REMOTE DICTIONARY SERVER

Mme Ines Slimene
Ines_slimene@yahoo.fr

Introduction

Redis est une solution open-source codée intégralement en C par Salvatore Sanfilippo

Il est sponsorisé par VMWare

Les sites Internet à fort traffic utilisent Redis

Redis est avant tout un serveur de données en mémoire. Il est vraiment rapide. Une machine classique traite sans problèmes 120 000 opérations par seconde.

Introduction

Redis est un système de gestion de base de données (SGBD) fonctionnant sur le principe clé-valeur scalable.

Ce système de type NoSQL est connu pour obtenir de très bonnes performances.

Redis est l'un des système de gestion de base de données les plus populaires.

Ce système est notamment utilisés sur de gros projets web tels que GitHub, Stack Overflow, Craigslist ou The Guardian.

Son nom viens de la concaténation du terme « REmote DIctionary Server », signifiant littéralement « serveur de dictionnaire distant ».

Introduction

La force caractéristique de Redis consiste à stocker les données en mémoire vive (RAM).

De nombreux langages de programmation supporte Redis, incluant notamment Java, PHP, Javascript (via node.js), C, C++, Objective-C, Perl, Python ou encore Ruby.

Redis est particulièrement utile pour manipuler des types de données simples tels que des chaînes de caractères, des tableaux associatifs, listes, ensembles et ensembles ordonnés.

NB : La clé doit être de type chaine de caractère mais la valeur peut être de type chaînes de caractères (String), des tableaux associatifs (Hash), listes (list), ensembles (set) et ensembles ordonnés (stored set).

Persistence

On dispose de trois moyens pour assurer la persistence de Redis: RDB, AOF et la commande SAVE.

Mechanisme RDB

RDB crée une copie de toutes les données en mémoire et les stockes dans un espace de stockage secondaire permanent. Ceci pour un intervalle de temp bien déterminé. Donc il y a possibilité de perte de données qui seront stockées après le dernier snapshot de RDB.

AOF

AOF crée des fichiers log pour toutes les opérations d'écriture effectuées dans le serveur. Le problème est que a chaque opération il y a une écriture sur le disque (fichier log) tache couteuse en terme de stockage et de temps d'exécution.

SAVE Command

On peut forcer le serveur Redis a creer des snapshots RDB a tout moment avec la commande SAVE.

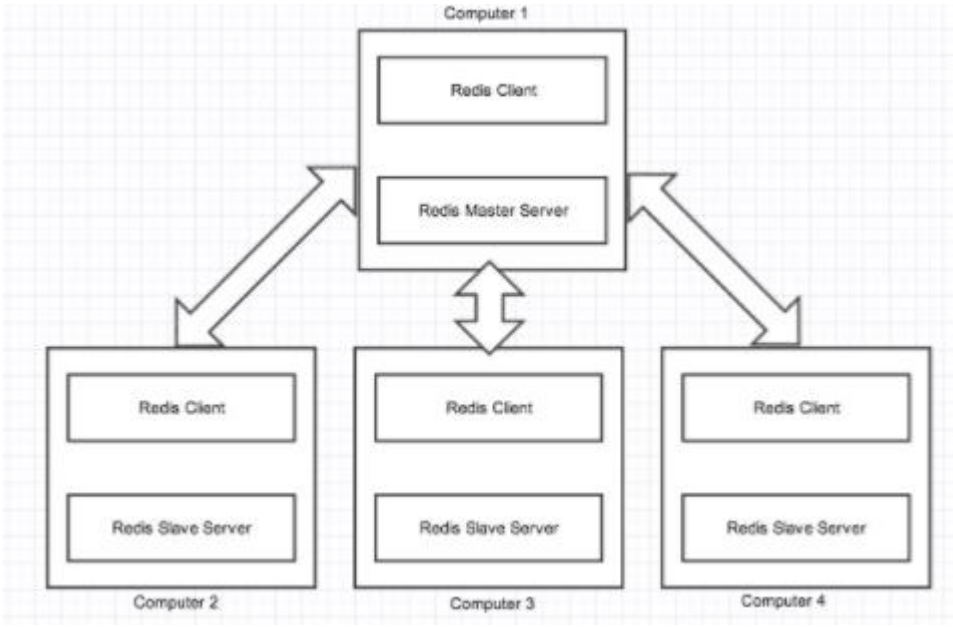
Sauvegarde et récupération des données

Redis ne fourni pas un mécanisme de sauvegarde et récupération. Donc en cas de problème on risqué de perdre toutes les données.

Possibilité de travailler avec les réplications

Réplication

Dans un environnement de réplication, un ensemble de machine partage les mêmes informations. Si l'un des nœuds tombe en panne les autres prennent la relève.



Réplication

Tous les slaves contiennent les même données que le master. Si on ajoute un slave au cluster, le master synchronise automatiquement les données dans le nouveau slave.

Toutes les opérations d'écriture sont dirigées vers le master. Quand une opération est effectuée le master réplique les Nouvelles données dans les slaves.

Si le nombre d'opération de lecture est grand, le master les distribue sur les slaves pour les exécuter a temp.

Si l'un des slave tombe en panne, tout l'environnement continuer a bien fonctionner. Et quand il reprend son travail, le master lui envoie les mises a jour nécessaire.

