Latency-sensitive hashing for collaborative Web caching

Presented by:

Xin Qi

Yong Yang

09/04/2002

About Authors:

- Kun-Lung Wu
  - Ph.D in University of Illinois at Urbana-Champaign
  - IBM Thomas J. Watson Research Center and currently manager of the Software Tools and Techniques group.
  - published more than 50 papers in refereed journals and conferences. Holds or has applied for 15 US patents.
- Philip S Yu
  - Ph.D in Stanford University
  - IBM Thomas J. Watson Research Center and currently manager of the Software Tools and Techniques group.
  - Published more than 270 papers in refereed journals and conferences. Holds or has applied for 92 US patents.

Content table

Why Caching?

- Internet grows very quick.
- The problems are network congestion and server overloading.
- User response times for accessing the web have become increasingly unsatisfactory.
- Web caching is needed to reduce network traffic.
- Three ways to cache: Caching at client, Caching at proxy and Caching at Servers.

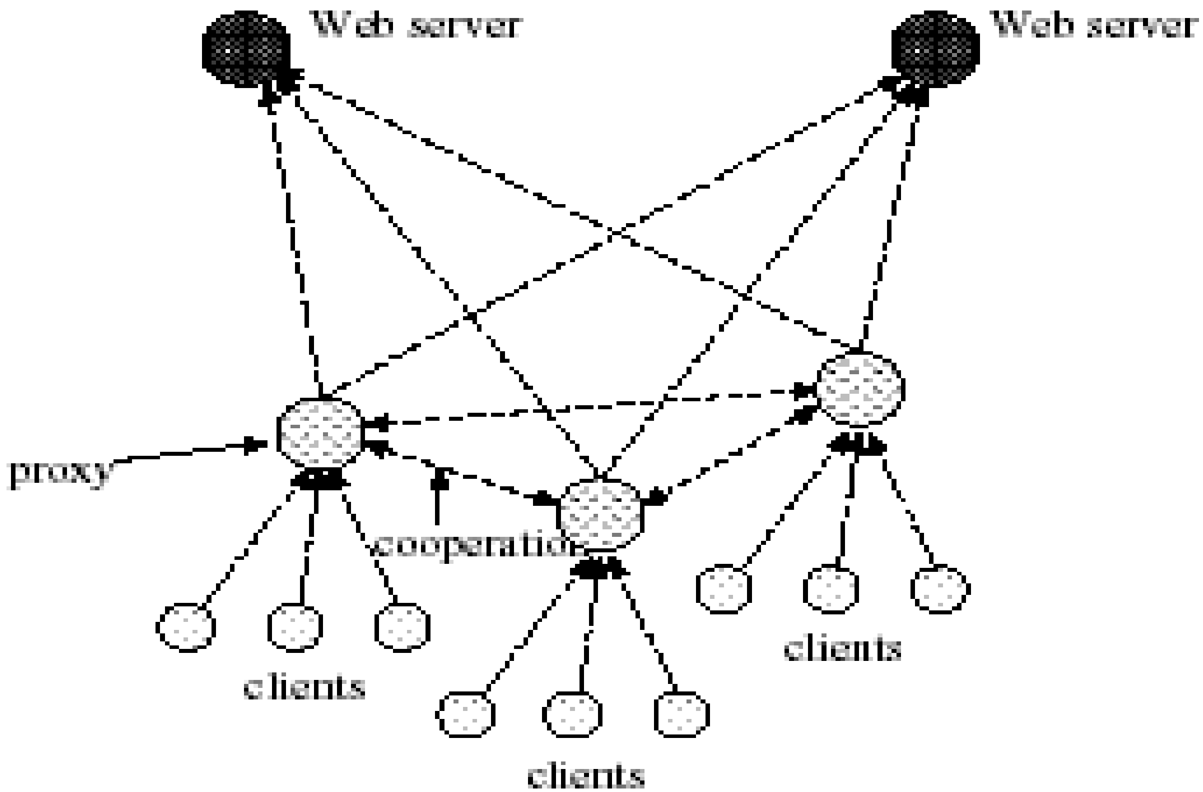
Why Proxy?

- Proxy was firstly used to allow accesses to the internet from users within a firewall.
- Proxy served a previous request and cached document for next one.
- Web caching at proxy server can not only save network bandwidth but also lower access latency for the clients.

Collaborative Web Caching

- A single server is single point of failure
- A single server cache is always a bottleneck
- Multiple proxies are used.

A generic WWW caching system



Geographically distributed proxies

- Response times tend to be negatively impacted for those requests hashed into geographically distant proxies or overloaded proxies.
- Distant proxies tend to incur longer network latency delays
- Overloaded proxies can cause significant delays too.
- Strong need to consider the latency issue in hashing based web caching among geographically distributed proxies.

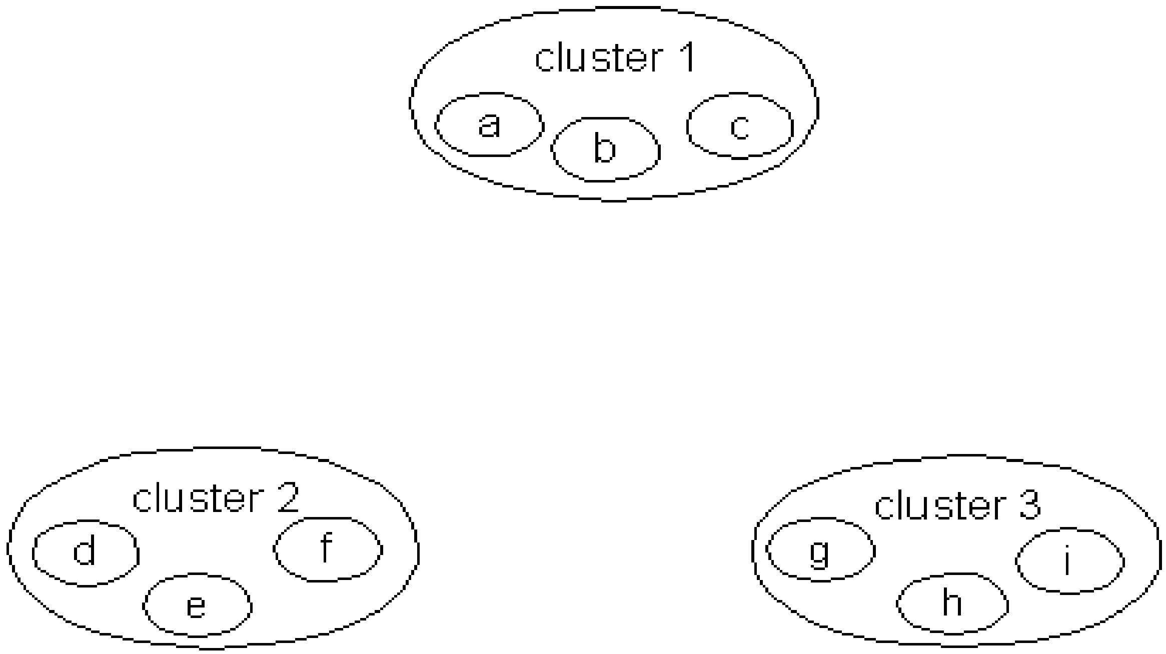
Geographically distributed proxies

- Geographically clustered hashing (GCH)
  - Requests are served only by proxies in a geographically close region.
  - Work well if the proxies within a region can adequately service all the requests originated within the same region.
  - However, proxies in one region may be overloaded while those in another region are under loaded
- Geographically distributed hashing (GDH)
  - Requests are hashed into all cooperating proxy caches regardless of geographical location.
  - Load tends to be more balanced among all the geographically distributed cooperating caches compared with GCH
  - However, GDH did not take into account network latency delays due to geographical distances.

