

Adaptive Web Cache: *Theory and Practice*

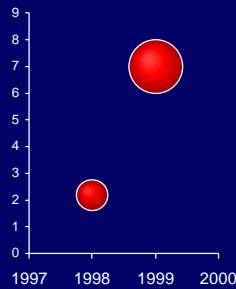
Yu Hen Hu
Dept. Electrical and Computer Engineering
University of Wisconsin-Madison
Madison, WI 53706
hu@engr.wisc.edu

Portion of this presentation is based on the Ph.D. dissertation by Dr. Annie Foong

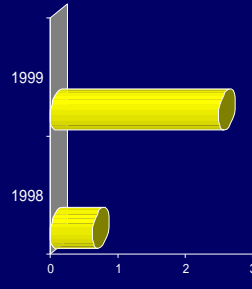
Outline

- Web Cache Problem
 - Distributed storage allocation
 - Intelligent proxy cache admission/prediction
 - Active serving of dynamic contents
- Cache admission Problem
- Simulation Results

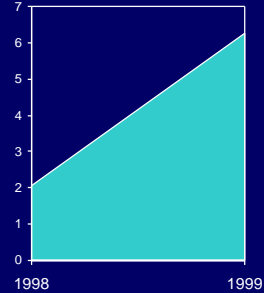
Growing Web From 1998 to 1999



Number of Web sites
grew from 2.2 M to
7.0 M



MCI backbone speed
up from 622 Mbps
(OC-12) to 2.5 Gbps
(OC-48)

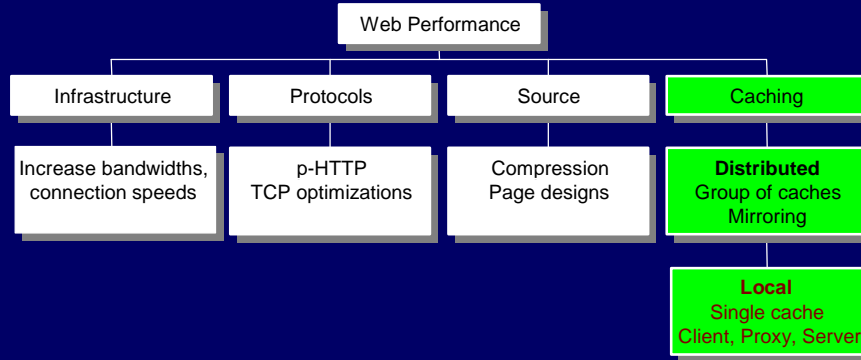


Average download
delay increases
from 2.05s to 6.26s!

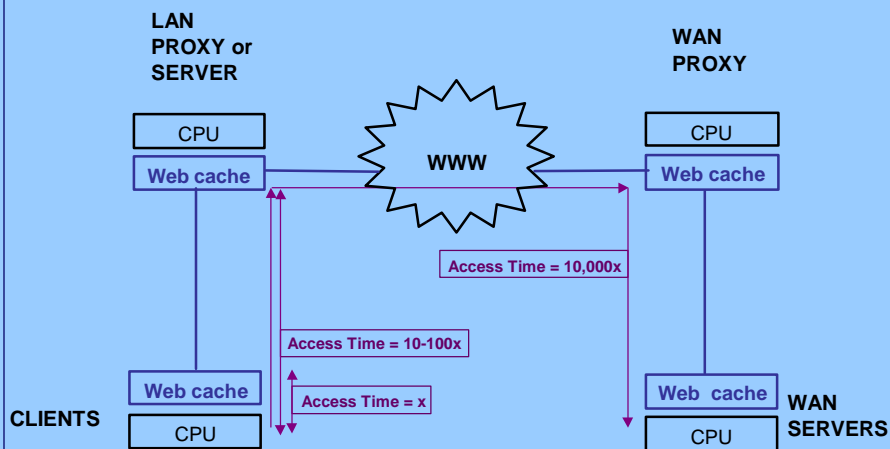
World Wide Wait

- Long delay (latency) due to
 - Increase in size and number of objects, especially multimedia data streams
 - Uneven bandwidth: from fiber optic links to 56k modem
- Human Computer Interface Issue
 - Users who are used to faster response of new PCs grow impatient to slower web delay.
 - Perceived delay sometimes more important!

