

La RéPLICATION dans les Systèmes de Bases de Données Distribuées

Rapport - Rayane Sendjakedine

1 Introduction

La réPLICATION est un mécanisme fondamental dans les systèmes de gestion de bases de données modernes, particulièrement dans le contexte des bases NoSQL. Ce rapport présente les concepts théoriques de la réPLICATION, son application dans une architecture maître-esclave, et une mise en œuvre concrète avec MongoDB.

Dans un contexte où les applications doivent gérer des volumes de données massifs tout en garantissant une haute disponibilité, la réPLICATION devient un élément central de l'architecture. Elle permet non seulement d'assurer la continuité du service en cas de panne, mais aussi d'améliorer les performances en distribuant la charge de travail.

2 Principe Général de la RéPLICATION

2.1 Définition et Objectifs

La réPLICATION est le processus qui consiste à copier et maintenir des données sur plusieurs serveurs (ou nœuds) au sein d'un même cluster. Cette technique vise à atteindre plusieurs objectifs critiques dans un système distribué.

Objectifs principaux :

- **Haute disponibilité** : Le système continue de fonctionner même si un ou plusieurs serveurs tombent en panne. Les utilisateurs peuvent accéder aux données sans interruption notable.
- **Tolérance aux pannes** : Les données ne sont pas perdues en cas de défaillance d'un nœud. Le système peut récupérer automatiquement après une panne.
- **Performance de lecture** : En distribuant les requêtes de lecture sur plusieurs nœuds répliqués, on réduit la charge sur chaque serveur individuel et on améliore le temps de réponse global.
- **Répartition géographique** : Les données peuvent être distribuées sur plusieurs sites géographiques, réduisant ainsi la latence pour les utilisateurs distants.

2.2 Architecture des Systèmes Distribués

Un système de base de données distribué repose sur un ensemble de nœuds interconnectés qui coopèrent pour stocker et servir les données. Tous les nœuds d'une grappe de serveurs sont connectés et échangent constamment des messages pour :

- Répliquer les données
- Transmettre des confirmations
- Vérifier l'état de chacun des nœuds

Cette interconnexion permet au système de rester cohérent et réactif même dans un environnement distribué.

2.3 Modèles de RéPLICATION

Il existe deux grands modèles d'organisation dans les bases distribuées :

2.3.1 Architecture Master-Slave (Maître-Esclave)

Dans ce modèle, un nœud est désigné comme master (maître) qui reçoit toutes les écritures, tandis que plusieurs nœuds slaves (esclaves) répliquent les données issues du

master. Les lectures peuvent être servies par le master ou les slaves selon la configuration.

2.3.2 Architecture Masterless (Sans Maître)

Dans un cluster masterless, tous les noeuds jouent un rôle homogène. Les écritures et lectures peuvent être envoyées à n'importe quel noeud, qui réplique ensuite les données aux autres. Ce modèle est utilisé par des systèmes comme Cassandra.

3 Architecture Maître-Esclave

3.1 Principe de Fonctionnement

L'architecture maître-esclave est un modèle de réPLICATION asymétrique où les rôles sont clairement séparés entre les différents noeuds du cluster.

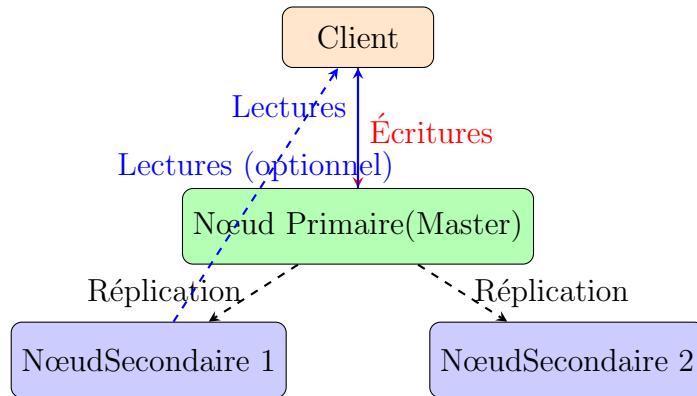


FIGURE 1 – Architecture Maître-Esclave - Flux de données

3.2 Rôles et Responsabilités

3.2.1 Le Nœud Maître (Master)

Le nœud maître, également appelé *primary* dans MongoDB, est l'unique responsable des écritures. Il centralise toutes les opérations d'insertion, de mise à jour et de suppression.