Geographically distributed proxies (cont'd)

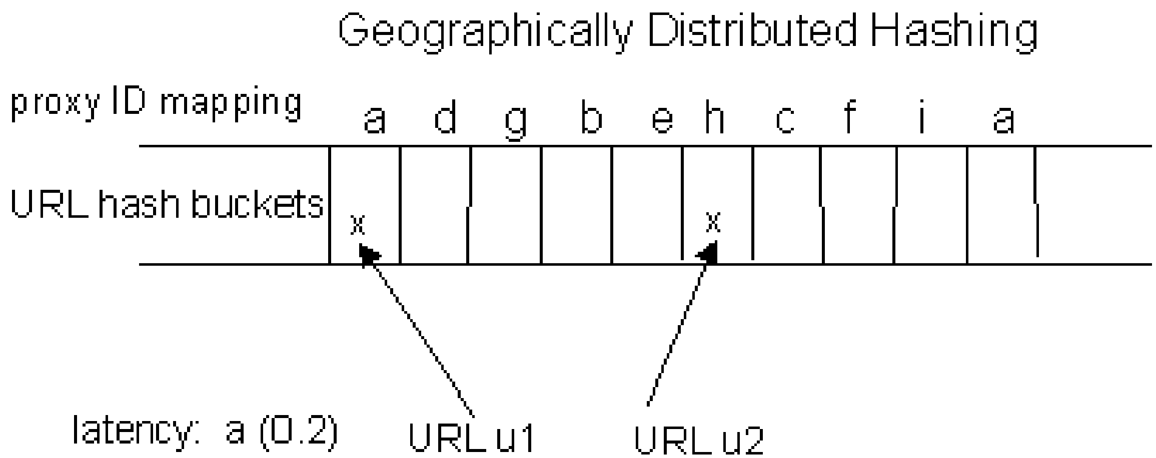
- Latency-sensitive hashing (LSH)
  - It hashes requests into all proxies
  - It counts latency delays and potential overloaded proxies.
  - Firstly, a request is hashed into an anchor hash bucket. Each hash bucket is mapped to one of the geographically distributed proxies.
  - Secondly, a selection algorithm is used to pick a proxy among a small number of hash buckets adjacent to the anchor hash bucket.
  - The selection is based on objective to reduce network latency and to avoid creating over-loaded proxies.

An example



Three Geographically distributed clusters of proxies

An example of a geographically distributed Hashing



latency:
a (0.2)
b (0.2)
c (0.2)
d (1.0)
e (1.0)
f (1.0)
g (1.0)
h (1.0)
i (1.0)

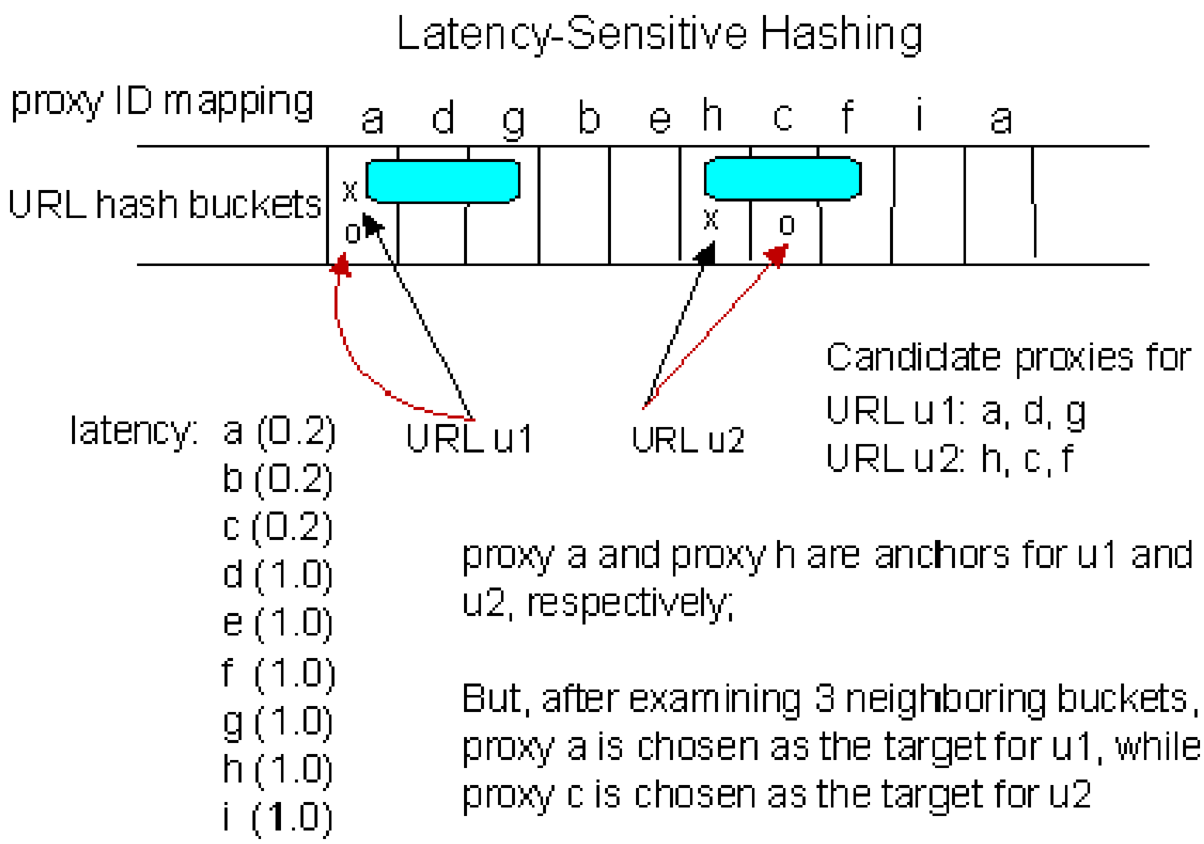
u1 and u2 originate from cluster 1
URL u1 is hashed into proxy a
URL u2 is hashed into proxy h

Big difference in latency delays for URL u1 and URL u2

An example of a geographically distributed Hashing(cont'd)

- This example shows the potential problem of hashing requests into more and more geographically distributed proxies.
- The network latency can be a problem for those that are hashed into geographically distant proxies.

