ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ
Π.Μ.Σ. Ανάπτυξη Λογισμικού και Νέφος

Διαχείριση και Αναλυτική Δεδομένων στο Υπολογιστικό Νέφος

Υλοποίηση εφαρμογής με χρήση του Redis

ΕΥΣΤΑΘΙΟΣ ΙΩΣΗΦΙΔΗΣ
AM: mai25017

ΘΕΣΣΑΛΟΝΙΚΗ ΑΠΡΙΛΙΟΣ 2025

ΕΙΣΑΓΩΓΗ

Η εργασία αυτή επικεντρώνεται στην ανάπτυξη μιας εφαρμογής συντόμευσης URL με χρήση της βάσης δεδομένων Redis. Οι βασικές λειτουργίες της εφαρμογής περιλαμβάνουν την εισαγωγή long URL και μετατροπή τους σε short URL, την αναζήτηση long URL με βάση το short URL, καθώς και την παρουσίαση στατιστικών στοιχείων. Η επιλογή του Redis ως τεχνολογίας βασίστηκε στις υψηλές επιδόσεις και την ευελιξία των δομών δεδομένων που προσφέρει, καθιστώντας το ιδανικό για εφαρμογές που απαιτούν γρήγορη πρόσβαση και ενημέρωση δεδομένων.

ΜΕΘΟΔΟΛΟΓΙΑ

1.1 ΣΤΗΣΙΜΟ ΥΠΟΔΟΜΗΣ (DOCKER REDIS)

Για τεχνικούς λόγους, δεν χρησιμοποιήθηκε η διαθέσιμη εικονική μηχανή στο Virtual Box, και επιλέχτηκε η λύση του Docker. Αρχικά παρακάτω ακολουθούν τα βήματα εγκατάστασης και χρήσης του Docker σε σύστημα Fedora Workstation 41 (ανάλογα γίνεται και σε συστήματα Ubuntu/Debian).

1. Εγκατάσταση Docker σε Fedora

Η εγκατάσταση γίνεται με την εντολή:

sudo dnf install docker-cli containerd docker-compose

Περισσότερα στην διεύθυνση:
https://docs.fedoraproject.org/en-US/quick-docs/installing-docker/

2. Εκκίνηση και ενεργοποίηση Docker

Συνήθως με την εγκατάσταση, ενεργοποιείται η υπηρεσία. Σε περίπτωση που δεν γίνει αυτό, δοκιμάζουμε τις παρακάτω εντολές:

sudo systemctl start docker
sudo systemctl enable docker

3. Προσθήκη χρήστη στην ομάδα docker (για να μην χρειάζεται sudo)

Ανάλογα την διανομή, μπορεί να μην χρειάζονται τα παρακάτω. Όμως για να μην χρειάζεται η χρήση του sudeo μπορείτε να βάλετε τον χρήστη στην ομάδα του docker και έτσι δεν θα χρειάζεται να εκτελείτε sudo docker …

sudo usermod -aG docker $USER

Για να ισχύσουν οι αλλαγές, εκτελούμε την εντολή:

newgrp docker

ή απλά κάνουμε logout/login

4. Λήψη της εικόνας Redis

Υπάρχουν δύο εναλλακτικές:
https://hub.docker.com/_/redis
https://hub.docker.com/r/redis/redis-stack

Στην εργασία προτιμήθηκε η πρώτη διεύθυση. Για να κατεβάσουμε το image και να εκτελέσουμε το container εκτελούμε την παρακάτω εντολή:

docker run --name my-redis -p 6379:6379 -d redis

Αυτή η εντολή θα ξεκινήσει το container που ονομάζεται "my-redis" στην πόρτα 6379. Μετά την εκτέλεση της, μπορούμε να επαληθεύσουμε ότι εκτελείται με:

docker ps

Θα πρέπει να δούμε το container "my-redis" στην λίστα με τα running containers.