Fonctionnement :

1. Le maître reçoit une requête d'écriture du client
2. Il enregistre l'opération dans un journal interne (oplog)
3. Il applique la modification à ses données locales
4. Il envoie un accusé de réception au client
5. Il initie ensuite la réPLICATION vers les esclaves

Cette centralisation garantit l'intégrité des données en évitant les conflits de mise à jour qui surviendraient si plusieurs nœuds pouvaient accepter des écritures simultanément.

3.2.2 Les Nœuds Esclaves (Slaves)

Les nœuds esclaves, ou *secondaries*, ont pour rôle principal de maintenir une copie à jour des données du maître.

Mécanisme de réPLICATION :

1. Chaque esclave se connecte au maître
2. Il lit en continu le journal d'opérations (oplog) du maître
3. Il applique les opérations pour maintenir sa copie synchronisée
4. Il peut servir les requêtes de lecture si configuré pour cela

3.3 Détection de Pannes et Récupération

3.3.1 Défaillance d'un Esclave

Si un esclave tombe en panne, le maître s'en aperçoit rapidement grâce aux échanges réguliers entre nœuds. L'absence de réponse ou un délai anormal est vite détecté. Le maître peut alors :

- Marquer le nœud comme inactif
- Réessayer la connexion plus tard
- Réaffecter la charge à un autre nœud actif

3.3.2 Défaillance du Maître

La défaillance du maître est beaucoup plus critique car c'est lui qui centralise les écritures et la coordination. Pour assurer la continuité du service, un processus d'élection automatique est déclenché.

Processus d'élection :

1. Les esclaves détectent l'absence du maître
2. Un algorithme de consensus distribué est lancé (Paxos, Raft)
3. Les nœuds restants négocient entre eux pour désigner un nouveau maître
4. L'esclave avec la priorité la plus élevée et les données les plus récentes est élu
5. Le nouveau maître commence à accepter les écritures

3.4 Problème du Split-Brain

Un problème critique peut survenir en cas de partitionnement réseau : le cluster est découpé en deux groupes qui ne peuvent plus communiquer entre eux. Chaque groupe pourrait croire que l'autre est tombé en panne et tenter de désigner un nouveau maître. On se retrouve alors avec deux maîtres simultanés, ce qui est inacceptable pour la cohérence du système.

Solution : Règle de la Majorité

Pour éviter ce scénario, une solution classique consiste à autoriser seul le groupe qui contient la majorité des nœuds à continuer à fonctionner. Le groupe majoritaire élit un maître et continue de traiter les requêtes, tandis que l'autre groupe reste inactif jusqu'au rétablissement du réseau.

C'est précisément cette stratégie qui est adoptée par MongoDB.

4 RéPLICATION SYNCHRONE VS ASYNCHRONE

4.1 RéPLICATION SYNCHRONE

Dans la réPLICATION SYNCHRONE, l'opéRATION d'écRITURE est CONFIRMÉE au client UNIQUEMENT après avoir été RéPLIQUÉE et CONFIRMÉE par une MAJORITé de secondaires.

Avantages :

- Cohérence forte garantie immédiatement sur tout le cluster
- Aucune perte de données en cas de panne du primaire

Inconvénients :

- Latence d'écriture plus élevée (attend la confirmation des secondaires)
- Performance d'écriture réduite

4.2 RéPLICATION ASYNCHRONE

Dans la réPLICATION ASYNCHRONE, l'opéRATION d'écRITURE est CONFIRMÉE au client DÈS QU'ELLE est VALIDÉE par le primaire. La réPLICATION vers les secondaires se fait EN ARRIÈRE-PLAN.

