Distributed Caching with Centralized Control

Sanjoy Paul[†] *and Zongming Fei*[⋆]
†: Edgix Corporation, New York, NY 10036, sanjoy@edgix.com
⋆: College of Computing, Georgia Tech, Atlanta, GA 30332, fei@cc.gatech.edu

Abstract

The benefits of using caches for reducing traffic in backbone trunk links and for improving web access times are well-known. However, there are some known problems with traditional web caching, namely, maintaining freshness of web objects, balancing load among a number of caches and providing protection against cache failure. This paper investigates in detail the advantages and disadvantages of a distributed architecture of caches which are coordinated through a central controller. In particular, the performance of a set of independent caches is compared against the performance of a set of coordinated distributed caches using extensive simulation. The conclusion is that a distributed architecture of coordinated caches *consistently* provides a better hit ratio, improves response time, provides better freshness, achieves load balancing, and increases the overall traffic handling capacity of a network while paying a small price in terms of additional control traffic. In particular, we have observed up to 40% improvement in hit ratio, 70% improvement in response time, 60% improvement in freshness and 25% improvement in traffic handling capacity of a network with caches.

1 Introduction

Internet service providers (ISPs) as well as corporate intranets are forced to deploy caches in their network to reduce traffic in their backbone trunk link and to provide faster web access to their customers. Reduction of traffic in the backbone trunk links translates to a direct cost reduction for ISPs and enterprise networks. Typically, ISPs deploy caches at the points of presence (POPs) and corporate networks deploy caches within each of their locations. Thus, each cache operates as an independent unit without taking advantage of the resources of the other caches within the same ISP (or enterprise network). We introduce a new architecture in this paper which treats the caches as a collective resource and thereby solves some of the fundamental caching problems, such as, *content freshness*, *load balancing* and *fault tolerance*.

End-users can take advantage of caching by pointing their browsers explicitly to a proxy cache. However, it is preferable to give the benefits of caching to end-users in a transparent manner. One way of transparently forwarding an end-user's web requests to a cache is to deploy a layer-4 (L4) switch[1] in the physical path between the end-user's machine and the cache. The L4 switch can then trap any traffic destined for Port 80 (web request) and forward it to a cache without the end-user's knowledge[2]. Although L4 switches have been used to transparently forward web requests to the caches, they have not been used to balance load among the caches in an intelligent way. Our proposed arhitecture enables intelligent load balancing among the caches.

Typically proxy caches are deployed in enterprise networks in various locations of a corporation. In the ISP networks, caches are deployed at the POPs. When a user sends a web request, it is forwarded to a nearby cache which then sends a response back to the user provided it has the requested object. If most of the requested objects can be served from the cache, user will see a perceptible improvement in accessing the web. The real-life deployment scenarios can be represented as in the left diagram of Figure 1 in which the clients are divided into n groups, where n is the number of caches in the network. The term CG_i is used to represent client group i which uses cache i (denoted as C_i) as its designated cache. However, we contend in this paper that such a deployment of caches, does not necessarily exploit the maximum potential benefit of a collection of caches in an ISP's network or an enterprise network. In fact, the caching resources are sub-optimally utilized in their network.

A lot of work has been done on cooperative caching [WC97,WC97b, CAB98, KD99, TVK99] mainly trying to improve cache hit ratio. For example, Inter Cache Protocol (ICP) [WC97, WC97b] is a protocol that is used by a cache to check if the requested objects reside at peer caches. In fact, ICP queries are sent *sequentially* to a *pre-configured* list of peer caches and if some cache has the requested object, an HTTP GET request is sent to that cache to actually fetch

[1]An L4 switch forwards packets based on five-tuple ⟨ Src.IP, Dst.IP, Src. Port, Dst. Port, Protocol ⟩.

[2]Note that when the cache receives the TCP SYN packet from the end-user's browser, it sends back a SYN/ACK packet with the *origin server's* IP address as the source address as opposed to its own IP address as the source address. Otherwise, the 3-way TCP handshake will fail.

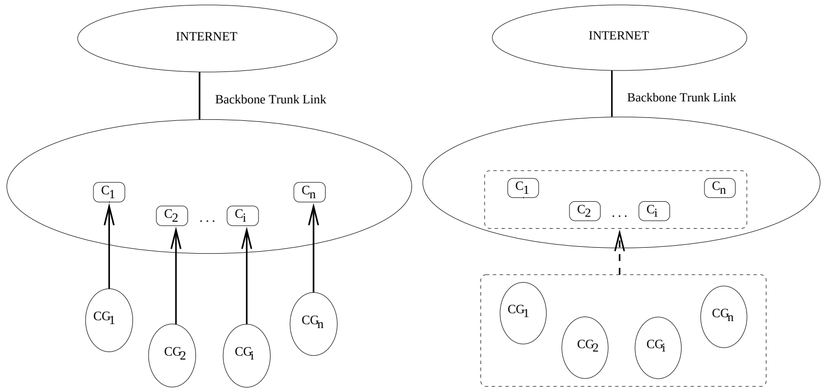


Figure 1: Common caching architecture and Coordinated caching architecture

the object. This mechanism is inefficient because every miss results in a sequence of ICP messages and the messages are not sent in parallel. These drawbacks can be avoided if a cache maintains a *summary* (or *digest*) of what web objects are stored at the peer caches and makes a *local* decision about which cache to go to for a specific web object [CAB98, HRW98]. This avoids the sequential polling of ICP and makes object look-up more efficient. This is known as *summary cache* [CAB98] or *cache digest*[HRW98]. Thus, summary cache and cache digest improve hit ratio. However, they do not explicitly help in balancing load across a number of caches or in reducing traffic for frequently changing web objects in the backbone trunk link.

There are several pieces of work [KD99, TDVK99] that require meta information to be maintained in the caching hierarchy to keep track of which objects are stored in which caches. This approach also uses a pool of caches in an efficient way to improve hit ratio. However, it cannot help in balancing load or in reducing traffic in the backbone trunk link.

WebWave [HM97, HMY97] is a diffusion-based caching protocol for a hierarchical structure of caches. It supports load balancing between the caches and improves client response time, but it is not effective in improving hit ratio or in reducing traffic across backbone trunk links.

WCCP [CF99] is a protocol used between a router and a set of caches mainly to provide transparent redirection of HTTP requests to the caches. It also has support for providing load balancing among the caches. Thus, WCCP helps in balancing load but it does not help in improving hit ratio or in reducing traffic across backbone trunk links.

Given that the existing work on cooperative caching has focussed on either improving hit ratio or in providing load balancing, there is a need for an architecture of cooperative caches which can do both and in addition, reduce traffic in the backbone trunk link for frequently changing web objects.

In this paper, we propose Distributed Caching Architecture with Centralized Control (DAC3) to address all of the above issues. DAC3 treats a set of n caches deployed in a network not as n independent caches, but rather as an *aggregate* pool such that if a web object (which can be an html page, a gif or jpg image) is stored in any one of the caches, it can be retrieved by a client request (the right diagram of Figure 1). Extensive simulation results show that this novel caching architecture provides the following benefits:

1. Hit ratio improves by 25% to 40% because the client's requested object is served from the cache as long as there is at least one cache which has the requested object. That is, if an object is in cache C_i and the request for the web object was generated from client group CG_j where $i \neq j$ and the web object is not cached in C_j, it is served from C_i, resulting in a "hit".

2. Freshness of objects served from the caches improves by as much as 60% for frequently changing objects because of centralized checking and update of the web objects.

