Rapport d'Analyse et d'Expérimentation de Redis

Houda FILALI

29 janvier 2025

1 Introduction et Présentation de Redis

FIGURE 1 – Logo de Redis

Dans le contexte de ce travail pratique sur les bases de données NoSQL, nous avons étudié Redis, un système de gestion de bases de données orienté clé-valeur. Cette étude s'inscrit dans une réflexion plus large sur les alternatives aux bases relationnelles classiques et sur les avantages des bases NoSQL dans des scénarios nécessitant des performances élevées et une grande flexibilité.

Pourquoi Redis ?
Redis se distingue des bases relationnelles comme MySQL ou PostgreSQL par son fonctionnement en mémoire. Contrairement aux bases de données relationnelles qui stockent leurs informations sur disque et nécessitent des accès séquentiels, Redis permet une gestion extrêmement rapide des données grâce à son stockage en RAM. Ce mode de fonctionnement lui confère des temps de réponse quasi-instantanés, ce qui en fait un choix privilégié pour les applications nécessitant des accès fréquents et rapides, telles que la mise en cache, la gestion de sessions ou la messagerie en temps réel.

Avantages Clés de Redis

- Stockage en mémoire pour des performances optimales.
- Flexibilité dans la gestion des structures de données.
- Adapté aux applications nécessitant un accès rapide aux informations.
- Possibilité de scalabilité et distribution des données.

Cependant, cette rapidité s'accompagne de plusieurs différences fondamentales par rapport aux bases relationnelles, notamment en termes de structuration des données, de persistance et de scalabilité.

1.1 Optimisation des Accès aux Données

L'un des inconvénients majeurs des bases de données relationnelles réside dans leur dépendance aux opérations d'entrée/sortie sur disque. Lorsqu'une donnée est demandée, plusieurs étapes sont nécessaires :

- Localisation de la donnée sur le disque et positionnement de la tête de lecture.
- Lecture d'un bloc entier de données (généralement 512 octets ou plus), même si seule une fraction est réellement utile.
- Transfert du bloc en mémoire principale avant son exploitation par le système.

Comparaison des temps d'accès :
- Temps d'accès mémoire : 10^{-8} à 10^{-7} secondes.
- Temps d'accès disque : environ 10^{-2} secondes, soit plusieurs millions de fois plus lent.
- Débit mémoire : plusieurs Go/s, contre quelques centaines de Mo/s pour un disque dur.

En supprimant l'étape de lecture sur disque, Redis garantit des accès quasi-instantanés aux données, ce qui le rend particulièrement adapté aux applications exigeant une faible latence.

1.2 Flexibilité et Modèle de Données

Comparaison avec les Bases Relationnelles

Bases Relationnelles :
- Structuration rigide avec des schémas définis.
- Relation entre les entités strictement établies.
- Optimisées pour les transactions complexes.

Redis :
- Aucune contrainte de schéma fixe.
- Stockage de données hétérogènes et dynamiques.
- Adapté aux applications nécessitant une flexibilité maximale.

Redis, en revanche, adopte une approche plus flexible. Grâce à son absence de schéma strict, il permet de stocker des objets hétérogènes sans contrainte de format. Par exemple, il est possible de créer une structure utilisateur avec certains attributs tout en définissant une autre structure d'un type complètement différent, sans lien formel entre eux. Cette caractéristique facilite l'adaptation aux besoins métier en constante évolution.

En outre, Redis intègre des structures de données avancées telles que les hashmaps, les listes, les ensembles et les sorted sets, qui offrent des solutions adaptées aux différents cas d'usage. Ces structures permettent non seulement un stockage optimisé, mais aussi des traitements avancés directement au niveau de la base de données, sans nécessiter d'opérations complexes côté applicatif.

2 Installation et Configuration de Redis avec Docker

L'installation de Redis via Docker permet de s'affranchir des contraintes de configuration locale.

```
docker run --name redis -d -p 6379:6379 redis
docker exec -it redis redis-cli ping  # V rification du fonctionnement de Redis
docker ps  # V rification des conteneurs actifs
```

Après l'exécution de ces commandes, Redis est prêt à être utilisé.

