Web Caching

based on:

- **Web Caching**, Geoff Huston
- **Web Caching and Zipf-like Distributions: Evidence and Implications**, L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker
- **On the scale and performance of cooperative Web proxy caching**, A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, H. Levy

Hypertext Transfer Protocol (HTTP)

based on an end-to-end model (the network is a passive element)

Pros:

- The server can modify the content;
- The server can track all content requests;
- The server can differentiate between different clients;
- The server can authenticate the client.

Cons:

- Popular servers are under continuous high stress
- high number of simultaneously opened connections
- high load on the surrounding network
- The network may not be efficiently utilized

Why Caching in the Network?

- ✔ reduce latency by avoiding slow links between client and origin server
  - ✗ low bandwidth links
  - ✗ congested links
- ✔ reduce traffic on links
- ✔ spread load of overloaded origin servers to caches

Caching points

- **content provider (server caching)**

- **ISP**
  - more than 70% of ISP traffic is Web based;
  - repeated requests at about 50% of the traffic;
  - ISPs interested in reducing the transmission costs;
  - provide high quality of service to the end users.

- **end client**

Caching Challenges

- ✔ Cache consistency
  - ✗ identify if an object is stale or fresh
- ✔ Dynamic content
  - ✗ caches should not store outputs of CGI scripts (Active Cache?)
- ✔ Hit counts and personalization
- ✔ Privacy concerned user
  - ✗ how do you get a user to point to a cache
- ✔ Access control
  - ✗ legal and security restrictions
- ✔ Large multimedia files

Web cache access pattern - (common thoughts)

- the distribution of page requests follows a Zipf-like distribution: the relative probability of a request for the i-th most popular page is proportional to $1/i^{\alpha}$ with $\alpha \leq 1$

- for an infinite sized cache, the hit-ratio grows in a log-like fashion as a function of the client population and of the number of requests
- the hit-ratio of a web cache grows in a log-like fashion as a function of cache size
- the probability that a document will be referenced k requests after it was last referenced is roughly proportional to $1/k$ (temporal locality)

Web cache access pattern - (P. Cao Observations)

- distribution of web requests follow a a Zipf-like distribution $\alpha \in [0.64, 0.83]$ (fig. 1)
- the 10/90 rule does not apply to web accesses (70% of accesses to about 25%-40% of the documents) (fig. 2)
- low statistical corellation between the frequency that a document is accessed and its size (fig. 3)
- low statistic correlation between a document's access frequency and its average modifications per request.
- request distribution to servers does not follow a Zipf law.; there is no single server contributing to the most popular pages (fig. 5)

Implications of Zipf-like behavior

- **Model:**
  - A cache that receives a stream of requests
  - N = total number of pages in the universe
  - $P_N(i)$ = probability that given the arrival of a page request, it is made for page i, $(i = \overline{1, N})$
  - Pages ranked in order of their popularity, page i is the i-th most popular
  - Zipf-like distribution for the arrivals:

$$P_N(i) = \Omega / i^{\alpha},$$

$$\Omega = \left(\sum_{i=1}^{N} 1/i^{\alpha} \right)^{-1}$$

- no pages are invalidated in the cache.

Implications of Zipf-like behavior

infinite cache, finite request stream:

$$H(R)\Big|_{=\ (\Omega/(1-\alpha))(R\Omega)^{1/\alpha-1},\ (0<\alpha<1)}^{=\ \Omega\log(\Omega R),\ (\alpha\ =\ 1)}$$

finite cache, infinite request stream:

$$H(C)\Big|_{=\ C^{1-\alpha},\ 0<\alpha<1}^{=\ \log C,\ (\alpha\ =\ 1)}$$

page request interarrival time:

$$d(k) \approx \frac{1}{k\log N}\left(\left(1-\frac{1}{N\log N}\right)^k - \left(1-\frac{1}{\log N}\right)^k\right)$$

H(R) = hit ratio

d(k) = probability distribution that the next request for page iis followed by k-1 requests for pages other than i