3. Traffic handling capacity of a network with caches improves by 25% because the load for *popular* objects is distributed among the caches. If the request rate r_i from client group CG_i for the popular objects is higher than what C_i can handle, then the requests are distributed among the other caches, and hence the caches are utilized in an efficient way.

4. Traffic through backbone trunk links reduces by as much as 45% over and above the reduction provided by the caches in a network because only one cache is responsible for checking freshness and fetching modified objects through the backbone trunk link.

This paper is organized as follows. Section 2 describes the distributed caching architecture followed by simulation results in section 3. Finally the main contributions and results of the paper are summarized and the directions for future work are described.

2 Distributed Architecture for Caching with Centralized Control (DAC3)

DAC3 is designed to *boost* the performance of a set of caches in an efficient manner. In particular, the goals are to improve hit ratio, increase level of content freshness, balance load and reduce traffic through backbone trunk links. While improving hit ratio is relatively easy, and load balancing can be satisfied with some additional effort, maintaining freshness of cached objects is quite difficult [GS96, KW97, KW97a, YBS99]. In order to do a reasonably good job without adding too much complexity, DAC3 concentrates on the so-called popular objects and keeps them fresh in the caches with very high probability. For the other objects, DAC3 does not do anything extra compared to what a cache would normally do for maintaining their freshness. Thus, DAC3 is based on the fundamental assumption that the web traffic has a Zipf-like distribution [BCF+99] which implies that a significant part of the traffic is generated due to a small set of web objects. In fact, about 25% of the objects contribute to 70% of the web traffic. The goal of the architecture is to isolate the top 25% of the web objects (which we refer to as *hot* objects in the rest of the paper) in a dynamic manner and optimize the performance of distributed caching system for these frequently accessed objects. In order to make this goal achievable, DAC3 uses a central control station called a WebController which constantly monitors the web traffic, identifies the hot objects dynamically and influences the traffic flow in the network. The other component of the DAC3 architecture is an L4 switch (called WebDirector in this paper) which provides *transparent* caching by forwarding web requests to a WebCache without the end-user's knowledge and balances load among a set of caches by distributing web requests such that none of the WebCaches get overloaded. DAC3 architecture works as follows:

1. WebCaches analyze their logs to determine which objects are hot and provide access statistics for hot objects to the WebController on a periodic basis.
2. WebDirector collects the number of requests for websites (based on destination IP address in the web requests) and computes the round-trip time between itself and the WebCaches and provides that information to the WebController.
3. The WebController does the following three things: (a) sends a list of hot objects to a special cache, called the *Active* WebCache; (b) computes load distribution among the WebCaches based on the information from WebCaches and the WebDirector; (c) downloads a forwarding table containing the load distribution (computed in the previous step) to the WebDirector so that the WebDirector can forward requests accordingly.
4. The Active WebCache, on behalf of the distributed set of WebCaches, checks with the origin server if a hot object has been modified. If so, it fetches the modified object and distributes it to the relevant WebCaches using reliable multicast. This ensures that the hot objects are always fresh at the WebCaches. The freshness checking interval is adjusted dynamically. That is, more frequently changing objects are checked more frequently while less frequently changing objects are checked less frequently. Note that active WebCache prefetches hot objects. The benefits of prefetching are well-known [PM96, KLM97]. Also, the common criticism associated with prefetching is that it increases consumption of bandwidth [CY97]. However, since the active WebCache prefetches only the hot objects which are frequently accessed, the above argument does not hold. Also, observe that by virtue of pushing hot objects to caches, the system tries to leverage the idea of push caching [GS96a].
5. When a client's request arrives at the WebDirector, the WebDirector uses the forwarding table to determine which WebCache the request should be forwarded to.
6. On receiving a client's request, the WebCache first checks to see if the request is for a hot object. If it is, the object is returned immediately to the client. If the object is not hot but exists in the WebCache, the WebCache may need to do an *if-modified-since* (IMS) check before returning the object to the client. Otherwise, the WebCache sends the request to the origin server, retrieves the object, stores it locally and sends it back to the client.

One fundamental aspect of the DAC3 architecture is the separation of control and data flows in the context of web caching. More specifically, the DAC3 architecture analyzes web traffic by processing the logs at the caches and by collecting information about HTTP requests that flow through the WebDirector. These information are enough to identify the *hot* objects and to obtain the number of requests over a predefined period of time for the *hot* objects. Once this meta information is available, the data flow for HTTP traffic in DAC3, is influenced such that the load is shared across multiple caches and also the freshness of hot objects is maintained across the set of WebCaches in the DAC3 architecture.

2.1 Data-Flow Architecture

The data-flow architecture consists of (see the left diagram of Figure 2): (1) Active WebCache, (2) WebCache and (3) WebDirector.

The active webcache maintains a list of *hot* objects and checks for freshness of the hot objects on behalf of a set of WebCaches. The frequency of checking for freshness is adjusted adaptively based on the life cycle of the objects. However, the checking interval has a minimum and a maximum value. The minimum value is provided to put a bound on the control traffic while the maximum value is provided to prevent objects from becoming permanently stale. The active webcache fetches modified hot objects and pushes them to WebCaches.

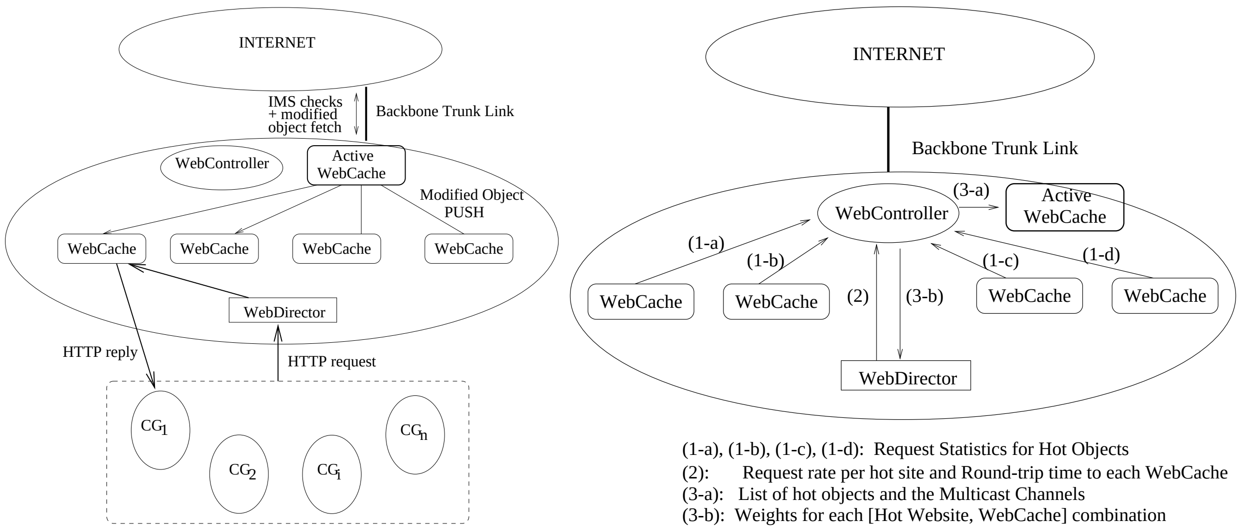


Figure 2: Data Flow and Control Flow Architecture

WebCaches are a set of machines which cache (store) objects. These WebCaches are distributed over a network. Ideally WebCaches are closer than the origin server to the clients and hence can respond to HTTP requests faster than the origin server can. In addition, if responses are sent from WebCaches to the clients, traffic load on the network segment between the origin server and the WebCaches is significantly reduced. A WebCache responds to HTTP requests from clients and sends HTTP requests to the origin server if the requested object is not stored in the cache. When the requested object is returned from the origin server, the WebCache stores it locally unless it is explicitly marked as non-cacheable. Web objects are replaced from the caches using LRU (least recently used) policy. One unique feature of the webcache is that it can receive objects pushed to it by the active webcache. In order to be able to receive the pushed objects, WebCache needs to tune in to one or more multicast addresses which are chosen by the active webcache for object distribution.

