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

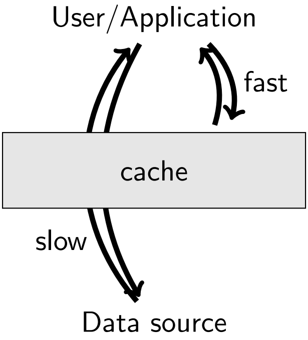
Nicolas Gast[1], Benny Van Houdt[2]

ITC 2016

September 13-15, Würzburg, Germany

[1]Inria
[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)

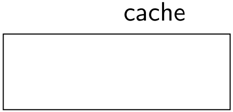
Contributions (and Outline)

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

Outline

The two policies generalize the LRU policy

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



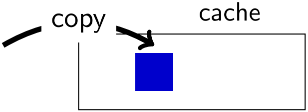
Example with this stream of requests:

(note: similar to RANDOM, FIFO)

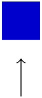
(Assumption: Object are assumed to have the same size.)

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:

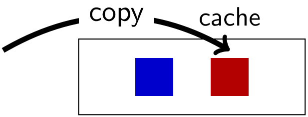


(note: similar to RANDOM, FIFO)

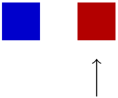
(Assumption: Object are assumed to have the same size.)

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:



(note: similar to RANDOM, FIFO)

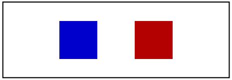
(Assumption: Object are assumed to have the same size.)

The two policies generalize the LRU policy

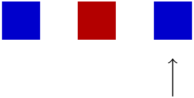
LRU:
- hit : do nothing
- miss : evict the LRU (least-recently used) item.

(do nothing)

cache



Example with this stream of requests:



(note: similar to RANDOM, FIFO)

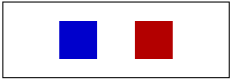
(Assumption: Object are assumed to have the same size.)

The two policies generalize the LRU policy

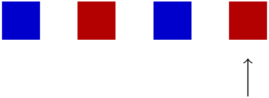
LRU:
- hit : do nothing
- miss : evict the LRU (least-recently used) item.

(do nothing)

cache



Example with this stream of requests:

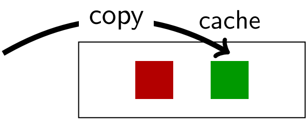


(note: similar to RANDOM, FIFO)

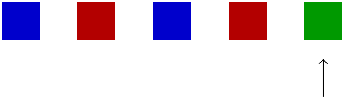
(Assumption: Object are assumed to have the same size.)

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:

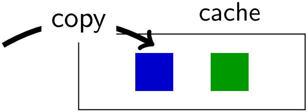


(note: similar to RANDOM, FIFO)

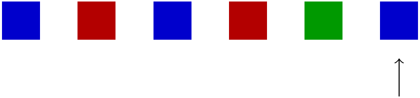
(Assumption: Object are assumed to have the same size.)

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:

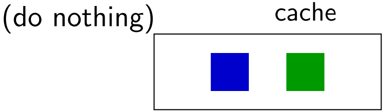


(note: similar to RANDOM, FIFO)

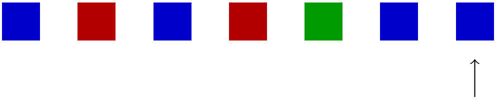
(Assumption: Object are assumed to have the same size.)

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:

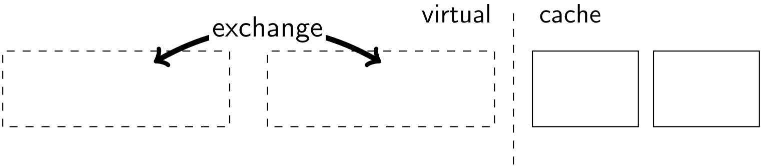


(note: similar to RANDOM, FIFO)

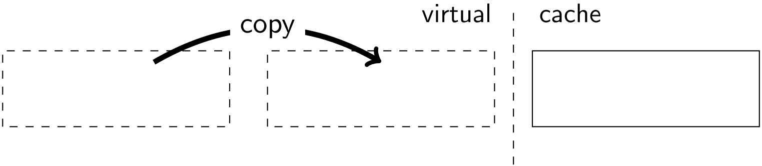
(Assumption: Object are assumed to have the same size.)

