Asymptotically Exact TTL-Approximations of the Cache Replacement Algorithms LRU(m) and h-LRU

Nicolas Gast\textsuperscript{1}, Benny Van Houdt\textsuperscript{2}

ITC 2016

September 13-15, Würzburg, Germany

\textsuperscript{1}Inria
\textsuperscript{2}University of Antwerp
Caches are everywhere

Examples:
- Processor
- Database
- CDN
Caching policies

- Popularity-oblivious policies
  - Cache-replacement policies\(^3\) (LRU, RANDOM),
  - TTL-caches\(^4\).

- Popularity-aware policies / learning
  - LFU and variants\(^5\)
  - Optimal policies for network of caches\(^6\)

\(^3\) started with [King 1971, Gelenbe 1973]
\(^4\) e.g., Fofack e al 2013, Berger et al. 2014
\(^5\) Optimizing TTL Caches under Heavy-Tailed Demands (Ferragut et al. 2016)
\(^6\) Adaptive Caching Networks with Optimality Guarantees (Ioannidis and Yeh, 2016)
Caching policies

- **Popularity-oblivious policies**
  - Cache-replacement policies\(^3\) (LRU, RANDOM),
  - TTL-caches\(^4\).

- **Popularity-aware policies / learning**
  - LFU and variants\(^5\)
  - Optimal policies for network of caches\(^6\)

---

\(^3\) started with [King 1971, Gelenbe 1973]
\(^4\) e.g., Fofack e al 2013, Berger et al. 2014
\(^5\) Optimizing TTL Caches under Heavy-Tailed Demands (Ferragut et al. 2016)
\(^6\) Adaptive Caching Networks with Optimality Guarantees (Ioannidis and Yeh, 2016)
Contributions (and Outline)

1. Two cache replacement policies
2. Performance analysis via TTL approximation
3. Asymptotic exactness of the approximation
4. Comparison between LRU, LRU($\bar{m}$) and $h$-LRU
5. Conclusion
Outline

1. Two cache replacement policies
2. Performance analysis via TTL approximation
3. Asymptotic exactness of the approximation
4. Comparison between LRU, LRU($\bar{m}$) and $h$-LRU
5. Conclusion
The two policies generalize the LRU policy

hit : do nothing
miss : evict the LRU (least-recently used) item.

Example with this stream of requests:

(note: similar to RANDOM, FIFO)

(Assumption: Object are assumed to have the same size.)
The two policies generalize the LRU policy

hit : do nothing
miss : evict the LRU (least-recently used) item.

Example with this stream of requests:

(note: similar to RANDOM, FIFO)

(Assumption: Object are assumed to have the same size.)
The two policies generalize the LRU policy

hit : do nothing

miss : evict the LRU (least-recently used) item.

Example with this stream of requests:

(note: similar to RANDOM, FIFO)

(Assumption: Object are assumed to have the same size.)
The two policies generalize the LRU policy

**LRU:**
- **hit**: do nothing
- **miss**: evict the LRU (least-recently used) item.

Example with this stream of requests:

- Blue  
- Red  
- Blue  

↑

(note: similar to RANDOM, FIFO)

(Assumption: Object are assumed to have the same size.)
The two policies generalize the LRU policy

hit : do nothing
miss : evict the LRU (least-recently used) item.

Example with this stream of requests:

(note: similar to RANDOM, FIFO)

(Assumption: Object are assumed to have the same size.)
The two policies generalize the LRU policy

hit : do nothing
miss : evict the LRU (least-recently used) item.

Example with this stream of requests:

(note: similar to RANDOM, FIFO)

(Assumption: Object are assumed to have the same size.)
The two policies generalize the LRU policy

hit : do nothing
miss : evict the LRU (least-recently used) item.

Example with this stream of requests:

(note: similar to RANDOM, FIFO)

(Assumption: Object are assumed to have the same size.)
The two policies generalize the LRU policy

- **hit**: do nothing
- **miss**: evict the LRU (least-recently used) item.