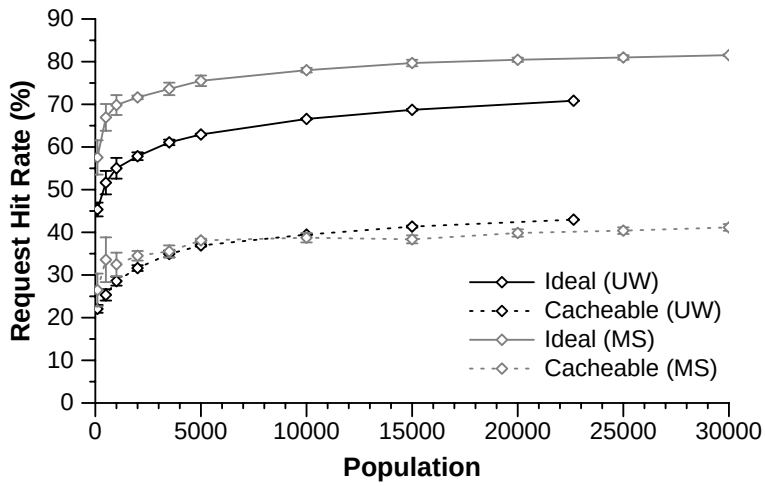
Cache Replacement Algorithms (fig. 9)

- GD-Size:
  - performs best for small sizes of cache.
- Perfect-LFU
  - performs comparably with GD-Size in hit-ratio and much better in byte hit-ratio for large cache sizes.
  - drawback: increased complexity
- In-Cache-LFU
  - performs the worst

Web document sharing and proxy caching

- What is the best performance one could achieve with “perfect” cooperative caching?
- For what range of client populations can cooperative caching work effectively?
- Does the way in which clients are assigned to caches matter?
- What cache hit rates are necessary to achieve worthwhile decreases in document access latency?

For what range of client populations can cooperative caching work effectively?

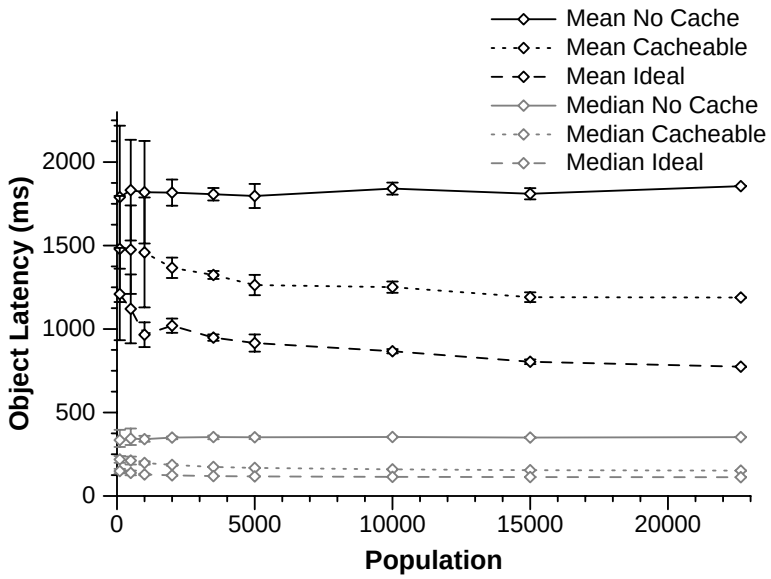


- for smaller populations, hit rate increases rapidly with population (cooperative caching can be used effectively)
- for larger population cooperative caching is unlikely to bring benefits

Conclusion:

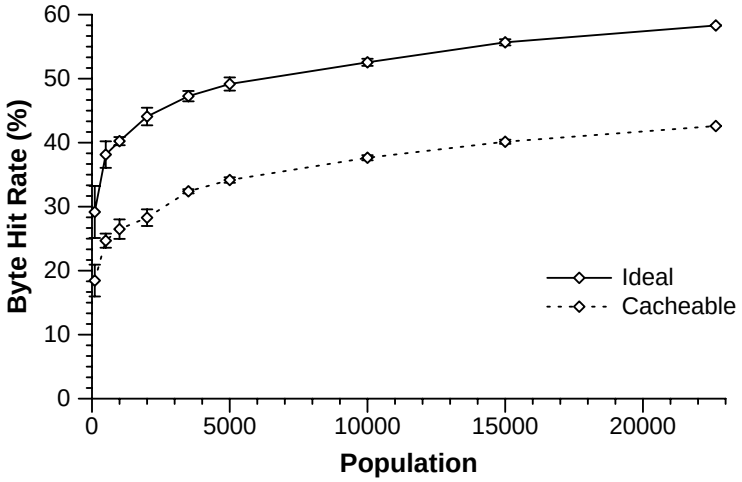
- use cooperative caching to adapt to proxy assignments made for political or geographical reasons.

