Proxy Ecology - Cooperative Proxies with Artificial Life

James Wang
Department of Computer Science
Clemson University
Clemson, SC 29634
864-656-7678
jzwang@cs.clemson.edu

Ratan Guha
School of EECS
University of Central Florida
Orlando, FL 32816
407-823-2956
guha@cs.ucf.edu

ABSTRACT

Proxy servers have been widely used by institutions to serve their clients behind firewalls. Recently many schemes have been proposed to organize the proxy servers into a cooperative proxy cache system. However most of existing proxy cache schemes require manual configuration of the cooperative proxies based on the network architecture. In this paper, we propose a novel P2P proxy caching scheme using an individual based model. We borrow the ideas from the ecological system as well as the economical system to manage the cooperative proxies through data and information exchange among individual proxies. Data flow among the proxy nodes creates artificial life for the proxies. Proxy servers with artificial life can automatically configure themselves into a Virtual Proxy Graph. The aggregate effect of caching actions by individual peer proxies forms a proxy ecology which automatically distributes data to nearest clients and balances workload. Our simulation results show that the proposed proxy caching scheme tremendously improves system performance. In addition, the individual based design model ensures simplicity and scalability of the cache system.

Keywords

Proxy ecology; P2P proxy cache; individual based model; artificial life; group behavior.

1. INTRODCUTION

The World Wide Web (WWW) can be viewed as a large distributed information system that provides access to shared data objects. The rapid growth of the internet and the WWW has enabled an increasing number of users to access vast amount of information stored at geographically distributed sites. The number of Web sites on the WWW has grown exponentially since the start of Internet era. Figure 1 shows the growing numbers of Web sites on the Internet since 1996, based on the survey conducted by Netcraft[*].

The WWW contains a wide range of information, such as news, education, sports, entertainment, shopping, etc., attracting different interests. Due to bandwidth limitation, the performance of Web surfing is suffering from network congestion and server overloading. Especially when some special event happens, Web servers who have the related information always experience unusual number of HTTP requests on those information. Recent studies [1, 2, 3, 4, 5] have shown that caching popular objects at locations closer to clients is an effective solution to improving Web performance. Caching can reduce both network traffic and document access latency. By caching replies to HTTP requests and using the cached replies whenever possible, client-side Web caches reduce the network traffic between clients and Web servers, reduce the load on the Web servers and reduce the average user-perceived latency of document retrieval. Because HTTP was designed to be stateless for servers, client-driven caching has been easier to deploy than server-driven replication.

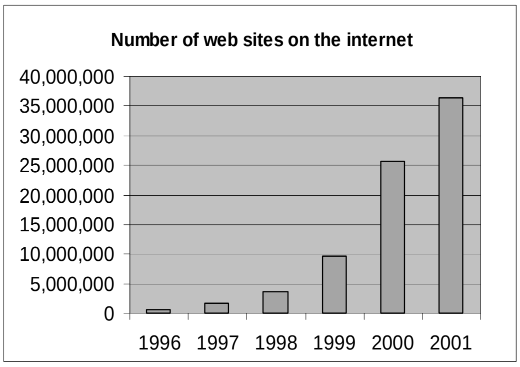


Figure 1: The growth of the WWW.

Caching can be implemented at various points on the network. Since early 90's, a special type of HTTP servers called "proxy" has been used to allow users hiding behind a firewall to access the internet [6]. For security reasons, organizations run proxy servers on their firewall machines. The proxy servers act as agents for the client browsers, to route the client requests to the remote Web servers and to send the replies back to the clients. Since the same proxy server is normally shared by all clients inside the firewall, those clients likely share common interests. Therefore caching documents from previous HTTP requests might result in future cache hits. Figure 2 depicts a typical proxy server.

Although using caching proxy can reduce both network traffic and document access latency, researchers have found that a single proxy cache can be a bottleneck due to its bandwidth and storage limitations [7, 8, 9, 10]. The proxy server tends to be overloaded as the client number increases

[*] http://www.netcraft.com/survey/

and hence causes a lot of cache missing. On the other hand, a single naive proxy cache system in a heterogeneous bandwidth environment might actually degrade the Web performance and introduce instability to the network [11]. To solve the problem, organizations normally use many servers to serve as a cooperative proxy cache system. Using current proxy caching schemes, those caching proxies can be manually configured as either a cluster or a hierarchy. Manual configuration of the cooperative proxy cache system requires knowledge of the network architecture as well as extensive administrators' involvement. Especially in a large corporation whose departments are physically located at different regions of the world, different departments maintain their own subnets and proxies. Configuring a distributed cooperative proxy cache system in such an environment requires significant collaboration among the administrators in different departments. The similar situation also exists in some single location institutions such as universities.

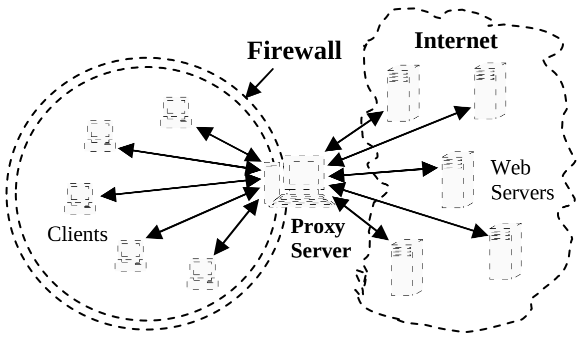


Figure 2: A proxy serving a group of clients inside the subnet.