The LRU($\vec{m}$) and h-LRU policies

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



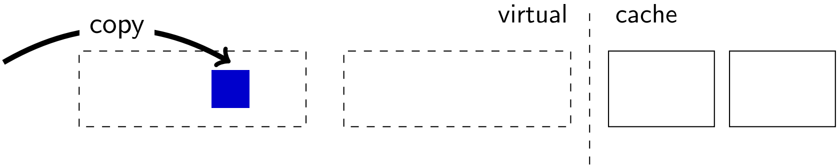
- h-LRU[8]: copy the requested item in the next list (and evict the LRU)



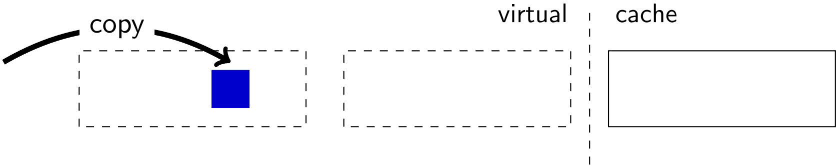
[7] Variant of RAND($\vec{m}$) of [G, Van Houdt 2015]
[8] Introduced as k-LRU in [Martina et al. 2014]

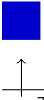
The LRU($\vec{m}$) and h-LRU policies

- 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)

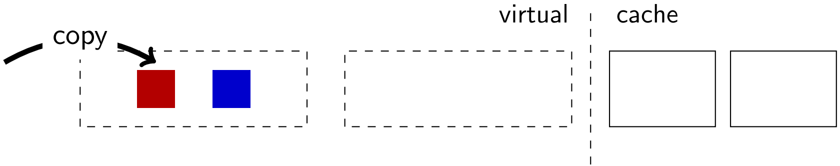




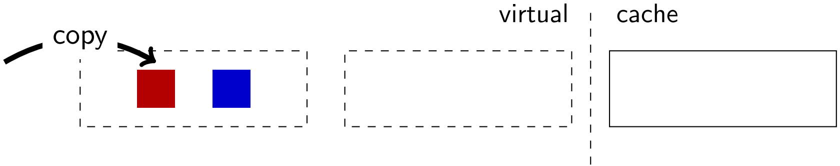
[7] Variant of RAND($\vec{m}$) of [G, Van Houdt 2015]
[8] Introduced as k-LRU in [Martina et al. 2014]

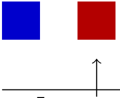
The LRU($\vec{m}$) and h-LRU policies

- 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)



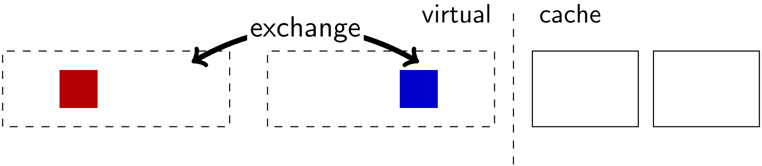


[7] Variant of RAND($\vec{m}$) of [G, Van Houdt 2015]

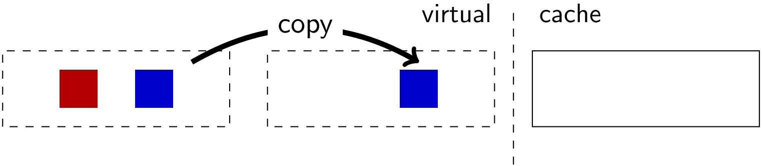
[8] Introduced as k-LRU in [Martina et al. 2014]

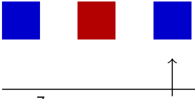
The LRU($\vec{m}$) and h-LRU policies

- 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)