An example of latency-sensitive hashing



An example of latency-sensitive hashing (cont'd)

- Compared with GDH, the proxy with lowest latency will be chosen.
- Mapping of hash buckets to proxies and the selection of window size are important to its performance.
- It is not obvious to do so if requests are evenly distributed to all proxies when there are different numbers of proxies within a cluster.

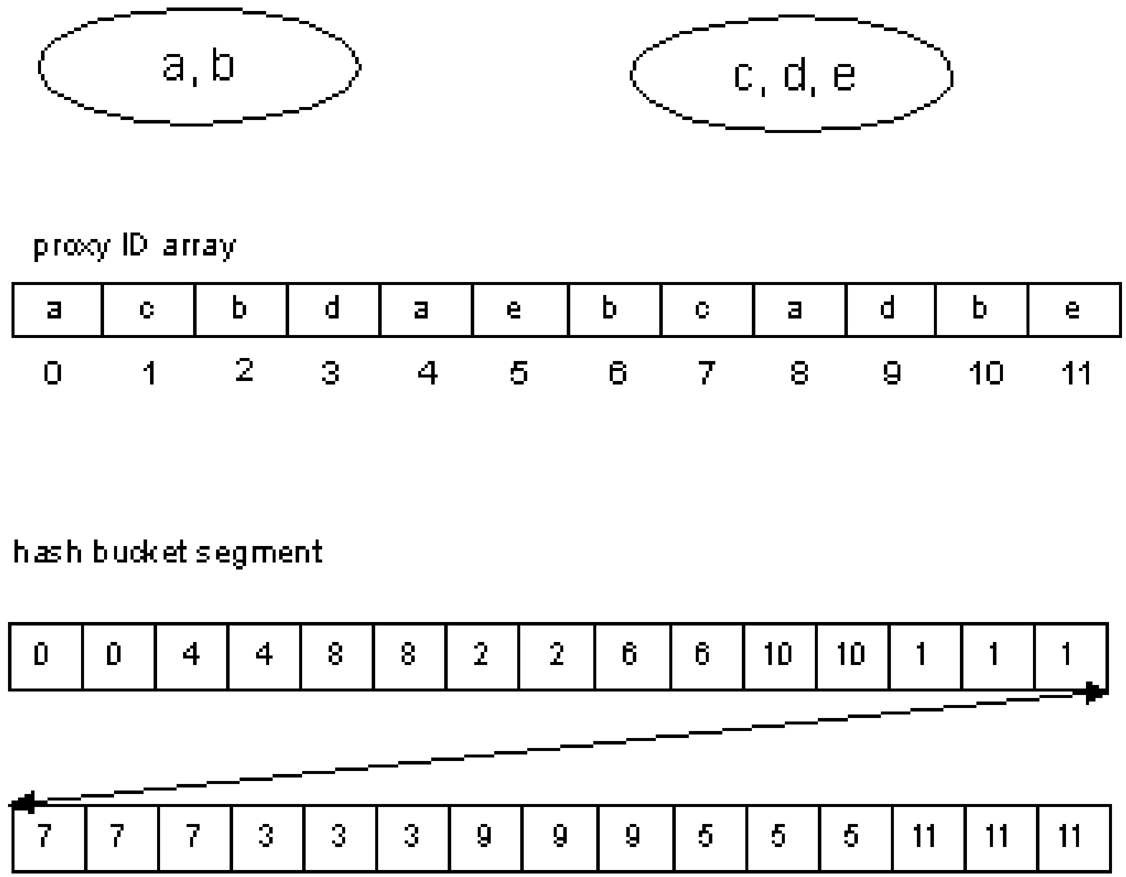
Indirect Mapping Scheme

- Map each hash bucket to an index of a proxy ID array instead of directly mapping each hash bucket into a proxy ID.
- From this proxy ID array, we then obtain the proxy ID for the hash bucket.
- Two parts for indirect mapping scheme:

  1. Construction of proxy ID array.

  2. The assignment of the indices of the proxy ID array to hash buckets.

An example of an indirect mapping scheme for LSH



Indirect Mapping

- Construction of Proxy ID array
- Two-level round-robin fashion
- Size of PA is N*LCMc, N is the number of clusters and LCM_c is the l.c.m. of C_i.
- Construction of hash bucket segment
- LCM_p is the l.c.m. of n_j
- The total size of the hash bucket segment is $LCM_p \cdot \sum C_i$

Load Balance

- Without considering Load Balance, the LSH degenerates into GCH.
- If the load of a proxy is too high, this proxy should not be selected.
- DNS is easy to detect the load condition of all proxies
- DNS is a better place to implement the LSH.

Performance Evaluation

- Trace driven simulator that models the three hashing schemes, GCH, GDH, and LSH.
- Nine proxies organized into three geographical clusters, each cluster has three proxies.
- Each Proxy has the same amount of computing resources.

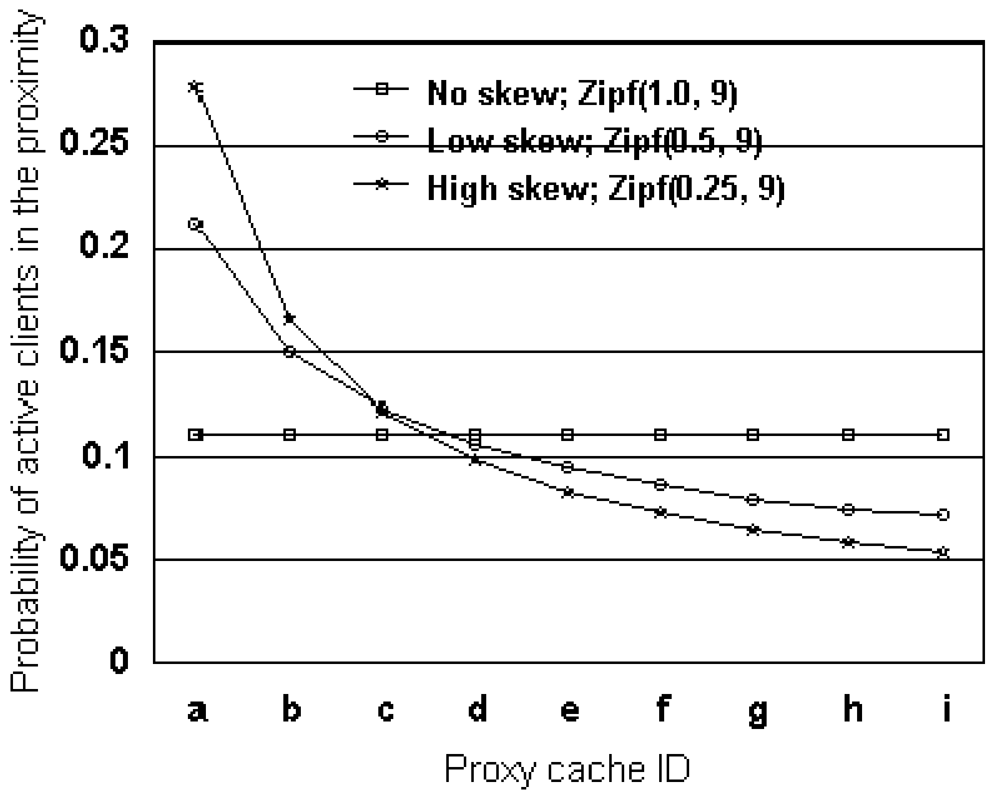
Performance Evaluation(cont'd)