There are many different cache architectures proposed in cooperative Web caching. Hierarchical cooperative caching was first introduced in the Harvest project [7]. The Harvest cache system organizes proxy caches in a hierarchy and uses a cache resolution protocol called Internet Cache Protocol (ICP) [12] to search the cached documents. Adaptive Web Caching [13] extends the Harvest cache hierarchy by grouping the cache servers into a tight mesh of overlapping multicast groups. There are many problems associated with the hierarchical cache systems [14, 15]. To setup a hierarchy, cache servers often need to be placed at the key access points in the network. It requires some manual configuration or significant coordination among the participating servers. In addition, higher levels of the hierarchy sometimes become the bottlenecks. To solve the problems, some distributed caching systems have been proposed [15, 16, 17, 18, 19, 20, 21, 22, 23]. In distributed Web caching systems, all cache proxies are viewed as the same level within the caching system. Several approaches have been proposed to design cooperative caches in a distributed environment. Internet Cache Protocol (ICP) [12] supports discovering and retrieving documents from sibling caches as well as parent caches despite its hierarchical origin. Adaptive Web Caching (AWC), an extension of Harvest cache hierarchy, is a cluster based distributed cooperative caching architecture. AWC relies on multicasting to discover and retrieve the cached documents. Similar to the idea of AWC, LSAM [16] is also a multicast based distributed Web cache architecture providing automated multicast push of Web pages to self-configuring interest groups. LSAM is essentially a hierarchical structure because the system includes server side pumps and client side filters managed by active middleware. Cache Array Routing Protocol (CARP) [17] divides the URL-space among an array of loosely coupled caches and lets each store only the documents of which URLs are hashed into it. Provey and Harrison proposed a hierarchical metadata-hierarchy [18], in which directory servers are used to replace the upper level caches in hierarchical cache structure, to efficiently distribute the location hints about the cached documents in proxies. Push Caching [15] proposed a similar distributed Web cache using a scalable hierarchy of location hints combined with caching these hints near clients. CRISP [19] cache adopts a centralized global mapping directory for caches. CacheMesh [20] builds a routing table for clients to forward the Web requests to the designated server who was selected to cache documents for a certain number of Web sites. Proxy sharing [21] tries to make multiple servers cooperate in such a way that a client can randomly pick a proxy server as the master server and the master server will multicast the requests to the other cooperative caches if it can not satisfy the client's request. Summary Cache [22] and Cache Digest [23] keep local directories to locate cached documents in the other caches. In those two cache systems, the cooperative servers have to exchange summary or digests of the documents in their caches to keep the local directories up to date.

In the hierarchical caching architecture, not only organizing and maintaining a cache hierarchy require significant administrative involvement, but also increasing levels of hierarchy tend to create bottleneck at higher levels. In distributed caching architectures, most schemes focus on maximizing the global cache hit ratio by implementing sophisticated directory lookup or search schemes. Increasing global hit ratio does not always imply reduction of request latencies [24] in distributed environment. To reduce the user request latencies, the Web documents need to be cached closer to clients. However, some of the current caching protocols often ignore the retrieving cost of the cached document. For instance, the popular Cache Array Routing Protocol (CARP) designates Web servers to certain proxy servers by hashing. The documents from the Web servers can only be cached in their hashed proxy servers. Using this protocol and some other similar protocols, clients get the cached documents only from the designated servers which might not be close to them. In those caching schemes, not all cache hits are good for user request latencies [24]. In addition, those cache protocols also create hot spots when some Web documents become very hot. To solve the hot spot problem, Karger et al proposed a consistent hashing technique to construct per-server distribution trees to balance the workload among proxy servers [25]. However implementing consistent hashing and random tree requires a lot of changes of current Internet infrastructure [26]. On the other hand, the distributed caching schemes, which use a centralized directory server or use multicast middleware to manage the complexity of Web caching, have to deal with high connection times, higher bandwidth usages and administrative issues [14]. Because most those schemes concentrate on reducing the miss rates, load balance among proxy servers are not thoroughly discussed. In reality, balancing the workload is very important in quality and fairness of services. Especially in a heterogeneous environment, load balance is not only affected by user request patterns, but also by characteristics of proxy servers, such as storage capacity and network bandwidth [27]. Kaiser et al. proposed a selective caching scheme [28] to distribute the web documents to cooperative proxies to achieve the load balance. But their approach focused on the clustering environment where all caching proxies are close in the network. Ideally a proxy cache system should possess the following properties [29]:

- *Minimized latencies:* Reducing user request latencies is the top priority of any cache system.
- *Robustness/Availability:* Availability is another important measurement of Web service. Self-recovery from failures is a very important feature for a proxy cache system.
- *Transparency*: A Web caching system should be transparent to users. It is desirable to have a self-configured and self-managed proxy cache system.
- *Efficiency*: A Web caching system should efficiently utilize all available resources in proxy servers.
- *Flexibility*: It is desirable that a Web caching system is not tied to any predetermined system configuration. Instead, it can adapt to the dynamic changing of user demand and network environment.
- *Stability*: A Web caching scheme should not introduce instabilities into network.
- *Load balancing*: A good caching protocol should automatically balance the workload among the cooperative proxies.
- *Scalability*: A good caching scheme should scale well along the increasing size and density of network and application.
- *Simplicity*: A Web caching mechanism should be simple to implement and easy to deploy. Complexity would hinder the scalability of the cache system and increase the cost of administration.

