Caching Intro

- A well-known concept in systems with multiple tiers of storage
  - Higher-level tiers have smaller capacities but faster access times
  - For simplicity, assume two tiers – the upper tier is the cache
- The system is presented with a continuous stream of requests (queries) for stored data items
  - If the requested item is in the cache (cache hit), it is quickly returned
  - Otherwise, a cache miss/fault occurs and the item is fetched from the slower tier; at that point it may be stored in the cache for future use
- A cache is governed by a replacement policy, that decides which item to evict whenever a new item is added to a full cache
Caching Intro (continued)

- Caches are effective when the request stream exhibits locality of reference – some correlation between past and future requests
- In other words – when the past can predict the future to some degree
  - In terms of information theory - when there’s redundancy in the request stream (i.e. it can be compressed)
- Hit ratio – the rate of cache hits – is the main metric of success
  - High hit ratios typically mean reduced latency and higher throughput in processing the request stream

Search Results Caching and Prefetching

- Exploits locality of reference in the query streams with which engines are faced:
  - Query popularities vary widely, from the extremely popular to the very rare
    - Popular queries are further divided into those that are “always” popular and those with bursty popularity
  - Significant numbers of users view multiple result pages per query (motivates prefetching)

Successful caching and prefetching of results can:
1. Lower the number/cost of query executions
2. Shorten the engine’s response time
Search Results Caching: Conceptual Flow

Query → Query Parse → Canonical Query → Results Cache

- Is cached? Y → Cached Results
- Is cached? N → Query Evaluation → S1, S2, S3, S4 → Computed Results

Search Results Caching Schemes

- Classic schemes: LRU, SLRU
  - First tested in the context of search results by Markatos in 2001; he also tested LRU/2 and FBR (not covered here)

- Schemes tailored for caching search results (non-comprehensive list):
  - PDC – Probability Driven Cache (Lempel & Moran, 2003)
  - AC – Admission-based Caching (Baeza-Yates, Junqueira, Plachouras and Witschel, 2007)
**LRU – Least Recently Used**

- Underlying data structure: a queue, $Q$, of certain capacity
- Upon a request of a page $p$:
  - If $p$ is not cached, insert $p$ into the front of $Q$
  - If $Q$ is full, evict its tail
  - If $p$ is cached, move $p$ to the front of $Q$

**SLRU – Segmented LRU**

- Contains two LRU segments: a *probationary* segment ($S_1$) and a *protected* segment ($S_2$)
- Upon a request of a page $p$:
  - If $p$ is not cached, insert $p$ into the front of $S_1$
    - If $S_1$ is full, evict its tail
  - If $p$ is cached, move $p$ to the front of $S_2$
    - If $S_2$ is full, evict its tail to the front of $S_1$
**LRU, SLRU: Results**

- Markatos conducted simulations on a query log containing a million queries submitted to Excite
  - SLRU outperformed LRU, but for large caches, the difference was slight
  - Achieved hit rates of almost 30% for large, warm caches
  - No prefetching was applied
- Consistent results were reported by other researchers on other query logs
- On some logs, the achieved hit rates are 80% of what an infinite cache (that never evicts anything) can achieve

---

**PDC – Probability Driven Cache**

- Assigns priorities to cached results pages
- Based on the following simple probabilistic model of a search session
  1. A search session starts when a user enters a \((t, 1)\) query for some topic \(t\), requesting the first page of results
  2. After requesting results page \((t, k)\), the user proceeds to request \((t, k+1)\) w.p. \(p_k\) or terminates the session w.p. \(1-p_k\)
  - These probabilities are estimated from the query log
- The priority of a page: the probability that at least one active search session will request it
Data Structures in PDC

- An SLRU segment ($S$), a *query window* ($QW$), and a *priority queue* ($PQ$).
- ($t, 1$)-pages (top result pages) are kept in $S$.
- $QW$ keeps all *last-in-session* queries submitted within the past $\tau$ time units (a multiset).
  - Heuristic for anonymous query logs: whenever a ($t, k$) query enters $QW$, the least recent ($t, k-1$) query is removed.
- ($t, k > 1$)-pages are cached in $PQ$, and are prioritized as follows:
  $$\text{PR}[ (t, k) ] = 1 - \prod_{(t, m) \in QW, m < k} (1 - p_{k|m})$$

Experiments on an Altavista Query Log

- The following slides report experiments with LRU, SLRU and PDC on a 2001 Altavista query log (see next slides).
- Cache sizes are measured in units of “results page of size 10”.
- The caching schemes are coupled with a *fetch unit* to form an integrated caching and prefetching policy.
- The semantics of a *fetch unit* of $m$:
  - The engine prepares $10m$ results, or $m$ result pages, per query execution.
  - Effectively implies the prefetching of $m - 1$ pages.
- All hit rates are reported for warm, full caches.
The 2001 Altavista Query Log

- Contains 7160190 queries submitted to AltaVista During September 2001
- Each query is a triplet q=(t,f,l):
  1. t - the topic (search phrase) entered by the user
  2. f - the first page of results requested
  3. l - the last page of results requested
- Result page: a page containing 10 search results per query
- Statistics:
  - 97.7% of queries requested 10 results (a single page of results – f equaled l)
  - Overall, the 7160190 queries requested 7549873 result pages
  - Terminology: 7549873 total page views

Query Log Statistics: Distribution of page views

- 4797186 of the 7549873 views (63.5%) were of first pages (top-10 results)
- 885601 of the views (11.7%) were of second pages (results 11-20)
Query Log Statistics:
Topic/Page Popularities