- For each proxy, implemented:
  - A CPU server
    - FIFO service queue
  - A cache manager
    - LRU stack
- Response time for a request whose object can be found locally = $L + T_{http} + T_{cache} + T_{http} + L + Q$
    - L: latency delay
    - T_{http}: service time for processing an HTTP request or reply
    - T cache: service time for looking up an object from its cache or storing an object into its cache
    - Q: the queue delay the request incurs waiting for the CPU

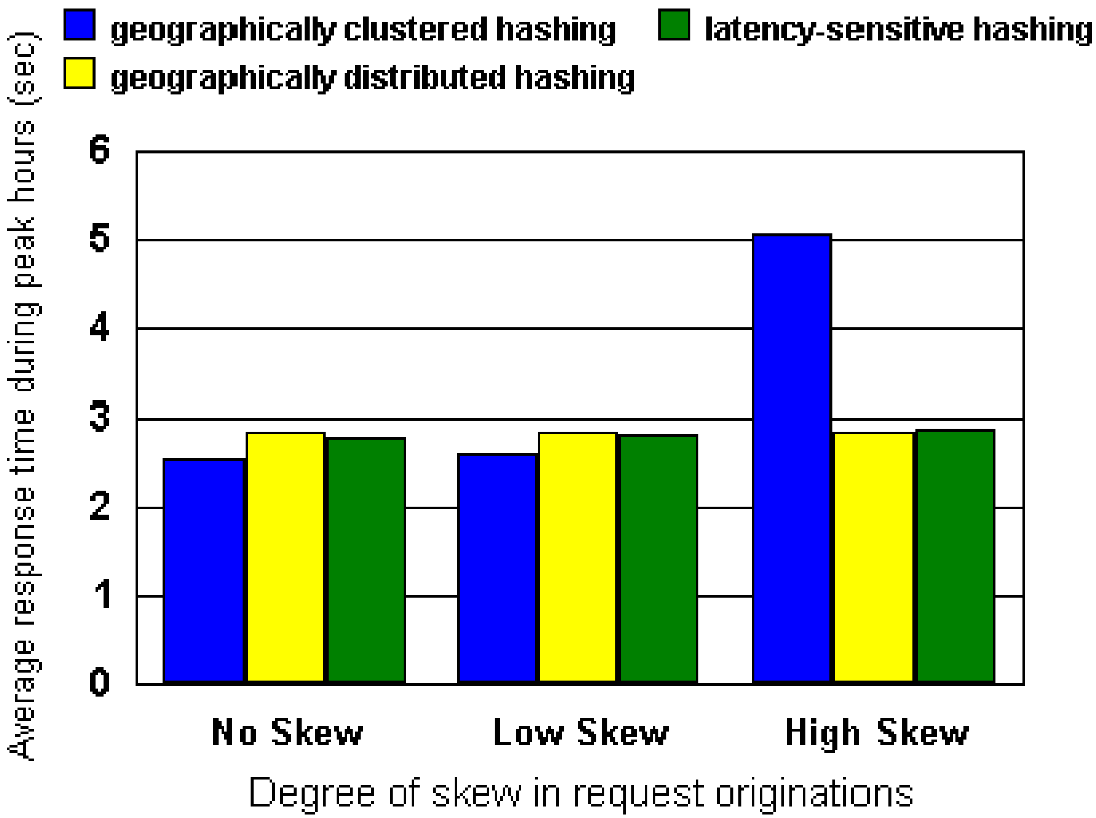
Performance Evaluation(cont'd)

- Response time for a request whose object is a cache miss $= L + T_{http} + T_{cache} + C_{miss} + T_{cache} + T_{http} + L + Q$
  - C_{miss}: A cache miss delay if the requested object can not found locally.
  - Assume $T_{cache} = 0.5*Thttp$
- Zipf-like distribution
  - Zipf(x, M) is a parametric distribution where the probability of selecting the ith item is proportional to $1/i^{(1-x)}$, where x is a parameter and i belongs to {1, …, M}

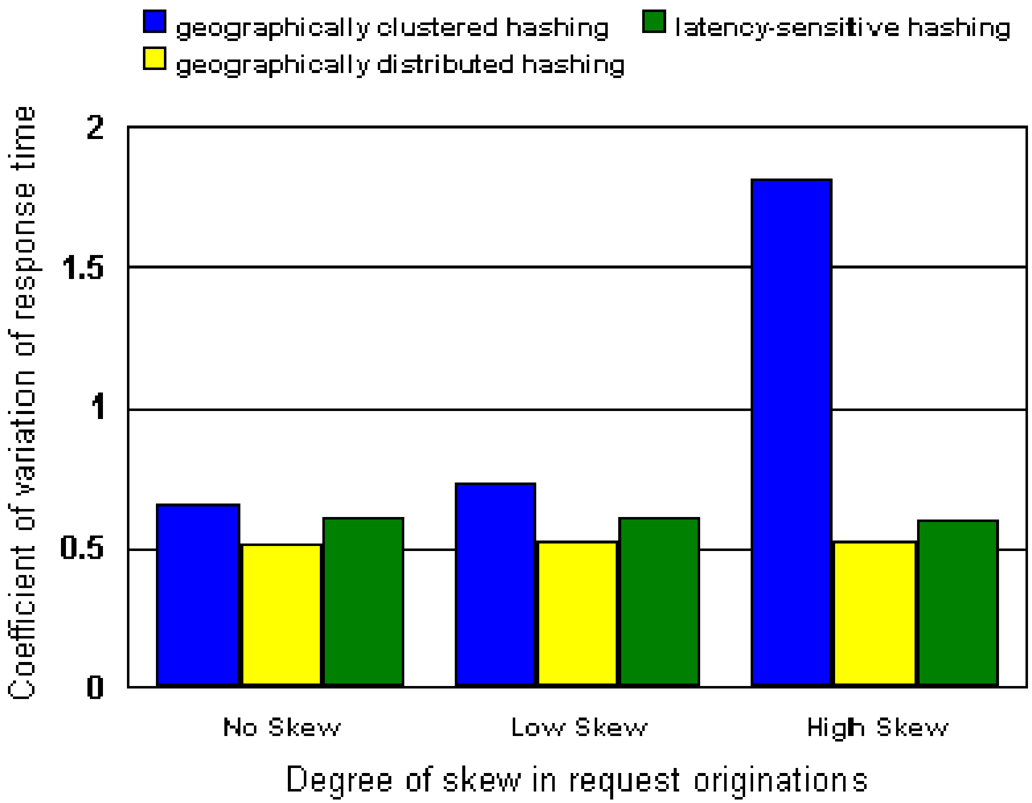
Distributions of clients around the proximity of each proxy cache