It is important to design and implement a Web caching scheme to possess all those properties. However, none of those aforementioned caching schemes satisfy all the requirements for such an ideal cache system. Scalability and simplicity are the biggest problems in current Web caching schemes. To solve the problems, we propose a novel P2P caching scheme using an individual based model in order to implement a self-configured, self-managed massively scalable cooperative proxy cache system. In this proxy cache system, cooperation among proxy servers is handled naturally by simulating an ecological system - flocking. Load balance is achieved by data caching and data replication based on demand. The design of this proxy cache system is so unique that it satisfies all desirable properties for a Web caching system mentioned above. We must note here that the proposed P2P proxy cache system is different from some client-level Web document sharing schemes [30, 31]. Those schemes adopt the concepts or search algorithms from some popular P2P file sharing systems [32, 33, 34]. However, they are not transparent to the clients and inherit the inevitable fairness, security, reliability and efficiency problems from the P2P file sharing systems as pointed out by recent studies [35, 36, 37].

The rest of the paper is organized as follows. In section 2, we discuss the fundamental of the proposed P2P proxy cache system and describe the design details. In section 3, we design a simulation model to exam our observations and prove the effectiveness of our proposed P2P proxy cache system. The simulation results are presented in section 4. We have our conclusion and discuss future studies in section 5.

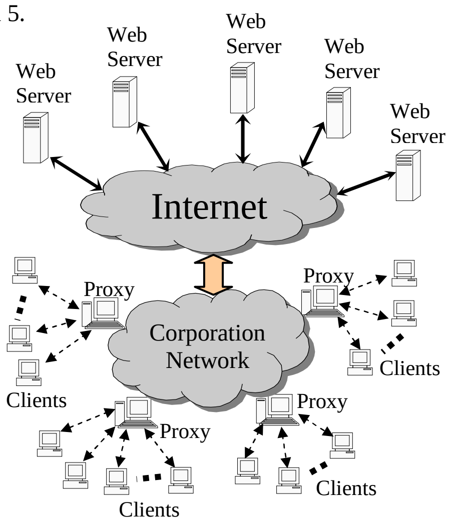


Figure 3: A typical distributed proxy cache system.

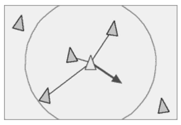
2. P2P PROXY CACHING SCHEME

A distributed proxy cache system consists of many proxy servers inter-connected through network. Each proxy server or a cluster of proxy servers normally serve a group of clients within an institution. In a large corporation, there might be many proxy servers or clusters distributed across a wide area network. Figure 3 depicts a typical distributed proxy cache system on the internet. In a distributed proxy cache system, proxies serve requests from their clients as well as requests from peer proxies. They fetch the requested documents either from Web servers or from the cooperative proxies to minimize the latencies for their clients. Due to historical and financial reasons, not all proxy servers have the same available caching resources, such as storage capacity and network bandwidth. Hence, managing a heterogeneous distributed proxy cache system is complicated. The complexity of management grows exponentially with the growth of network and cache system. Further more, using complex caching scheme adds more complexity to the cache system and in turn hinders manageability and scalability of the cache system.

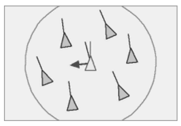
2.1 An Individual-Based Model

In real world, there are two proven mechanisms that can successfully self-manage massive distributed cooperative systems. One is the economic system in which millions of clothes can be distributed among the same magnitude number of people without the centralized control. The distribution of clothes follows a simple supply-by-demand policy. The other mechanism is the ecological system where within a specific environment natural selection creates cumulative advantages for evolving entities. For many years, people have enjoyed the beauty of bird flocks and fish schools in natural world. A flock exhibits many contrasts. It is made up of discrete birds yet overall motion seems fluids; it is simple in concept yet is so visually complex; it seems randomly arrayed and yet is magnificently synchronized. Perhaps most puzzling is the strong impression of intentional, centralized control. Yet all evidence indicates that flock motion must be merely the aggregate result of actions of individual animals acting solely based on the basis of their own local perception of the world.

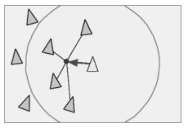
Researchers have tried to find a simple model to simulate the nature flock in computer animation until the paper "*Boids*†" published at SIGRAPH in 1987 [38]. Since then the *Boids* model has become an oft-cited example of principles of Artificial Life. The basic flocking model consists of three simple steering behaviors which describe how an individual boid maneuvers based on the positions and velocities of its nearby flockmates:



Separation: steer to avoid crowding local flockmates.



Alignment: steer towards the average heading of local flockmates.



Cohesion: steer to move toward the average position of local flockmates.

In the *Boids* model, interaction between simple behaviors of individuals produces complex yet organized group behavior. The component behaviors are inherently nonlinear, so mixing them gives the emergent group dynamics a chaotic aspect. At the same time, the negative feedback provided by the behavioral controllers tends to keep the group dynamics ordered.