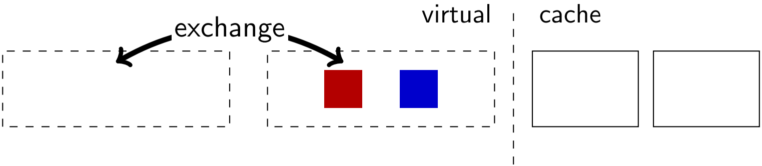




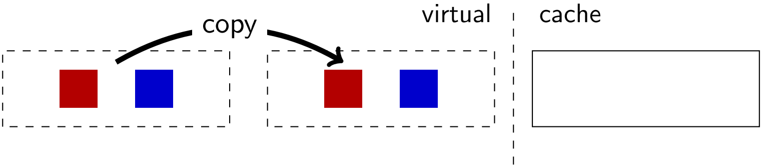
[7]Variant of RAND($\vec{m}$) of [G, Van Houdt 2015]
[8]Introduced as k-LRU in [Martina et al. 2014]

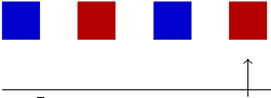
The LRU($\vec{m}$) and h-LRU policies

- 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)

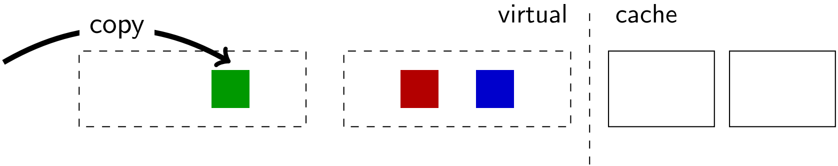




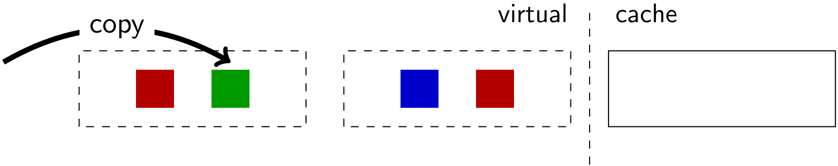
[7] Variant of RAND($\vec{m}$) of [G, Van Houdt 2015]
[8] Introduced as k-LRU in [Martina et al. 2014]

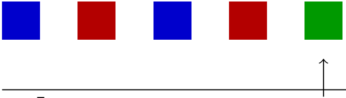
The LRU($\vec{m}$) and h-LRU policies

- 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)

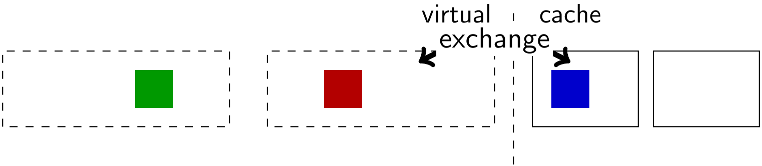




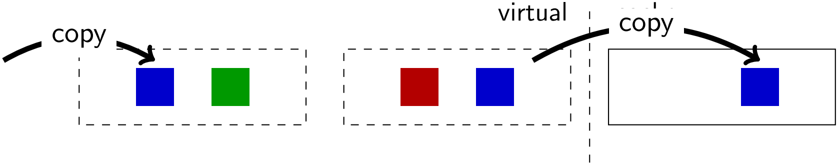
[7]Variant of RAND($\vec{m}$) of [G, Van Houdt 2015]
[8]Introduced as k-LRU in [Martina et al. 2014]

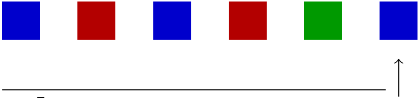
The LRU($\vec{m}$) and h-LRU policies

- 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)

