Rapport TP1 : Redis

ABDOULAHI SALAHANNE Ahmed 12306430

Ing3 Informatique

Enseignant : Mr. Y.Samir

Table des matières

Introduction 2
Prise en main de Redis 2
Exemple de CRUD clé-valeur 2
Utilisation typique : Compteur de visiteurs 3
Durée de vie d'une clé 3
Structures de données dans Redis 3
    Exemple de liste 3
    Exemple de Set (ensemble) 4
    Union de deux ensembles 4
Ensembles ordonnés (Sorted Sets) 5
Système de Publication/Abonnement (Pub/Sub) 5
Gestion des bases de données Redis 6
Reprise après panne (Persistence) 6
    RDB (Redis DataBase) 6
    AOF (Append Only File) 6
Conclusion 6

Introduction

Dans la famille NoSQL, il existe quatre grandes catégories :

1. Bases de données orientées **clés-valeurs**,
2. Bases orientées **colonnes**,
3. Bases orientées **documents**,
4. Bases orientées **graphes**.

Dans ce rapport, nous allons nous concentrer sur Redis, une base de données orientée **clés-valeurs**. Redis est principalement utilisée conjointement avec une base de données relationnelle pour accélérer l'accès aux données grâce à son stockage en RAM. Bien que Redis puisse également persister les données sur disque, cela nécessite une configuration spécifique.

Prise en main de Redis

```
redis-server
```

– La commande redis-server lance le serveur Redis. Il s'agit du processus principal qui va écouter les requêtes des clients et gérer la base de données en mémoire.

```
redis-cli
```

– La commande redis-cli démarre le client en ligne de commande, permettant d'interagir avec le serveur Redis. – Une fois cette commande exécutée, vous pouvez envoyer des commandes Redis au serveur.

Exemple de CRUD clé-valeur

```
127.0.0.1:6379> SET demo "Bonjour"
OK
```

– La commande SET permet d'ajouter une nouvelle clé-valeur dans la base de données Redis. Ici, on associe la clé "demo" à la valeur "Bonjour". Redis répond avec "OK" pour indiquer que l'opération a réussi.

```
127.0.0.1:6379> SET user:1234 "Samir"
OK
```

– Ici, nous utilisons à nouveau SET pour ajouter un utilisateur avec l'ID "1234" et le nom "Samir". – La syntaxe "user :1234" est utilisée pour représenter une clé unique.

```
127.0.0.1:6379> GET user:1234
"Samir"
```

– La commande GET est utilisée pour récupérer la valeur associée à une clé donnée. Ici, "user :1234" est la clé, et Redis renvoie "Samir" comme valeur associée.

```
127.0.0.1:6379> DEL user:1234
(integer) 1
```

– La commande DEL supprime la clé spécifiée et sa valeur. Le nombre entier "1" indique que la clé a été supprimée avec succès.

```
127.0.0.1:6379> DEL user:12346
(integer) 0
```

– La commande DEL essaie de supprimer la clé "user :12346". Comme cette clé n'existe pas, Redis renvoie "0", ce qui signifie qu'aucune clé n'a été supprimée.

Utilisation typique : Compteur de visiteurs

```
127.0.0.1:6379> SET 1mars 0
OK
```

– Nous initialisons une clé "1mars" à zéro pour indiquer le nombre initial de visiteurs le 1er mars. – Cette clé est associée à la valeur "0".

```
127.0.0.1:6379> INCR 1mars
(integer) 1
```

– La commande INCR incrémente la valeur associée à la clé donnée de 1. Dans ce cas, la valeur de "1mars" passe de 0 à 1.

```
127.0.0.1:6379> INCR 1mars
(integer) 2
```

– INCR est à nouveau utilisée pour ajouter 1 à la valeur de "1mars". Cette fois, la valeur devient 2.

```
127.0.0.1:6379> DECR 1mars
(integer) 1
```

– La commande DECR décrémente la valeur de la clé "1mars" de 1. La nouvelle valeur est donc 1.