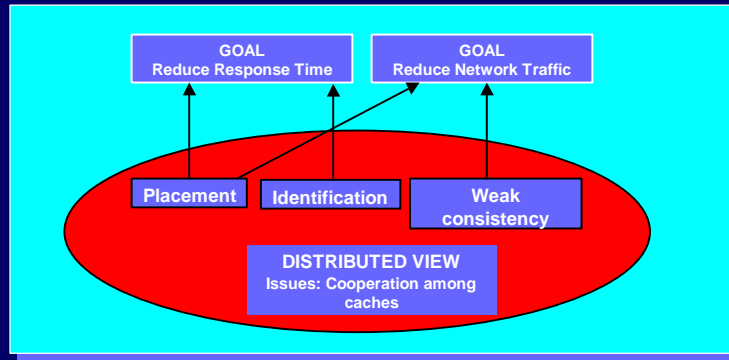
Factors Affecting Web Speed



Web Cache Architecture



Distributive Caching



- Effective Placement & Identification: Harvest, Squid [Chunkhunhood96], Summary cache [Fan98]
- Weak consistency: TTL [HTTP/1.1], manual setting

Http Cache Support

- HTTP/1.0 request:
 - request method, e.g. http://
 - URL
 - set of request headers, e.g. GET, POST
- Conditional GET has a *If-Modified-Since* header. Server can reply "Not Modified"
- HTTP/1.1 *cache control* header allows both client and server to override default caching algorithms
- MAX-AGE directive allow client to specify the *age* of an object in cache before it needs be re-fetched from server.

ICP: Internet Cache Protocol

- To facilitate inter-cache communication
- To support complex cache hierarchies
- Light weight process containing a header and a payload (URL)
- Does NOT match HTTP
- Increases request latency
- Implemented on Harvest, Squid

OPcode	Version	Packet Length
Request Number		
Options		
Padding		
Sender Host Address		

ICP header

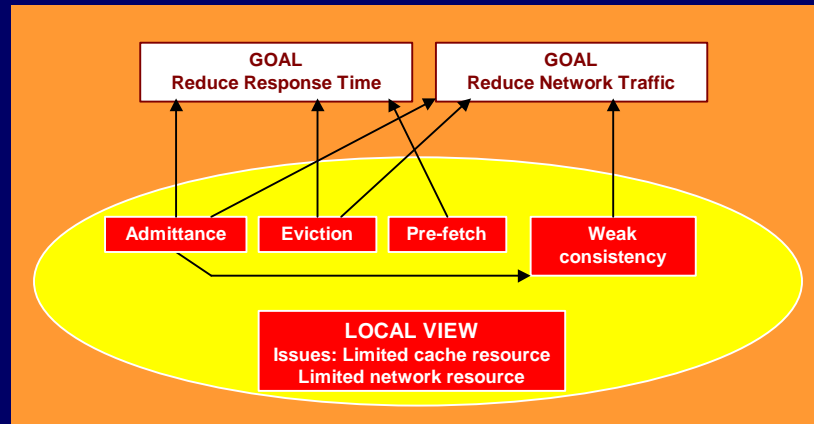
[Http://ds.internic.net/rfc/rfc2186.txt](http://ds.internic.net/rfc/rfc2186.txt)

