Proxy Web Server

Autori: Livadariu Vlad-Constantin, Blanaru Razvan-Stefan, Bucur Mario-Valentin

ITBI 2025-2026

Cuprins

1 Context și scop

Scriptul este gândit să ruleze într-un mediu **CGI** și să primească un parametru `url=` din variabila de mediu `QUERY_STRING`. Scopurile principale:

- **Cache local** pentru răspunsuri: accelerează cererile repetate pentru același URL.
- **Control concurență**: previne descărcări paralele duplicate pentru același URL.
- **Limită spațiu**: menține cache-ul sub 512 MiB printr-o strategie tip **LRU** (Least Recently Used) bazată pe atime.
- **Prefetch asset-uri**: după ce HTML-ul este salvat, încearcă să descarce în fundal imagini/CSS/JS/fonturi referite în pagină.

2 Dependențe și presupuneri

2.1 Dependențe

Scriptul folosește:

- `bash`
- `wget` (cu `-save-headers`)
- `sha256sum`, `du`, `find`, `sort`, `cut`, `awk`, `sed`, `grep`, `stat`, `date`, `touch`
- `flock` (locking)
- `inotifywait` (așteptare evenimente filesystem)
- `aria2c` (download paralel de asset-uri)

2.2 Structura cache-ului

Rădăcina: `/mnt/proxy-cache`

Pentru fiecare URL se creează un director cu numele `SHA-256(url)` și conține:

- `main.html` — corpul răspunsului (body) fără headere HTTP
- `content-type.txt` — valoarea Content-Type extrasă din răspuns
- `.lock` — fișier de lock pentru sincronizarea descărcărilor
- `assets/` — director pentru asset-uri preîncărcate
- `.prefetched_assets_at` — marcaj cu timestamp pentru throttle
- `.prefetch_assets.lock` — lock specific pentru prefetch

3 Explicație detaliată: linie cu linie și argument cu argument

3.1 Header, siguranță și configurare

`#!/bin/bash`

- Specifică interpreter-ul: rulează scriptul cu `bash`.

`trap 'rm -f "$partial" 2>/dev/null' EXIT`

- `trap ... EXIT`: rulează comanda la ieșirea din script (normal sau cu eroare).
- `rm -f`: șterge fișiere, fără eroare dacă nu există.
- `"$partial"`: fișier temporar de descărcare (definit mai târziu).
- `2>/dev/null`: ascunde erorile (stderr).

`CACHE_ROOT="/mnt/proxy-cache"`

Setează rădăcina cache-ului.

`CACHE_SIZE=$((512*1024*1024))`

- 512 MiB în bytes.

3.2 Funcția `urldecode()`

```
urldecode() {
    local url_encoded="${1//+/ }"
    printf '%b' "${url_encoded//%/\\x}"
}
```

- `local url_encoded=...`: variabilă locală funcției.
- `"${1//+/ }"`: înlocuiește toate caracterele `+` cu spațiu.
- `printf '%b'`: interpretează secvențe escape (`\xNN`).
- `"${url_encoded//%/\\x}"`: transformă `%NN` în `\xNN` pentru a fi decodat de `printf`.

3.3 Funcția `current_cache_size()`

```
current_cache_size(){
    du -sb "$CACHE_ROOT" 2>/dev/null | awk '{print $1}'
}
```

- `du -sb`: dimensiune totală (`-s`) în bytes (`-b`).
- `2>/dev/null`: ignoră erorile.
- `awk '{print $1}'`: extrage prima coloană (numărul de bytes).

3.4 Funcția `make_room_cache(protected)` (evicție LRU)

```
make_room_cache() {
    local protected="$1"
    local current_size
    current_size=$(current_cache_size)

    if [ "$current_size" -le "$CACHE_SIZE" ]; then return 0; fi

    find "$CACHE_ROOT" -mindepth 1 -maxdepth 1 -type d -printf '%A@ %p\n' | \
    sort -n | cut -d' ' -f2- | while read -r dir_to_delete; do
        [ "$dir_to_delete" = "$protected" ] && continue
        rm -rf "$dir_to_delete"
        current_size=$(current_cache_size)
        [ "$current_size" -le "$CACHE_SIZE" ] && break
    done
}
```

Semnificația fiecărui element

- `protected="$1"`: directorul „protejat” (cel curent), care nu trebuie șters.
- `current_size=$(current_cache_size)`: calculează dimensiunea actuală.
- `[ "$current_size" -le "$CACHE_SIZE" ]`: dacă e sub limită, ieșire.