A distributed proxy cache system can be viewed as a flock of individual cache proxies having life-like group behavior. A significant property of life-like behavior is *unpredictability* over moderate time scales while being predictable within a short time span. Data caching in a distributed proxy cache system possesses this property due to the *unpredictability* of Web requests over a longer period of time. In a shorter time frame, Web requests seem to be very predictable because of the flock like behavior of human interests. Thus modeling the distributed proxy cache system using similar approach as *Boids'* is feasible. Actually long before Internet flourished, flocks and schools were given as examples of robust self-organizing distributed systems in the literature of parallel and distributed computer systems [39]. The *Boids* model is an example of an individual-based model‡, in which a class of simulation used to capture the global behavior of a large number of interacting autonomous agents. Individual-based models are also used in biology, ecology, economics and other fields of study.

In this paper, we use individual-based model to design a self-configured, self-managed proxy ecology in which individual proxy servers exchange data and information using some simple rules. The aggregate effect of caching actions by individual proxy servers automatically distributes web documents to nearest clients and also automatically balances the workload.

2.2 Virtual Proxy Graph

As an individual-based cache model, the proxy needs to find its neighbors with whom it exchanges data and information. Unlike some other existing schemes, in which architecture of the proxy cache system is manually configured, individual proxy servers in our proxy cache system should automatically discover their neighbors and virtually link the cooperative proxies together into a Virtual Proxy Graph (VPG). By forming the VPG, proxies are restricted to only exchange data and information with their neighbors. The neighbor proxies help each others in searching for cached data and balancing workload. This is similar to Boids model where individual boid maneuvers based on the positions and velocities of its nearby flockmates. Figure 4 demonstrates a VPG with 7 nodes.

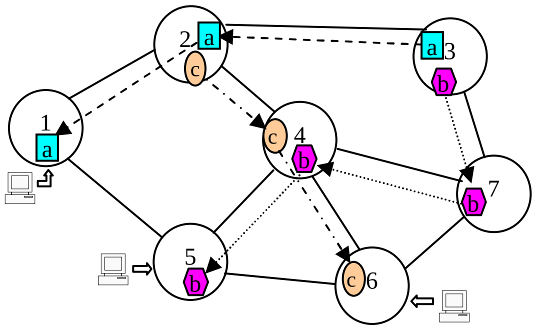


Figure 4: Data flow among the proxies in a VPG

The construction of the VPG follows some simple rules. First, neighbors must be close to each others. This rule makes sure the network distance between two neighbors is short so that communication latencies among the neighbors are low. Second, the number of allowable links to a proxy is determined by its available cache space, its network bandwidth, and its computing power. Normally a high power proxy server who has more cache space and large

† http://www.red3d.com/cwr/boids/index.html

‡ http://www.red3d.com/cwr/ibm.html

network bandwidth will link with more neighbors. On the other hand, a proxy server that has limited network bandwidth should not link with many neighbors. Assigning the number of links to a proxy based on its computing resources is the key for automatic load balance. We briefly discuss the automatic VPG configuration process in following paragraphs.

When a proxy server wants to join the proxy cache system, the following steps are needed:

1. ***Token request:*** A proxy server must request for a permission token before joining the proxy cache system. It first broadcasts a token-request message to all the nodes in the network asking for the token. If within certain period of time the proxy server does not get any response from any other nodes or no proxy has been found holding the token, it will create a token and assume itself to be the first proxy in the proxy cache system. If there are existing proxies in the proxy cache system, one of them must hold the token. This proxy will send the token to the requesting proxy if it has finished its own configuration, otherwise it tells the requesting proxy to wait. The other proxies in the cache system will only notify the requesting proxy that they are part of the proxy cache system.

2. ***Information gathering:*** Once the proxy gets the token, it starts configuring itself into the proxy cache system. It first multicasts a request-for-join message to the existing cache proxies. An existing cache proxy replies the request-for-join message with an acknowledge-to-join message that contains its characteristic information, including storage capacity, network bandwidth, the maximum allowed distance to its neighbors, and the number of its linked neighbors or the IP addresses of its linked neighbors. Based on information gathered from the other existing proxies, the requesting proxy decides which existing proxies to link with to form a new VPG.

3. ***Configuration:*** Many factors affect the number of links that a new proxy can establish. Besides its own storage capacity, network bandwidth and the user's tolerance to response latencies, the characteristics of all other exiting proxies also contribute to the final decision. In general, the neighbors should be close to each other in network in order to get fast responses for the requests sent to neighbor proxies; the number of links for each cache proxy should not exceed its processing capability. Without getting into too much detail, we give a simple description on how this new proxy selects its neighbors here. First, the new proxy determines the maximum allowed distance to its neighbors. It is decided by its user's tolerance to request latencies. Normally the round trip distance to the neighbors should not be larger than the average user request latency directly from the Web servers. Second, the new proxy needs to decide how many of those close proxies it wants to link to. Using the information it gathered from the existing proxies, the new proxy can calculate the average *Storage Capacity Per Link* (SCPL) and the average *Network Bandwidth Per Link* (NBPL) for the existing proxies. Then it sets up its allowable links to a number so that it SCPL and NBPL are comparable to it peers. Third, after the number of allowable links has been determined, the new proxy selects the closest proxies to link. For any selected proxy, the round trip distances from this proxy to the new proxy must be less than its maximum allowable neighbor distance. After selecting the links, the new proxy also recalculates the SCPL and NBPL for the involving proxies to make sure the new SCPLs and NBPLs are still comparable. Selection adjustment can be made at this point if necessary.