Related Works

- Dynamic Server Locating
 - SONAR: simple message format expressing proximity(?) of a set of server addresses
 - HOPS
 - Adaptive web caching (Floyd et al): use multicast to disseminate objects among proxy cache groups.
- Replication (Mirroring): currently are done manually.
- Push Caching: replicate popular data in advance based on geography

www.netlib.org/utk/projects/sonar
www.ingrid.org/hops/wp.html

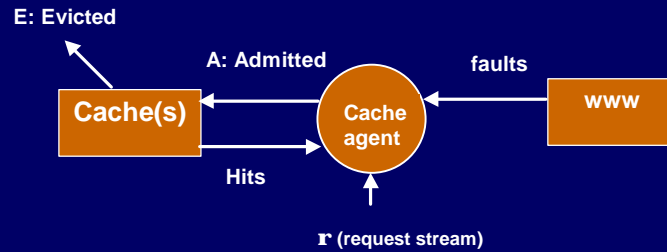
Local Caching



Local Caching Focus on ...

- Caching strategy within a single cache site.
- Cache maintenance issues:
 - Object admission
 - Object eviction
 - Object pre-fetch
- Performance issues:
 - Metrics: delay, quality
- Human factors
 - Predicting user's needs
 - Partial caching
- Dynamic contents
 - Active Proxy Cache
- Multimedia streams
 - SWORD

Cache Maintenance Process



Cache Maintenance

- Object Admission Policy
 - When an object not in cache is fetched, should it be admitted into cache?
 - Statistical hypothesis testing problem.
- Object Eviction Policy
 - When the cache is full, choose one or more object to be evicted.
 - Combinatorial optimization problem
- Object Pre-fetch Policy (Pre-push policy)
 - If an object is likely to be requested, fetch it (from proxy side) or push it (from server side) before the request is made.
 - Statistical stream prediction problem.

Performance Metrics

$$\text{Object hit ratio, } h = \frac{\text{Number of objects served from cache}}{\text{Total number of objects requested}}$$

$$\text{Byte hit ratio, } b = \frac{\text{Number of bytes served from cache}}{\text{Total number of bytes requested}}$$

$$\text{Object turnover ratio, } v = \frac{\text{Number of objects replaced from cache}}{\text{Total number of objects requested}}$$

A General Cost Model

- r_i : i -th object request
- f_k : = 0 if r_k in cache, = 1 if r_k not in cache
- s_k : size of r_k
- c_k : cost of requesting r_k . It may be unity (object cost), proportional to s_k (byte cost), or assume some other assigned values
- COST FUNCTION

$$F_k = \frac{\sum_{i=1}^k f_i c_i}{\sum_{i=1}^k c_i}$$

- If $c_k = 1$, then $F_k = h$
- If $c_k = s_k$, then $F_k = b$

Cache Pre-fetch Problem

- Given the partial object request sequence made to a particular cache agent
 $\rho_k = \{r_1, r_2, \dots, r_k\}$
 predict the next object to be accessed r_{k+1} such that $\Pr.\{f_{k+1} = 1\}$ is minimized.
- Denote the predicted object as r'_{k+1} . If it is in cache, do nothing. Otherwise, *pre-fetch* r'_{k+1} *before the request is made*.
- Denote by $O = \{o_m\}$ as the set of unique object identities in the entire web.
- $p_m(k)$: probability o_m will be accessed at time index k
- Minimize $\Pr.\{f_{k+1} = 1\}$
 \Leftrightarrow Maximize $p_m(k+1)$
- HOWEVER, ...
 - O is only partially known
 - subsequent requests may not be from the same context

Pre-fetch ...

- Pre-fetch requires bandwidth! But it should not interfere with human user's normal web object access.
- Pre-fetch will be useful only when average time between two successive requests made to the same cache agent is large compared to pre-fetch time.
- Stream prediction: Given a collection of object request traces $\{\rho_k\}$. Compute empirical probability
 $\Pr\{r_{N+1} = o_m | \rho_N\}$.
- This is *N-Gram model* used in statistical continuous speech recognition. Usually, $N \leq 5$.
- Difficulty: different traces may not be of the same context.
- 10% improvement observed.