Object Latency vs. Population



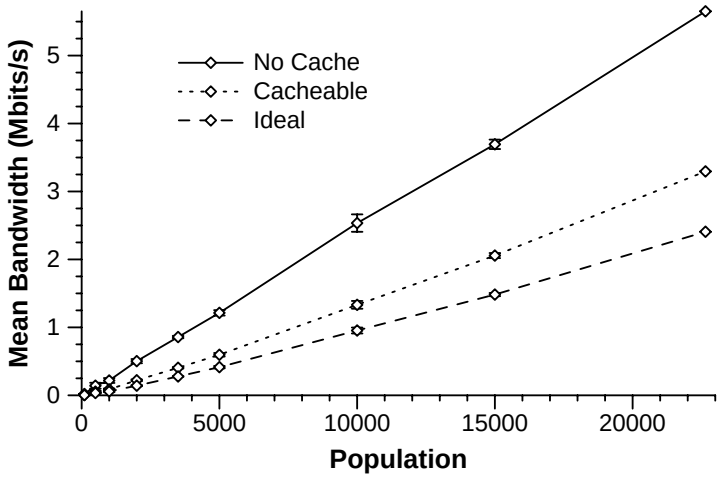
- Latency stays unchanged when population increases
- Caching will have little effect on mean and median latency beyond very small client population (?!)

Byte Hit Rate vs. Population



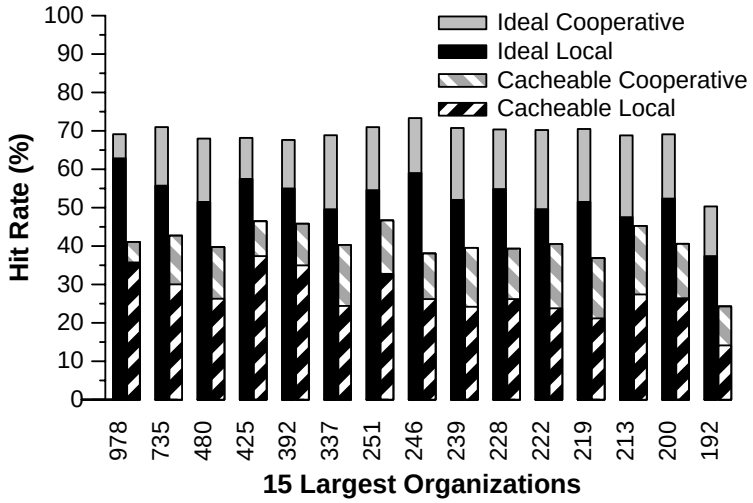
- same knee at about 2500 clients
- shared documents are smaller

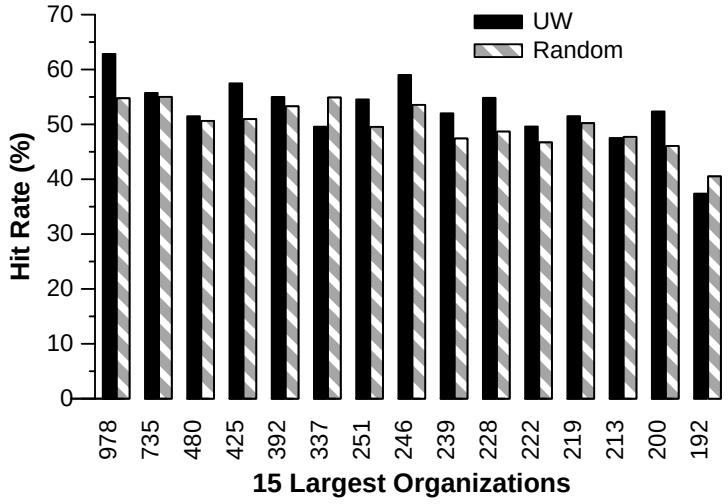
Bandwith vs. population



- while caching reduces bandwidth consumption there is no benefit to increased client population.