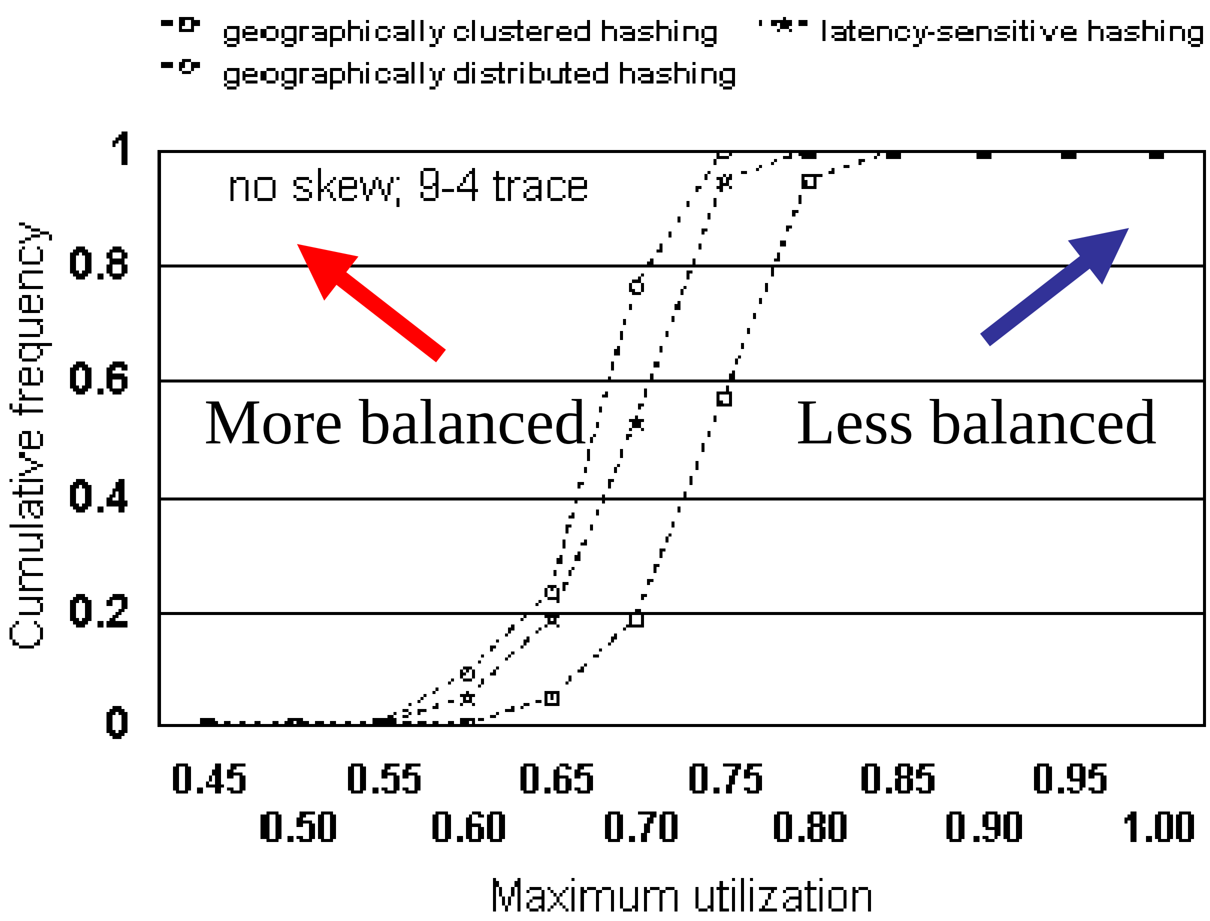
The impact of request origination skew on average response time



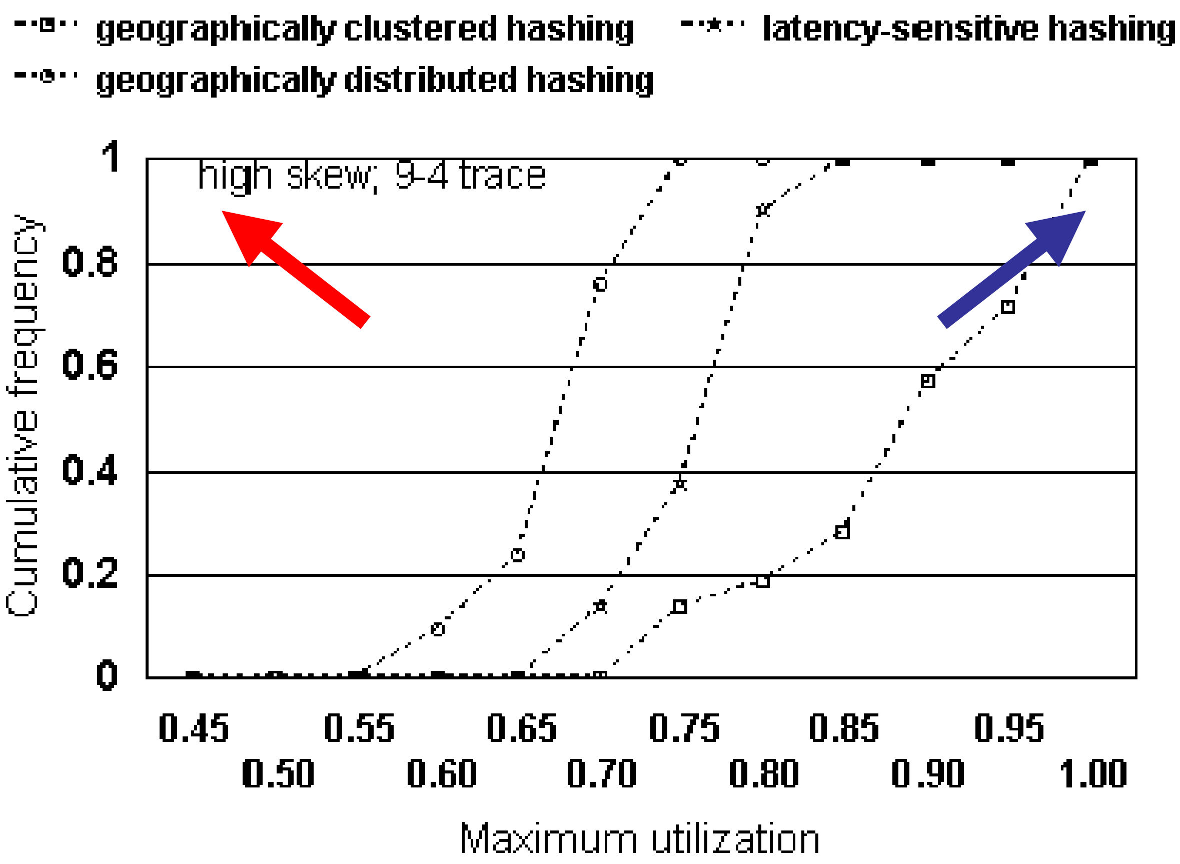
The impact of request origination skew on coefficient of variation