Cache Admission Problem

- Given that the partial object request trace up to time k is $\rho_k = \{r_1, r_2, \dots, r_k\}$ and the current content of cache is C_k . Assume that $r_{k+1} \notin C_k$ and has to be fetched. Decide whether to *admit* r_{k+1} into the proxy cache.
 - Why not admit all?
 - Dynamic object (expires)
 - Will not be requested again.
 - Reduce replacement cost
- Admission criterion: If for some $k', k+1 < k' < k+K_o$, $r_{k'} = r_{k+1}$, that is if r_{k+1} will be *re-accessed* in the future, then it may be worthy of being cached. K_o : a pre-defined window.
- Probability of re-access: $P_a = \Pr\{r_{k'} = r_{k+1} \mid \rho_k, C_k, k+1 < k' < k+K_o\}$
- Many factors contribute to the estimation of P_a .

Cache Eviction Problem

Problem Statement:

Develop a cache admission and eviction policy to minimize

$$F_k = \sum_{i=1}^k f_i c_i / \sum_{i=1}^k c_i$$

subject to cache size constraint:

$$S(k) = \sum_{i=1}^k \mathbf{1}_{r_i \in C_k} s_i \leq S$$

- Ideal Case: Entire (not partial) object request trace $\rho(K) = \{r_1, r_2, \dots, r_K\}$ is available. Re-access needs are known. Assume $K_o = K-1$
- Constraints:
 - do not cache objects that are needed only once
 - objects that will not be re-accessed should be evicted

Life-Span

- Assume: object is not fetched until requested and will be evicted if no longer needed. I.e. cache only store objects that are to be re-accessed.
- Lift span of a request r_k , $L(k)$ is the time duration from k to its immediate next access.
- Eviction of an object corresponding to r_k during $L(k)$ will incur a miss ($f_k = 1$) and increase F by c_k .
- Cache size demand function*:

$$S(k) = \sum_{\substack{i \leq k \\ r_i \text{ will be re-accessed}}} S_i$$
- When $S(k) > S$, a subset of objects in C_k will be evicted.
- The optimal solution (may not be unique) is one that minimizes F while satisfying $S(k) \leq S$.
- Number of possible solutions grow exponentially w.r.t. $|C_k|$.

Cache Maintenance: Reality

- The complete trace is not known. Only partial trace up to time k is available.
 - Life-span may be weighted by the size of the object. Larger objects should be evicted first.
- Life span $L(i)$ is unknown. However, one may try to predict $L(i)$ for each object in Cache.
- Heuristics:
 - Object with larger expected life span has smaller probability of being re-accessed within time window K_o
- Statistical Tool for life-span prediction: current time = k ,

$$\Pr\{L(i) \leq k + K_o\}$$

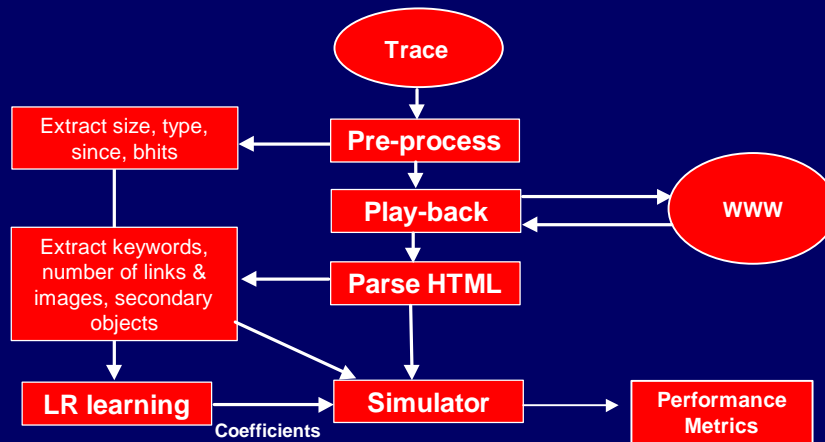
$$= \Pr\{r_{k'} = r_i \mid k < k' < k + K_o\}$$

$$= \text{Probability } r_i \text{ will be re-accessed at least once within the time window } (k, k + K_o].$$