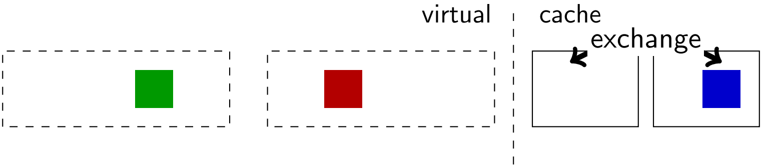




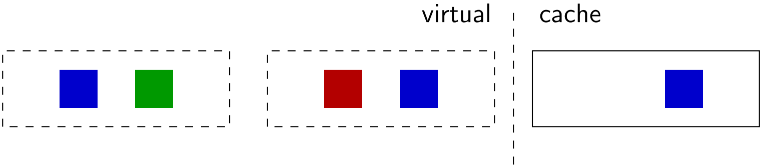
[7]Variant of RAND($\vec{m}$) of [G, Van Houdt 2015]
[8]Introduced as k-LRU in [Martina et al. 2014]

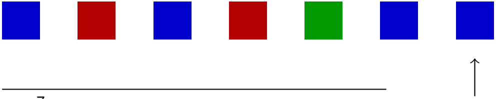
The LRU($\vec{m}$) and h-LRU policies

- 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)



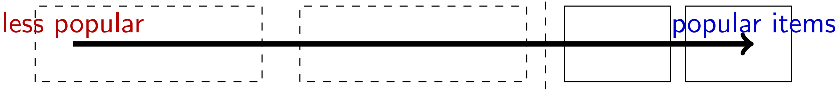


[7] Variant of RAND($\vec{m}$) of [G, Van Houdt 2015]
[8] Introduced as k-LRU in [Martina et al. 2014]

Outline

In this talk: Performance analysis and comparison

Qualitatively:



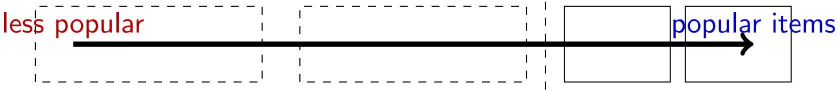
- It takes time to adapt

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

[10]heuristic for h-LRU [Martina et al. 2014]

In this talk: Performance analysis and comparison

Qualitatively:



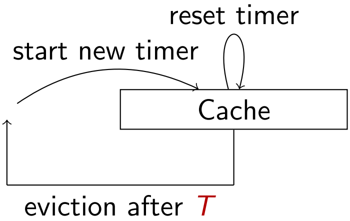
- It takes time to adapt

Quantitatively:

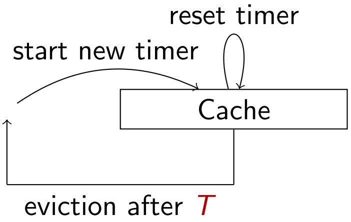
- Related work: Variants[9] or less accurate approximation[10]
- We present TTL approximations for MAP arrival (in the talk: IRM).

[9](RAND($\vec{m}$) 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



Pure LRU: the Che-approximation

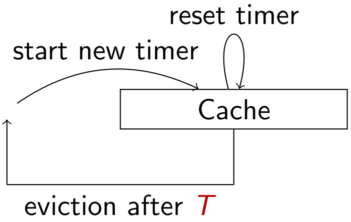


If the request of object k is a Poisson process of intensity λ_k:

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

(TTL)

Pure LRU: the Che-approximation



If the request of object k is a Poisson process of intensity λ_k:

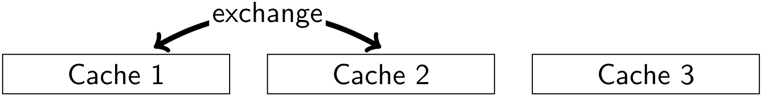
- Object k is in cache with probability $\pi_k(T) = 1 - e^{-\lambda_k T}$

(TTL)

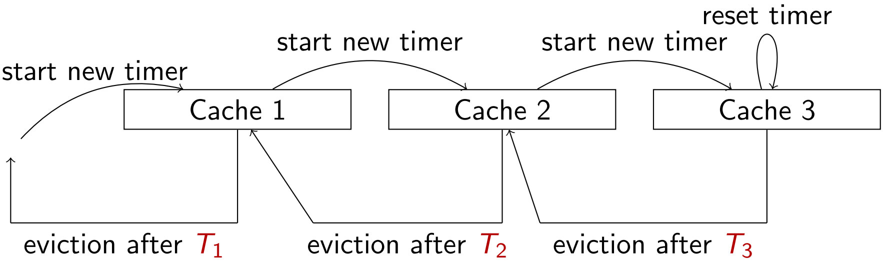
T satisfies $\sum_k \pi_k(T) =$ cache size .