Αφού έχουμε ήδη δημιουργήσει το container με το όνομα "my-redis", δεν χρειάζεται να το ξαναδημιουργήσουμε (παραπάνω εντολή). Απλώς πρέπει να το εκκινήσουμε. Οι εντολές για εκκίνηση και τερματισμό του container είναι:

docker stop my-redis
docker start my-redis

Για δοκιμή, δημιουργήθηκε το script σε python:

```
#!/usr/bin/env python3

import redis

# Σύνδεση με Redis
r = redis.Redis(host='localhost', port=6379, db=0)

# Δοκιμαστική λειτουργία
r.set('test_key', 'Hello Redis!')
value = r.get('test_key')
print(value.decode('utf-8'))
```

Αυτό βοήθησε στο έλεγχο σύνδεσης με το Redis.

Για να δοκιμάσουμε εάν εκτελείται η μηχανή με το Redis, μπορουμε να έχουμε το τερματικό με την εντολή:

docker exec -it my-redis redis-cli

Για να δούμε τις εγγραφές, μπορούμε να χρησιμοποιήσουμε την εντολή (προφανώς η βάση είναι κενή):

KEYS *

Σημαντκά:
Με **exit** εξερχόμαστε από το τερματικό του docker και με **flushall**, αδειάζουμε την βάση από εγγραφές (χρήσιμο για να αδειάζουμε την βάση από τα δοκιμαστικά δεδομένα).

1.2 ΜΕΛΕΤΗ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΑΝΑΛΥΣΗ ΜΕ ΤΗΝ ΒΟΗΘΕΙΑ ΤΟΥ DEEPSEEK

Για την μελέτη της διάλεξης αλλά και βοήθεια για την εργασία, χρησιμοποιήθηκε ο βοηθός τεχνητής νοημοσύνης Deepseek. Η βοήθεια και τα prompts που χρησιμοποιήθηκαν, αναλύονται παρακάτω.

1. Δομές δεδομένων

Ρωτήθηκε ποιες είναι οι δομές δεδομένων στο Redis. Το prompt που δώθηκε ήταν:

Ποιες δομές δεδομένων υποστηρίζει το Redis;

Η περιληπτική απάντηση (ουσιαστικά ήταν η σελίδα https://redis.io/docs/data-types/), βοήθησε στην κατανόηση των δομών δεδομένων που υποστηρίζει το Redis. Έχουν καταγραφεί για καλύτερη μελέτη στο μέλλον και βρίσκονται στον σύνδεσμο: https://github.com/iosifidis/msc_applied_informatics/blob/main/s2/Data%20Management%20and%20Analytics%20in%20the%20Cloud/theory/5th_Redis_Data_Types.md

2. Κόστος λειτουργιών

Επειδή η εργασία ζητούσε το κόστος των λειτουργιών, ρωτήθηκε μια περιληπτική παρουσίαση.Το prompt που δώθηκε ήταν:

Για όλες τις δομές δεδομένων, δώσε μου το κόστος των λειτουργιών τους Big O notation

Η απάντηση (ουσιαστικά πήρε κομμάτια από την σελίδα https://redis.io/docs/data-types/) βοήθησε στην κατανόηση του κόστους που έχει η κάθε δομή δεδομένων του Redis. Επίσης η απάντηση έχει καταγραφή για καλύτερη μελέτη στο μέλλον και βρίσκεται στον σύνδεσμο: https://github.com/iosifidis/msc_applied_informatics/blob/main/s2/Data%20Management%20and%20Analytics%20in%20the%20Cloud/theory/5th_lecture_Redis_Big_O_Notation.md

3. Περιπτώσεις χρήσης δομών δεδομένων Redis

Για να γίνει αντιληπτό, ποιες δομές δεδομένων θα χρησιμοποιήσουμε για την επίλυση του ζητούμενου ερωτήματος, θεωρήθηκε σκόπιμο να ερωτηθεί το Deepseek για περιπτώσεις