- The 7160190 queries requested 4496503 distinct result pages, pertaining to 2657410 distinct topics
  1. 1792104 of the topics (67%) were only requested once; the most popular topic was queried 31546 times
  2. 48% of the result pages were only requested once; the 50 most requested pages account for almost 2% of the number of page views (150131 of 7549873)
- Topic and page popularities follow a power-law distribution; the relative frequency of topics/pages requested \(n\) times is proportional to \(n^{-c}\)
  - \(c\) is about 2.8 for pages, 2.4 for topics
- Very popular topics/pages are over represented in the log

![Frequency of topic requests (log-log plot)](image1)

![Frequency of page requests (log-log plot)](image2)
Upper Bounds on the Hit Ratio

The Altavista query log implies the following upper bounds on the achievable hit ratio:

<table>
<thead>
<tr>
<th>Fetch unit</th>
<th>Hit ratio</th>
<th>Fetch unit</th>
<th>Hit Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.372</td>
<td>5</td>
<td>0.600</td>
</tr>
<tr>
<td>2</td>
<td>0.515</td>
<td>10</td>
<td>0.620</td>
</tr>
<tr>
<td>3</td>
<td>0.567</td>
<td>20</td>
<td>0.629</td>
</tr>
<tr>
<td>4</td>
<td>0.586</td>
<td>32</td>
<td>0.629</td>
</tr>
</tbody>
</table>

Results - LRU

Optimizing the fetch unit doubles the hit ratios of small caches, and improves by 50% those of large caches.

An LRU of size $s$ and fetch unit 3 is always better than an LRU of size $4s$ and fetch unit 1.

The optimal fetch unit grows as the cache grows larger.
Results – SLRU

- The behavior with respect to the fetch unit is consistent with that of LRU.

- The optimal ratio between the size of the two LRU segments:
  - The protected segment grows as the cache size grows (to about 40% of the cache size)
  - Affects the hit ratios of small caches by about 10%
  - Hardly matters for large caches

Results – PDC

PDC, 5-Minute Window, 64000 Pages, Fetch Unit=5

- PQ=28800
- PQ=25600
- PQ=22400
- PQ=19200

Hit Ratio vs. Probationary SLRU segment
Results – Scheme Comparison

Comparing All Policies

Conclusions – LRU, SLRU, PDC

- The fetch unit is the most important optimization parameter:
  1. Doubles the hit ratio of small caches
  2. Increases the hit ratios of large caches by 50%
  3. Optimal fetch unit grows with size of the cache

- Large caches:
  1. Respect the past – large protected segment in SLRU, large query window in PDC
  2. Plan for the future – large fetch units
SDC – Static Dynamic Cache

- SDC divides its cache into two areas:
  1. A read-only (static) cache of results for the perpetually popular queries
  2. A dynamic cache for the results of the rest of the queries, using any replacement policy
- Using a static cache for the results of queries that remain popular over time has the following advantages:
  1. The results of these queries are not subject to eviction in the rare case that the query stream exhibits some “dry spell” with respect to them.
  2. By using ROM, multiple query threads can access the cache simultaneously without the need to synchronize access or lock portions of the memory, increasing its throughput

SDC – Prefetching Policy

- Instead of working with a fixed fetch unit, SDC proposes the following policy:
  - Whenever a query \((t,1)\) results in a cache miss, compute \((t,1)\) and prefetch \((t,2)\)
  - Whenever a query \((t,k>1)\) results in a cache miss, compute \((t,k)\) and prefetch \(m\) additional pages
  - Whenever a query \((t,k>1)\) results in a cache hit, return it but still prefetch \(m\) additional pages
    - This introduces speculative query executions
  - This more elaborate prefetching policy was shown to outperform the fixed fetch-unit policy
Complexities of Caching Schemes

Memory for bookkeeping:
1. Negligible for LRU, SLRU
2. PDC: $O(\text{capacity of query window})$ - several bytes per query in window
3. SDC – none for static portion; dynamic portion depends on specific policy used

Complexity of cache operation:
1. LRU, SLRU: $O(1)$.
2. PDC: amortized complexity of $O(\log(\text{capacity of priority queue}))$
3. SDC – none for static portion; dynamic portion depends on specific policy used

AC – Admission-based Caching

- Like PDC and SDC, also divides its cache into two parts:
  1. A controlled portion for queries that are perceived as having high chance of repeat
     - Entry into this exclusive portion is governed by an admission policy
  2. A non-controlled portion for the rest of the queries
     - Intuition: identify the large fraction of “tail” queries and relegate them to the non-controlled area
     - Prevent them from evicting more worthy results
     - Both cache portions are dynamic and can use arbitrary eviction policies
AC – Admission Functions

Admission functions determine, per query, to which portion it belongs

- Stateless functions: depend only on query characteristics (e.g. length, use of syntax)
- Stateful functions consider also usage statistics
  - Not necessarily at the entire query level – perhaps at more atomic units like individual terms
- Stateful policies achieve higher hit rates, but are also more costly in terms of both memory and computation time
- Experiments showed that even simple admission policies can result in higher hit rates than LRU, PDC and SDC

Richer Prefetching Models due to Richer Result Pages
Additional Notes on Caching

- When many “head” queries result in cache hits, the workload per cache miss experienced by the backend changes
  - E.g. cache miss queries are longer, with more use of syntax
- What about caching in situations with decreasing locality of reference:
  - Engines that personalize search results
  - Engines with incremental or rapidly changing indices, where cached results may quickly become stale?
    - Blanco et al., SIGIR 2010: Cache Invalidation Predictors
- Can use low query load times to proactively refresh cache [Cambazoglu et al. WWW’2010]

Additional Notes on Caching

- Should caching of results be done only at the QI (query integrator) level, or throughout the distributed index?
  - Perhaps the RAM of individual index segments is better utilized by caching postings lists?
- “Smart” caching of data at all levels of the search engine is crucial for decreasing cost-per-search