4. ***Updating VPG:*** After the new proxy selects its neighbors, it builds a neighbor table for later lookup and then notifies its neighbors on the selections. An entry in the neighbor table includes an ID which is normally a small integer and the IP address to the neighbor. If the IP addresses of the neighbors for all the existing cache proxies have been sent to the new proxy, a VPG can be built and stored locally for later reference. If each proxy only stores its neighbor table, the VPG is actually distributed among all the proxies. Nevertheless, storing the VPG locally at every proxy might be a good idea because it would definitely help in quick searching the cached documents. Because our individual based model does not require the individual proxy to exchange information with the non-neighbor proxies, storing the VPG locally at every proxy is only an enhancement option.

There are several advantages of constructing a VPG. With the VPG, the proxies only exchange the data and messages with their neighbors. So the management at each proxy is simpler than those schemes that use multicast or broadcast to retrieve documents from all other proxies. A VPG can be easily reconstructed based on the dynamic changing of workload and network environment. This reconfigurable feature makes the cache system not only adaptive to the changes in network environment but also robust to the failure of some proxies. With the VPG, the aggregate effect of data movement among the proxy neighbors creates a life-like group behavior that automatically distributes the data close to clients with less complexity. The VPG configuration process also provides a well balanced distributed cache structure that does not depend on the underlying network architecture.

2.3 Network Cache Model

Cache was originally designed for hierarchical storage systems in computer architecture for fast access the frequently accessed data. Traditionally a cache line consists of cached data, tag and state bits. Tag field is used to identify the data page or instruction; State bits are used for cache coherency protocols. The size and content of the State bits vary depending on the cache coherency protocols. Figure 5 shows a traditional cache line.

Cached Data	Tag	State Bits

Figure 5: A traditional cache line

Unlike in traditional hierarchical cache where cached data is normally a data page or an instruction from lower level storage devices, the cached data in Web caching is a Web document which may migrate to current proxy node from its neighbors instead of directly from the original Web server. To cope with the new data type and proxy cache architecture, a Web cache line should include more information than that in traditional cache line. Figure 6 depicts a Web cache line in our proxy cache system.

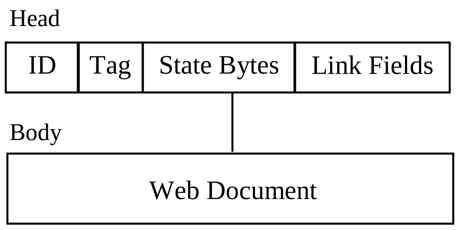


Figure 6: A web cache line.

A Web cache line is made up of two portions. The body portion is used to store the Web document itself. The head portion consists of ID, state bytes and link fields. The ID field contains an UUID which the URL of the cached Web document will be hashed to. The Tag is the name of the document. State bytes are used to store information for cache coherency control as well as some other statistical data, such as number of replicas of the Web document known to this node in the proxy cache system and shortest distance to the nearby replicas. Because the cached document might be replicated from other nodes in the proxy cache system, link fields are needed in a Web cache line to provide the link information. A link field is organized as a pair of integer <NID, Dist>. NID is an integer used to lookup the neighbor table to get the neighbor's IP address. Dist is round trip distance in finding the cached document through the neighbor specified by NID.

2.4 Data Search and Data Flow

The heart of this proposed proxy cache system is to link the Web documents across the proxies by flowing Web cache lines or Web cache line heads among the neighbor proxies. The data flow should bring the *Artificial Life* to the proxies and generate a *life-like* group behavior so that the aggregate effect of caching actions by individual cache proxies automatically distributes data close to clients interested in the cached data. To achieve this goal, we must find a set of well designed rules that individual cache proxies would use to route the search messages and cache the Web documents.

Efficiently searching the cached Web documents is the most important function in the proxy cache system. A good search algorithm should possess the following properties:

- ***Quickness:*** A search algorithm should quickly find the location of the cached Web document.
- ***Low cost:*** The overhead of the search should be low.
- ***Simplicity:*** The algorithm should be very simple to implement in individual proxies.
- ***Load balance:*** The search algorithm should not cause hot spots in the proxy cache system.

Because our P2P proxy cache system is designed on an individual-based model, the search algorithm runs on all proxies simultaneously and each proxy only queries its neighbors for Web documents. When a new request for a document comes in to certain proxy from its clients, the proxy server takes actions based on three different situations as follows:

1. ***The entire Web cache line is cached:*** In this case, the Web document is sent to the client and the request is satisfied.

2. ***Only head of the Web cache line is cached:***

   - If the proxy contains only the Head of the Web cache line, it checks the shortest distance to a replica stored in the Head. If the distance to the nearest replica is larger than its expected response time by requesting from the Web server directly, it issues a HTTP request to the Web server.
   - If the shortest distance is less than the expected response time by requesting directly from the Web server, the proxy searches the neighbor table to find the link with the shortest distance to a replica. Then a query is sent to the neighbor that link points to. A query message should contain ID and Tag of the Web document, the time when the message is sent, and maximum waiting time the requesting proxy will tolerate.
   - If the requesting proxy could not get a response within the maximum waiting time, a query will be issued directly to the Web server.
   - When a proxy gets a request from its neighbor, it checks current status of the query message first. If the waiting time is expired, the query message is discarded. This will prevent the query message being sent to unnecessary proxy nodes.
   - If the waiting time is not expired, the proxy checks the proper Web cache line status to see if the Web document is cached.
   - If only the head of the Web cache line is cached, it checks the link fields of the Web cache line and finds a link field having shortest distance to the cached replica. Then the query message is routed to its neighbor that the link points to.
   - Using this method, the nodes linked by Web cache lines relay the query message to eventually reach a proxy that has a cached Web document.