Example with this stream of requests:

```
[blue] [red] [blue] [red] [green] [blue] [blue]
```

(note: similar to RANDOM, FIFO)

(Assumption: Object are assumed to have the same size.)
The LRU($\vec{m}$) and $h$-LRU policies

- **LRU($\vec{m}$)$^7$:** exchange the requested item with the LRU of next list

- **$h$-LRU$^8$:** copy the requested item in the next list (and evict the LRU)

---

$^7$Variant of RAND($\vec{m}$) of [G, Van Houdt 2015]

$^8$Introduced as $k$-LRU in [Martina et al. 2014]
The LRU($\vec{m}$) and $h$-LRU policies

- **LRU($\vec{m}$)**\(^7\): exchange the requested item with the LRU of next list

- **$h$-LRU**\(^8\): copy the requested item in the next list (and evict the LRU)

\(^7\)Variant of RAND($\vec{m}$) of [G, Van Houdt 2015]
\(^8\)Introduced as $k$-LRU in [Martina et al. 2014]
The LRU($\tilde{m}$) and $h$-LRU policies

- **LRU($\tilde{m}$)$^7$:** exchange the requested item with the LRU of next list

- **$h$-LRU$^8$:** copy the requested item in the next list (and evict the LRU)

---

$^7$ Variant of RAND($\tilde{m}$) of [G, Van Houdt 2015]

$^8$ Introduced as $k$-LRU in [Martina et al. 2014]
The LRU($m$) and $h$-LRU policies

- **LRU($m$)**\(^7\): exchange the requested item with the LRU of next list

- **$h$-LRU**\(^8\): copy the requested item in the next list (and evict the LRU)

\(^7\)Variant of RAND($m$) of [G, Van Houdt 2015]

\(^8\)Introduced as $k$-LRU in [Martina et al. 2014]
The LRU($\tilde{m}$) and $h$-LRU policies

- **LRU($\tilde{m}$)**: exchange the requested item with the LRU of next list

- **$h$-LRU**: copy the requested item in the next list (and evict the LRU)

---

$^7$Variant of RAND($\tilde{m}$) of [G, Van Houdt 2015]

$^8$Introduced as $k$-LRU in [Martina et al. 2014]
The LRU($\tilde{m}$) and $h$-LRU policies

- **LRU($\tilde{m}$)**: exchange the requested item with the LRU of next list

- **$h$-LRU**: copy the requested item in the next list (and evict the LRU)

---

Variant of RAND($\tilde{m}$) of [G, Van Houdt 2015]

Introduced as $k$-LRU in [Martina et al. 2014]
The LRU($\vec{m}$) and $h$-LRU policies

- **LRU($\vec{m}$)**$^7$: exchange the requested item with the LRU of next list

- **$h$-LRU**$^8$: copy the requested item in the next list (and evict the LRU)

---

$^7$Variant of RAND($\vec{m}$) of [G, Van Houdt 2015]

$^8$Introduced as $k$-LRU in [Martina et al. 2014]
The LRU(\(m\)) and \(h\)-LRU policies

- **LRU(\(m\))\(^7\):** exchange the requested item with the LRU of next list

- **\(h\)-LRU\(^8\):** copy the requested item in the next list (and evict the LRU)

\(^7\)Variant of RAND(\(m\)) of [G, Van Houdt 2015]

\(^8\)Introduced as \(k\)-LRU in [Martina et al. 2014]
Outline

1. Two cache replacement policies
2. Performance analysis via TTL approximation
3. Asymptotic exactness of the approximation
4. Comparison between LRU, LRU($\bar{m}$) and $h$-LRU
5. Conclusion
In this talk: Performance analysis and comparison

Qualitatively:

- It takes time to adapt

Quantitatively:

- Related work: Variants
- 9 or less accurate approximation
- We present TTL approximations for MAP arrival (in the talk: IRM).

\[\text{RAND}(\bar{m})\text{ in [G, Van Houdt, 2015]}\) for which product form solution exist.

\[\text{heuristic for } h\text{-LRU [Martina et al. 2014]}\]
In this talk: Performance analysis and comparison

Qualitatively:
- less popular
- popular items
- It takes time to adapt

Quantitatively:
- Related work: Variants\(^9\) or less accurate approximation\(^{10}\)
- We present TTL approximations for MAP arrival (in the talk: IRM).

---

\(^9\) (RAND(\(\bar{m}\)) in [G, Van Houdt, 2015]) for which product form solution exist. 
\(^{10}\) heuristic for \(h\)-LRU [Martina et al. 2014]
Pure LRU: the Che-approximation

If the request of object $k$ is a Poisson process of intensity $\lambda_k$:

Object $k$ is in cache with probability $\pi_k(T) = 1 - e^{-\lambda_k T(TTL)}$.

$T$ satisfies $\sum_k \pi_k(T) = \text{cache size}$.