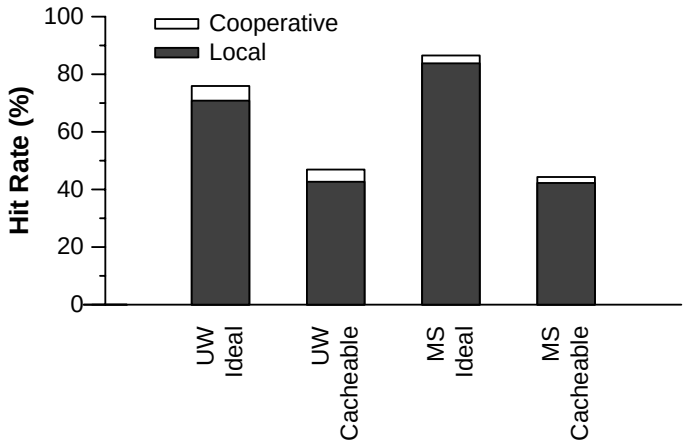
Proxies and organizations





- there is some locality in organizational membership, but the impact is not significant

Impact of larger population size

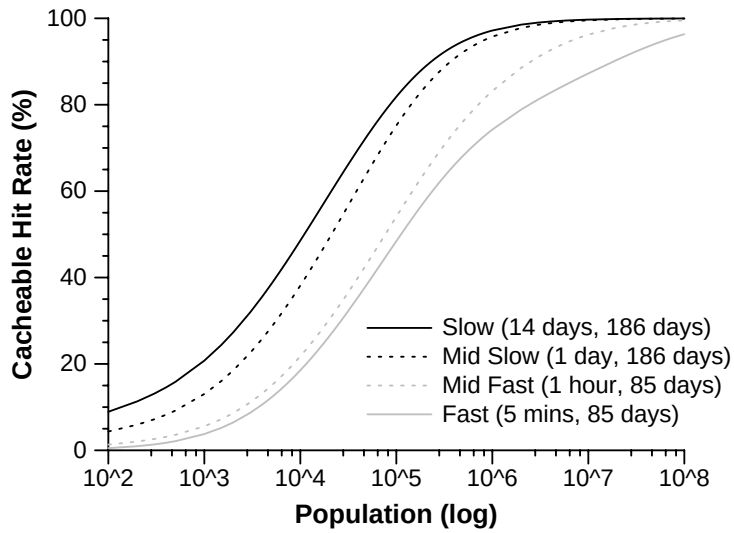


- unpopular documents are universally unpopular => unlikely that a miss in one large group of population to get a hit in another group.

Performance of large scale proxy caching - model

- N = clients in population
- n = total number of documents ((aprox. 3.2 billions)
- pi = fraction of all requests that are for the i-th most popular document ($p_i \approx \frac{1}{i^{\alpha}}$, $\alpha = 0.8$)
- exponential distribution of the requests done by client with parameter λN, where λ = 590 reqs/day = average client request rate
- exponential distribution of time for document change, with parameter μ (unpopular document $\mu_u = 1/186$ days, popular documents $\mu_p = 1/14$ days)
- p_c = 0.6 = probability that the documents are cacheable
- E(S) = 7.7KB = avg. document size
- E(L) = 1.9 sec = last-byte latency to server

