Performance Study for New Web Caching Strategies Combining LRU with Score Based Object Selection

G. Hasslinger¹, K. Ntougias², F. Hasslinger³, O. Hohlfeld⁴

¹Deutsche Telekom, ²Athens Information Tech., ³TU Darmstadt, ⁴RWTH Aachen

- Caching strategies: Score-Gated (SG-)LRU, LFU & alternatives
- Simulation of caching for Zipf distributed request pattern
- Hit rate comparison for LRU and score-based strategies
- Evaluations based on Wikipedia top-1000 request statistics
- Conclusions & Outlook
Caching: Goals & Strategies

➢ Main Goals of Web Caching:
  - Reduce Transport Paths, Delay, Traffic Load
  - Reduce Traffic Peaks: Prefetching, Overnight Cache Updates
  - Increase Throughput (shorten RTTs)
  - Distributed, Replicated Data on Net (Info. Centric Networks)

➢ Caching Strategies: Which data to put into / delete from the cache?
  - Least Recently Used (LRU): Simple standard method
  - Least Frequently Used (LFU): Score-based, non-dynamic
  - Score-Gated (SG-)LRU: Flexible, includes LRU & LFU
    SG-LRU: Puts requested object into the cache only if it has higher score than the object at the bottom of the cache
    (Sorted cache list due to scores has too high update effort)
Scope

Main focus: Web caches for a large user population
⇒ Strategies with score-based selection of cached objects
⇒ Comparison of caching strategies to improve LRU
⇒ Optimization for maximum hit rate as KPI
⇒ Zipf distributed access pattern
⇒ Only for cacheable data following RFC 7234 guidelines

Not in scope:
Caches for computing and database storage systems
⇒ Partly periodic access pattern e.g. due to loops in programs
Caches in local network, for single devices, users:
⇒ Papers show efficiency of nano data centers
⇒ Papers also show efficiency of client-side caching e.g. in browsers
Objects of different size; general cost functions:
⇒ can be partly covered by extended/combined score functions
Motivation: Pros. & Cons. of the State of the Art

→ Improves LRU hit rate by 10 - 20% for many traces of web requests

Our main contributions:
10-20% goal obtained by a simple LRU variant; general confirmation for Zipf IRM requests

ARC policy:

```
ARC(c)  INITIALIZE    T₁ = B₁ = T₂ = B₂ = 0, p = 0.  x - requested page.

Case I. x ∈ T₁ ∪ T₂ (a hit in ARC(c) and DEL(2c)): Move x to the top of T₂.

Case II. x ∈ B₁ (a miss in ARC(c), a hit in DEL(2c)):
   Adapt p = min[c, p + max{|B₂|/|B₁|, 1}] \rightarrow REPLACE(p). Move x to the top of T₂ and place it in the cache.

Case III. x ∈ B₂ (a miss in ARC(c), a hit in DEL(2c)):
   Adapt p = max[0, p - max{|B₁|/|B₂|, 1}] \rightarrow REPLACE(p). Move x to the top of T₁ and place it in the cache.

Case IV. x ∈ L₁ ∪ L₂ (a miss in DEL(2c) and ARC(c)):
   case (i) |L₁| = c:
      If |T₁| < c then delete the LRU page of B₁ and REPLACE(p).
      else delete LRU page of T₁ and remove it from the cache.
   case (ii) |L₁| < c and |L₁| + |L₂| ≥ c:
      if |L₁| + |L₂| = 2c then delete the LRU page of B₂.
      REPLACE(p).
      Put x at the top of T₁ and place it in the cache.

Subroutine REPLACE(p)
if (|T₁| ≥ 1) and (x ∈ B₂ and |T₁| = p) or (|T₁| > p) then move the LRU page of T₁ to the top of B₁ and remove it from the cache.
else move the LRU page in T₂ to the top of B₂ and remove it from the cache.
```

Constant update effort, but more complex than LRU
Score-Gated LRU Caching Strategy

3 cases for (SG-)LRU cache updates after each request

Put $O_w$ on top in case of a cache hit requesting $O_w$

or load a new object $O_n$ in case of a cache miss for LRU and SG-LRU*

*if Score($O_n$) > Score($O_z$)

or put $O_z$ on top in case of a cache miss only for SG-LRU

if Score($O_n$) ≤ Score($O_z$)