3 Manipulation des Données avec Redis

3.1 Clé-Valeur

Redis fonctionne principalement avec des paires clé-valeur, permettant un accès rapide aux données. Chaque clé est unique et associée à une valeur qui peut être une chaîne de caractères, un nombre, ou même une structure plus complexe.

```
SET user:1234 "Samir"
GET user:1234
DEL user:1234
```

Ce modèle est utilisé pour le stockage de sessions, de données temporaires et de caches. Il permet un accès direct aux informations en un temps constant $O(1)$.

3.2 Incrémentation des Valeurs Numériques

Redis permet d'incrémenter une valeur numérique stockée sous une clé sans devoir la récupérer et la modifier manuellement. Cette fonctionnalité est particulièrement utile pour la gestion de compteurs, comme les vues d'une page, les identifiants uniques ou le suivi d'événements.

```
SET compteur 0
INCR compteur
INCR compteur
INCR compteur
GET compteur
```

Dans cet exemple, la clé `compteur` est initialisée à 0, puis incrémentée plusieurs fois. À chaque appel de `INCR`, la valeur est augmentée de 1.

3.3 Expiration des Clés (TTL - Time To Live)

Redis permet d'attribuer une durée de vie aux clés pour les supprimer automatiquement après un certain temps. Cette fonctionnalité est essentielle pour gérer des sessions utilisateurs, des jetons d'authentification ou des caches temporaires.

```
SET temporaire "valeur"
EXPIRE temporaire 120  # Expiration apr s 120 secondes
TTL temporaire         # Temps restant avant expiration
```

- `EXPIRE` définit une durée de vie en secondes. - `TTL` permet de vérifier le temps restant avant suppression. Une valeur de `-1` signifie que la clé n'a pas de durée de vie définie.

Une fois le temps écoulé, la clé est automatiquement supprimée, évitant ainsi l'encombrement de la base de données avec des informations obsolètes.

3.4 Listes

Les listes permettent de stocker une séquence ordonnée de valeurs, où les éléments peuvent être ajoutés à gauche ou à droite, et récupérés selon un ordre défini. Contrairement aux ensembles, elles acceptent les doublons.

```
RPUSH mesCours "BDA"
RPUSH mesCours "Services Web"
LRANGE mesCours 0 -1  # Affiche tous les    lments    de la liste
LPOP mesCours         # Supprime et retourne le premier    lment
```

Les listes sont particulièrement adaptées à la gestion de files d'attente, de journaux d'événements et de flux de messages.

3.5 Hashmaps (Tables de Hachage)

Les hashmaps permettent de stocker des objets sous forme de paires clé-champs-valeur, facilitant l'organisation et la récupération rapide des informations.

```
HSET user:11 username "Youssef"  # D finit le champ "username"    "Youssef" pour
   user:11
HSET user:11 age 31  # D finit le champ "age"    31 pour user:11
HSET user:11 email "samir.youces@"  # D finit le champ "email"    "samir.youces@"
   pour user:11

HGETALL user:11  # R cup re tous les champs et valeurs associ s    user:11

HMSET user:4 username "houda" age 22 email "hfilali@"  # D finit plusieurs champs
   pour user:4
HGETALL user:4  # V rifie les donn es stock es pour user:4

HINCRBY user:4 age 23  # Incr mente la valeur du champ "age" de 23
```

```
HGET user:4 age  # V rifie la nouvelle valeur du champ "age"

HVALS user:4  # Retourne uniquement les valeurs des champs de user:4
```

Les hashmaps sont idéales pour stocker des profils utilisateurs ou des objets structurés sans contrainte de schéma, offrant ainsi une grande flexibilité.

3.6 Ensembles

Les ensembles permettent de stocker des éléments uniques, évitant ainsi les doublons. Ils sont utiles pour les relations utilisateur, la gestion de tags ou encore les abonnements.

```
SADD utilisateurs "Augustin"
SADD utilisateurs "Houda"
SADD utilisateurs "Marc"
SREM utilisateurs "Marc"
SMEMBERS utilisateurs  # Liste des membres de l'ensemble
SUNION utilisateurs autresUtilisateurs  # Union de deux ensembles
```

Les ensembles garantissent que chaque valeur est unique et prennent en charge des opérations avancées comme les unions et les intersections.