χρήσης της κάθε δομής δεδομένων. Η πληροφορία βρίσκεται διάσπαρτη στην τεκμηρίωση του Redis https://redis.io/docs/latest/develop/data-types/. Το prompt που δώθηκε ήταν:

Σε ποιες περιπτώσεις χρησιμοποιούνται οι δομές δεδομένων του Redis;

Η απάντηση βοήθησε στην κατανόηση των περιπτώσεων που χρησιμοποιούμε την κάθε δομή δεδομένων. Η απάντηση έχει καταγραφή και είναι διαθέσιμη στον σύνδεσμο: https://github.com/iosifidis/msc_applied_informatics/blob/main/s2/Data%20Management%20and%20Analytics%20in%20the%20Cloud/theory/5th_Redis_Data_Types_usecases.md

4. Ποιες δομές να χρησιμοποιήσω στην εργασία;

Το τελευταίο prompt που χρησιμοποιήθηκε, θα επιβεβαίωνε την απόφαση των δομών δεδομένων που θα χρησιμοποιηθούν. Πιο συγκεκριμένα, από τις παραπάνω απαντήσεις, εκτιμήθηκε ότι θα χρησιμοποιηθεί string, hash, set. Το prompt που δώθηκε ήταν:

Έχω να υλοποιήσω μια εφαρμογή και θέλω να μου πεις ποιες δομές να χρησιμοποιήσω, γιατί να τις χρησιμοποιήσω και να μου δώσεις και το κόστος λειτουργιών τους
Η εφαρμογή θα πρέπει να κάνει:
- Insertion: ο χρήστης πληκτρολογεί ένα username και ένα long URL και του επιστρέφεται από το σύστημα ένα short URL.
- Query: ο χρήστης πληκτρολογεί ένα short URL και (αν αυτό υπάρχει) το σύστημα επιστρέφει ένα long URL.
- Statistics: το σύστημα εκτυπώνει κάποια στατιστικά στοιχεία.

η απάντηση ήταν η αναμενόμενη:

Strings:

- Χρησιμοποιούνται για την αποθήκευση μετρητών, όπως το total_users και total_urls.
- Χρησιμοποιούνται επίσης για την αποθήκευση του συντόμευσης του URL με βάση το μακρύ URL (long_url:{long_url}).

Hashes:

- Χρησιμοποιούνται για την αποθήκευση πληροφοριών σχετικά με κάθε σύντομο URL, όπως το μακρύ URL, τον δημιουργό και τον αριθμό των κλικ (url:{short_code}).

Sets:

- Χρησιμοποιούνται για την αποθήκευση των συντομευμένων URL που έχει δημιουργήσει ένας χρήστης (user:{username}:urls).

Αφού μελετήθηκαν τα ζητούμενα, αποφασίστηκε τι θα χρησιμοποιηθεί από πλευράς Redis, επόμενη απόφαση ήταν της γλώσσας προγραμματισμού. Επιλέχτηκε η Python.

Μελετήθηκαν (κατά το δυνατόν) οι:

1. https://redis.io/docs/latest/commands/?group=string
2. https://redis.io/docs/latest/commands/?group=hash
3. https://redis.io/docs/latest/commands/?group=set

σκοπός ήταν για να γίνει η εύρεση των σωστών εντολών στην βάση αλλά και της σωστής σύνταξης για την γλώσσα προγραμματισμού Python.

Εκκινήθηκε το docker και η βάση Redis και εκτελέστηκε σε τερματικό

docker exec -it my-redis redis-cli

Σκοπός ήταν να δοθούν οι εντολές που σημειώθηκαν και θα χρειαστούν για την υλοποίηση.

Μετά την συγκέντρωση των εντολών, έγινε μελέτη του αρχείου java που προτάθηκε, έτσι ώστε να δομηθεί ανάλογα και το αρχείο python.