Logit-Regression Model

- Define random variable $Y(i) = 1$ if r_i is to be re-accessed within next K_0 requests; $= 0$ otherwise.
- $P_{LR} - \Pr\{Y(i) = 1 | x_1, \dots, x_M\}$: probability $Y(i) = 1$ given a set of observations (features)
$$P_{LR}(i) = 1 / (1 + \exp(-\sum_{m=1}^M b_m x_m))$$
- Objects with smaller $P_{LR}(i)$ will be evicted when cache is full.
- Feature selection (x_m):
 - time since last access
 - hits within past N' requests
 - size of object
 - type of object
 - # of hyperlinks
 - # of images
 - # of keywords in top 10 list
- Use Maximum Likelihood (ML) method to estimate β .
- Adaptively updating β is possible using on-line learning algorithms.

Simulation Setup

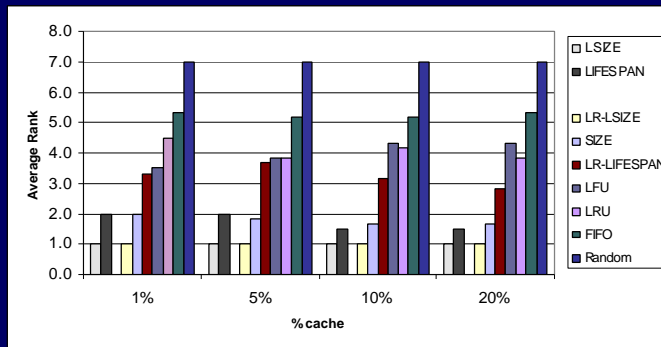


Methods Compared

Policy	Criteria
LSIZE	Evict object with largest weighted-lifetime. Admit only if smaller.
LIFESPAN	Evict object with the largest lifespan. Admit only if smaller
LR-LSIZE	Evict object with the largest predicted weighted-lifespan. Admit only if smaller
LR-LIFESPAN	Evict object with the largest predicted lifespan. Admit only if smaller
LRU	Evict object with the largest time since last access
LFU	Evict object with the smallest number of accesses
FIFO	Evict object with the earliest time of entry to cache
SIZE	Evict object with the largest size.
Random	Randomly chosen

Use entire trace, a priori algorithms

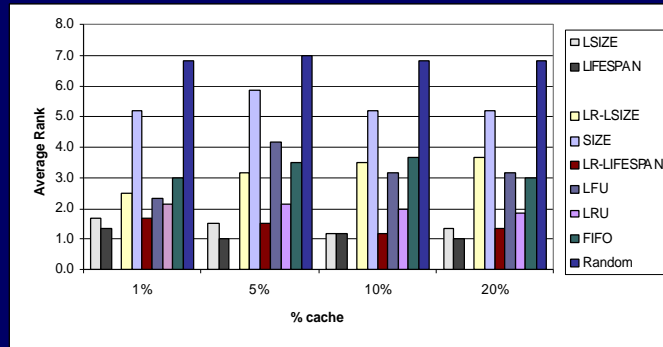
Object Hit Ratio Results



Best a priori algorithm: LSIZE

Best predictive algorithm: LR-LSIZE (SIZE is 2nd best)

Byte Hit Ratio Results



Best a priori algorithm: LIFESPAN

Best predictive algorithm: LR-LIFESPAN (LRU is 2nd best)

Conclusion

- Web cache is not the same as system cache due to different types of objects to be cached
- Two directions:
 - distributive caching and local caching
- We only focus on single proxy cache
- Many open issues remain to be explored for different cache infra-structures.