Pipeline pentru alegerea „celor mai vechi” directoare

- `find "$CACHE_ROOT"`: enumeră elemente.
- `-mindepth 1`: exclude rădăcina.

- `-maxdepth 1`: doar copii direcți.
- `-type d`: doar directoare.
- `-printf '%A@ %p\n'`: tipărește **timp acces** (epoch) + cale.
- `sort -n`: sort numeric ascendent (cele mai vechi accesări primele).
- `cut -d' ' -f2-`: elimină timestamp-ul, păstrează calea.
- `while read -r`: parcurge fiecare director, fără interpretare de backslash.

Ștergere până sub limită

- `[ "$dir" = "$protected" ] && continue`: sari peste directorul protejat.
- `rm -rf`: șterge recursiv.
- recalculare dimensiune + `break` când e sub limită.

3.5 Funcția `prefetch_assets(url, dir, html_file)`

```
prefetch_assets() {
  local url="$1"
  local dir="$2"
  local html_file="$3"

  local mark="$dir/.prefetched_assets_at"

  if [ -e "$mark" ] && [ $(( $(date +%s) - $(stat -c %Y "$mark" 2>/dev/null ||
      echo 0) )) -lt 1800 ]; then
    return 0
  fi

  local total; total=$(current_cache_size)
  [ "$total" -gt $((CACHE_SIZE - 50*1024*1024)) ] && return 0

  (
    exec 8>"$dir/.prefetch_assets.lock"
    flock -n 8 || exit 0

    mkdir -p "$dir/assets" || exit 0

    local domain; domain=$(echo "$url" | awk -F/ '{print $1"//"$3}')
```

```
    local base; base="${url%/*}/"

    grep -oE '(src|href)="[^"]+\.(jpg|jpeg|png|gif|css|js|svg|woff|woff2|ttf|ico
        )"' "$html_file" | \
    cut -d'"' -f2 | \
    sort -u | \
    while read -r asset_path; do
        if [[ "$asset_path" == //* ]]; then
            echo "https:$asset_path"
        elif [[ "$asset_path" == /* ]]; then
             echo "${domain}${asset_path}"
        elif [[ "$asset_path" == http* ]]; then
             echo "$asset_path"
        else
             echo "${base}${asset_path}"
        fi
    done | \
    aria2c -i - \
      --dir="$dir/assets" \
      --j=16 \
      --quiet=true \
      --connect-timeout=5 \
      --timeout=10 \
      --auto-file-renaming=false \
      --allow-overwrite=false \
      --user-agent="Mozilla/5.0 (Compatible; AriaCache/1.0)" >/dev/null 2>&1

    date +%s > "$mark"
  ) &
}
```

Throttle (30 minute)

- `mark`: fișier marker cu ultima preîncărcare.
- `date +%s`: timpul curent epoch.
- `stat -c %Y`: timpul de modificare al marker-ului.
- dacă diferența < 1800 secunde, iese fără a face nimic.

Protecție spațiu

- dacă totalul cache-ului este în ultimele 50 MiB până la limită, nu prefetch.

Rulare în fundal + locking

- blocul `( ... ) &` rulează într-un subshell în fundal.
- `exec 8>file` deschide fișierul de lock pe FD 8.
- `flock -n 8`: încearcă lock non-blocking; dacă e ocupat, iese.

Normalizare URL asset-uri

- `//...` (scheme-relative) → prefix `https:`
- `/...` (root-relative) → prefix `domain`
- `http...` → păstrează
- altfel (relative) → prefix `base`

Descărcare paralelă cu `aria2c`

- `-i -`: citește lista de URL-uri din stdin.
- `-dir=`: director de salvare.
- `-j=16`: 16 download-uri în paralel.
- timeouts, fără redenumire automată, fără overwrite.

3.6 Execuția principală

Parsare query string

```
raw_url=$(echo "$QUERY_STRING" | grep -oE 'url=[^&]+' | cut -d= -f2-)
url=$(urldecode "$raw_url")
```

- `QUERY_STRING`: șirul brut de parametri CGI.
- `grep -oE 'url=[&]+'`: extrage doar porțiunea `url=...` până la `&`.
- `cut -d= -f2-`: elimină partea `url=` și păstrează valoarea.
- `urldecode`: decodează percent-encoding.