Μετά από πολλές προσπάθειες, κατέληξα σε μια βασική δομή. Χρησιμοποιήθηκε το Deepseek με ερωτήματα σχετικά με το τι σημαίνουν τα σφάλματα που εμφανιζόταν.

Κάποια από αυτά ήταν:

Σφάλμα:
clicks = int(self.r.hget(f"url:{short_code}", 'clicks'))
TypeError: int() argument must be a string, a bytes-like object or a number, not 'NoneType'

Αιτία:
Το πεδίο clicks δεν υπάρχει στο hash url:<short_code>, πιθανόν λόγω κακής εισαγωγής ή χειροκίνητης παρέμβασης στη βάση.

Σφάλμα:
username = user_key.split(':')[1]
IndexError: list index out of range

Αιτία:
Η scan_iter("user:*:count") μπορεί να επιστρέψει κλειδιά που δεν έχουν την αναμενόμενη μορφή user:<username>:count, ειδικά αν υπάρχει χειροκίνητη παρέμβαση στη βάση.

Τέλος, ζητήθηκε από το Deepseek να κατασκευάσει τον κώδικα της διεπαφής (με τα νούμερα), κάτι αντίστοιχο του κώδικα java.

Για το τέλος, ρωτήθηκε το Deepseek το εξής:

Έχω τον κώδικα του αρχείου. Μπορεί να γίνει κάποια βελτίωση έτσι ώστε να έχει λιγότερο κόστος λειτουργιών;

Απάντησε να διορθωθεί το:
stats['total_urls'] = len(self.r.keys("url:"))*

σε έναν μετρητή που ενημερώνεται κατά την εισαγωγή νέου URL:
self.r.incr("total_urls")

καθώς και πρότεινε την αλλαγή του *keys* σε *scan_iter*.

Σύμφωνα με το Deepseek, αυτό μειώνει το κόστος από O(n) (γραμμικό, ανάλογα με τα κλειδιά) σε O(1) (σταθερό).

Βασικές Βελτιστοποιήσεις:

- Μετρητές (total_users, total_urls)
    - Αντικαθιστούν τις keys με O(1) ερωτήματα.
    - Ενημερώνονται αυτόματα κατά την εισαγωγή νέων δεδομένων.
- Ασφαλής αναζήτηση με scan_iter
    - Χρησιμοποιείται για την ανάκτηση χρηστών, χωρίς να μπλοκάρει το Redis.
- Αποδοτική διαχείριση clicks
    - Τα clicks μετρώνται με hincrby (atomic και γρήγορο).
- Ευανάγνωστος κώδικας
    - Σχόλια και ονόματα μεταβλητών στα Ελληνικά για καλύτερη κατανόηση.

Τι Αλλάξαμε Συγκεκριμένα:

- Προστέθηκε init_counters() για αρχική ρύθμιση.
- Αντικαταστάθηκαν όλες οι keys με scan_iter ή μετρητές.
- Βελτιώθηκε η απόδοση των get_stats() (από O(n) σε O(1) για τα κύρια στατιστικά)

Οπότε κατέληξα στον τελικό κώδικα.

ΕΥΡΗΜΑΤΑ

Στον παρόντα κώδικα υλοποίησης υπηρεσίας συντόμευσης URL με χρήση Redis, η επιλογή των δομών δεδομένων έγινε με γνώμονα την αποδοτικότητα σε χρόνους προσπέλασης και ενημέρωσης, όπως τεκμηριώνεται και από τη Big O σημειογραφία του Redis. Συγκεκριμένα, για την αποθήκευση των πληροφοριών κάθε συντόμευσης (όπως το αρχικό URL, τον δημιουργό και τα clicks), επιλέχτηκε η δομή Hash (HSET, HGETALL, HINCRBY). Η χρήση Hash είναι ιδιαίτερα αποδοτική (O(1) για πρόσβαση και ενημέρωση), επιτρέποντας την αποδοτική αναζήτηση και καταμέτρηση των clicks για κάθε short code χωρίς την ανάγκη πλήρους αναζήτησης σε όλες τις εγγραφές.