3. ***Nothing is cached at this proxy:***

   - If the proxy does not contain a Web cache line of the requested Web document, then this proxy node has to multicast a query message to all its neighbors. After that, the requesting proxy waits for the query results until the waiting time expires. A query message should include *ID* and

Tag of the Web document. The *ID* is created by hashing the Web document URL into an *UUID*. The *Tag* can be the name of the Web document. The message issuing time and maximum waiting time of the requesting proxy will also be included in the query message.

- If a proxy receives a query message from its neighbor, it first checks whether the message waiting time is expired. If the time is expired, it discards the query message. Otherwise, the proxy searches the Web cache lines in its cache to find a match.
- Once a proxy finds a Web cache line matching the *ID* and *Tag* given in query message, it sends a response to its neighbor where the query message comes from. Then the proxy involved in search routes the response all the way back to the requesting proxy.
- A response message must include a field to indicate the distance to the cached replica from the responding server and the time of the response message is sent. The proxies who route the response message should be able to create a Head of the Web cache line from the response message and cache the Head in their own caches to benefit the later search. The distance to replica field in the response message should be updated (increase) along the path on which the response message is propagated to the requesting proxy.
- During the response message travels back to the requesting server, a proxy may discard the response message if it finds the maximum waiting time in the original query message is expired.
- The requesting proxy checks all response messages received before the waiting time is expired and creates appropriate Web cache lines to benefit later requests. The requesting proxy then finds the shortest distance to a replica among in those new Web cache lines. The rest of the process would be same as in case 2.
- If the requesting proxy does not get any response within the maximum waiting time, it queries the Web document directly from the Web server.

Once a cached Web document is found in the proxy cache system, it can be routed to the requesting proxy along the way the query message was propagated. The proxies involved in search process then have a chance to cache the Web document at their caches to serve the later requests when they route the cached Web document to the requesting proxy. Alternatively, the cached Web document can be sent to the requesting proxy directly from the proxy where the cached replica being located. In this later case, we need include the IP address of the requesting proxy in the query message. Obviously, including requesting proxy's IP will increase the size of the query message. We must note here that sending the cached Web document directly to the requesting proxy does not necessarily reduce the network traffic in the proxy cache system, because the Web document might travel through the same set of proxies transparently due to the network routing. But routing the Web document via searching proxies would add process overhead to the intermediate proxies.

The effect of the data flow in a proxy cache system is illustrated by the VPG depicted in Figure 4. Assume document *a*, *b* and *c* are originally cached in nodes 3, 3 and 2 respectively. After clients at node 1, 5, and 6 request the document *a*, *b*, and *c* correspondingly, those documents will be routed to and replicated at the requesting proxy nodes through the search paths. The accumulative result of the data caching and data flow distributes the data to where they are mostly demanded and hence reduces the user response latencies. If we store the VPG in every proxy nodes, we should be able to find a better search algorithm. We will not discuss the search algorithm with the VPG stored locally because it is out of the scope of this paper.

2.5 Cache Replacement

An efficient cache replacement algorithm can increase the cache hit ratio. A good cache replacement algorithm considers many parameters such as storage capacity, access frequency to the objects, size of the objects, user access prediction, and some other historical statistic information. Besides the traditional Least Recent Used (LRU) and Lease Frequently Used (LFU) schemes, some other replacement policies have been proposed in recent years. Pitkow and Pecker [40] proposed a cache replacement algorithm based on dynamic user access patterns. A size weighted Web cache replace policy is discussed in [41]. A cost based replacement algorithm is proposed in [42]. Some LRU variants, such as LRU-MIN [43], LRU-Threshold [43], Size-Adjusted LRU [44] have also been discussed in past years. Rizzo and Vicisano replace the Web document based on Lowest Relative Value (LRV) [45]. A Least Normalized Cost Replacement algorithm (LCN-R) [46] employs a rational function of the access frequency, the transfer time and the size to select replacement victims. A sample based randomized algorithm is discussed in [47]. Starobinski and Tse introduced new randomized replacement policies [48] based on an extended version of the IR model. Kelly et al present a biased cache replacement algorithm [49] which is a simple generalization of LFU algorithm that is sensitive to varying levels of server valuation for cache hits. Some studies [50, 27] have found that Web access pattern follows a Zipf like distribution and proposed the cache replacement policies according to this observation.

Most existing cache replacement policies are proposed for a single proxy or a proxy cluster. Cache replacement policies have not been thoroughly studied in a cooperative proxy environment. Especially, our proposed proxy cache system is an individual-based cooperative proxy cache system. Unlike some other proxy cache systems where a directory server is aware of all cached documents in the cache system, the individual proxy in our proxy cache system has only limited global information from its neighbors. On the other hand, querying the other proxies for global data distribution information is not desirable. Thus the cache replacement policy can only base on individual proxy's own local perception of the cache system. In addition, our network cache model divides the Web cache line into two portions. The head portion contains link information for cached Web document. So our cache replacement algorithm must treat the head and body of the Web cache line differently. Usually we should leave the head of the Web cache line in the cache system as long as possible to maintain the links among the cached replicas. In case we have to remove the entire Web cache line, an invalidating message should be sent to its neighbors to update the link information in the related Web cache lines. The state bytes also play an important role in selecting the victim for replacement. For instance, it might not be wise to replace a cached Web document when it is the last replica in the linked proxy group.

