Compromised Search Engine Caching
Overview

• Intro to Web Cache
• Engineering Web Cache Consistency
• Design Trade-offs for search Engine caching
• Similarity Search
• Predictive caching and prefetching in search engines
• Utilizing navigational Queries for result representation in search engines
Web

1. Web

- Web, as the name goes, is a large network of interconnected systems.

- Tim Berners-Lee is the founder of web, and the rest is history and also the future; known to everyone.

**Let's go with some interesting facts:**

- **Before there were web search engines there was a complete list of all webservers. The list was edited by Tim Berners-Lee and hosted on the CERN webserver.**

- **The prime reason the Google home page is so bare is due to the fact that the founders didn’t know HTML and just wanted a quick interface.**

- **One of the biggest leap in search usage came about when they introduced their much improved spell checker giving birth to the “Did you mean...” feature... It almost doubled!!**

- **Quote: Integrate Sensibly**
2. Caching

- "the process of storing frequently used data in a medium that is faster to access than accessing it from the original medium" [6]

Move the data “close” to those that need it
Reduces latencies
Reduces network traffic
Can also reduce sever load

Perfect Cache:

Cache Size = Collection Size !!!
We need a cache, to cache the cache!!
3. Search Engines and Caching

- To reduce the load on busy web servers web server accelerators are frequently used.

- The inefficiency of the cache consistency protocols in the current HTTP prevents this potential from being fully realized.

- Client polling is used, in which clients query servers to update their cached objects.

- There is no rigid bonding between the search engines and their caches.
4. Engineering Web Cache Consistency

• A Simulated study was done to evaluate the web cache consistency: The goals are
  (i) to understand the interaction of server-driven consistency with a class of workloads; and
  (ii) to provide a baseline for more detailed evaluations

• Consistency Problem:
  There are two conditions under which the system has to contact the server to satisfy a read.

  *First, when the requested object is not cached, We call this a cache miss.*

  *Second, when even if the requested object is cached locally, the consistency protocol may need to contact the server to determine whether the cached copy is valid. We call this a consistency miss.*

  Thus Cache Inconsistency increases the read latency.
4.1. Consistency with dynamic workloads

- Because of the inconsistency in dynamic data, a large portion of dynamic cached data is unused
- The dynamic objects in a server accounted to 60.8% of all objects
- Requests to dynamic objects account for 12% of all requests

| Table I. Classifying Objects and Requests in the IBM Trace According to URL Types |
|---------------------------------|-----------------|-----------------|
|                                 | Object          | Request         |
|                                 | Number  | Percent | Number  | Percent |
| image                           | 9027    | 32.4    | 6165803 | 70.7    |
| non-image                       | 18857   | 67.6    | 2553543 | 29.3    |
| dynamic                         | 16960   | 60.8    | 1044712 | 12.0    |
| other non-image                 | 1897    | 6.8     | 1508831 | 17.3    |
| total                           | 27884   | 100     | 8719346 | 100     |

| Table II. Classifying Objects and Requests in the E-Commerce Trace According to URL Types |
|---------------------------------|-----------------|-----------------|
|                                 | Object          | Request         |
|                                 | Number  | Percent | Number  | Percent |
| image                           | 9255    | 13.3    | 8004528 | 84.2    |
| non-image                       | 60353   | 86.7    | 1500425 | 15.8    |
| dynamic                         | 59083   | 84.9    | 606073  | 6.4     |
| other non-image                 | 1270    | 1.8     | 894352  | 9.4     |
| total                           | 69608   | 100     | 9504953 | 100     |
4.2 Dynamicity is bad for caching!

- The dynamic and other nonimage data are bad for two reasons:

  First, few current systems allow dynamically generated content to be cached. Several studies have suggested that uncachable data significantly limits achievable cache performance [Wolman et al. 1999a; 1999b]

  Second, the cache behavior of these subsets of data may disproportionately affect end-user response time. This is because dynamically generated pages and nonimage objects may form the bottleneck in response time, since they must often be fetched before images may be fetched

- The search engines have to overcome these two hurdles to achieve the best possible cache mechanism, which couldn't be achieved till date. The solutions for the above problems would let the search engine efficiently respond to all user requests, by actually letting the search engine learn which dynamic objects to cache and which not to.
5. Design Trade-Offs for Search Engine Caching

• Some Design choices:

1) Caching query results in lower hit ratios compared with caching of posting lists for query terms, it is faster because there is no need for query evaluation.

2) A mixed policy of combining static and dynamic cache for the problem of caching posting lists, performs better than either static or dynamic caching alone

3) Static caching of terms in a search engine can be more effective than dynamic caching
5.1. Caching Query vs Caching Postings List

- Out of all queries in the stream composing the query log, the upper threshold on hit ratio is 56%. This is because only 56% of all the queries comprise queries that have multiple occurrences.

- A compulsory miss occurs when the cache receives a query for the first time. A capacity misses, occurs due to space constraints on the amount of memory the cache uses.

