée est plus grand
  - Dans le cas d'une requête portant sur des données se trouvant dans un intervalle défini, le système doit parcourir plusieurs shards

Partitionnement des données (4/4)

Aiway
Your way to AI

- Partitionnement basé sur les tags
  - Les administrateurs peuvent définir des tags, qu'ils associent à des intervalles de clés
  - Ils associent ces tags aux différents shards en essayant de respecter la distribution équitable des données
  - Un balancer migre les données taggées vers les shards adéquats
  - Le meilleur moyen pour assurer une bonne répartition des données

Maintien d'une distribution équitable

Aiway
Your way to AI

- L'ajout de nouvelles données ou nouveaux serveurs peut rendre la distribution des données déséquilibrée

- **Splitting :**
  - Évite d'avoir des *chunks* trop larges
  - Quand la taille d'un chunk augmente au delà d'une valeur prédéfinie (*chunk size*), MongoDB divise cet ensemble de données en deux sur le même shard
  - Les insertions et les modifications déclenchent les splits
  - Un split change les méta-données, mais ne fait pas migrer les données ni n'affecte le contenu des shards

Maintien d'une distribution équitable

Aiway
Your way to AI

- **Balancing :**
  - *Balancer* : processus en arrière plan gérant les migrations de chunks
  - Peut être lancé à partir de n'importe quel query router
  - Quand la distribution des données est déséquilibrée, le balancer fait migrer des chunks du shard ayant le plus de chunks vers celui qui en a le moins, jusqu'à ce que la collection devienne équitablement répartie
  - Étapes :
    - Le shard destination reçoit tous les documents du chunk à migrer
    - Le shard destination applique tous les changements faits aux données durant le processus de migration
    - Les méta-données sur l'emplacement du chunk sont modifiées

Ajout ou suppression de shards

Aiway
Your way to AI

- Dans le cas d'un ajout de shard
  - Un déséquilibre est créé entre les shards du cluster, car le nouveau shard n'a pas de chunks
  - MongoDB peut commencer le processus de migration immédiatement, mais cela risque de prendre du temps avant que le cluster ne soit équilibré
- Dans le cas d'une suppression de shard
  - Le balancier migre tous les chunks de ce shard vers les autres shards
  - Une fois toutes les données migrées et les méta-données mises à jour, la suppression peut avoir lieu

Stratégies de réplication (1/5)

Aiway
Your way to AI

- Définition d'un *replicat set* : groupe de processus mongo qui fournit la redondance et la haute disponibilité

- Types de membres dans un replicat set :
  - Primaire : Reçoit toutes les opérations d'écriture
  - Secondaire : réplication des opérations du primaire pour maintenir des répliques identiques à l'original
  - Arbitre : ne conserve pas de copie de la donnée, mais joue un rôle dans les élections qui sélectionnent un membre primaire, si le primaire actuel est indisponible

Stratégies de réplication (2/5)

Aiway
Your way to AI

- Primaire
  - Seul membre qui reçoit les opérations d'écriture
  - MongoDB applique les opérations d'écriture sur le primaire, puis les enregistre dans le log (oplog)
  - Les membres secondaires dupliquent ce log et appliquent les opérations sur leur données
  - Tous les membres d'un replicat set peuvent accepter une opération de lecture
    - Par défaut, une application dirige ces opérations vers le primaire
  - Un replicat set a au plus un primaire
  - Si le primaire tombe en panne, une élection a lieu pour en choisir un autre

Stratégies de réplication (3/5)

Aiway
Your way to AI

- Secondaires
  - Maintiennent une copie des données du primaire
  - Pour répliquer les données, un secondaire applique les opérations du *oplog* du primaire sur ses propres données, de manière asynchrone
  - Un replicat set peut avoir un ou plusieurs secondaires
  - Même si les clients ne peuvent pas écrire des données sur les secondaires, ils peuvent en lire
  - Un secondaire peut devenir un primaire suite à une élection
  - Il est possible de configurer un secondaire :
    - L'empêcher d'être élu, lui permettant de rester un backup
    - Empêcher certaines applications de lire à partir de lui
    - Exécuter des snapshots périodiques pour garder un historique