Avantages :

- Faible latence d'écriture (performances rapides)
- Meilleur débit d'opérations

Inconvénients :

- Introduit un léger décalage (lag) où les secondaires peuvent avoir des données légèrement obsolètes
- Cohérence éventuelle (eventual consistency)

Choix de MongoDB : Par défaut, MongoDB utilise une réPLICATION ASYNCHRONE. Cependant, le développeur peut exiger un "write concern" pour simuler un comportement SYNCHRONE en contrepartie d'une latence accrue.

5 Implémentation avec MongoDB

5.1 Le Replica Set de MongoDB

MongoDB implémente la réPLICATION à travers un mécanisme appelé **Replica Set**. Un Replica Set est un groupe de processus `mongod` qui gère de manière collaborative le même ensemble de données.

5.1.1 Terminologie

MongoDB utilise les termes suivants :

- **Primary** (Primaire) : équivalent du Master
- **Secondary** (Secondaire) : équivalent du Slave
- **Arbiter** (Arbitre) : nœud spécial qui participe au vote mais ne stocke pas de données

5.2 Configuration d'un Replica Set

5.2.1 Prérequis

Pour mettre en place un Replica Set, chaque instance doit être configurée avec :

- Un nom de Replica Set (identique pour tous les membres)
- Un port d'écoute unique
- Un répertoire de données dédié

5.2.2 Exemple de Configuration

Voici les étapes pour créer un Replica Set avec trois serveurs :

Étape 1 : Création des répertoires

```
1 mkdir disque1 disque2 disque3
```

Étape 2 : Démarrage des serveurs

Serveur 1 (port 27018) :

```
1 mongod --replSet monReplica7 --port 27018 --dbpath disque1
```

Serveur 2 (port 27019) :

```
1 mongod --replSet monReplica7 --port 27019 --dbpath disque2
```

Serveur 3 (port 27020) :

```
1 mongod --replSet monReplica7 --port 27020 --dbpath disque3
```

Étape 3 : Initialisation du Replica Set

Se connecter au premier serveur :

```
1 mongo --port 27018
```

Initialiser le Replica Set :

```
1 rs.initiate()
```

Étape 4 : Ajout des membres

```

1 rs.add("localhost:27019")
2 rs.add("localhost:27020")

```

5.3 Gestion des Données

5.3.1 Opérations d'Écriture

Les écritures se font uniquement sur le nœud primaire :

```

1 // Connexion au primaire (port 27018)
2 use demo1
3 db.createCollection("personne")
4
5 // Insertion de données
6 db.personne.insert({nom: "Dupont"})
7 db.personne.insert({nom: "Godard"})
8 db.personne.insert({nom: "Martin"})
9
10 // Lecture des données
11 db.personne.find()

```

Ces opérations réussissent car nous sommes connectés au primaire qui accepte les écritures.

5.3.2 Opérations de Lecture sur un Secondaire

Par défaut, les secondaires n'acceptent pas les lectures. Il faut explicitement autoriser cette opération :

```

1 // Connexion à un secondaire (port 27019)
2 mongo --port 27019
3
4 use demo1
5
6 // Cette commande choue par défaut
7 show collections // Erreur: "not master"
8
9 // Autoriser les lectures sur ce secondaire
10 rs.secondaryOk()
11
12 // Maintenant les lectures fonctionnent
13 show collections
14 db.personne.find() // Retourne les 3 documents

```

Important : Lorsqu'on lit depuis un secondaire, on prend le risque d'obtenir des données légèrement obsolètes en raison du délai de réPLICATION asynchrone. L'application doit être consciente de ce compromis entre performance et cohérence.

5.3.3 Tentative d'Écriture sur un Secondaire

Les écritures sur un secondaire sont toujours refusées :

```

1 // Sur le secondaire (port 27019)
2 db.personne.insert({nom: "Nouveau"})
3 // Erreur: "not master"

```

Cette restriction garantit la cohérence des données en évitant les conflits d'écriture.

5.4 Simulation d'une Panne

5.4.1 Arrêt du Primaire

Pour simuler une panne du primaire, on l'arrête brutalement :

```

1 # Sur le terminal du serveur 1 (port 27018)
2 Ctrl+C

```

Les deux autres nœuds détectent immédiatement la panne et lancent le processus d'élection. Après quelques secondes, un nouveau primaire est élu (généralement le serveur sur le port 27019 ou 27020).