WebDirector is an L4 switch which forwards HTTP requests based on the five-tuple: $\langle$ source IP, destination IP, source Port, destination Port, Protocol $\rangle$ to the WebCaches. The WebDirector is capable of bypassing the WebCaches based on a set of rules. In addition, WebDirector plays the role of load balancer. When a HTTP request arrives at the WebDirector, it looks up a forwarding table and dispatches the request to one of many WebCaches, thereby balancing load among the WebCaches. It maintains a forwarding table which consists of a set of *deterministic* rules and a set of *probabilistic* rules. The deterministic rules could apply to specific source(s), specific destination(s), or to a combination. That is, for example, if a HTTP request is generated from a certain set of source IP addresses or is destined for certain IP addresses, it can be forwarded to specific WebCaches. The probabilistic rules apply in cases where the contents of the same website is replicated in more than one WebCache and the WebDirector forwards HTTP requests to one of these WebCaches with certain probability. The probability numbers are obtained in DAC3 architecture from the control-flow part (as described later on). Note that this type of probabilistic dispatching can be thought of as *two-dimensional weighted round robin* scheduling where a weight w_{ij} is the probability of dispatching a request destined for website i to WebCache j. In addition, when HTTP requests are destined for IP addresses that do not belong to any one of the above two categories (this is normally the case for "non-hot" objects as opposed to the *hot* objects), the WebDirector hashes on the destination address to decide which WebCache to forward the request to.

2.2 Control-Flow Architecture

The control-flow architecture consists of (see the right diagram of Figure 2): (1) WebCaches, (2) WebDirector, (3) WebController and (4) Active WebCache.

WebCache is as much a component of the data-flow architecture as it is a component of the control-flow architecture. Each WebCache uses a hot-object threshold T_{ho} such that if the number of accesses to an object over a period of time exceeds that threshold, it is marked as hot. A WebCache provides a list of hot objects and the number of accesses to each one of them to the WebController.

WebDirector is the key element for *load balancing* in DAC3. It distributes requests for a given site among multiple WebCaches with different probabilities (same as weighted round-robin distribution). Finally, the WebDirector sends probe messages to WebCaches to check for their liveness. Thus it can remove a WebCache from its forwarding table as soon as it detects the unavailability of a WebCache.

The WebController collects information from different components of the system, consolidates the information and distributes relevant information to different components. The WebCaches periodically provide a list of hot objects

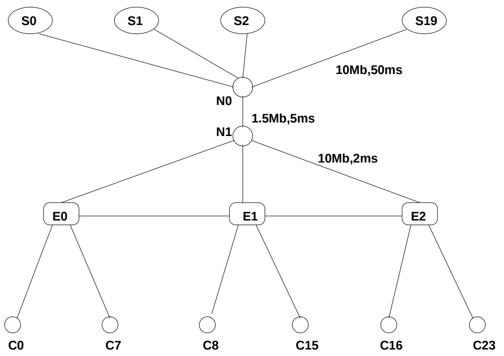


Figure 3: The network setup

and number of accesses to each one of them. The WebController collects this information from all the WebCaches, adds the number of accesses for each object provided by the WebCaches and compares the total number against the hot-object threshold T_{ho} such that if the total number of accesses exceeds T_{ho}, the object is classified as *hot*. A simple example will clarify why a WebController needs to do the thresholding in addition to a WebCache. In the beginning, there are no hot objects and hence the WebDirector simply uses a hash function to redirect a web request to one specific WebCache. Thus, *all* requests for a specific object will go to a *single* WebCache. Therefore, the only way a hot object can be identified is by every WebCache comparing the number of requests for an object against the hot-object threshold T_{ho}. However, once the WebController knows that an object is hot, it starts distributing the requests for the object among multiple WebCaches. Now, since the requests for a specific object gets distributed among many WebCaches, the number of requests for a specific object on a specific WebCache probably will not exceed the hot object threshold T_{ho}. However, the sum of requests to a specific object on multiple WebCaches will most likely exceed the threshold T_{ho}. Therefore, in the steady state, when requests for hot objects are distributed among multiple WebCaches, the thresholding needs to be done by the WebController. Once a list of hot objects is compiled, the sites to which the hot objects belong are referred to as *hot sites*. Load balancing is performed at the granularity of hot-sites because the WebDirector is an L4 switch and performs forwarding of requests based on IP address and Port number. WebController also collects information from the WebDirectors about their round-trip delay to each WebCache and the number of requests for the hot-sites as seen by each WebDirector. Based on the information from the WebCaches and the WebDirectors, the WebController runs a load balancing algorithm [RP98] and decides on the distribution of load among the WebCaches. The load distribution, in the form of a two dimensional forwarding table is then downloaded to the WebDirector. That is, each (website, WebCache) combination has a weighting factor. A special case of the optimal load balancing is to have a hot object replicated on all WebCaches so that the weighting factor for every (website, WebCache) combination is $1/N_c$ where N_c is the number of WebCaches. The WebController also provides the list of *hot* objects to the Active Cache.

Active WebCache receives from the WebController the list of hot objects and the list of WebCaches for each hot site. It then uses a multicast control channel to broadcast to all the WebCaches the information about which WebCaches should subscribe to which multicast data channels where a multicast data channel and a hot site has a one-to-one mapping. Eventually, the Active WebCache uses the multicast data channels to push hot objects to the WebCaches.

3 Performance Evaluation

3.1 Experimental Setup

We evaluate the performance of our proposed caching scheme by simulation written in *ns-2* [ULV97]. The simulation topology is shown in Figure 3. We have 24 client nodes (C_0 to C_{23}), 3 cache nodes (E_0 to E_2) and 20 server nodes (S_0 to S_{19}). Every 8 client nodes are directly connected with a cache node. The remote line bandwidth is 10Mbps and the delay is 50ms. Assume we have one $T1$ line (N_0N_1) and the bandwidth is 1.5Mbps and delay is 5ms. All other local lines have a bandwidth of 10Mbps and their delays are 2ms. The line are assumed to be duplex links. Each client node is attached with a client and each server node is attached with a server. We experiment with three different configurations.

1. No Cache. We do not have any cache in the system. Each client directly makes requests to servers.
2. Independent Cache. Each cache node is attached with a cache, but there is no coordination between caches. Web request from a client always goes to its nearest cache.
3. Coordinated Cache. Each cache node is attached with a cache and a WebDirector. A WebController is located in one of the cache nodes. When a client wants to make a request, it is sent to the nearest WebDirector. The WebDirector decides which cache the request should be sent to and forwards the request to the selected cache. Control information is exchanged between WebCaches, WebDirector and WebController.

Trace	No. of Reqs	No. of Objects	Avg. Object Size (bytes)	Avg. Req Size(bytes)
NLANR	162379	99104	4755	4256
DEC	277276	148898	13026	10232
UCB	473531	232269	9446	7307

Table 1: Trace Statistics

3.2 Metrics of Interests

We are interested in the following performance metrics:

1. Response Time This is the duration between the time a client makes a request and the time it gets all the requested data. The average is computed over all clients.
2. Traffic Generated This is the traffic generated by each scheme at the bottleneck link ($N_0 N_1$).
3. Hit Ratio When a client send a request to a cache, the document may or may not be in the cache. If it is in the cache, the cache checks if it is fresh or not. If it is fresh, it sends the document back to the client immediately. The hit ratio is the number of fresh hits over the total number of requests.

