Analysis and design of list-based cache replacement policies

Nicolas Gast (Inria)

Inria (joint work with Benny Van Houdt (Univ. of Antwerp))

POLARIS / DataMove Seminar, Jan.2016, Inria

---

Caches are everywhere

Examples:
- Processor
- Database
- CDN
- Single cache / hierarchy of caches
In this talk, I focus on a single cache.

The question is: which item to replace?

Application

requests

| cache |

data source
In this talk, I focus on a single cache.

The question is: which item to replace?

![Diagram of cache system]

- Application
- Data source
- Cache
- Requests
- Hit
- LRU, RAND, FIFO, CLIMB
- Other approaches: Time to live

Data source
In this talk, I focus on a single cache.

The question is: which item to replace?

Classical cache replacement policies:
- RAND
- FIFO
- LRU
- CLIMB

Other approaches:
- Time to live
The analysis of cache performance has a growing interest

- Theoretical studies: started with [King 1971, Gelenbe 1973]

Nowadays:
- New applications: CDN / CON (replication\(^2\))
- New analysis techniques (Che approximation\(^3,4\))

\(^2\)Borst et al. 2010 Distributed Caching Algorithms for Content Distribution Networks
\(^3\)Che et al 2002 Hierarchical web caching systems: modeling, design and experimental results.
\(^4\)Fricker et al. 2012 A versatile and accurate approximation for lru cache performance
Outline of the talk

1. What are the classical models?
Outline of the talk

1. What are the classical models?
2. We introduce a family of policies for which the cache is (virtually) divided into lists (generalization of FIFO/RANDOM)
   - We can compute in polynomial time the steady-state distribution
     ★ Disprove old conjectures.

3. We develop a mean-field approximation and show that it is accurate
   ★ Fast approximation of the steady-state distribution.
   ★ We can characterize the transient behavior:

4. We provide guidelines of how to tune the parameters by using IRM and trace-based simulation
Outline of the talk

1 What are the classical models?

2 We introduce a family of policies for which the cache is (virtually) divided into lists (generalization of FIFO/RANDOM)
   1 We can compute in polynomial time the steady-state distribution
     * Disprove old conjectures.
   2 We develop a mean-field approximation and show that it is accurate
     * Fast approximation of the steady-state distribution.
     * We can characterize the transient behavior:

Simulation

ODE approximation

Nicolas Gast – 5 / 31
Outline of the talk

1. What are the classical models?

2. We introduce a family of policies for which the cache is (virtually) divided into lists (generalization of FIFO/RANDOM)
   - We can compute in polynomial time the steady-state distribution
     - Disprove old conjectures.

2. We develop a mean-field approximation and show that it is accurate
   - Fast approximation of the steady-state distribution.
   - We can characterize the transient behavior:

3. We provide guidelines of how to tune the parameters by using IRM and trace-based simulation
Outline

1 Performance models of caches

2 List-based cache replacement algorithms
   - Steady-state performance under the IRM model
   - Transient behavior via mean-field approximation

3 Parameters tuning and practical guidelines

4 Conclusion
Outline

1. Performance models of caches
2. List-based cache replacement algorithms
   - Steady-state performance under the IRM model
   - Transient behavior via mean-field approximation
3. Parameters tuning and practical guidelines
4. Conclusion
Our performance metric will be the hit probability

\[
\text{hit probability} = \frac{\text{number of items served from cache}}{\text{total number of items served}} = 1 - \text{miss probability}
\]

Goal: find a policy to maximize the hit probability.
The offline problem is easy...
The offline problem is easy...

If you know the sequence of requests:

**MIN policy**
At time $t$, if $X_t$ is not in the cache, evict an item in the cache whose next request occurs furthest in the future.

**Theorem (Maston et al. 1970)**
MIN is optimal
The offline problem is easy... but with unbounded competitive ratio

Theorem

- No deterministic online algorithm for caching can achieve a better competitive ratio than $m$.
- LRU has a competitive ratio of $m$. 

Application

requests

hit

miss

cache (size $m$)

replace one item

data source
To compare policies, we need more...

- We can use trace-based simulations.

---

To compare policies, we need more...

- We can use trace-based simulations.
- We can model request as stochastic processes (Started with [King 1971, Gelenbe 1973])

**Independent reference model (IRM)**

At each time step, item $i$ is requested with probability $p_i$.