The level of load imbalance with no skew



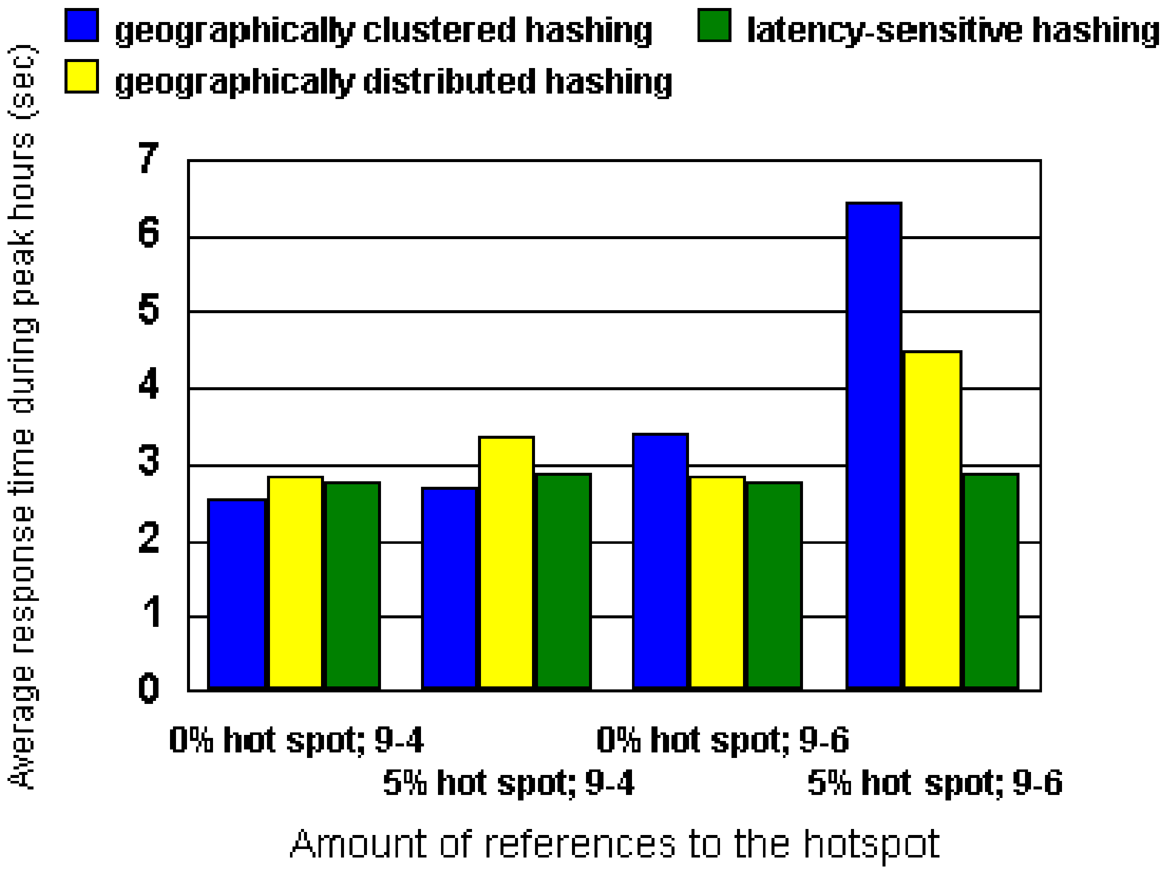
The level of load imbalance with high skew



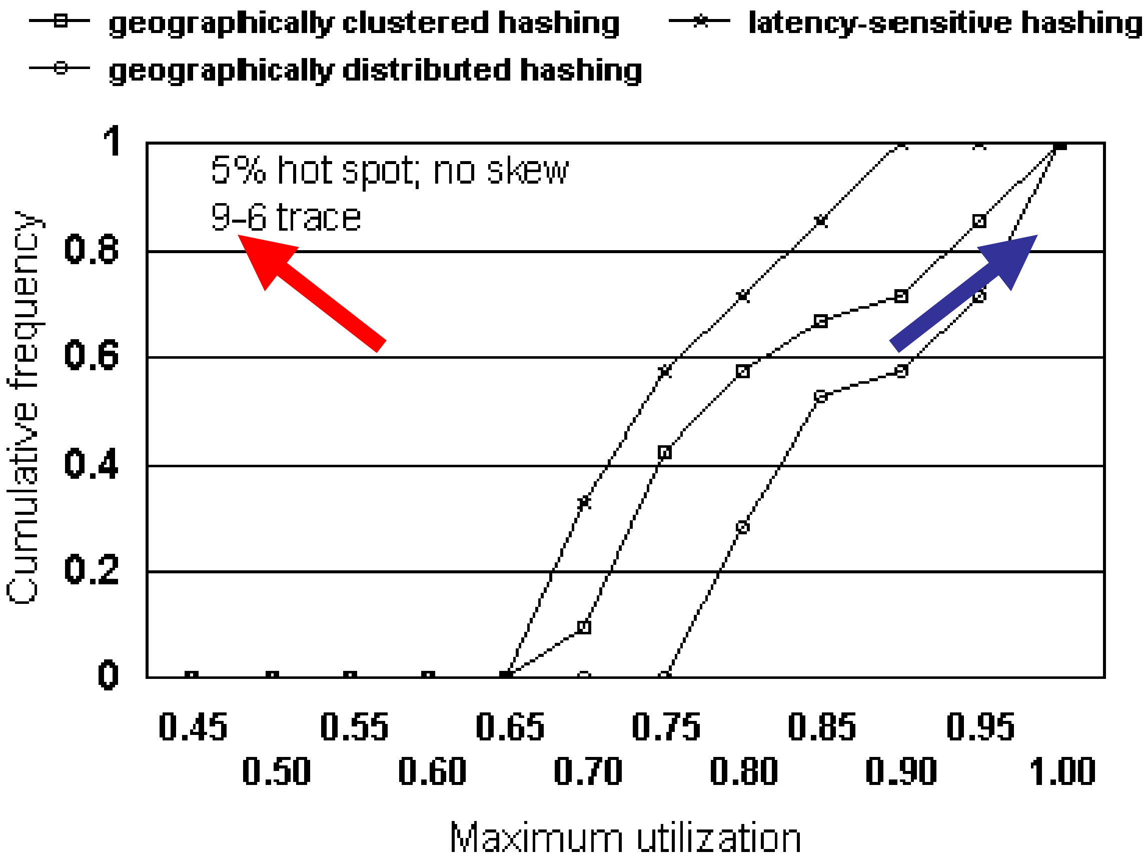
Simulation Results

- GCH is very sensitive to skew in request origination
  - GCH can not effectively utilize proxies in other clusters to help balance the load
- GDH is immune to the skew in request origination
  - Hashing is based on URL and thus the load distribution among the proxies remains the same regardless of skew in request origination.
- LSH can distribute requests among all the proxies, but it is slightly less balanced compared with GDH
  - In order to lower latency delays, LSH tends to choose a proxy within the same cluster as the browser originating the request.