In addition to the above metrics, we also show some other results, like the load on each cache, freshness of objects served from cache, and the control traffic generated.

3.3 Design of the Experiments

We perform two categories of experiments. The first set of experiments are driven by actual traces. The request time and the document request follows strictly from the trace. This is used by clients. We also have the size information for each document, which can be used by the servers.

The second category of experiments is driven by synthetic traffic. This traffic is generated according to the following specifications. The popularity of objects in the request stream is modeled using Zipf-like distribution. The interarrival time is exponentially distributed with a specified mean value.

The modification model of web objects at the server is an important factor. We divide them into three categories. On one end of the spectrum are objects that change very frequently. The mean lifetime for these objects is in several minutes. On the other end of the spectrum are objects that do not change in a very long time. The average lifetime for these objects are in several months. In between them is the third category which consists of objects that change moderately frequently. The mean lifetime for these objects is in several hours. Lifetime of documents in each category is modeled using exponential distribution.

We experiment with several different average lifetime of middle category and show how the metrics are affected.

3.4 Numerical Results

3.4.1 Trace based experiments

We use three traces that are publicly available.

1. NLANR Trace [NLA97] We use one day trace of 06/29/99.
2. Dec Trace [KMM96] We use one day trace of 09/21/96.
3. UCB Trace [GB97a] This is UC Berkeley home IP web traces. We use one day trace from 11/14/96 12:47:04.

The statistics of these traces are given in Table 1.

Response time, Traffic in backbone trunk link, Hit Ratio

The response time of each case is shown in the left plot of Figure 4. Note that while caching (independent) can reduce response time by a factor of 3, coordinated caches (DAC3) can provide another factor of 2 reduction over independent caches. In particular, for the NLANR trace, the response time for no-cache is more than 3 sec, for independent caches is close to a second, while for coordinated caches, it is less than 500 ms.

We show the traffic generated at the bottleneck links in the right plot of Figure 4. The point to be noted is that the traffic for coordinated caches (DAC3) is consistently lower than thet for independent caches although the differences are not significant. However, as we increase the number of caches, the difference increases (refer to the left plot of Figure 13) and the traffic generated in the backbone trunk link for independent caches can be as much as 75% more than that in the case of coordinated caches.

The third metric of comparison is hit ratio. Figure 4 shows that coordinated caches (DAC3) consistently provide better hit ratio than independent caches. In fact, in the NLANR trace, hit ratio is improved from 10% to 15% which is a 50% improvement. There are two factors that contribute to this improved hit ratio. The first is the fact that *all* requests for hot-objects, regardless of where they come from, are satisfied in DAC3. This is not necessarily the case in independent caches because requests for objects which are not hot *locally* are not satisfied from local proxy caches. The second reason is to do with object freshness. Hot objects are always fresh and hence requests for hot-objects are always hits in DAC3. In independent caches, the requested object may reside in a cache but may not be fresh. A stale hit is counted as *miss* in our simulation.

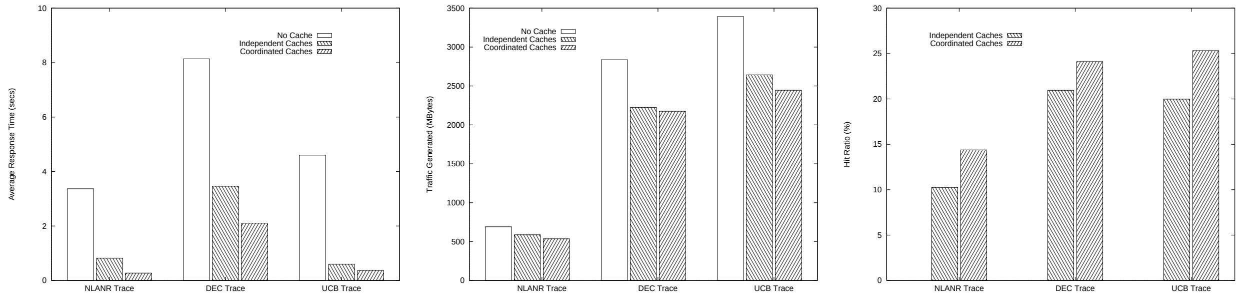


Figure 4: Response Time, Traffic Generated and Hit Ratio

3.4.2 Experiments with Generated Requests

These results are based on the synthetic workloads described in section 4.3. Note that there are three types of objects: very frequently-changing, moderately frequently-changing and less frequently-changing. The average lifetime of moderately frequently-changing objects is varied. Three different values were used: 3.5 minutes, 15 minutes and 60 minutes. The threshold T_{ho} is also varied from 2 to 64.

Response time, Traffic in backbone trunk link, Hit Ratio

First we show how three schemes (no-cache, independent caches and coordinated caches) perform with regard to the metrics we are interested in. The left plot of Figure 5 shows the response times. While the average response time for independent caches is more than 3 times lower than that for a no-cache scheme, it is further reduced in coordinated caching systems. The differences are more dramatic in the case of trace-based simulations (the left plot of Figure 4).

The right plot of Figure 5 shows the generated traffic at the backbone trunk links. Note that coordinated caches (DAC3) consistently provides about 25% reduction over independent caches.

The left plot of Figure 6 gives the hit ratio for independent caches and coordinated caches. Once again, coordinated caches consistently provide 15% to 30% improvement in hit ratio. Note that the increase in hit ratio is more prominent with the more frequently-changing objects. This happens because DAC3 keeps the frequently-changing hot-objects always fresh and any request to the hot-objects is a guaranteed hit.

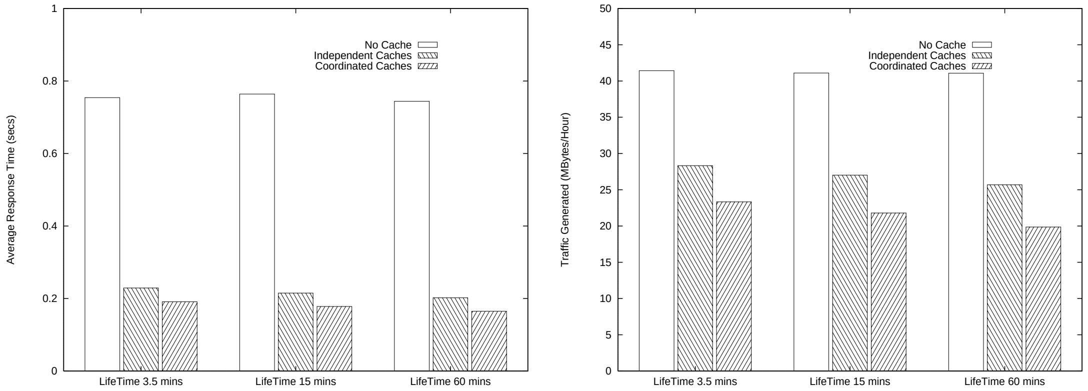
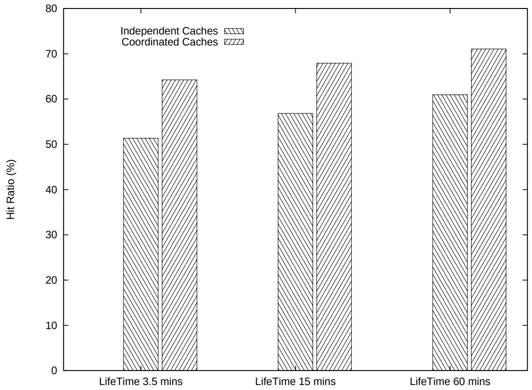