Si le master crache et perd ses données, l'un des slave sera converti en master. On n'ajoute pas un nouveau nœud car il ne contiendra aucune données.

Si le master crache mais conserve encore ses données (persistance physique) il suffit de redémarrer le serveur.

Clustering

Le clustering est une technique qui permet de découper les données et les stockées sur plusieurs nœuds. L'avantage est de pouvoir stocker un volume de données plus grand.

Supposons qu'on un serveur redis avec 64G de mémoire. Si on utilise 10 serveurs on aura 640 G de ram.

Inconvénients

Redis demande un temps pour bien comprendre ses structures.

Redis étant une pure base en mémoire, il faut activer soit la sauvegarde sur disque, soir un slave pour sauver vos données.

Vu la taille limitée de la mémoire, Redis ne peut pas stocker des fichiers volumineux, il ne peut stocker que des informations textuelles de petite taille et qui nécessite un accès en modification ou insertion très rapide

Par défaut, la base n'est pas sécurisée. Un client peut se connecter dessus et lancer un FLUSHDB, qui videra les données. Il faut activer soit la protection par clé, soit simplement faire en sorte que les serveurs Redis ne soient pas sur Internet.

Sécurité :

- Vérifier valeur mot de passe : CONFIG get requirepass
- Modifier mot de passe : CONFIG set requirepass « redis"
- S'authentifier avec mot de passe : AUTH password

String

C'est le type le plus simple : à une clef, on peut y associer une valeur. Cette valeur peut-être une string de tout type (json, valeur d'une image jpeg), mais aussi un entier.

La limite est de 1 Go.

Les principales commandes : SET, GET, INCR, DECR, et GETSET.

redis 127.0.0.1:6379> SET compteur 10

redis 127.0.0.1:6379> INCR compteur

reredis 127.0.0.1:6379> GET compteur

redis 127.0.0.1:6379> GETSET compteur 123

Hash

Une hash permet de stocker dans un même enregistrement plusieurs couples de clef/valeurs.

Chaque hash peut stocker jusqu'à 2^{32}-1 (~ 4 milliard) paire

Commandes : HSET, HGET, HLEN, HDEL

HGETALL pour obtenir tous les couples clef-valeur,

HKEYS et HVALS pour obtenir toutes les clefs ou valeurs.

redis 127.0.0.1:6379> HSET nosql redis "clef/valeur"

redis 127.0.0.1:6379> HGET nosql redis

redis 127.0.0.1:6379> HGETALL nosql

redis 127.0.0.1:6379> HKEYS nosql

redis 127.0.0.1:6379> HVALS nosql

Listes

Les listes de redis sont des listes liées (linked list). Cela veut dire que même si vous avez des millions d'enregistrement dans une liste, cela sera toujours aussi rapide d'insérer un élément en tête ou en queue de liste.

Chaque liste peut stocker jusqu'à 2^{32}-1 (~ 4 milliard) éléments

Commandes : RPUSH et LPUSH qui permettent respectivement d'ajouter un élément en fin ou en début de liste,

LRANGE pour obtenir une partie des éléments de la liste,

LINDEX pour obtenir un seul élément de la liste, LLEN pour obtenir la taille de la liste.

redis 127.0.0.1:6379> RPUSH messages "Bienvenue"

redis 127.0.0.1:6379> LPUSH messages "Merci"

redis 127.0.0.1:6379> LRANGE messages 0 3

Sets

Les Sets sont des collections d'objets non ordonnées.

Les commandes : SADD pour ajouter une valeur à un set, SCARD pour obtenir la taille d'un set,

SINTER, SUNION, SDIFF qui permettent respectivement d'obtenir l'intersection, l'union et la différences entre 2 sets.

Ces commandes existent en version "STORE" ; ainsi SINTERSTORE permet de stocker dans un nouveau set l'intersection de 2 autres.

redis 127.0.0.1:6379> SADD supermarche pommes

redis 127.0.0.1:6379> SADD monFrigo pommes

redis 127.0.0.1:6379> SCARD monFrigo

redis 127.0.0.1:6379> SINTER supermarche monFrigo

Sets triés

Les sets triés (Sorted Set) sont similaires à des Sets mais ajoutent une notion de score aux valeurs ajoutées aux Set, ce qui permet de faire des tris.

Les commandes commencent toutes par Z comme Zorted Set.

Exemple : ZRANGE, ZRANGEBYSCORE et ZRANK qui tirent parti des scores des données stockées.

redis 127.0.0.1:6379> ZADD savants 1571 "Johannes Kepler"

redis 127.0.0.1:6379> ZADD savants 1879 "Albert Einstein"

redis 127.0.0.1:6379> ZADD savants 1858 "Max Planck"

redis 127.0.0.1:6379> ZRANGE savants 0 -1

sredis 127.0.0.1:6379> ZREVRANGE savants 0 -1 WITHSCORES

redis 127.0.0.1:6379> ZRANGEBYSCORE savants 1800 2010

Domaines d’utilisation

- Redis comme mémoire cache
  - Gestion des sessions – hashes
  - Excellent outil pour la gestion des scores – sorted sets
- Redis comme base de données
  - Persistance dans le disque
  - Haute performance

Redis comme bus de message

- Basé sur les fonctionnalités Pub/Sub
- Queues avec des structures en liste
- Resque – Ruby library for creating background jobs