IRM is OK for web-caching\(^5\)

---

Example: analysis of LRU: from King [71] to Che [2002]

[King 71]: Under IRM model, in steady-state, the probability of having a sequence of distinct items \( i_1 \ldots i_n \) is

\[
\mathbb{P}(i_1 \ldots i_m) = p_{i_1} \frac{p_{i_2}}{1 - p_{i_1}} \ldots \frac{p_{i_m}}{1 - p_{i_1} - \ldots - p_{i_{m-1}}}
\]

Hit probability is:

\[
\sum_{\text{distinct sequences } i_1 \ldots i_m} (p_{i_1} + \ldots + p_{i_m}) \mathbb{P}(i_1 \ldots i_m).
\]
Example: analysis of LRU: from King [71] to Che [2002]

[King 71]: Under IRM model, in steady-state, the probability of having a sequence of distinct items $i_1 \ldots i_n$ is

$$
\mathbb{P}(i_1 \ldots i_m) = p_{i_1} \frac{p_{i_2}}{1 - p_{i_1}} \ldots \frac{p_{i_m}}{1 - p_{i_1} - \ldots p_{i_{m-1}}}
$$

Hit probability is:

$$
\sum_{\text{distinct sequences } i_1 \ldots i_m} (p_{i_1} + \ldots + p_{i_m}) \mathbb{P}(i_1 \ldots i_m).
$$

[Che approximation 2002]: an item spends approximately $T$ in the cache.

$$
\mathbb{P}(\text{item } i \text{ in cache}) \approx 1 - e^{-p_i T},
$$

where $T$ is such that $\sum_{i=1}^{n} 1 - e^{-p_i T}$
Even when the popularity is constant, LFU is not optimal.

- LFU is optimal under IRM (it maximizes the steady-state hit probability).

\[\text{e.g.} \quad \text{time between two requests of item 1} = 1 \text{ with probability } 99, \quad \text{time between two requests of item 2} = 5.\]
Even when the popularity is constant, LFU is not optimal.

- LFU is optimal under IRM (it maximizes the steady-state hit probability).

- LFU is not optimal under general distribution:
  - e.g. time between two requests of item 1 = 1 with probability .99, 1000 with probability .01. Time between two requests of item 2 is 5. LRU outperforms LFU.
Outline

1 Performance models of caches

2 List-based cache replacement algorithms
   - Steady-state performance under the IRM model
   - Transient behavior via mean-field approximation

3 Parameters tuning and practical guidelines

4 Conclusion
I consider a cache (virtually) divided into lists

Application

list 1 ... list j list j+1 ... list h

data source

IRM At each time step, item $i$ is requested with probability $p_i$ (IRM assumption$^3$)

---

I consider a cache (virtually) divided into lists

**IRM** At each time step, item $i$ is requested with probability $p_i$ (IRM assumption$^3$)

**MISS** If item $i$ is not in the cache, it is exchanged with a item from list 1 (FIFO or RAND).

---

I consider a cache (virtually) divided into lists

Application

miss

hit

data source

list 1  ...  list j  list j+1  ...  list h

IRM  At each time step, item $i$ is requested with probability $p_i$ (IRM assumption\(^3\))

MISS  If item $i$ is not in the cache, it is exchanged with an item from list 1 (FIFO or RAND).

HIT  If item $i$ is list $j$, it is exchanged with an item from list $j + 1$ (FIFO or RAND).

---

Items on higher lists are (supposedly) more popular.

\[
\text{cache size } = m = m_1 + \cdots + m_h
\]

These algorithms are referred to as RAND(m) and FIFO(m).
The steady-state is a product-form distribution

**Theorem 1.** The steady state probabilities \( \pi_{\text{RAND}}(m)(c) \) and \( \pi_{\text{FIFO}}(m)(c) \), with \( c \in C_n(m) \), can be written as

\[
\pi_{\text{FIFO}}(m)(c) = \pi_{\text{RAND}}(m)(c) = \pi(c) \triangleq \frac{1}{Z(m)} \prod_{i=1}^{h} \left( \prod_{j=1}^{m_i} p_{c(i,j)} \right)^i,
\]

(1)

where \( Z(m) = \sum_{c \in C_n(m)} \prod_{i=1}^{h} \left( \prod_{j=1}^{m_i} p_{c(i,j)} \right)^i \).

- Same for RAND and FIFO.
The steady-state is a product-form distribution

**Theorem 1.** The steady state probabilities \( \pi_{\text{RAND}}(m)(c) \) and \( \pi_{\text{FIFO}}(m)(c) \), with \( c \in C_n(m) \), can be written as

\[
\pi_{\text{FIFO}}(m)(c) = \pi_{\text{RAND}}(m)(c) =
\]

\[
\pi(c) \triangleq \frac{1}{Z(m)} \prod_{i=1}^{h} \left( \prod_{j=1}^{m_i} p_{c(i,j)} \right)^i,
\]

where \( Z(m) = \sum_{c \in C_n(m)} \prod_{i=1}^{h} \left( \prod_{j=1}^{m_i} p_{c(i,j)} \right)^i \).