Figure 5: Response Time and Traffic Generated

Effect of T_{ho}

It is easy to see that the performance of DAC3 is very much dependent on the value of the hot-object threshold T_{ho}. If the value of T_{ho} is 0, all objects will be hot and hence the traffic generated due to freshness checking on the backbone trunk link will be enormous. Similarly, control traffic generated in DAC3 due to reporting of statistics of hot-objects



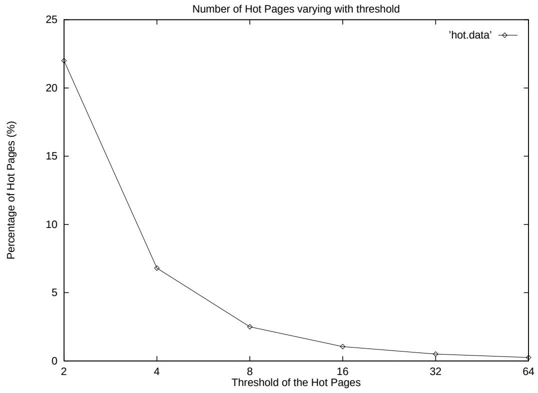
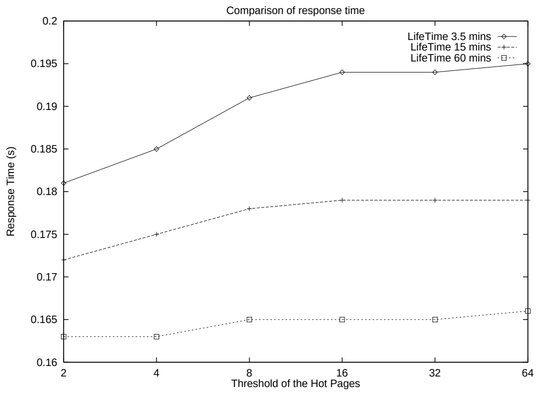


Figure 6: Hit Ratio and Percentage of Hot-Objects

and hot-sites will be very high. On the other hand, if the value of T_{ho} is chosen to be ∞, DAC3 will degenerate to an independent caching system and there will be no additional traffic for pro-active freshness checking or any control traffic for reporting of statistics because there will be no hot-objects at all. The next plots explore the effect of T_{ho} on the performance of DAC3.

The right plot of Figure 6 shows what percentage of objects are hot. The left plot of Figure 7 shows the response times. Note that as the threshold increases from 2 to 64, the response time increases. This is because, as the number of hot-objects is reduced, the probability of finding a fresh object in a cache reduces. In addition, note that the response time for less frequently-changing objects is lower than that for more frequently-changing objects. This happens because the lower is the frequency of change, the higher is the probability of finding a fresh object in a cache.

The right plot of Figure 7 shows the generated traffic at the backbone trunk links. Here also the traffic is higher for more frequently-changing objects compared to that for less frequently-changing objects because the frequency checking freshness is higher for more frequently-changing objects. In addition, as the threshold increases, the traffic reduces until it becomes stable. This is because the number of hot-objects reduces with the increase in the value of threshold resulting in a lower rate of checking freshness. The left plot of Figure 8 shows the effect of T_{ho} on hit ratio. Note that as expected, as the threshold increases, number of hot-objects reduces and the hit ratio goes down. The effect is more pronounced for frequently-changing objects as opposed to for less frequently-changing objects.



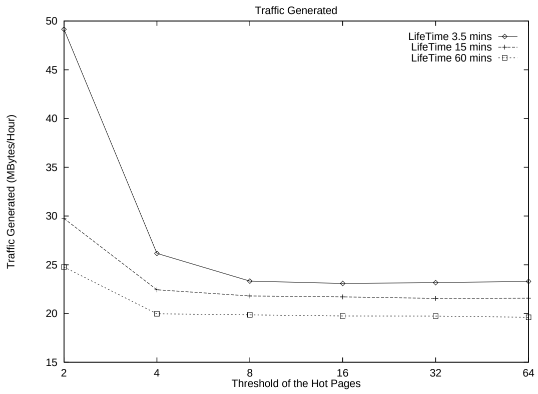


Figure 7: Response Time and Traffic at the backbone trunk link

Overhead in DAC3

Next, we evaluate the overhead of co-ordinating a set of distributed caches in DAC3. First, we show the overhead due to centralized freshness checking of hot-objects. This is represented by the if-modified-since (IMS) check traffic in the backbone trunk link. The right plot of Figure 8 compares the IMS traffic in an independent caching system with that in a coordinated caching (DAC3) system. Note that, as expected, the IMS traffic is higher in DAC3 compared to an independent caching system. However, the difference is not significant. In addition, if we compare the IMS traffic with the data traffic generated in the backbone trunk link, we see that the control traffic is a very small portion (less than 5%) of the data traffic. This implies that DAC3 pays negligible price for centralized freshness checking when the

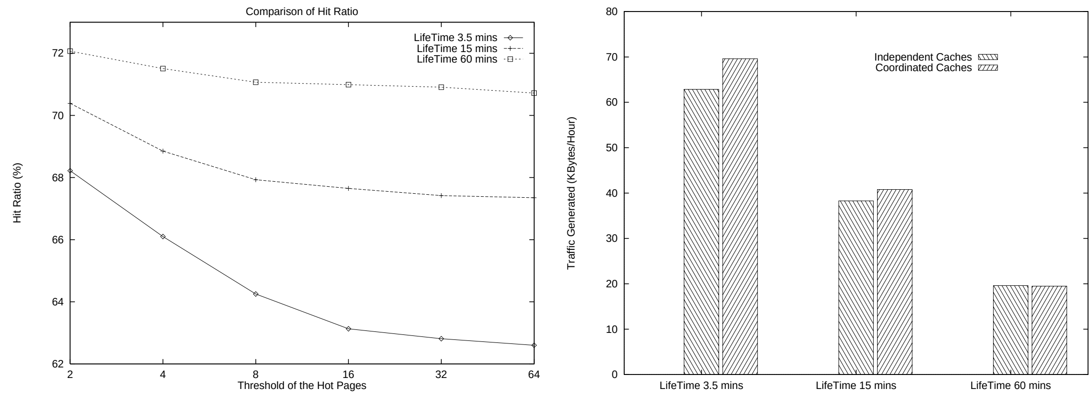


Figure 8: Hit Ratio and IMS Traffic

overhead is estimated in terms of additional traffic in the backbone trunk link. This is illustrated in the right plot of Figure 9. The left plot of Figure 9 shows how the IMS traffic in DAC3 varies with the hot-object threshold T_{ho}. As expected, the IMS traffic increases as the value of T_{ho} decreases. This is because a decrease in the value of T_{ho} implies more hot-objects which in turn implies more number of IMS check messages.

We also recognize that control traffic is generated when WebCaches send hot-site/object information to the WebController. However, this is done every half an hour. The packet size is dependent on the number of hot-sites and hot-objects. The total control traffic generated in our simulation is less than 6 Kbytes/Hour which is less than 40% of the IMS traffic generated in the backbone trunk link in 60 minute case and is about 10% of the IMS traffic generated in the 3.5 minute case. If we compare this control traffic with the data traffic, we can see that the control traffic is less than 0.1% of the data traffic. Once again, this shows that the overhead of co-ordinating caches in a centralized manner is significant while the benefits are significant.