5.4.2 Vérification du Nouveau Primaire

```

1 // Connexion au port 27019
2 mongo --port 27019
3
4 // Vérifier le statut
5 rs.status()
6
7 // Si ce nœud est devenu primaire, on peutcrire
8 use demo1
9 db.personne.find() // Les données sont toujours présentes

```

5.4.3 Réintégration de l'Ancien Primaire

Lorsqu'on redémarre le serveur 1, il réintègre automatiquement le Replica Set, mais cette fois en tant que secondaire. Il ne reprend pas automatiquement le rôle de primaire.

5.5 Le Nœud Arbitre

Pour éviter les situations de partitionnement réseau sans majorité, MongoDB permet d'ajouter un nœud arbitre.

Caractéristiques de l'arbitre :

- Ne stocke pas de données
- Participe uniquement aux votes d'élection
- Permet d'atteindre une majorité même avec un nombre pair de nœuds

Configuration :

```

1 # Créer le répertoire
2 mkdir arbitre1
3
4 # Démarrer l'arbitre
5 mongod --replicaSet monReplica7 --port 27021 --dbpath arbitre1

```

Ajout au Replica Set :

```
1 rs.addArb("localhost:27021")
```

5.6 Commandes de Surveillance**5.6.1 rs.config()**

Affiche la configuration statique du Replica Set :

- Liste des membres avec leur ID
- Adresses (host :port)
- Priorité de chaque nœud
- Droit de vote

5.6.2 rs.status()

Affiche l'état dynamique en temps réel :

- Quel nœud est actuellement primaire
- État de synchronisation des secondaires
- Temps de décalage (replication lag)
- Santé de chaque nœud (health : 1 = en ligne, 0 = hors ligne)

5.6.3 rs.isMaster()

Indique si le nœud connecté est le primaire et fournit des informations sur le Replica Set.

5.7 Gestion de la Cohérence

MongoDB privilégie par défaut une cohérence forte en forçant les lectures et écritures sur le primaire. Cependant, il est possible de configurer ce comportement avec les paramètres suivants :

Read Preference (Préférence de Lecture) :

- **primary** : Lecture uniquement depuis le primaire (défaut)
- **primaryPreferred** : Primaire si disponible, sinon secondaire
- **secondary** : Lecture depuis les secondaires uniquement
- **secondaryPreferred** : Secondaire si disponible, sinon primaire
- **nearest** : Lecture depuis le nœud le plus proche (latence minimale)

Write Concern (Niveau de Garantie d'Écriture) :

- **w**: 1 : Confirmation après écriture sur le primaire uniquement (défaut)
- **w**: "majority" : Confirmation après réplication sur la majorité des nœuds
- **w**: <nombre> : Confirmation après réplication sur N nœuds

6 Avantages et Limites de la RéPLICATION

6.1 Avantages

- **Haute disponibilité** : Le système continue de fonctionner même en cas de panne d'un ou plusieurs noeuds
- **Protection des données** : Les données sont sauvegardées sur plusieurs serveurs, réduisant le risque de perte
- **Amélioration des performances de lecture** : Distribution de la charge sur plusieurs noeuds
- **Reprise après sinistre** : Possibilité de récupération rapide en cas de défaillance matérielle
- **Maintenance sans interruption** : Possibilité de mettre à jour ou maintenir un nœud sans arrêter le service

6.2 Limites

- **Complexité accrue** : Gestion plus complexe qu'un système centralisé
- **Latence de réPLICATION** : Délai entre l'écriture sur le primaire et sa disponibilité sur les secondaires
- **Pas de scalabilité d'écriture** : Toutes les écritures passent par le primaire, qui peut devenir un goulot d'étranglement
- **Capacité limitée** : Chaque nœud contient une copie complète des données, limitée par la capacité du plus grand nœud
- **Coût de stockage** : Multiplication du stockage nécessaire (N fois les données pour N nœuds)

Note importante : La réPLICATION n'est pas la stratégie utilisée par MongoDB pour la montée en charge. Pour cela, MongoDB propose le sharding (partitionnement des données), qui permet de distribuer les données sur plusieurs shards, chacun étant un Replica Set.