Stratégies de réplication (4/5)

Aiway
Your way to AI

- Arbitre
  - N’a pas de copie des données et ne peut pas devenir un primaire
  - Permet de voter pour un primaire
  - Doit être défini pour les replicat sets avec un nombre pair de membres

Stratégies de réplication (5/5)

Aiway
Your way to AI

- Élections
  - Ont lieu à la création d'un replicat set, ou bien quand un primaire devient indisponible
  - Processus qui prend du temps, et qui fait passer le replicat set en mode readonly
  - Chaque membre a une priorité déterminant son éligibilité à devenir primaire
    - L'élection choisit le membre avec la plus haute priorité
    - Par défaut, tous les membres ont la même priorité (1)
    - Il est possible de modifier la priorité selon leur localisation par exemple
  - Chaque membre a la possibilité de voter pour un seul autre membre et celui qui reçoit le plus de votes devient primaire

Write Concern

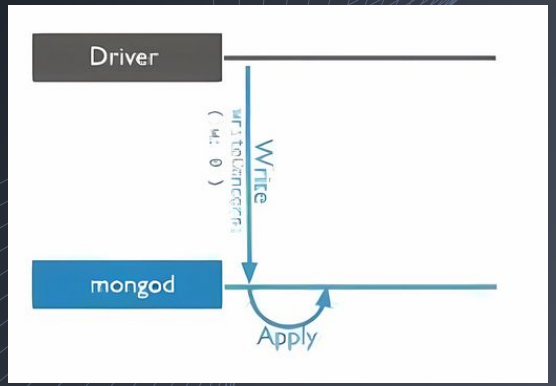
Aiway
Your way to AI

- Représente la garantie que MongoDB fournit en déclarant qu'une opération d'écriture a été réalisée avec succès
- Si les insertions, modifications et suppressions ont un *write concern* faible
  - Les opérations retournent rapidement
  - En cas d'échec, les données peuvent ne pas être persistées
- Si l'écriture a un *write concern* plus fort, les clients doivent attendre après chaque opération d'écriture que MongoDB la confirme
- Depuis la version 2.6, le *write concern* peut être intégré avec l'opération d'écriture
  - Au lieu d'être dans une commande séparée (getLastError)

Write Concern : niveaux (1/4)

Aiway
Your way to AI

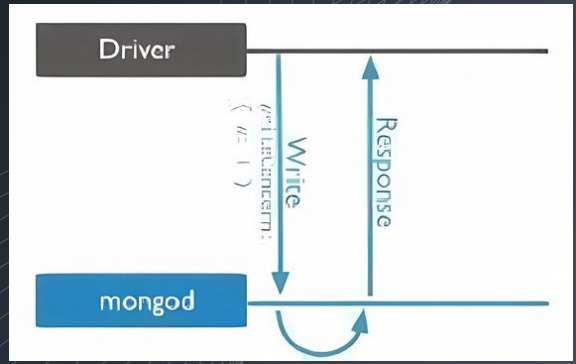
- Unacknowledged
  - Similaire à “Errors Ignored”
  - MongoDB n’envoie pas d’acquittement après une opération d’écriture
  - Le driver tentera de gérer les erreurs réseau tant que possible, selon la configuration du système
  - Avant, c’était le niveau par défaut



Write Concern : niveaux (2/4)

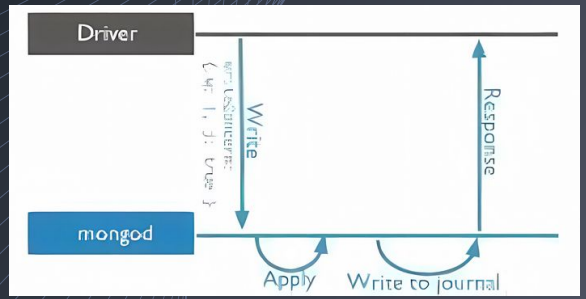
Aiway
Your way to AI

- Acknowledged
  - Niveau par défaut
  - MongoDB confirme la réception de l'opération d'écriture et applique les changements à la vue *in-memory* des données
  - Permet aux clients de détecter les erreurs dues au réseau, aux clés dupliquées...
  - Elle **ne confirme pas** que les données sont correctement écrites sur le disque !