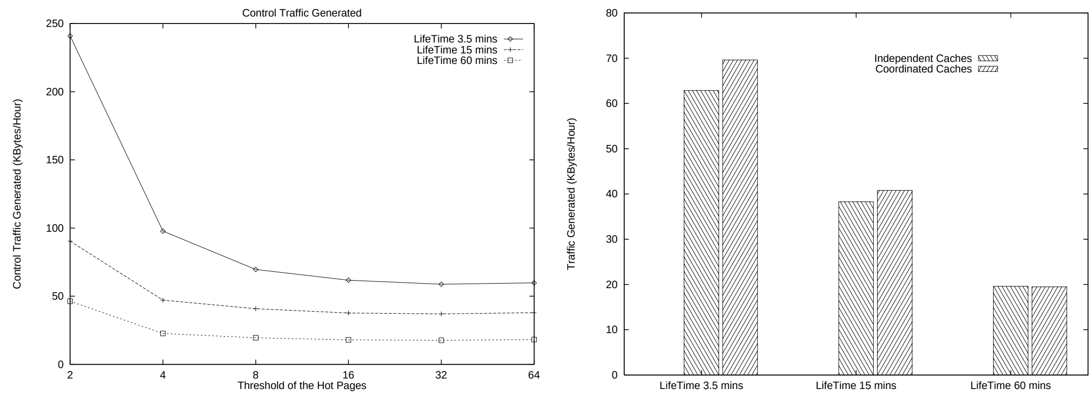


Figure 9: IMS Traffic vs. T_{ho}; Data and IMS Traffic

Load balancing

To illustrate how DAC3 can balance the load on WebCaches, we let clients generate request at different rate. We let the clients attached to WebCaches 0, 1 and 2 with relative request rate 6:3:2. We illustrate the load on each WebCache in the left plot of Figure 10. With independent caches, the load distribution among the WebCaches is exactly in the relative ratio of 6:3:2. However, in DAC3, the load is evenly distributed among three WebCaches (left plot of Figure 10).

Traffic handling capacity

To explore the traffic handling capacity of a caching system, we increase the request rates until the bottleneck link (backbone trunk link which is 1.5 Mbps in our simulation network) is saturated. We show in the right plot of Figure 10 the response times in the three schemes: no-cache, independent cache and coordinated cache, as the request rate is increased. Note that if there are no caches at all, the response time increases sharply when the request rate reaches 40 requests/s. While the backbone trunk link gets saturated at 80 requests/sec for an independent caching system, it does not get saturated until 100 requests/sec for a coordinated caching system like DAC3. This implies that DAC3

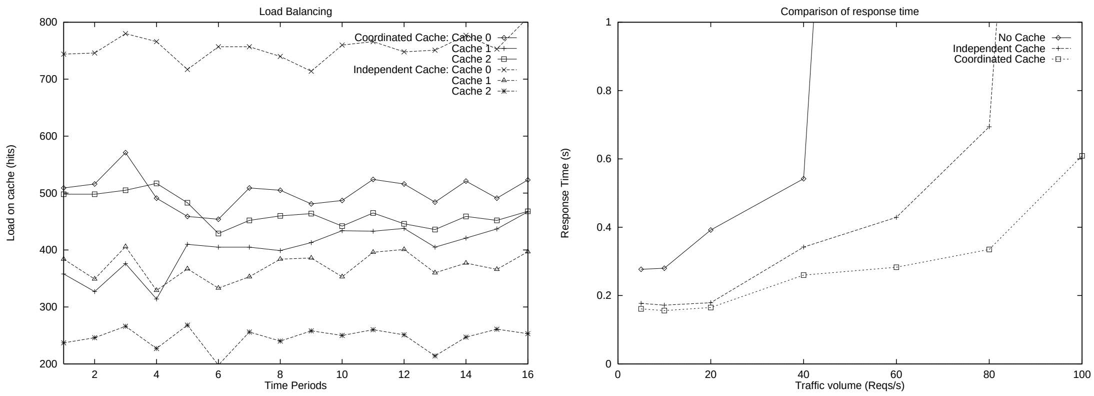


Figure 10: Load Balancing and Traffic Handling Capacity

increases the capacity of a network by 25%. Another way of interpreting the same result is that to achieve the same performance, a network has to deploy 25% less number of WebCaches if DAC3 architecture is used instead of simply deploying independent caches.

Effect of the number of caches

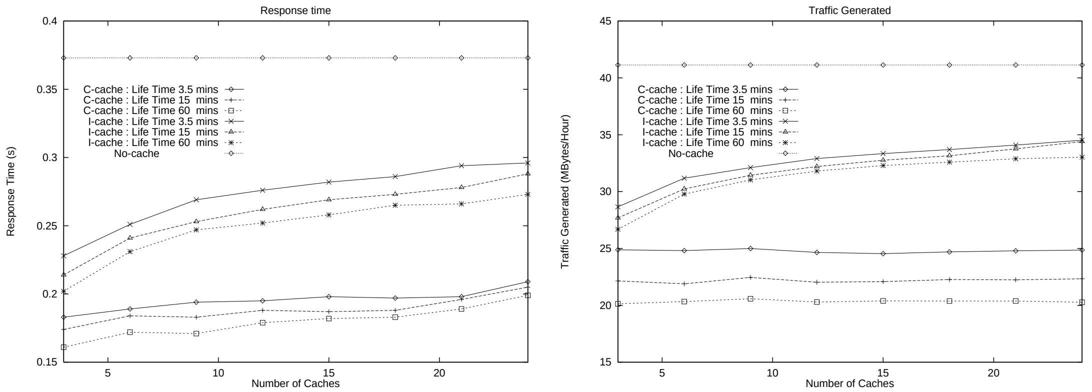


Figure 11: Response Time and Traffic vs. Number of Caches

In our first experiment, we change the number of caches from 3 to 24 and compare the three caching schemes: No-cache, I-cache (independent cache) and C-cache (DAC3) with respect to the three metrics of interest: response time, traffic in backbone trunk link and hit ratio. Three different values of lifetime (3.5 minutes, 15 minutes and 60 minutes) are used for the moderately frequently changing objects in the simulation. The left plot of Figure 11 shows how response time changes with the number of caches. The total number of requests from the clients is the same in each case. Note that the average response time for no-cache case is fixed at 370 ms. In case of independent caches, it increases from 200 ms to 250 ms for the plot corresponding to average lifetime of 60 minutes. The corresponding numbers are 160 ms and 190 ms for coordinated caches (DAC3). The main point to observe is that the difference between independent caches and coordinated caches increases as the number of caches is increased. The increase in response time with number of caches can be explained by the fact that the hit ratio decreases and the caches have to retrieve the requested objects from the origin servers.

The right plot of Figure 11 shows the traffic generated in the backbone trunk link. It has two components: (1) IMS traffic and (2) actual data traffic due to fetching of modified objects. The latter is the dominant component. In the coordinated cache scheme, the traffic generated does not increase as the number of caches increases. However, the traffic generated in the independent cache scheme increases as the number of caches increases. This happens because in the independent cache scheme, each cache generates IMS checks and fetches objects when they are modified. However, in DAC3, *only* the active WebCache generates IMS checks and fetches modified objects on behalf of all the WebCaches. The other point to note is that in the C-cache 60 minute case, the traffic generated is less than half of the traffic generated in the no-cache scheme.

The left plot of Figure 12 shows the change in hit ratio with the change in the number of caches. For independent caches, hit ratio decreases sharply (from 51% to 30% for I-cache 3.5 minutes case) as the number of caches increases.

