Prefetching in Web Applications

Noa Avigdor-Elgrabli  Ronny Lempel  Yuval Rabani

Motivation
- Search engines need to compute query answer pages, delaying response time.
- Caching recurring queries reduces latency.
- The time saved on cache hits can be used to prefetch answers to anticipated queries.

The Model

The prefetching problem:
- Cache size = \(k\)
- Possible query answers (pages) = \(n\) \((n > k)\)
- At each step, some queries arrive:
  - On a cache hit: prefetch any single page.
  - On a cache miss: the missed page may be cached.
- Objective: maximize the rate of cache hits.

The input’s stochastic model:
- Each time a query arrives, the distribution of its next arrival time is known to the algorithm.*
- Page distributions are independent.
- Each distribution is on an interval of length \(\leq L\).

*Identical queries need not arrive with the same distribution.

Why Simple Algorithms will not Work?
- Type 1 will be requested at the next step w.p. \(1 - \varepsilon\).
- After expected number of \(1/\varepsilon\) hits type 1 prefetching will fail and we will need to wait \(L\) steps to recover.
- Type 2 will only arrive in \(L/2\) steps, but afterwards we will score \(L/2\) sure hits.

The best page to prefetch may depend on the entire state of the system.

Classifying Next Arrival Time Distributions
- We classify a distribution by a sequence of points which we call the \((\alpha, \lambda)\)-profile of the distribution.
- We approximate the \((\alpha, \lambda)\)-profile by rounding the distances between consecutive points to the closest power of 2. This is called a \((\alpha, \lambda)\)-sketch of the distribution.
- Notice that at each step after the last arrival of a page it has a different conditional next arrival time distribution.

Classifying Hit Attempts
- A hit attempt is trying to score a hit on a specific segment of the distribution.
- We classify hit attempts by:
  - The length of the segment, \(B = 2^i\).
  - The profile point \(i_x\) such that \(i_x \leq B < i_{x+1}\).
  - The \((\alpha, \lambda)\)-sketch of the distribution the page will have at step \(B\) assuming the hit attempt failed (ske).

Recovery
In order to resume a specific kind of hit attempts, after a failed attempt, one must “recover”:
- Self recovery is waiting for a later hit on the same page, classified by:
  - A profile point \(i_x\) of the distribution at step \(B\) that until this point we are willing to wait.
- External recovery is using a different type of hit on a prefetched page to resume attempts on the specific targeted type. The recovery page is classified by:
  - The length of the segment, \(B' = 2^i\).
  - The profile point \(i_x\) such that \(i_x \leq B' < i_{x+1}\).

Any recovery is a combination of both self and external recoveries.

The Algorithm for a Cache of Size 1
Choose uniformly at random one type of hit attempt \(B, \sigma, \text{ske}\), self recovery \(r\), and external recovery \(B', \sigma'\).
- Upon a hit: prefetch a candidate page that will match \(B, \sigma, \text{ske}\) the earliest.
- The cache holds a \(B, \sigma, \text{ske}\) page: hold it for \(8\) steps.
- A requested page is a better candidate: replace the current candidate.
- Upon failing a hit attempt:
  1. Self recovery: wait for a later hit for another \(i\) steps.
  2. External recovery:
     - The cache holds a \(B', \sigma'\) page: hold it for \(B'\) steps.
     - A requested page is a better recovery candidate: replace the current page in the cache.

Analysis
- Counting hits of one type chosen at random costs us the number of types.
- The basic observations of the analysis are:
  - \(\mathbb{E} \left[ \text{# consecutive successful attempts} \right] \leq \frac{1}{1 - \text{Pr}[0, i_{x}]}\)
  - An algorithm must wait at least \(B/2\) steps before an attempt, otherwise it is treated as another type.
- Using these observations we bound the expected number of \(B, \sigma, \text{ske}\) hits of any online algorithm between two of our hits.
- There is a separate argument for each phase of our algorithm (regular, self recovery, external recovery).

Our bound follows from balancing the number of possible sketches and the inaccuracy \(\sigma\) in a sketch.

Our algorithm achieves a competitive ratio of \(\exp(\sqrt{\log L \cdot \log \log L})\)

A Larger Cache
- We attempt hits on the set of earliest \(B, \sigma, \text{ske}\) pages.
- We attempt external recoveries on the set of earliest \(B', \sigma'\) pages.
- We map cached pages to cache entries dynamically and apply the \(k=1\) analysis.

The queries:

The cache:

We can cache this page as it was requested.

We can prefetch this page as this place got a hit.