3.7 Sorted Sets (ZSET)

Les **sorted sets** (ou ensembles triés) fonctionnent comme des ensembles, mais chaque élément est associé à un score qui permet de les trier automatiquement.

```
ZADD score4 19 "Augustin"  # Ajoute "Augustin" avec un score de 19
ZADD score4 18 "Sara"       # Ajoute "Sara" avec un score de 18
ZADD score4 10 "Houda"      # Ajoute "Houda" avec un score de 10

ZRANGE score4 0 -1  # Affiche les    lments    tri s par score croissant
ZREVRANGE score4 0 -1  # Affiche les    lments    tri s par score d croissant
ZRANK score4 "Augustin" # Retourne le rang (position) de "Augustin" dans le
   classement (ordre croissant)
```

Les ensembles triés sont souvent utilisés pour classer des scores de jeux, organiser des classements dynamiques ou gérer des priorités dans des files d'attente.

3.8 Système de Messagerie Pub/Sub

Redis offre un mécanisme de messagerie en temps réel basé sur le modèle **publish/subscribe (Pub/Sub)**. Ce modèle permet aux clients de s'abonner à des canaux et de recevoir des messages lorsqu'un éditeur publie du contenu sur ces canaux.

3.8.1 Abonnement à un Canal (Subscriber)

Les clients peuvent s'abonner à un ou plusieurs canaux pour recevoir des messages envoyés par les éditeurs.

```
SUBSCRIBE mescours user:1  # Abonnement aux canaux "mescours" et "user:1"
PSUBSCRIBE mes*  # Abonnement    tous les canaux commen ant par "mes"
```

Lorsque l'un de ces canaux reçoit un message, tous les abonnés reçoivent automatiquement la mise à jour en temps réel :

```
1) "message"
2) "mescours"
3) "Un nouveau cours MongoDB"

1) "message"
2) "mescours"
```

```
3) "Houda Filali"

1) "pmessage"
2) "mes*"
3) "mesnotes"
4) "Une nouvelle note est arriv e"
```

3.8.2 Publication de Messages (Publisher)

Les éditeurs peuvent envoyer des messages sur un canal auquel des clients sont abonnés. Les abonnés recevront ces messages en temps réel.

```
PUBLISH mescours "Un nouveau cours MongoDB"
PUBLISH mescours "Houda Filali"
PUBLISH mesnotes "Une nouvelle note est arriv e"
```

Chaque commande `PUBLISH` envoie un message au canal spécifié, et le nombre d'abonnés ayant reçu le message est retourné par Redis.

3.8.3 Utilisation du Pub/Sub

Ce système est particulièrement utile pour :

- La notification en temps réel (ex. mises à jour en direct dans une application).
- La communication entre microservices sans nécessiter une base de données intermédiaire.
- La gestion des événements asynchrones dans une architecture distribuée.

3.9 Gestion des Bases de Données Multiples

Redis prend en charge plusieurs bases de données distinctes au sein d'une même instance. Par défaut, Redis offre **16 bases de données** indexées de `0` à `15`. Il est possible de basculer entre ces bases avec la commande `SELECT`.

```
SELECT 1  # Bascule vers la base de donn es index e 1
OK
SELECT 0  # Retour    la base de donn es par d faut (index 0)
OK
```

Chaque base de données est indépendante, et les clés stockées dans une base ne sont pas accessibles depuis une autre. On peut afficher les clés d'une base spécifique avec :

```
SELECT 0  # S lection de la base de donn es 0
KEYS *  # Liste toutes les cl s de cette base
1) "29janvier"
2) "user:11"
3) "mesCours"
4) "autresUtilisateurs"
5) "demo"
6) "user:4"
7) "utilisateurs"
8) "score4"
```

Cela permet d'organiser les données de manière compartimentée et d'isoler certaines informations pour des cas d'utilisation spécifiques.