- Hence compulsory miss happens with the use of both query and postings-list caching. But a capacity miss occurs only with the query caching because of its exhaustive capacity.
5.2. Static vs Dynamic Caching in Search Engines

![Graph showing hit rate vs cache size for different caching strategies.

- Static QTF/DF
- LRU
- LFU
- Dyn-QTF/DF
- QTF

The graph illustrates the performance of different caching strategies in terms of hit rate as a function of cache size for a UK dataset.]
6. A Metric Cache for Similarity Search

- Content-Based Image Retrieval (CBIR) techniques are growing rapidly, but their retrieval is still done with methods that exploit only the text and tags.

- CBIR process is inherently expensive and the query processing costs grows with the size of the collection.

- A cache in front of the CBIR can be used to tackle the scalability issues and by reducing the average cost of query resolution.

- Reduce the distance between the query object and collection of objects. Similarity Search can be seen as a process of retrieving the most relevant objects.
What is a metric cache?

- A Metric Cache is a cache that assumes that the Queries may be approximate.

- The cache stores result-objects, not only result-pointers.  
  *e.g.*: *documents vs. documents ids*

- The cache is a peculiar sample of the whole dataset of the set of objects most recently seen by the users (= most interesting !?!)  

- **Claim:**  
  An interesting object may be used to answer approximately if it is sufficiently similar to the query.
Similarity queries & Top-K results

Front-end of the CBIR System

Cache $C$

Similarity queries not solved by the cache

Parallel & Distributed CBIR System

Index of MM objects

Unit 1

Index of MM objects

Unit 2

Index of MM objects

Unit $n$
6.1. Approximate Search in Metric Space

- Approximate similarity search has improved efficiency at the cost of precision.

- This can be done in one of two ways:
  - Transformations of the metric space
  - Reduce subset of data to be examined

- In the first approach, the object representation is changed to reduce the query distance, thus stopping the search before the precise end.

- In the second approach, the search avoids the data regions that are not likely to contain close objects by either early termination or relaxed branching.
6.2. Caching CBIR Query Results

- To compute the similarity between the cache content and the submitted query, the cache stores not only the identifiers of the objects returned but also the features associated with them.

- The object features are used to measure the metric distances.

- This approach will increase the complexity and makes it more expensive computationally.

- There are two algorithms to approach this computation: 
  - $R_{cache}$
  - $Q_{Cache}$
6.2.1. RCACHE

- It uses hash table to store and retrieve queries and results lists efficiently.

- When a cache is looked-up for $k$ objects that are most similar to a given query $q$, $H$ is first accessed to check for an exact hit.

- If it doesn't get an exact hit, the cache metric index $M$ is accessed to find the closest stored object in the cache.

- It also adopts Metric index $M$ to perform $k$NN searches over the cached objects to return approximate results when possible.

- A $k$NN query returns the $k$ nearest objects to query object $q$, denoted with $kNND(q, k)$. 
6.2.2 QCACHE

- Unlike the Rcache, the Qcache does not index every single object in the result sets of the cached queries.

- Qcache builds metric index M only over the query objects.

- The idea is to search a set of suitable queries among the cached ones and use their neighbors to produce an approximate answer.

- In this algorithm the results are stored independently, thus leading to some redundancy.

- Cost of indexed queries in Qcache= $O(Sz/K)$, whereas Cost of indexed queries in Rcache= $O(Sz)$. 
6.2.3 Rcache vs QCache

- Considering the hit rate, approximate hit rate and the miss rate, it is seen that the Rcache slightly outperforms Qcache.

- It is because Rcache can store more objects than Qcache due to the absence of redundant objects.

- But this difference is still quite small and increases with the increase in redundant objects.
7. Predictive Caching and Prefetching of Query Results in search engines

• Prefetching improves the cache performance usually and when tested on real-life queries in Altavista, prefetching out performed normal query response

• A Probability Driven Cache (PDC) is based on probabilistic model of search engine user's browsing sessions and PDC consistently out performed the LRU and Segemented LRU replacement policies and attained hit ratios exceeding 53% in large caches
8. Utilization of Navigational Queries for result presentation and caching in search engines

- A Navigational query is intended to find a particular web site that a user has in mind

- However, web search engines usually cache and return the results of the query in sets of ten in each page

- For Navigational queries, it is waste of cache space and network bandwidth
8.1. Scheme

- The experimental results showed that for some queries that are identified as navigational in the training log and have all the clicks for the top2 results

- Hence the scheme would never cache the second page, whatever the size of the cache may be

- Or a confidence factor can be included into the scheme, where the second page will have low confidence rate and stored in the cache. But retrieved by a request only when the confidence rate is above a certain value. Hence reducing the bandwidth problems
Conclusion

I believe there are many compromises that every search has to make to balance all the features. But it is the way you make the compromise and what you compromise is what matters in the current search engine world. By far, Google made the right choices in most of its features, might not be all.

Mastery in one feature like Pre-fetching might lead to unnecessary use of bandwidth. Also approximation might not satisfy all users, and all users might not be looking for an exact match either. 70% of the time I search Google I type the characters out of order.

Yeah!! Integrate Sensibly
Best questions first please!!!