SG-LRU: Basic LRU + Score-based decision about cache uploads
- Includes score per object: more update effort for more flexibility
- Score Fct.: e.g. Sliding Window: Count over last $W$ requests
- Score Fct.: e.g. Geometric Fading: Weighted request count

SG-LRU avoids many LRU uploads ⇒ Update effort undercuts LRU!
ZIPF REQUEST PATTERN MAKES WEB CACHES EFFICIENT

Popularity of objects in a large set requested by a large community is given by Zipf laws: 80:20 (90:10) rules:
80% of requests to 20% of objects
Example: $Z(k) = 0.065 \cdot k^{-0.8}$ for $k = 1, \ldots, 1000$ objects

$Z(1) = 6.5\%$
$Z(2) = 3.7\%$
$Z(3) = 2.7\%$
$Z(4) = 2.1\%$
$Z(5) = 1.8\%$

$\Rightarrow \sum_{1}^{10} Z(k) \approx 23\%$

$\Rightarrow$ A cache storing the top ten objects:
1% of objects achieves 23% hit rate
$\Rightarrow$ Small caches can be efficient
$\Rightarrow$ Often confirmed for YouTube, IPTV, book selling, word frequency, etc.

$Z(10) = 1.0\%$
$Z(100) = 0.16\%$
$Z(1000) = 0.026\%$

Breslau et al. (1999) $0.64 \leq \beta \leq 0.83$
Che et al. (2002) $0.64 \leq \beta \leq 0.75$
Fricker et al. (2012) $\beta \approx 0.7; \beta \approx 0.82$
Cha et al. (2008) $0.56 \leq \beta \leq 0.88$
Hefeeda, Saleh (2008) $0.6 \leq \beta \leq 0.78$
Simulation requires efficient random Zipf rank generator

Mathematica recommends an accept/reject method only for $\beta > 1$

We use and confirm

$$r^*(R) = N \left[ 1 - (1 - R)(1 - (\frac{1}{2})^{1-\beta}) / (1 - Z_{CDF}(\frac{N}{2})) \right]^{1-\beta}$$

as an almost perfect inversion method

Rank deviations of $r^*(R)$ are checked via

$$\Delta(r) = r^*(Z_{CDF}(r)) - r$$

We checked

$$\forall r: 1 \leq r \leq N: |\Delta(r)| \leq 1$$

for $N = 3, 4, \ldots, 10^5, 10^5 + 100, 10^5 + 200, \ldots, 10^6$ and for $\beta = 0.1, 0.2, \ldots, 3$

We lack an analytical confirmation or proof
Simulation requires control of the variance in the results: 2nd Order Statistics ≈ Variance in Different Time Scales

2nd order statistics is taken during steady state phase of the simulation over $10^{R+1}$ requests where the first quarter of requests is excluded ($R$ sufficiently large)

We compute hit rate estimates in time scales $K = 10, 10^2, \ldots, 10^R$ i.e. mean values over $10^k$ requests and their variance during sim.

$$\pi_{(K)}(j) = \frac{1}{K} \sum_{k=(j-1)K+1}^{jK} \pi(k);$$

$$\sigma(\pi_{(K)}) = \sqrt{E(\pi_{(K)}^2)(j) - \mu^2(\pi)}; \quad \mu(\pi) = E(\pi_{(K)}(j)) = E(\pi(k)).$$

$$\sigma(\pi_{(K)}) = \sqrt{\sum_{j=1}^{10^{R+1}/K} \pi_{(K)}^2(j) - \mu^2(\pi) / \left( \frac{10^{R+1}}{K} - 1 \right)}; \quad \mu(\pi) = \frac{1}{10^{R+1}} \sum_{j=1}^{10^{R+1}} \pi(j).$$
2nd Order Statistics: Variance in Different Time Scales for Hit Rate Estimation $h_C$: Hit Count versus $\pi_C$: Cache Prob.