Write Concern : niveaux (3/4)

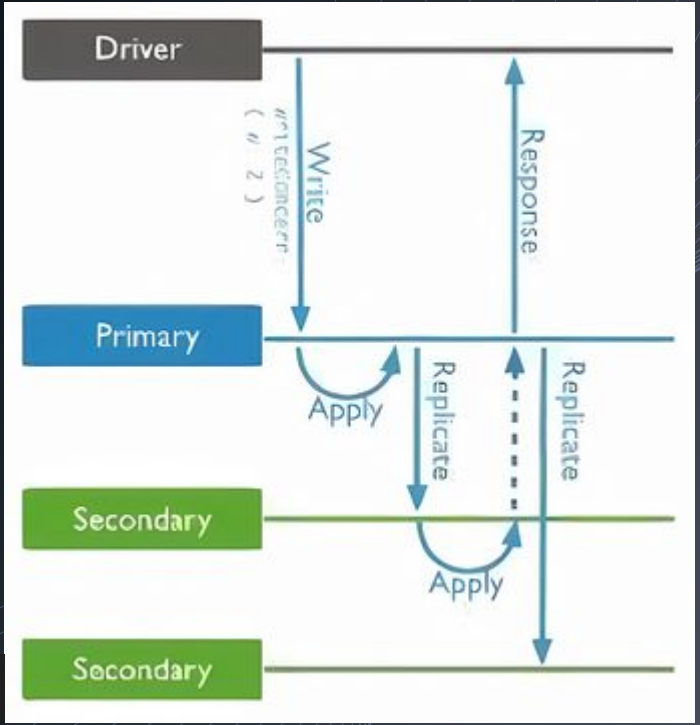
- Journaled
  - MongoDB valide les opérations d'écriture uniquement après avoir réalisé l'opération d'écriture sur le journal (log permettant de sauvegarder les transactions et événements)
  - MongoDB peut ainsi récupérer les données à la suite d'un shutdown ou d'une panne électrique
  - Le *journaling* doit être activé pour utiliser cette option
  - Pour réduire la latence de ces opérations, MongoDB augmente la fréquence des commit
  - La journalisation est requise uniquement pour la donnée primaire



Write Concern : niveaux (4/4)

Aiway
Your way to AI

- Replica Acknowledged
  - Garantit que l'opération d'écriture s'est propagée à un nombre spécifié de répliques
  - Exemple :
    - La méthode retourne seulement si l'écriture se propage à la donnée primaire et au moins à une réplique
    - Sinon la méthode lance un *timeout* au bout de 5 s.




```
db.products.insert(
    { item: "spatula", qty: 100, "type": "golden" },
    { writeConcern: { w: 2, wtimeout: 5000 } }
)
```

Read Isolation

Aiway
Your way to AI

- MongoDB permet aux clients de lire des documents insérés ou modifiés **avant** qu'ils ne soient écrits sur le disque, indépendamment du *write concern* défini ou de la configuration des journalisation
- Comme résultat, les applications peuvent observer deux classes de comportement
  - Pour les systèmes avec des lecteurs et écrivains concurrents, MongoDB autorise les résultats de lecture avant que l'opération d'écriture ne retourne
  - Si MongoDB termine avant que la journalisation ne soit faite, et même si une opération d'écriture a retourné avec succès, le client peut avoir lu des données qui n'existeront plus

Read Preference

Aiway
Your way to AI

- Décrit comment les clients MongoDB routent les opérations de lecture vers les répliques
- Par défaut, une application oriente ses opérations de lecture vers la donnée primaire
  - Donne une garantie de fraîcheur, car les opérations d'écriture par défaut opèrent sur la donnée primaire

- Pour une application qui ne nécessite pas de données de toute fraîcheur, il est possible d'améliorer la latence de lecture en distribuant certaines opérations de lecture vers les répliques