Hit rate, latency and bandwidth



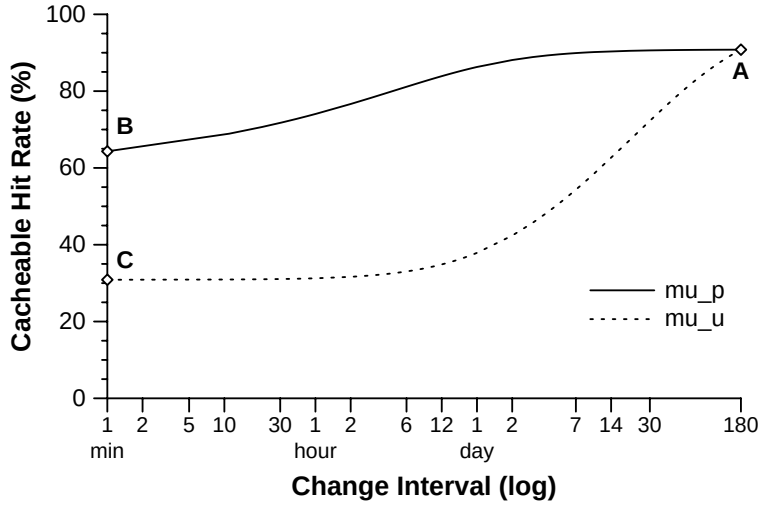
Hit rate = f(Population) has three areas:

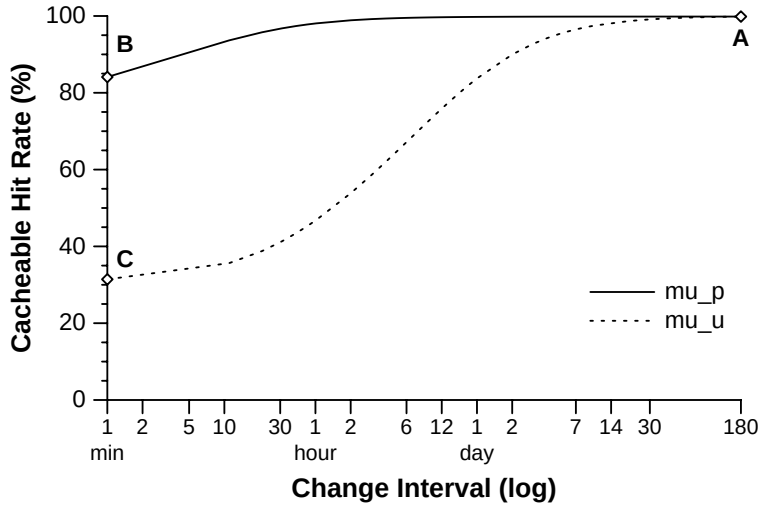
- ✔ 1. the request rate too low to dominate the rate of change for unpopular documents
- ✔ 2. marks a significant increase in the hit rate of the unpopular documents
- ✔ 3. the request rate is high enough to cope with the document modifications

Latency $lat_{req} = (1 - H_N) \cdot E(L) + H_N \cdot lat_{hit}$

Bandwidth $B_N = H_N \cdot E(S)$

Document rate of change





Proxy cache hit is very sensitive to the change rates of both popular and unpopular documents with a decrease on the time scales at which hit rate is sensitive once the size of the population increases.

Comparing cooperative caching scheme

Three types of cache schemes: hierarchical, hash based and summary caches;

Conclusions:

- the achievable benefits in terms of latency are achieved at the level of city-level cooperative caching;
- a flat cooperative scheme is no longer effective if the served area increases after a limit
- in a hierarchical caching scheme the higher level might be a bottleneck in serving the requests.

Hierarchical Caching (Squid):

- a hierarchy of k levels
- a client request is forwarded up in the hierarchy until a cache hit occurs.
- a copy of the requested object is then stored in all caches of the request path

Hash-based caching system:

- assumes a total of m caches cumulatively serving a population of size N
- upon a request for an URL the client determines the cache in charge and forwards the request to it.
- if the cache has the URL the page is returned to the client, otherwise the request is forwarded to the server

Directory based system (Summary cache)

- a total of m caches serving cumulatively a population of size N
- the population is partitioned into subpopulations of size N/m and each population is served by a single cache
- each cache maintains a directory that summarizes the documents stored at each of the other cache in the system
- when a client request can not be served by a local cache, it is randomly forwarded to a cache which is registered to have the document. If no document has the document the request is forwarded to the original server