- Same for RAND and FIFO.

---

**Example of a cache of size 4 with 3 lists and** \( m = (1, 2, 1) \)

\[
\begin{array}{|c|c|c|c|}
\hline
i & j & k & \ell \\
\hline
\end{array}
\]

Probability of \((i, j, k, \ell)\) is proportional to \( p_i(p_jp_k)^2(p_\ell)^3 \).
We can compute the miss probability by using a dynamic programming approach (Generalization of [Fagin,Price]⁸).

We want to compute

\[ M(m) = \sum_{c \in C_n(m)} \left( \sum_{k \not\in c} p_k \right) \pi(c) = \frac{E(m + e_1, n)}{E(m, n)}, \]

where

\[ E(r, k) = \sum_{c \in C_k(r)} \prod_{i=1}^{h} \left( \prod_{j=1}^{r_i} p_{c(i,j)} \right)^i. \]

We obtain a recursion formula on \( E(r, k) \): solvable in \( O(n \times m_1 \ldots m_h) \).

The Dan and Towsley⁷ approximation is not needed for polynomial time.


A higher cache size and more lists (usually) leads to a lower steady-state miss probability.

\[ h = \infty \text{ corresponds to LFU}. \]
Is increasing the number of lists always better? 

\[ m_1, \ldots, m_j, m_{j+1}, \ldots, m_h \]

hit

less popular popular items

Six lists: \( m = (1, 1, 1, 1, 1, 1) \)

Three lists: \( m = (1, 1, 4) \).

\[ \geq? \]

\[ ^9 \text{conjectured in 1987!} \]

Is increasing the number of lists always better\(^9\)?

Six lists: \( m = (1, 1, 1, 1, 1, 1) \)

Three lists: \( m = (1, 1, 4) \).

<table>
<thead>
<tr>
<th>policy</th>
<th>( m )</th>
<th>( M(m) )</th>
<th>lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>RAND(1,1,4)</td>
<td>0.005284</td>
<td>0.004925</td>
</tr>
<tr>
<td></td>
<td>RAND(1,1,3,1)</td>
<td>0.005299</td>
<td>0.004884</td>
</tr>
<tr>
<td></td>
<td>RAND(1,1,2,2)</td>
<td>0.005317</td>
<td>0.004884</td>
</tr>
<tr>
<td></td>
<td>RAND(1,1,2,1,1)</td>
<td>0.005321</td>
<td>0.004879</td>
</tr>
<tr>
<td></td>
<td>RAND(1,1,1,3)</td>
<td>0.005338</td>
<td>0.004884</td>
</tr>
<tr>
<td></td>
<td>RAND(1,1,1,2,1)</td>
<td>0.005343</td>
<td>0.004879</td>
</tr>
<tr>
<td></td>
<td>RAND(1,1,1,1,2)</td>
<td>0.005347</td>
<td>0.004879</td>
</tr>
<tr>
<td>CLIMB</td>
<td>RAND(1,1,1,1,1)</td>
<td>0.005348</td>
<td>0.004878</td>
</tr>
<tr>
<td></td>
<td>RAND(1,2,3)</td>
<td>0.005428</td>
<td>0.004925</td>
</tr>
<tr>
<td></td>
<td>RAND(1,2,2,1)</td>
<td>0.005439</td>
<td>0.004884</td>
</tr>
<tr>
<td>LRU</td>
<td>LRU(6)</td>
<td>0.005880</td>
<td>–</td>
</tr>
<tr>
<td>RANDOM</td>
<td>RAND(6)</td>
<td>0.015350</td>
<td>0.015350</td>
</tr>
</tbody>
</table>

Table 1: CLIMB is not optimal for IRM model: \( p = (49, 49, 49, 49, 7, 1, 1) \)/205 and \( m = 6 \).

Outline

1 Performance models of caches

2 List-based cache replacement algorithms
   - Steady-state performance under the IRM model
   - Transient behavior via mean-field approximation

3 Parameters tuning and practical guidelines

4 Conclusion
We want to study at which speed the caches fills

Figure: Popularities of objects change every 2000 steps.
We want to study at which speed the caches fills

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

- We develop an ODE approximation
We want to study at which speed the caches fills

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

- We develop an ODE approximation
- We show that it is accurate
We construct an ODE by assuming independence

Let $H_i(t)$ be the popularity in list $i$. 

![Diagram of states and transitions]

- $p_k$: Population at state $k$
- $H_i(t)$: Popularity in list $i$ at time $t$
- $m_1$, $m_2$, $m_3$, $m_h$: Transition rates

This is similar to a TTL approximation.
We construct an ODE by assuming independence

Let $H_i(t)$ be the popularity in list $i$.