Diagram:

- Cache
  - reset timer
  - start new timer
  - eviction after $T$
Pure LRU: the Che-approximation

If the request of object $k$ is a Poisson process of intensity $\lambda_k$:

- Object $k$ is in cache with probability $\pi_k(T) = 1 - e^{-\lambda_k T}$
If the request of object $k$ is a Poisson process of intensity $\lambda_k$:

- Object $k$ is in cache with probability $\pi_k(T) = 1 - e^{-\lambda_k T}$

$T$ satisfies $\sum_k \pi_k(T) = \text{cache size}$.

(Fixed point)
The TTL-approximation for LRU(m)

If the request of object \( k \) is a Poisson process of intensity \( \lambda_k \):

Object \( k \) is in cache \( \ell \) with probability \( \pi_k, i(\ell) \propto \ell \prod_{i=1}^{\ell} (e^{\lambda_k T_i} - 1) \).

\( T_1 \ldots T_h \) satisfy

\[ \sum_k \pi_k, i(\ell) (T_1 \ldots T_h) = \text{size of list } i. \]
The TTL-approximation for LRU(m)

If the request of object $k$ is a Poisson process of intensity $\lambda_k$:

- Object $k$ is in cache $\ell$ with probability $\pi_{k,i}(T_1 \ldots T_h) \propto \prod_{i=1}^{\ell} (e^{\lambda_k T_i} - 1)$

$T_1 \ldots T_h$ satisfy $\sum_k \pi_{k,i}(T_1 \ldots T_h) = \text{size of list } i$. 
The TTL-approximation for $h$-LRU

First idea: track the lists in which an object are. [Martina et al. 14]
- Problem: number of states $= 2^h$. 

Diagram:

```
Cache 1 ----> copy ----> Cache 2 ----> Cache 3
```
The TTL-approximation for $h$-LRU

Solution: change model (track the greatest ID of the list in which the item appears by assuming that $T_1 \leq T_2 \leq \ldots T_h$)

The TTL model can be solved exactly (see paper).

Once $T_1 \ldots T_k$ have been computed, $T_{k+1}$ satisfies a fixed point equation.
Outline

1. Two cache replacement policies
2. Performance analysis via TTL approximation
3. Asymptotic exactness of the approximation
4. Comparison between LRU, LRU($\bar{m}$) and $h$-LRU
5. Conclusion
Is the approximation accurate?

Example (10-LRU, with a cache size $n/10$ and a Zipf popularity)

<table>
<thead>
<tr>
<th></th>
<th>Simulation</th>
<th>(Our approximation)</th>
<th>[Martina et al. 14])</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 1000$</td>
<td>0.51506</td>
<td>0.51552 (+0.088%)</td>
<td>0.50796 (-1.380%)</td>
</tr>
<tr>
<td>$n = 10000$</td>
<td>0.56124</td>
<td>0.56130 (+0.012%)</td>
<td>0.55447 (-1.206%)</td>
</tr>
</tbody>
</table>

Numerically, TTL approximation have proven to be very accurate [Dan and Towsley 1990, Martina et al. 14, Che, 2002]. Theoretical guarantees exist for LRU [Fricker et al. 12]. We prove that our approximation is asymptotically exact.
Is the approximation accurate?

Example (10-LRU, with a cache size $n/10$ and a Zipf popularity)

<table>
<thead>
<tr>
<th>$n$</th>
<th>Simulation</th>
<th>(Our approximation)</th>
<th>[Martina et al. 14])</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 1000$</td>
<td>0.51506</td>
<td>0.51552 (+0.088%)</td>
<td>0.50796 (-1.380%)</td>
</tr>
<tr>
<td>$n = 10000$</td>
<td>0.56124</td>
<td>0.56130 (+0.012%)</td>
<td>0.55447 (-1.206%)</td>
</tr>
</tbody>
</table>

- Numerically, TTL approximation have proven to be very accurate [Dan and Towsley 1990, Martina at al. 14, Che, 2002]
- Theoretical guarantees exist for LRU [Fricker et al. 12]

We prove that our approximation is asymptotically exact.
Asymptotic exactness of the approximation

We develop an ODE approximation

We show that it is accurate

This ODE has the same fixed point as the TTL approximation
Asymptotic exactness of the approximation

![Graphs showing probability in cache over number of requests for different scenarios: simulation, 1 list (200), 4 lists (50/50/50/50), ODE approximation (1 list), ODE approximation (4 lists).]