The impact of hot-spot references on average response time



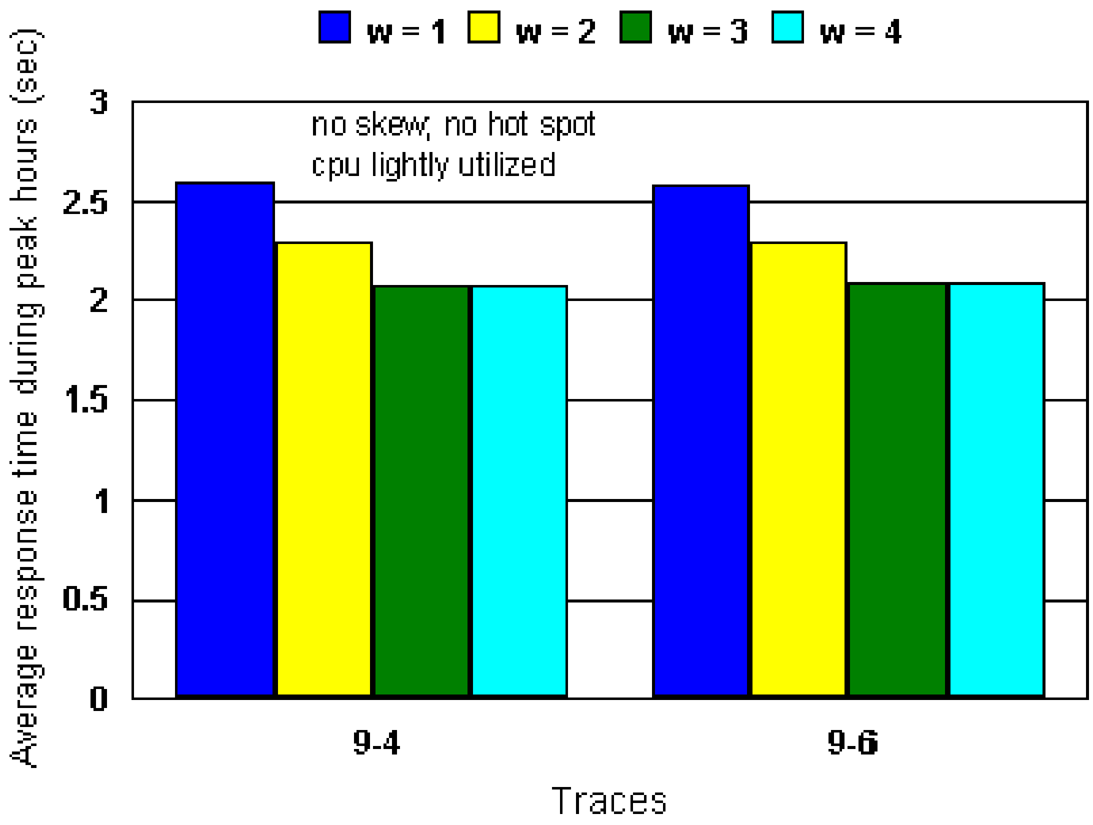
The level of load imbalance with hot-spot references



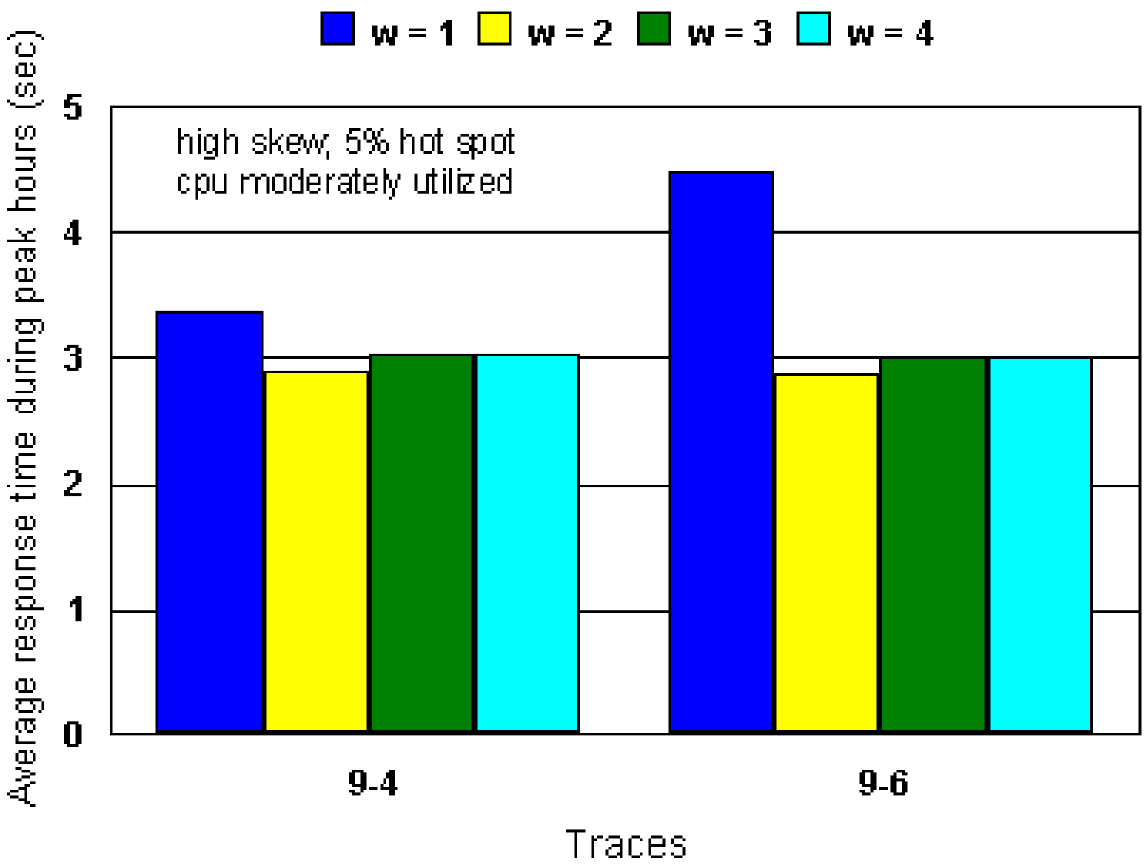
Simulation Results

- GDH can become quite unbalanced in the presence of hot-spot references
  - Each UTL is hashed into the same proxy cache no matter which browser issues the request.
- GCH is less susceptible to 9-4 trace hot-spot references, but highly sensitive to 9-6 trace.
- LSH handles is almost insensitive to hot-spot references.
  - LSH can select different proxies to offload the hot-spot references originating from different browsers.