Validare

```
if [ -z "$url" ]; then
  ...
fi

if [[ ! "$url" =~ ^https?:// ]]; then
  ...
fi
```

- `[ -z "$url" ]`: url gol.
- `[[ ! "$url" = ĥttps?:// ]]`: acceptă doar HTTP/HTTPS.
- răspunsul CGI include `Content-Type`, apoi o linie goală, apoi body HTML de eroare.

Hash și căi de fișiere

```
hashed_url=$(printf "%s" "$url" | sha256sum | awk '{print $1}')
dir="$CACHE_ROOT/$hashed_url"

mkdir -p "$dir"
partial="$dir/.part.$$.tmp"
final="$dir/main.html"
meta_type="$dir/content-type.txt"
lock="$dir/.lock"
```

- `printf "%s"`: tipărește fără newline.
- `sha256sum`: hash stabil pentru nume de director.
- `awk '{print $1}'`: ia doar hash-ul.
- `$$`: PID-ul procesului curent (evită coliziuni între request-uri).

Cache HIT

```
if [ -s "$final" ]; then
  touch -a "$dir" "$final" 2>/dev/null
  c_type="text/html"
  [ -f "$meta_type" ] && c_type=$(cat "$meta_type")

  echo "Content-Type: $c_type"
```

```
  echo "X-Cache: HIT"
  echo ""
  cat "$final"
  exit 0
fi
```

- `[ -s file ]`: există și are dimensiune > 0.
- `touch -a`: actualizează atime pentru LRU.
- citește tipul din `content-type.txt` dacă există.
- trimite header-ul CGI plus `X-Cache: HIT` i body.

Cache MISS + locking

```
exec 3>"$lock"

if flock -n 3; then
  make_room_cache "$dir"

  if wget --quiet --tries=3 --timeout=15 --save-headers \
          --output-document="$partial" -- "$url"; then

    grep -i "^Content-Type:" "$partial" | head -n 1 | cut -d: -f2- | tr -d '\r'
        | xargs > "$meta_type"
    sed '1,/^\r$/d' "$partial" > "$final"

    c_type=$(cat "$meta_type")
    [ -z "$c_type" ] && c_type="text/html"

    echo "Content-Type: $c_type"
    echo "X-Cache: MISS"
    echo ""
    cat "$final"

    rm -f "$partial"
    prefetch_assets "$url" "$dir" "$final"
  else
    ...
  fi
else
```

```
  ...
fi
```

Semnificație `exec 3> "$lock"` Deschide fișierul `.lock` pe descriptorul de fișier 3 (creându-l dacă nu există).

Semnificație `flock -n 3`

- încearcă să obțină lock exclusiv pe FD 3.
- `-n` = non-blocking: dacă e ocupat, nu așteaptă, ci intră pe ramura `else`.

Argumente `wget`

- `-quiet`: fără output.
- `-tries=3`: reîncercări.
- `-timeout=15`: timeout de rețea (sec).
- `-save-headers`: include headerele HTTP în fișierul rezultat.
- `-output-document="$partial"`: scrie în fișierul temporar.
- `- "$url"`: `-` oprește parsarea opțiunilor; protejează împotriva URL-urilor care încep cu `-`.

Extracție Content-Type

- `grep -i "^Content-Type:"`: caută header-ul.
- `head -n 1`: primul rezultat.
- `cut -d: -f2-`: elimină numele header-ului.
- `tr -d '\r'`: elimină CR din CRLF.
- `xargs`: face trim la whitespace.
- scrie în `content-type.txt`.

Separare body de headere

- `sed '1,/^\r$/d'` : *șterge liniile de la 1 până la linia separatoare.*

Ramura de concurență (HIT-WAIT)

```
inotifywait -q -e close_write --timeout 15 "$dir" >/dev/null 2>&1

if [ -s "$final" ]; then
  touch -a "$dir" "$final" 2>/dev/null
  c_type="text/html"
  [ -f "$meta_type" ] && c_type=$(cat "$meta_type")

  echo "Content-Type: $c_type"
  echo "X-Cache: HIT-WAIT"
  echo ""
  cat "$final"
else
  ...
fi
```

- dacă un alt proces descarcă deja, acest proces așteaptă un eveniment de tip `close_write` (scriere finalizată).
- timeout 15 secunde.
- dacă `main.html` apare și are mărime > 0, servește cu `X-Cache: HIT-WAIT`.

4 Concluzie

Scriptul oferă un proxy cu cache eficient pentru scenarii ușoare, combinând:

- indexare prin hash,
- control concurență cu `flock` + `inotifywait`,
- politică de evicție tip LRU bazată pe `atime`,
- prefetch de asset-uri în paralel cu `aria2c`.