Durée de vie d'une clé

```
127.0.0.1:6379> SET macle mavaleur
OK
```

– Nous utilisons SET pour créer une clé "macle" avec la valeur "mavaleur". Redis confirme que l'opération a réussi en renvoyant "OK".

```
127.0.0.1:6379> EXPIRE macle 120
(integer) 1
```

– La commande EXPIRE définit une durée de vie pour la clé "macle". Ici, la clé expirera après 120 secondes (2 minutes). – Redis répond "1" pour indiquer que la durée a bien été définie.

```
127.0.0.1:6379> TTL macle
(integer) 112
```

– La commande TTL (Time To Live) permet de vérifier combien de temps il reste avant que la clé "macle" expire. Ici, il reste 112 secondes avant expiration.

```
127.0.0.1:6379> DEL macle
(integer) 1
```

– La commande DEL est utilisée pour supprimer la clé "macle". Redis renvoie "1", indiquant que la suppression a bien eu lieu.

Structures de données dans Redis

Exemple de liste

```
127.0.0.1:6379> RPUSH mesCours "BDA"
(integer) 1
```

– La commande RPUSH permet d'ajouter un élément à la fin d'une liste. Ici, nous ajoutons "BDA" à la liste "mesCours". – Redis répond avec "1", le nouveau nombre d'éléments dans la liste.

```
127.0.0.1:6379> RPUSH mesCours "Services Web"
(integer) 2
```

– Nous ajoutons un autre élément "Services Web" à la liste "mesCours", portant ainsi le nombre d'éléments dans la liste à 2.

```
127.0.0.1:6379> LRANGE mesCours 0 -1
1) "BDA"
2) "Services Web"
```

– La commande LRANGE permet de récupérer un sous-ensemble d'une liste, dans ce cas, tous les éléments de "mesCours". – Ici, nous demandons tous les éléments de l'index 0 à l'index -1 (fin de la liste).

```
127.0.0.1:6379> LPOP mesCours
"BDA"
```

– La commande LPOP permet de retirer et de renvoyer le premier élément de la liste. Ici, "BDA" est retiré de "mesCours".

Exemple de Set (ensemble)

```
127.0.0.1:6379> SADD utilisateurs "Ines"
(integer) 1
```

– La commande SADD permet d'ajouter un élément à un ensemble. Ici, nous ajoutons "Ines" à l'ensemble "utilisateurs". – Redis renvoie "1" pour indiquer que l'élément a bien été ajouté.

```
127.0.0.1:6379> SADD utilisateurs "Samir"
(integer) 1
```

– Nous ajoutons "Samir" à l'ensemble "utilisateurs". Le nombre d'éléments dans l'ensemble augmente.

```
127.0.0.1:6379> SMEMBERS utilisateurs
1) "Ines"
2) "Samir"
```

– La commande SMEMBERS permet de lister tous les éléments d'un ensemble. Ici, nous voyons que l'ensemble "utilisateurs" contient "Ines" et "Samir".

```
127.0.0.1:6379> SREM utilisateurs "Samir"
(integer) 1
```

– La commande SREM supprime un élément d'un ensemble. Ici, "Samir" est retiré de l'ensemble "utilisateurs".

```
127.0.0.1:6379> SMEMBERS utilisateurs
1) "Ines"
```

– Après la suppression de "Samir", l'ensemble "utilisateurs" ne contient plus que "Ines".

Union de deux ensembles

```
127.0.0.1:6379> SADD autresUtilisateurs "Antoine"
(integer) 1
```

– Nous ajoutons "Antoine" à l'ensemble "autresUtilisateurs".

```
127.0.0.1:6379> SADD autresUtilisateurs "Philippe"
(integer) 1
```

– Nous ajoutons également "Philippe" à l'ensemble "autresUtilisateurs".

```
127.0.0.1:6379> SUNION utilisateurs autresUtilisateurs
1) "Antoine"
2) "Philippe"
3) "Ines"
```

– La commande SUNION permet d'obtenir l'union de deux ensembles. Ici, l'union des ensembles "utilisateurs" et "autresUtilisateurs" contient "Antoine", "Philippe" et "Ines".