Για την αντιστοίχιση μεταξύ του αρχικού URL και του short code χρησιμοποιείται η String (μέσω SET και GET), η οποία επίσης προσφέρει χρόνο πρόσβασης O(1), καθιστώντας την ιδανική για ελέγχους ύπαρξης (π.χ. αν έχει ήδη καταχωρηθεί το URL). Αντίστοιχα, τα counters των URLs και των χρηστών υλοποιούνται επίσης με Strings που αυξάνονται με INCR, εξασφαλίζοντας σταθερό χρόνο λειτουργίας (O(1)).

Η χρήση της δομής Set για την παρακολούθηση όλων των short codes που έχει δημιουργήσει κάθε χρήστη (SADD, SMEMBERS) είναι κρίσιμη για την αποδοτική συλλογή των URLs ανά χρήστη, καθώς οι περισσότερες σχετικές εντολές έχουν χρόνο O(1) ή O(N), όπου N είναι ο αριθμός στοιχείων στο set. Αυτό επιτρέπει την ταχεία ανάκτηση των short

codes και την εξαγωγή στατιστικών ανά χρήστη, διατηρώντας χαμηλό τον υπολογιστικό φόρτο.

Τέλος, η χρήση του SCAN_ITER για την εύρεση όλων των χρηστών κατά την εξαγωγή στατιστικών, αποτελεί επιλογή φιλική προς την απόδοση, αφού επιτρέπει την προοδευτική σάρωση των κλειδιών χωρίς να μπλοκάρει τον Redis server, σε αντίθεση με εντολές όπως KEYS που έχουν χρόνο O(N) και δεν είναι κατάλληλες για παραγωγικά περιβάλλοντα. Συνολικά, η επιλογή των δομών βασίζεται στην ανάγκη για σταθερό και χαμηλό κόστος λειτουργιών, ακόμα και σε κλίμακα, γεγονός που καθιστά το σύστημα επεκτάσιμο και αποδοτικό.

Αποτελέσματα

Η εφαρμογή λειτουργεί με επιτυχία και πληροί όλες τις απαιτήσεις:

Εισαγωγή URL: Ο χρήστης εισάγει ένα long URL και λαμβάνει ένα αντίστοιχο short URL. Αν το URL υπάρχει ήδη, επιστρέφεται το υπάρχον short URL.

Αναζήτηση URL: Ο χρήστης εισάγει ένα short URL και το σύστημα επιστρέφει το αντίστοιχο long URL, ενώ αυξάνει τον μετρητή clicks.

Στατιστικά: Το σύστημα παρουσιάζει στατιστικά όπως το πλήθος εισαγωγών ανά χρήστη και τον μέσο όρο των κλικ.

Οι επιλογές των δομών δεδομένων (Strings, Hashes, Sets) επικυρώθηκαν ως αποδοτικές, με χρόνους πρόσβασης O(1) για τις περισσότερες λειτουργίες. Η χρήση του scan_iter αντί του keys βελτίωσε την απόδοση, ιδιαίτερα σε μεγάλες βάσεις δεδομένων.

Συμπεράσματα

Η εργασία αυτή καταδεικνύει την αποτελεσματικότητα του Redis στην ανάπτυξη εφαρμογών που απαιτούν γρήγορη πρόσβαση και ενημέρωση δεδομένων. Οι επιλογές των δομών δεδομένων και οι βελτιστοποιήσεις που πραγματοποιήθηκαν (όπως η χρήση μετρητών και του scan_iter) έδειξαν σημαντική βελτίωση στην απόδοση.

Παρά τις δυσκολίες αντιμετώπισης σφαλμάτων, η εφαρμογή λειτουργεί αξιόπιστα και μπορεί να επεκταθεί για να υποστηρίξει περισσότερες λειτουργίες ή μεγαλύτερο φόρτο εργασίας.