When a proxy runs out of cache space, a replacement victim must be selected. The criteria for choosing the replacement victim include access frequency of the document, size of the document, number of replicas of the document and time since the last access to the document. Assume access frequency, size, replica number and time since last access are *f, s, r* and *t* respectively for a cached Web document, the replacement value of the document *V* should be a function of those four parameters: $V = F(f, s, r, t)$. For instance, *V* could be calculated using the following simple function:

$$V = \frac{\alpha}{f} + \beta \cdot s + \mu \cdot r + \nu \cdot t \qquad (1)$$

where α, β, μ, ν are some predetermined constants. We calculate replacement value *V* for all cached Web document and choose the document whose *V* value is largest as the replacement victim. We may determine the four constants (α, β, μ, ν) using experimental study. Due to space limitation, we will not discuss how to determine those constants. Instead we use the simple LRU algorithm in our simulation in this paper. We may also use another enhancement option for our cache replacement policy. When a proxy server runs out of cache space, it may flow its cached replica to one of its neighbors instead of removing the Web document from the cache system.

2.6 Cache Coherency

Currently there are two types of Web cache coherency protocols provide different level of cache consistency. Strong cache consistency is normally implemented through *client validation* or *server invalidation*. Client validation is implemented by client polling server for every access [51]. Server invalidation is accomplished by server sending an invalidating message to the clients whenever there is a change on the Web document at server side [52, 53]. Since an ideal proxy cache system should be transparent to clients, client validation is not a good approach. On the other hand, since our cache coherency protocol should not depend on server actions, we will not use the server invalidation approach as well. Contrary to the strong cache consistency protocols, the other type of cache coherency protocols who provide weak cache consistency is very popular in Web caching. The adaptive TTL [54] and its variants [55] take advantage of the fact that file lifetime distribution tends to be bimodal [56]. They handle the problem by adjusting a document's Time-To-Live (TTL) based on observations of its life-time. Most existing proxy servers [6, 7, 12] use the adaptive TTL approach. Another weak cache consistency protocol is Piggyback Cache Validation (PCV) [57] which capitalizes on requests sent from the proxy cache to the Web server to improve cache coherency.

Cache coherency protocols influence the performance of a cooperative proxy cache system [58]. In our proxy cache system, a cache coherency protocol not only needs to maintain the consistency between the original Web document and the cached Web document, but also needs to maintain the consistency among the cached documents in cooperative proxies. Furthermore our cache coherency protocol must be individual-based, i.e., individual cache proxy validates/invalidates the Web documents based on its local prospect of the replica. Because of our unique network cache model, our cache coherency protocol can take advantage of peer cooperation to design a Peer Invalidation (PI) method to maintain the consistency of the cached documents in the proxy cache system. Our cache coherency protocol is described as follows.

Each individual cache node maintains a Time-To-Live (TTL) field in cached Web document's URL. TTL is a priori estimate of how long the document will remain unchanged. This field is supported by current HTTP protocol for cache consistency. Whenever there is a request for the cached Web document, the proxy checks if the TTL of the document is expired. If the TTL is expired, the cache proxy sends an "if-modified-since" HTTP request to the Web server. This special HTTP request contains the URL of the document and a timestamp. Upon receiving the request, the Web server checks whether the document has been modified since the timestamp. The Web server returns the status code "200" and the new data to the proxy cache if the data has been modified. Otherwise, a code "304" is returned.

Once the new data received at the proxy, it invalidates the other cached replicas in the proxy cache system. The Peer Invalidation (PI) process can be implemented in two different modes:

Aggressive Mode: The invalidating message will be propagated to all proxy servers in the cache system so that all staled replicas can be invalidated.

Conservative Mode: The invalidating message will only be sent to the neighbors that current Web cache line has a link to so that only the staled replicas in a linked group will be invalidated.

The Peer Invalidation may be invoked after a proxy has retrieved a Web document directly from Web server due to the existing cached replicas are too far from the current proxy. Final step of the invalidation process is to update the staled replicas. A staled replica can be removed from the cache system or be replaced by the new Web document. In either case, the relative Web cache lines need to be modified.

Besides maintaining the cache consistency, there are some other fields need to be updated to reflect current state of the cached Web replicas, whenever a replication or removal takes place. For instance, when a Web document is replicated or a cached Web replica is removed, the number-of-replica fields and the related link fields in every linked web cache lines have to be updated. The updates on any proxy node will be solely based on the information passed in by its neighbors and the information stored locally. The proxy nodes will also propagate the updated information to the linked neighbors for them to update their state bytes and link bytes accordingly.

2.7 Dynamic Reconfiguration

Availability is an important measurement of the service of quality for the distributed systems [59, 60]. When some proxies crash or shut down for maintenance, the other proxies in the P2P proxy cache system should continue operating normally without being noticed by their respective clients, although the efficiency of the Web caching and the load balance might temporarily suffer from the failures of those proxies. Because our proxy cache system is not tied to any predetermined hardware configuration or network structure, it can automatically recover from the failure of some proxies and automatically reconfigure a new VPG to efficiently serve the clients with the available resources. The automatic system reconfiguration has not been studied thoroughly in existing proxy cache systems because some of them require manual configuration and the others rely on the network routing protocols for failure recovery. The dynamic reconfiguration scheme needs deal with two scenarios.