If $x_{k,i}(t)$ is the probability that item $k$ is in list $i$ at time $t$, we approximately have:

This is similar to a TTL approximation.
We show that this approximation is accurate, theoretically and by simulation.

**Theorem 6.** For any $T > 0$, there exists a constant $C > 0$ that depends on $T$ such that, for any probability distribution over $n$ items and list sizes $m_1 \ldots m_h$, we have:

$$
\mathbb{E} \left[ \sup_{t \in \{0 \ldots \tau\}, i \in \{0 \ldots h\}} |H_i(t) - \delta_i(t)| \right] \leq C \sqrt{\max_{k=1}^n p_k + \max_{i=0}^h \frac{1}{m_i}},
$$

where $\tau := [T/(\max_{k=1}^n p_k + \max_{i=0}^h \frac{1}{m_i})]$. 

![Graph representing the probability in cache over the number of requests for different list configurations.

---

Nicolas Gast – 23 / 31
This approximation can also be used to compute stationary distribution.

**Theorem 7.** The mean-field model (8) has a unique fixed point. For this fixed point, the probability that item $k$ is part of list $i$, for $k = 1, \ldots, n$ and $i = 0, \ldots, h$, is given by

$$x_{k,i} = \frac{p_k^i z_i}{1 + \sum_{j=1}^{h} p_{k}^j z_j},$$

where $z = (z_1, \ldots, z_n)$ is the unique solution of the equation

$$\sum_{k=1}^{n} \frac{p_k^i z_i}{1 + \sum_{j=1}^{h} p_{k}^j z_j} = m_i. \quad (14)$$

<table>
<thead>
<tr>
<th>$m_1$</th>
<th>$m_2$</th>
<th>$m_3$</th>
<th>$m_4$</th>
<th>exact</th>
<th>mean field</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>96</td>
<td></td>
<td>0.3166</td>
<td>0.3169</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>60</td>
<td>-</td>
<td>0.3296</td>
<td>0.3299</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>78</td>
<td>-</td>
<td>0.3273</td>
<td>0.3276</td>
</tr>
<tr>
<td>90</td>
<td>8</td>
<td>2</td>
<td>-</td>
<td>0.4094</td>
<td>0.4100</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>10</td>
<td>85</td>
<td>0.3039</td>
<td>0.3041</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>25</td>
<td>55</td>
<td>0.3136</td>
<td>0.3139</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>0.3345</td>
<td>0.3348</td>
</tr>
<tr>
<td>60</td>
<td>2</td>
<td>2</td>
<td>36</td>
<td>0.3514</td>
<td>0.3517</td>
</tr>
</tbody>
</table>

- **Very accurate:**

- **Map is contracting:** computation in $O(nh)$, compared to $O(nm_1 \ldots m_h)$ for the exact.
Outline

1. Performance models of caches

2. List-based cache replacement algorithms
   - Steady-state performance under the IRM model
   - Transient behavior via mean-field approximation

3. Parameters tuning and practical guidelines

4. Conclusion
Under the IRM model, a smaller first list (usually) means a higher hit probability but a larger time to fill the cache.
Under the IRM model, the time to fill the cache mainly depend on the size of the first list.

- In a dynamic setting, a good choice seems to be \( m_1 \geq m_2 \cdots \geq m_h \) with \( m_1 \) “large-enough”. 

\[
\begin{array}{c|c|c|c|c}
\text{Number of Requests} & \text{Hit Probability} & m = (40,160), \text{ODE} & m = (40,160), \text{simul} & m = (40,40,120), \text{ODE} & m = (40,40,120) \text{ simul} & m = (40,40,40,80), \text{ODE} & m = (40,40,40,80), \text{simul} \\
\end{array}
\]
We verified on a trace of youtube videos\textsuperscript{10}, that reserving at least 30\% of the cache for the first list seems important.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{hit_probability.png}
\caption{Comparison of hit probability for different cache sizes and list configurations.}
\end{figure}

Outline

1 Performance models of caches

2 List-based cache replacement algorithms
   - Steady-state performance under the IRM model
   - Transient behavior via mean-field approximation

3 Parameters tuning and practical guidelines

4 Conclusion
Recap

- Unified framework for studying list-based replacement policies.
- Steady-state miss probability in polynomial time.
- Accurate ODE approximation
- Guidelines on how to use such a replacement algorithm: the size of the first list is important.

Two theoretical interests of this work:
- provides a unified framework and disproves old conjectures.
- ODE approximation

Future work

- Network of caches?
- Applications?
Thank you!

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

nicolas.gast@inria.fr