(SG-)LRU cache simulations ($N = 10^6$; Zipf requests $\beta = 0.8$) 4 different cache sizes to obtain 10%, 25%, 50% & 75% hit rate
Simulative hit rate comparison: LRU ↔ SG-LRU for independent (IRM) Zipf distributed requests

Results of 7·8·99 simulation runs covering the relevant param. range $N \times \beta \times M \in \{10^2, 10^3, ..., 10^7\} \times \{0.4, 0.6, ..., 1.1\} \times \{M_{1\%}, M_{2\%}, ..., M_{99\%}\}$ for 3 parameters

- $N$: Fixed number of objects
- $\beta$: Shape parameter of the Zipf distribution
- $M$: Cache size
- $M_x\%$: Minimum cache size to achieve an LRU hit rate of $x\%$

Each simulation runs at least
- $10^8$ requests in the evaluation phase
- until the hit rate estimation of the std. deviation is below $5 \cdot 10^{-5}$
Main Result: Hit Rate Gain $h_{LFU} - h_{LRU}$ for IRM Zipf requests

Minimum, mean and maximum gain for 6·8 cases for cache sizes $M_x$ for $x\%$ hit rate & $\beta \times N \in \{0.4, 0.5, \ldots, 1.1\} \times \{10^2, 10^3 \ldots, 10^7\}$
Simulation based on requests to daily top-1000 Wikipedia pages

Number of requests to top-1000 Wikipedia pages per day is available since Aug. 2015 and evaluated until Jan. 2016. Zipf distributions with $0.5 < \beta < 0.75$ make a good or perfect fit.

Daily top-1000 Wikipedia request statistics

<table>
<thead>
<tr>
<th>Top $k$: $k =$</th>
<th>1</th>
<th>100</th>
<th>1000</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>$1.85 \cdot 10^5$</td>
<td>$4.74 \cdot 10^6$</td>
<td>$1.63 \cdot 10^7$</td>
<td>$2.37 \cdot 10^8$</td>
</tr>
<tr>
<td>Mean</td>
<td>$7.80 \cdot 10^5$</td>
<td>$7.82 \cdot 10^6$</td>
<td>$2.11 \cdot 10^7$</td>
<td>$2.85 \cdot 10^8$</td>
</tr>
<tr>
<td>Maximum</td>
<td>$6.95 \cdot 10^6$</td>
<td>$2.50 \cdot 10^7$</td>
<td>$4.55 \cdot 10^7$</td>
<td>$3.43 \cdot 10^8$</td>
</tr>
</tbody>
</table>

Dynamics in daily top-1000 Wikipedia pages

<table>
<thead>
<tr>
<th>Top-$k$ pages: $k =$</th>
<th>25</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y's Top-$k \rightarrow$ Top-$k$</td>
<td>54.7%</td>
<td>58.8%</td>
<td>62.4%</td>
<td>66.7%</td>
<td>71.9%</td>
<td>76.1%</td>
</tr>
<tr>
<td>$1 - (Y\text{'s Top-1000} \rightarrow$ Top-$k)$</td>
<td>22.6%</td>
<td>21.8%</td>
<td>21.2%</td>
<td>20.8%</td>
<td>20.9%</td>
<td>23.9%</td>
</tr>
</tbody>
</table>
Simulation of $R_d/N_C$ request per day, equally distributed over $N_C$ caches

Independent requests due to the daily changing top-1000 statistics:

Daily dynamics becomes relevant for $N_C \geq 200$,
i.e. for < 50,000 requests per cache per day
Conclusions

- Gain of LFU and SG-LRU over pure LRU for indep. Zipf requests in the most relevant range of 10% - 50% LRU hit rates:
  - LFU improves LRU hit rates by at least 9.6%
  - in average by 13.7% (2306 simulated cases in the 10%-50% range)
  - by up to 19.3%
  
  Relative differences are largest for small LRU hit rates:
  - LFU yields > 2-fold hit rates when LRU is below 10%

- The Che approx. for LRU hit rates is highly precise for Zipf-IRM

- Evaluation for daily changing top-1000 Wikipedia pages:
  - SG-LRU hit rate is close to LFU limit for > 50 000 requests/day
  - cache performance for smaller population depends on dynamics in object popularity, local preferences etc.

- Future work: More traces & predictive score functions