**Figure:** Popularities of objects change every 2000 steps.

- We develop an ODE approximation
- We show that it is accurate
- This ODE has the same fixed point as the TTL approximation
Theorem 1. Let $H_\ell(t)$ be the sum of the popularity of the items of list $\ell$ and $h_\ell(t)$ be the corresponding ODE approximation (Equation (18) for $h$-LRU and Equation (22) for LRU($m$)). Then: for any time $T$, there exists a constant $C$ such that

$$\mathbb{E} \left[ \sup_{t \leq T/\sqrt{\max_k p_k}} |H_\ell(t) - h_\ell(t)| \right] \leq C \sqrt{\max_k p_k},$$

where $C$ does not depend on the probabilities $p_1 \ldots p_n$, the cache size $m$ or the number of items $n$.

Idea of the proof.

- We study the empirical distribution of the request dates.
- We use stochastic approximation to prove the convergence to an infinite dimensional deterministic ODE.
Outline

1. Two cache replacement policies
2. Performance analysis via TTL approximation
3. Asymptotic exactness of the approximation
4. Comparison between LRU, LRU($\bar{m}$) and $h$-LRU
5. Conclusion
In general, adding more lists:

- Improves the steady-state performance\(^a\),
- Decreases the response time.

\(^a\)This is not true in full generality, even for IRM. The same counter-example as in [G., Van Houdt 2015] works.
Quantitative remark 1: On synthetic traces: \( \text{LRU}(m, m) \) and 2-LRU perform similarly

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>Hit Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zipf ( \alpha = 0.8 ), ( n = 1000 ), correlation in IRTs</td>
<td></td>
</tr>
</tbody>
</table>

Hypothesis 2:
- LRU
- 2-LRU
- LRM\((m, m)\)

Hypothesis 10:
- LRU
- 2-LRU
- LRU\((m, m)\)

Nicolas Gast – 19 / 24
Quantitative remark 1: On synthetic traces: $\text{LRU}(m, m)$ and 2-LRU perform similarly.

$\text{LRU}(m, m)$: exchange

$\text{2-LRU}$: copy

Zipf $\alpha = 0.8$, $n = 1000$, correlation in IRTs

Hit Probability

Cache Size

Hypo2 LRU
Hypo2 2-LRU
Hypo2 LRM($m, m$)
Hypo10 LRU
Hypo10 2-LRU
Hypo10 LRU($m, m$)
Quantitative remark 1: LRU is insensitive to correlations between requests time

![Graph showing hit probability vs. Lag-1 autocorrelation](image)

- Zipf $\alpha=0.8$, $n=1000$, $m=100$
- LRU
- 2-LRU
- LRU($m,m$)
- LRU($m/2,m/2$)
Quantitative remark 1: LRU is insensitive to correlations between requests time

**Theorem 2.** Assume that the items’ request processes are stationary, independent of each other and that the expected number of requests per unit time is positive and finite. Then, the hit probability of LRU only depends on the inter-arrival time distribution. In particular, it does not depend on the correlation between inter-arrival times.
Quantitative remark 1: We verified on a web trace\textsuperscript{11} that having virtual list seems to improve performance.

\begin{align*}
\text{LRU}(m,m) \\
\text{LRU}(m/2,m/2) \\
\text{LRU}(m,m/2,m/2)
\end{align*}

\textsuperscript{11}[Bianchi et al. 2013]
Quantitative remark 1: We verified on a web trace\textsuperscript{11} that having virtual list seems to improve performance.

\textsuperscript{11}[Bianchi et al. 2013]
Outline

1 Two cache replacement policies
2 Performance analysis via TTL approximation
3 Asymptotic exactness of the approximation
4 Comparison between LRU, LRU(\(\tilde{m}\)) and \(h\)-LRU
5 Conclusion
Conclusion

- Characterize list-based cache replacement policies
- We provide TTL approximation
  - New or improved approximations
  - Exact for large cache

- Theoretical interests:
  - Prove equivalence between TTL and cache replacement policies
  - Show that these approximation work for MAP

- Practical applications:
  - Comparison of LRU($m$) and $h$-LRU.
  - Our results can be used to tune such algorithms.
Questions or comments?

http://mescal.imag.fr/membres/nicolas.gast

nicolas.gast@inria.fr

Supported by EU project quanticol http://www.quanticol.eu
Hyperexponential

\[ z - \frac{qz}{1 + z} \]

\[ q \frac{z}{1 + z} \]

\[ 1/z - \frac{q}{1 + z} \]

\[ q/(1 + z) \]

Fire rate:

- Proba(0) = \( z/(1 + z) \). Fire rate = \( z \).
- Proba(1) = \( 1/(1 + z) \). Fire rate = \( 1/z \).

Coefficient of variation:

\[
\frac{z}{1 + z} \frac{2}{z^2} + \frac{1}{1 + z} 2z^2 - 1.
\]