(Fixed point)

The TTL-approximation for LRU(m)



The TTL-approximation for LRU(m)

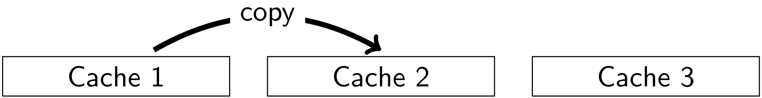


If the request of object k is a Poisson process of intensity λ_k:

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

$T_1 \dots T_h$ satisfy $\sum_k \pi_{k,i}(T_1 \dots T_h) =$ size of list i.

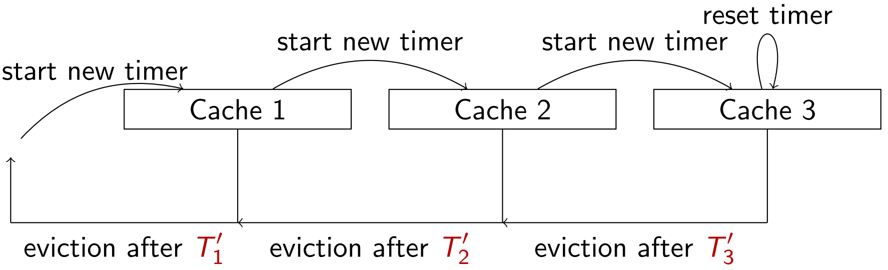
The TTL-approximation for h-LRU



First idea: track the lists in which an object are. [Martina et al. 14]

- Problem: number of states $= 2^h$.

The TTL-approximation for h-LRU



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

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

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

Outline

Is the approximation accurate?

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

	Simulation	(Our approximation)	[Martina et al. 14])
$n = 1000$	0.51506	0.51552 (+0.088%)	0.50796 (-1.380%)
$n = 10000$	0.56124	0.56130 (+0.012%)	0.55447 (-1.206%)

Is the approximation accurate?

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

	Simulation	(Our approximation)	[Martina et al. 14])
$n = 1000$	0.51506	0.51552 (+0.088%)	0.50796 (-1.380%)
$n = 10000$	0.56124	0.56130 (+0.012%)	0.55447 (-1.206%)

- 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

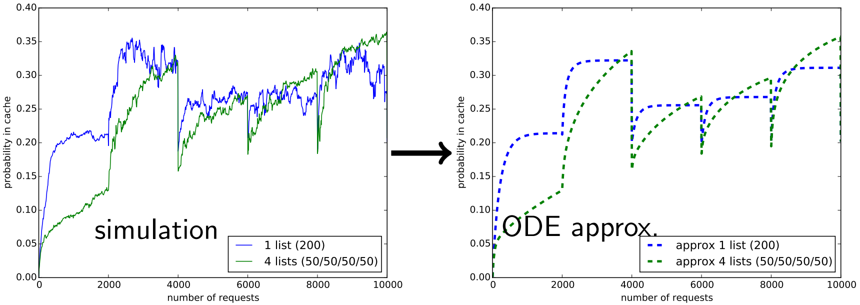


Figure: Popularities of objects change every 2000 steps.

- We develop an ODE approximation
- We show that it is accurate
- This ODE has the same fixed point as the TTL approximation

Asymptotic exactness of the approximation

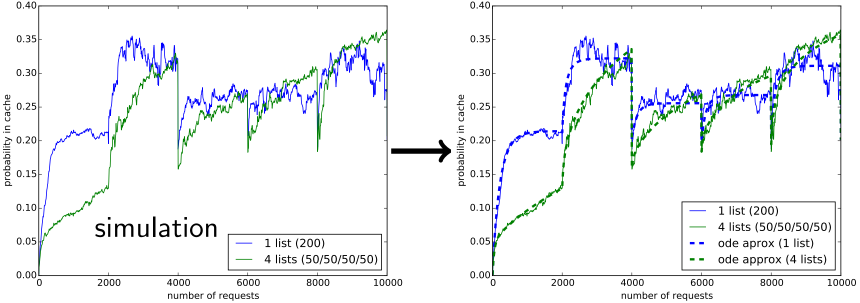


Figure: Popularities of objects change every 2000 steps.