Ensembles ordonnés (Sorted Sets)

```
127.0.0.1:6379> ZADD score4 19 "Augustin"
(integer) 1
```

– La commande ZADD ajoute l'élément "Augustin" avec un score de 19 à l'ensemble ordonné "score4".
– Redis répond avec "1", ce qui signifie que l'élément a été ajouté avec succès.

```
127.0.0.1:6379> ZADD score4 18 "Ines"
(integer) 1
127.0.0.1:6379> ZADD score4 10 "Samir"
(integer) 1
```

– Ici, nous ajoutons "Ines" avec un score de 18 et "Samir" avec un score de 10 à l'ensemble ordonné "score4". – Redis confirme chaque ajout avec "1".

```
127.0.0.1:6379> ZRANGE score4 0 -1
1) "Samir"
2) "Ines"
3) "Augustin"
```

– La commande ZRANGE permet de récupérer les éléments triés de l'ensemble ordonné "score4" par score, dans l'ordre croissant. – Ici, "Samir" a le score le plus bas, suivi de "Ines" et "Augustin".

Vous pouvez également trier par ordre décroissant avec la commande `ZREVRANGE` :

```
127.0.0.1:6379> ZREVRANGE score4 0 -1
1) "Augustin"
2) "Ines"
3) "Samir"
```

– La commande ZREVRANGE trie les éléments de l'ensemble ordonné "score4" par score, dans l'ordre décroissant. – Ici, "Augustin" a le score le plus élevé, suivi de "Ines" et "Samir".

```
127.0.0.1:6379> ZRANK score4 "Augustin"
(integer) 2
```

– La commande ZRANK retourne le rang de l'élément "Augustin" dans l'ensemble ordonné "score4".
– Ici, "Augustin" a le score le plus élevé, donc son rang est 2 (en comptant à partir de 0).

Système de Publication/Abonnement (Pub/Sub)

```
127.0.0.1:6379> SUBSCRIBE mescours
```

– La commande SUBSCRIBE permet au client 1 de s'abonner au canal "mescours". – Il recevra tous les messages publiés sur ce canal.

```
127.0.0.1:6379> PUBLISH mescours "Un nouveau cours sur MongoDB"
```

– La commande PUBLISH permet au client 2 de publier un message sur le canal "mescours". – Le message est "Un nouveau cours sur MongoDB".

```
1) "Un nouveau cours sur MongoDB"
```

– Le client 1, abonné au canal "mescours", reçoit immédiatement le message publié par le client 2.

Vous pouvez aussi vous inscrire à plusieurs canaux, par exemple avec la commande `PSUBSCRIBE` :

```
127.0.0.1:6379> PSUBSCRIBE mes*
```

– La commande PSUBSCRIBE permet au client de s'abonner à plusieurs canaux qui correspondent au motif "mes*". – Cela permet d'écouter tous les canaux dont le nom commence par "mes".

Gestion des bases de données Redis

```
127.0.0.1:6379> SELECT 1
OK
127.0.0.1:6379[1]> KEYS *
(empty array)
```

– La commande SELECT permet de changer de base de données. Ici, nous sélectionnons la base de données avec l'index 1. – La commande KEYS * permet d'afficher toutes les clés de la base de données. Comme elle est vide, Redis renvoie un tableau vide.

```
127.0.0.1:6379[1]> SELECT 0
OK
127.0.0.1:6379> KEYS *
1) "users"
2) "user:1"
3) "user:123"
4) "counter:1mars"
```

– La commande SELECT 0 permet de revenir à la base de données par défaut, celle avec l'index 0. – La commande KEYS * affiche toutes les clés présentes dans la base de données par défaut. – Ici, les clés sont "users", "user :1", "user :123" et "counter :1mars".

Reprise après panne (Persistence)

Redis peut sauvegarder les données en disque pour pouvoir les récupérer après une panne. Cependant, si la persistance n'est pas activée, les données sont perdues en cas de redémarrage.

RDB (Redis DataBase)

Redis peut effectuer des sauvegardes de la base de données à intervalles réguliers.

AOF (Append Only File)

Redis peut également enregistrer chaque commande dans un fichier, permettant une récupération plus fine des données.

Conclusion

Redis est une base de données extrêmement rapide et flexible qui permet de gérer efficacement des données en mémoire. Il est particulièrement adapté aux cas où la performance et la réactivité sont essentielles, comme pour les caches ou les comptages en temps réel. Cependant, il nécessite une gestion de la persistance des données (RDB, AOF) pour éviter toute perte en cas de redémarrage.