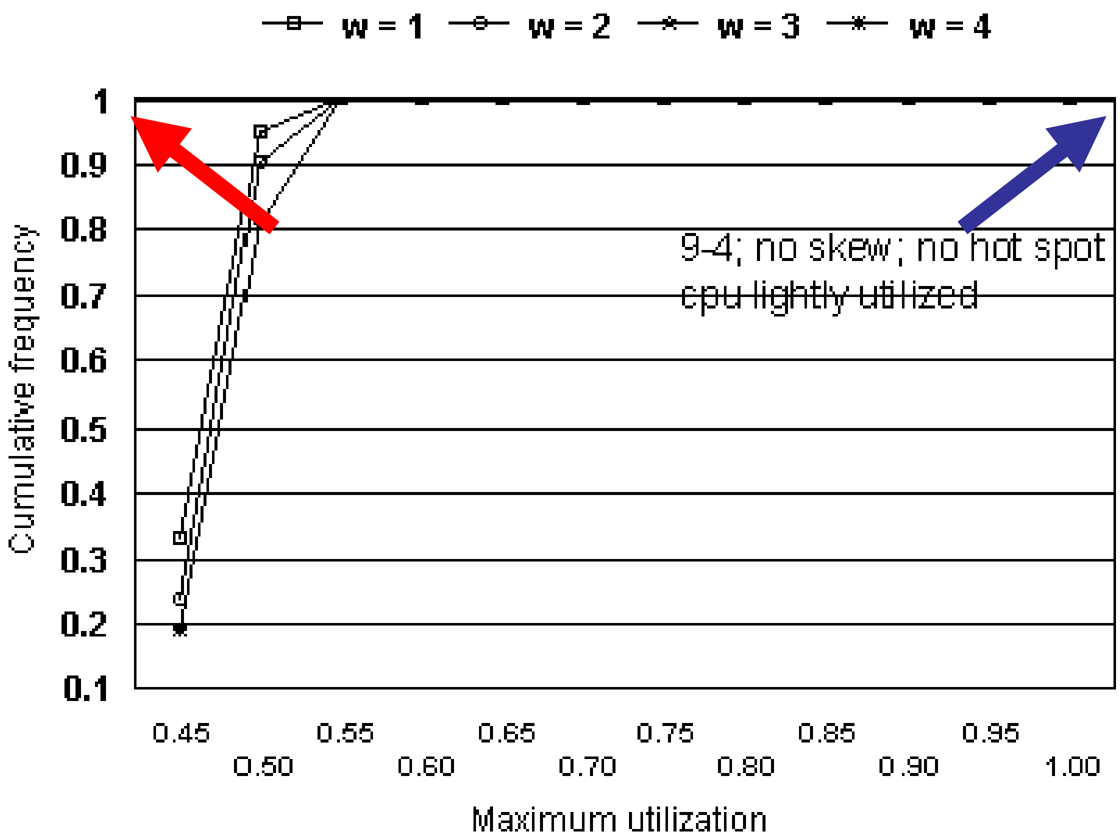
The impact of selection window size when the system is lightly loaded and balanced



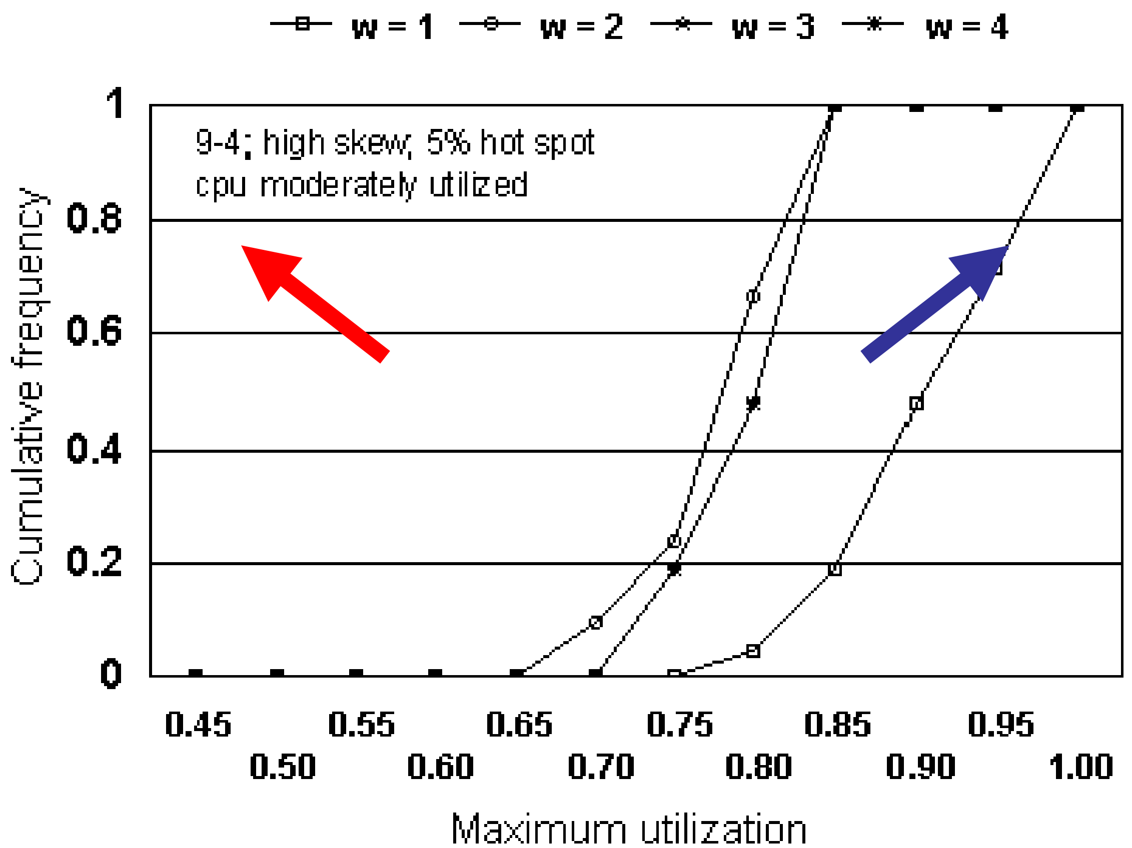
The impact of selection window size when the system is moderately loaded and unbalanced



The level of load imbalance when the system is lightly loaded and well balanced



The level of load imbalance when the system is moderately loaded and unbalanced



Simulation Results

- For light load and relatively well balanced system, a larger w enables more requests to be hashed into geographically closer proxies. The average response time is better.
- For a moderately loaded and unbalanced system, w=3 may cause too many requests to be hashed into geographically closer proxies, resulting in slightly less balanced system compared with w=2. When W=1, system is highly unbalanced.

Conclusion

- GCH hashes requests originated from one region into proxies within the same region. It’s performance is poor.
- GDH hashes requests to all proxies regardless of geographical locations. It fails in the presence of hot-spot references.
- LSH effectively handles both skew in request origination and hot spot references by hashing requests among geographically distributed proxies.
- Overall system is lightly loaded, LSH effectively reduces latency delays by hashing requests in to geographically closer proxies.

Questions?