- We develop an ODE approximation
- We show that it is accurate
- This ODE has the same fixed point as the TTL approximation

Convergence result and idea of the proof

Theorem 1. *Let $H_\ell(t)$ be the sum of the popularity of the items of list ℓ and $h_\ell(t)$ be the corresponding ODE approximation (Equation (18) for h-LRU and Equation (22) for LRU($\mathbf{m}$)). Then: for any time T, there exists a constant C such that*

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

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

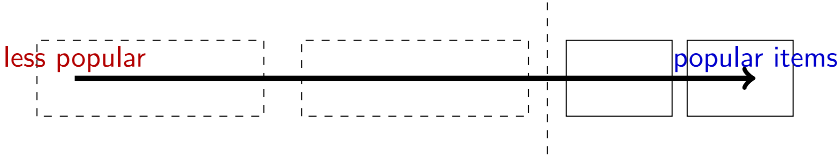
Idea of the proof.

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

$\square$

Outline

Qualitative remarks

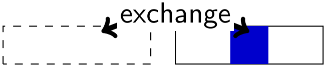


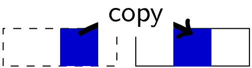
In general, adding more lists:

- Improves the steady-state performance[a],
- Decreases the response time.

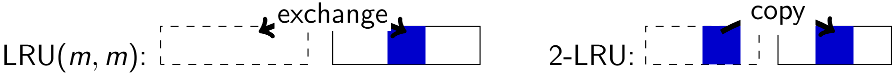
[a] This is not true in full generality, even for IRM. The same counter-example as in [G., Van Houdt 2015] works.

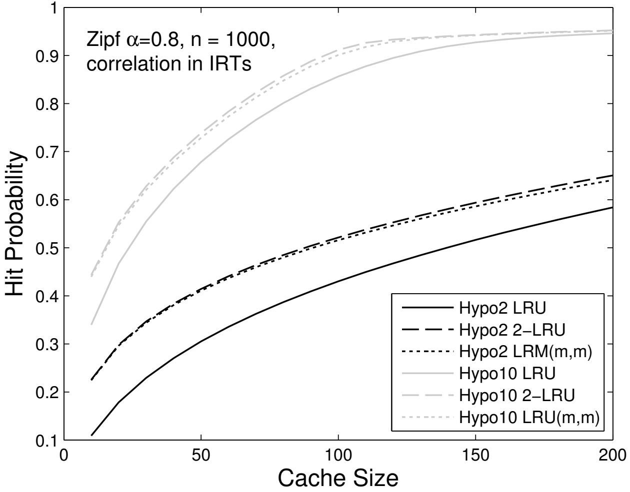
Quantitative remark 1: On synthetic traces: LRU(m, m) and 2-LRU perform similarly

LRU(m, m): 

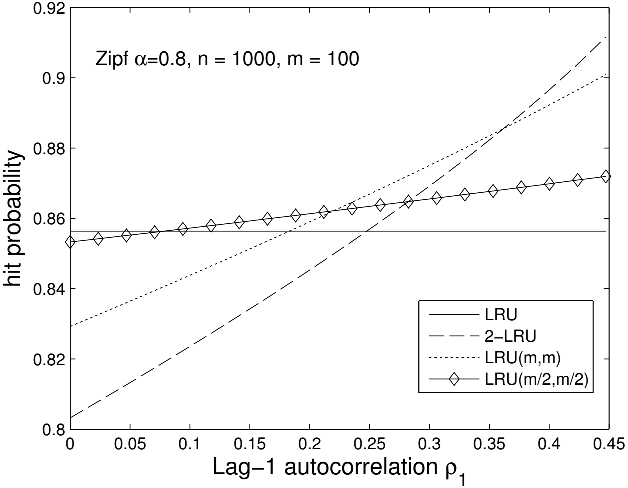
2-LRU: 