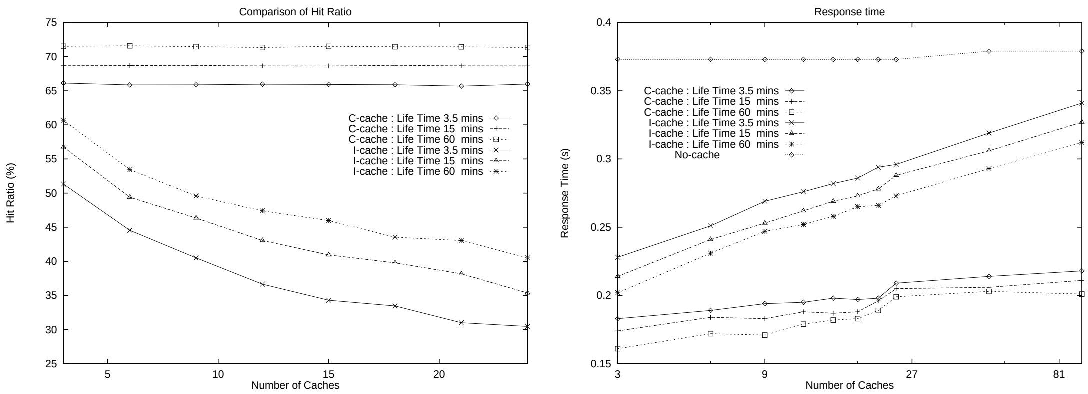


Figure 12: Hit Ratio and Response Time vs. Number of Caches

However, for coordinated caches, hit ratio remains constant. This happens because there is no centralized freshness checking mechanism in the case of independent caches and when requests are distributed over all the caches, the likelihood of finding a fresh copy of an object reduces. This implies lower hit ratio. On the contrary, since freshness of hot-objects is maintained across all the caches in DAC3, fresh objects are always found in the caches resulting in a constant hit ratio which is higher than that for independent caches.

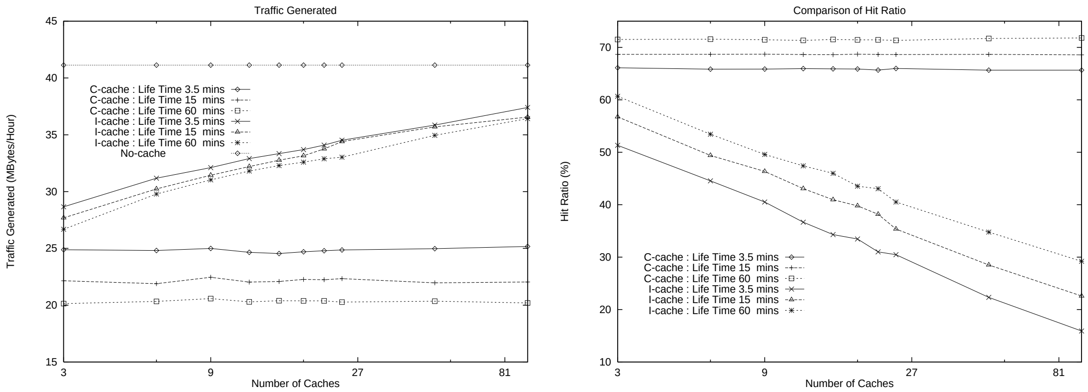


Figure 13: Traffic and Hit Ratio vs. Number of Caches

We present a second set of plots which show the effects of increasing the number of caches on response time, backbone trunk traffic and hit ratio more vividly. In these plots the number of caches is increased to 96. Of particular interest are the increase in response time from 200 ms to 310 ms (The right plot of Figure 12) for I-cache 60 minutes case, the increase in backbone trunk traffic to 35 MBytes/hr for 96 caches approaching that of the no-cache case (The left plot of Figure 13) and the sharp drop in hit ratio from 51% to 16% for I-cache 3.5 minutes case (The right plot of Figure 13). Side-by-side we also present a plot (left plot of Figure 14). showing the overhead in terms of the control traffic generated in DAC3 to enable the above kind of performance improvement. Note in particular that the control traffic increases linearly with the number of caches because each cache generates approximately the same amount of control traffic. Also, for a 96-cache system, the control traffic increases to about 0.2 MBytes/hr (or equivalently 0.5 Kbits/sec) which is still negligible compared to the amount of data traffic in the network.

Freshness

In the right plot of Figure 14, we compare the cache freshness in the DAC3 scheme and the independent cache scheme. When a object is hit in the cache, it may be fresh or stale. We measure the percentage of the fresh objects over the total hits which we call *freshness index*. To make two schemes comparable, we only measure the freshness index for the moderately frequently changing objects. The average lifetime of these objects are set to 3.5 minutes, 15 minutes and 60 minutes and the number of caches is varied from 3 to 24. In the 15 cache case, if the lifetime is 3.5 minutes, the freshness is 80% for the coordinated caches and it is 54% for the independent caches. When the lifetime is 15 minutes, the freshness for coordinated caches is 87% while that for independent caches is 74%. When the lifetime is longer (for example, 60 minutes), the freshness percentage is increased in both cases. While the number can be as high

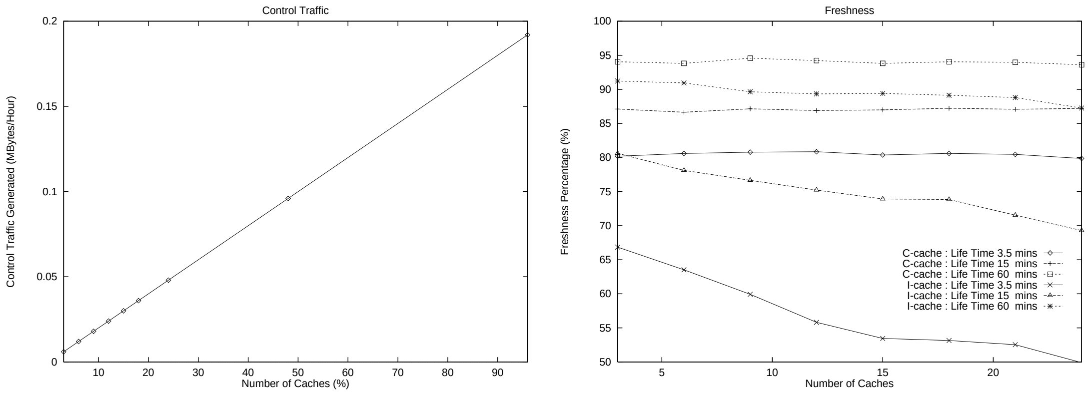


Figure 14: Control traffic and Freshness Index

as 94% for the coordinated caches, it can be as high as 89% for the independent caches. When the number of caches increases from 3 to 24, the freshness percentage for coordinated caches does not change much, while it decreases for independent caches. This is because the degree of sharing decreases for independent caches while it remains unchanged for coordinated caches with the increase in number of caches.

4 Summary and Future Work

In summary, this paper presents a novel distributed architecture of web caches with centralized control (referred to as DAC3) and evaluates it with respect to three important metrics: hit ratio, response time and traffic on backbone trunk links. The main finding of the paper is that such an architecture consistently provides improved performance over an architecture where the caches are independent. The degree of improvement depends on several factors, the most important of which is the hot-object threshold T_{ho}. [PF99] shows using analytical modeling how the optimal hot-object threshold T_{ho} can be computed. In addition to showing how a coordinated caching system can utilize the caching resources in an optimal manner with respect to the above three metrics, the paper also illustrates that such an architecture can consistently provide higher freshness of content and can balance load among the multiple caches in the network.

The authors view the network of caches as an overlay network on top of an IP network and believe that this network can be used in innovative ways to provide novel services. Future work will be focussed on systems issues such as defining an API for the network of caches and building new components which can be plugged into the architecture to provide new services. In addition, there is ongoing work on how to plan and design a network of caches for a backbone network service provider given the unique characteristics of its network.

References

[BCF+99] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. "Web Caching, and Zipf-like Distributions: Evidence, and Implications," *Proceedings of IEEE INFOCOM,* 1999.

[BDH+94] Bowman, P.B. Danzig, Hardy, Manber, M.F. Schwartz, and D. Wessels. "Harvest: A scalable, customizable discovery, and access system," *Technical Report CU-CS-732-94,* Department of Computer Science, University of Colorado, August 1994.

[CAB98] P. Cao, J. Almeida, and A.Z. Broder. "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *Proceedings of ACM SIGCOMM'98,* Vancouver, British Columbia, August 1998.

[CBM95] C. Cunha, A. Bestavros, and M.E. Crovella. "Characteristics of WWW client-based traces," *Technical Report TR-95-010,* Boston University, Computer Science Department, April 1995.

[CC95] R.L. Carter, and M.E. Crovella. "Dynamic Server Selection in the Internet," *Proceedings of the Third IEEE Workshop on the Architecture and the Implementation of High Performance Communication Subsystems (HPCS'95),* August 1995.

[CC96] R.L. Carter, and M.E. Crovella. "Dynamic server selection using bandwidth probing in wide-area networks," *Technical Report TR-96-007,* Boston University, Computer Science Department, March 1996.

[CDN+95] A. Chunkhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell. "A hierarchical internet object cache," *Technical Report 95-611,* University of Southern California, March 1995.

[CF99] M. Cieslak, and D. Forster. "Web Cache Coordination Protocol V1.0," Internet Draft *draft-ietf-wrec-web-pro-00.txt,*.

[CY97] K. ichi Chinen, and S. Yamaguchi. "An interactive prefetching proxy server for improvements of WWW latency," *Proceedings of INET'97,* Kuala Lumpur, Malayasia, JUne 1997.

[DFK97] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. Mogul. "Rate of Change and other Metrics: A Live Study of the WWW," *Proceedings of USENIX Symposium on Internet Technology, and Systems,* Monterey, CA, pp. 147-158, December 1997.

[DMF97] B.M. Duska, D. Marwood, and M.J. Feeley. "The measured access characteristics of world-wide-web client proxy caches," *Proceedings of USENIX Symposium on Internet Technology, and Systems,* Monterey, CA, December 1997.

[G94] S. Glassman. "A caching relay for the world wide web," *Proceedings of First International Conference on the World-Wide Web,* May 1994.

[GB97] S. Gribble, and E. Brewer. "System design issues for internet middleware services: Deductions from a large client trace," *Proceedings of USENIX Symposium on Internet Technology, and Systems,* Monterey, CA, December 1997.

[GB97a] S. Gribble, and E. Brewer. UCB home IP HTTP traces. http:/www.cs.berkeley.edu/gribble/traces/index.html, June 1997.

[GuS95] A.J.D. Guyton, and M.F. Schwartz. "Locating nearby copies of replicated internet servers," *Proceedings of ACM SIGCOMM'95,* Cambridge, MA, pp. 288-298, August 1995.

[GS96] J. Gwertzman, and M. Seltzer. "World-wide web consistency," *Proceedings of the 1996 USENIX Technical Conference,* January 1996.

[GS96a] J. Gwertzman, and M. Seltzer. "The Case for Geographical Push Caching," *Technical Report,* Harvard University.

[HGKM98] G.D.H. Hunt, G.S. Goldszmidt, R.P. King, and R. Mukherjee. "Network Dispatcher: a connection router for scalable Internet services," *Computer Networks and ISDN Systems,* 30 (1998), pp. 347-357.

[HM97] A. Heddaya, and S. Mirdad, "WebWave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents", *Proc. 17th IEEE Intl. Conf. on Distributed Computing Systems*, Baltimore, Maryland, May 1997.

[HMY97] A. Heddaya, S. Mirdad, and D. Yates, "Diffusion-based Caching Along Routing Paths", *Proceedings of 2nd Web Caching Workshop*, Boulder, Colorado, June 1997.

[HRW98] M. Hamilton, A. Rousskov, and D. Wessels, "Cache Digest Specification - version 5", http://www.squid-cache.org/CacheDigest/cache-digest-v5.txt, December 1998.

[KD99] M.R. Korupolu, and M. Dahlin. "Coordinated Placement and Replacement for Large-Scale Distributed Caches," *Proceedings of IEEE Workshop on Internet Applications,* pages 62-71, July 1999.

[KLM97] T.M. Kroeger, D.D.E. Long, and J. Mogul. "Exploring the bounds of web latency reduction from caching, and prefetching," *Proceedings of USENIX Symposium on Internet Technology, and Systems,* Monterey, CA, December 1997.

[KMM96] T.M. Kroeger, J. Mogul, and C. Maltzahn. Digital's web proxy traces. ftp:/ftp.digital.com/pub/DEC/traces/proxy/webtraces.html, August 1996.

[KW97] B. Krishnamurthy, and C. Wills. "Study of Piggyback Cache Validation for Proxy Caches in the WWW," *Proceedings of USENIX Symposium on Internet Technology, and Systems,* Monterey, CA, pp. 1-12, December 1997.

[KW97a] B. Krishnamurthy, and C. Wills. "Piggyback Server Invalidation for Proxy Cache Coherency," *Proceedings of the WWW-7 Conference,* Brisbane, Australia, pp. 185-194, April 1998.

[NLA97] National Laboratory for Applied Network Research. ftp:/ircache.nlanr.net/Traces/, July 1997.

[PF99] S. Paul, and Z. Fei. "Distributed Coordinated Caching," submitted for publication, *available from the authors: sanjoy@bell-labs.com,* December, 1999.

[PM96] V.N. Padmanabhan, and J.C. Mogul. "Using predictive prefetching to improve world wide web latency," *ACM SIGCOMM Computer Communications Review,* July 1996.

[RP98] S. Rangarajan, and S. Paul. "Load Balancing, Load Sharing and Fault-Tolerance through Weighted Redirection," *available from the authors: sanjoy@bell-labs.com,* November 1998.

[TDVK99] R. Tewari, M. Dahlin, H.M. Vin, and J.S. Kay. "Design considerations for distributed caching on the Internet," *Proceedings of 19th. International Conference on Distributed Computing Systems,* 1996.

[ULV97] UCB/LBNL/VINT Network Simulator - ns (version 2). http:/www-mash.cs.berkeley.edu/ns/

[W97] D. Wessels. "Squid internet object cache," http:/squid.nlanr.net/

[WC97] D. Wessels, and K. Claffy. "Internet Cache Protocol (ICP), version 2," *RFC-2186,* September 1997.

[WC97a] D. Wessels, and K. Claffy. "Applications of Internet Cache Protocol (ICP), version 2," *RFC-2186,* September 1997.

[WC97b] D. Wessels, and K. Claffy. "ICP and the Squid Web Cache," http:/www.ircache.net/squid/reading.html

[YBS99] H. Yu, L. Breslau, and S. Shenker. "A Scalable Web Cache Consistency Architecture," *Proceedings of ACM SIGCOMM'99,* Cambridge, MA, August 1999.

[Z29] G.K. Zipf. "Relative Frequency as a determinant of phonetic change," *Reprinted from the Harvard Studies in Classical Philology,* Volume XL, 1929.