Quantitative remark 1: On synthetic traces: LRU(m, m) and 2-LRU perform similarly



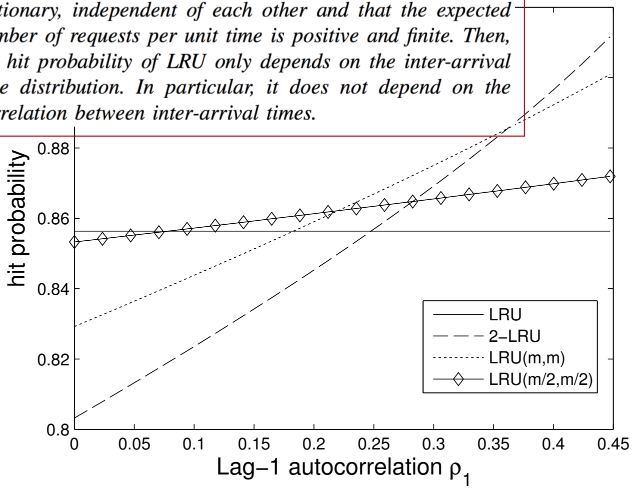


Quantitative remark 1: LRU is insensitive to correlations between requests time



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.*

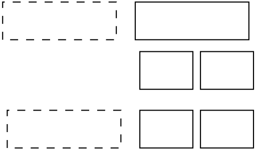


Quantitative remark 1: We verified on a web trace[11] that having virtual list seems to improve performance.

LRU(m,m)

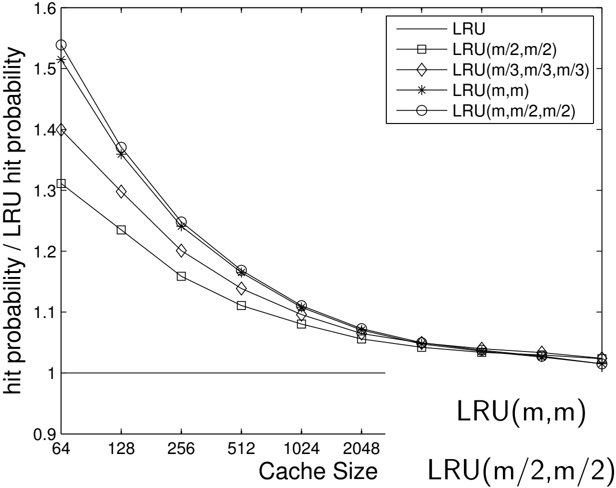
LRU(m/2,m/2)

LRU(m,m/2,m/2)



[11] [Bianchi et al. 2013]

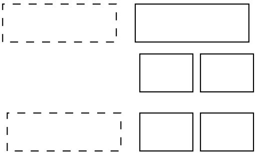
Quantitative remark 1: We verified on a web trace[11] that having virtual list seems to improve performance.



LRU(m,m)

LRU(m/2,m/2)

LRU(m,m/2,m/2)



[11] [Bianchi et al. 2013]

Outline

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.

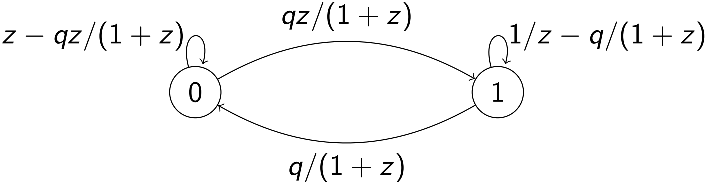
Questions or comments?

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

nicolas.gast@inria.fr

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

Hyperexponential



Fire rate:

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

Coefficient of variation: $\frac{z}{1+z}\frac{2}{z^2} + \frac{1}{1+z}2z^2 - 1$.