A proxy server shuts down for maintenance: The proxy being brought down should inform the neighbors of its departure. The neighbors would update the cached information at their sites and propagate the information to the other proxies to reflect the current state of the system. The proper actions should be taken to reconfigure the VPG.

A proxy server crashes: If a proxy has not received any query or response message from one of its neighbor for certain period of time, it sends a status checking message to that proxy in order to see if it is still alive. Once a proxy is determined to be dead, the similar reconfiguration process should be invoked.

3. SIMULATION MODEL

There are many factors affect the performance of the system. Those factors include user request pattern, storage capacity of the proxy servers, and network bandwidth of the proxy servers. In this paper, we focus on study the feasibility of our proxy caching scheme. Without loss of generality, we configure a proxy cache system based on some simple assumptions. Then we examine the hit ratios and user response times on this simplified simulation model. First we assume there are 64 proxy servers existing in the network. The distribution of those 64 servers is so uniform that we can automatically configure them into a mesh-like VPG so that all proxies have 4 neighbors each, excepting those proxies on the edge and the corner. Those edge and corner proxies have 3 and 2 neighbors each respectively. We further assume all the servers are identical in computation power, storage capacity and network bandwidth. The distances between the neighbors are all equal.

We assume the users have equal probability issuing Web requests at any proxies. We also assume all Web documents have equal size. The response time for a Web document is the time when first part of the document arrives at client's workstation. We compare the user response times using our proxy cache system with that using a single proxy server. We assume the distance from a client to its proxy server is 100 ms. If the requested document is cached in the proxy, the user response time is 200ms ignoring the other overheads. We assume the distance between the neighbor proxies is 200 ms, which is double the distance from client to its proxy. So if a requested document is found in a neighbor node, the user response time is 600 ms. We also assume there are 10,000 distinct Web documents on the Web servers. The average distance from the clients to the Web servers is 1500 ms. Thus the user response time for a cache miss is 3000 ms, ignoring all other overheads. Those assumptions are based on the data collected by running the benchmark Polygraph[§] on a Web caching appliance developed by Swell Technology[**].

Web requests follow a Zipf-like distribution according to our observation [27] and some other studies [48, 50]. The access frequency for each Web document i is determined as follows:

$$f_i = \frac{1}{i^z \cdot \sum_{j=1}^{m} 1/j^z},$$

where m is the number of the Web documents in the system, and $0 \le z \le 1$ is the Zipf factor [27]. A larger z value corresponds to a more skew condition, i.e., some objects are accessed considerably more frequently than other objects. When $z = 0$, the distribution is uniform, i.e., all the objects have the same access frequency. We simulate our proxy caching scheme on this automatically configured VPG. Each node in the VPG performs as a single proxy as well.

4. SIMULATION RESULTS

We could have used Web traces[††] to evaluate our P2P proxy cache system. However, based on study conducted by Breslau et al [50], Web requests follow zipf-like distribution with various zipf factors in different Web caches. We want to study how the Web request skew conditions affect the system performance, so using synthetic Web requests are easier to implement and using trace based performance study won't help more in our evaluation. Our simulator can run in two different modes, individual mode and cooperative mode. The individual

[§] http://www.measurement-factory.com/

[**] http://www.swelltech.com/products/caching.html

[††] http://www.web-caching.com/traces-logs.html

mode is used to simulate the single proxy server. The cooperative mode is used to simulate our P2P proxy cache system. We collect the statistic data on simulations running in those two modes. As in most of the previous studies, we use hit ratio and average response time as system performance metrics. Total of 200,000 Web requests issued to the cooperative proxy cache simulator. We start to collect statistic data after the first 50,000 Web requests being processed to avoid the inaccurate statistic data due to the startup of the simulation. We observe the user response times and cache hit ratios at all proxies.

4.1 Performance under Different Cache Rates

In this performance study, we vary the cache rate on individual cache proxy from 0.2% to 2% while keep the Zipf factor at 0.7. Increasing the cache rate will increase the cache hit ratios and in turn improve the system average response time for requests. This is true in both individual proxy server and the cooperative proxy cache system. Simulation study shows our P2P proxy cache system increases cache hit ratios tremendously over the individual proxy server as shown in Figure 7.

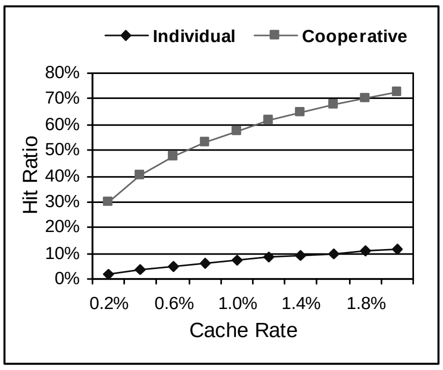


Figure 7: Hit ratios under different cache rates

When each individual proxy caches only 1% of the total Web documents, the average hit ratio for individual proxies is only 7%. On the other hand, our automatically configured P2P proxy cache system yields 57% hit ratio. In terms of user response times, Figure 8 shows our system also outperform individual proxy cache by large margin.

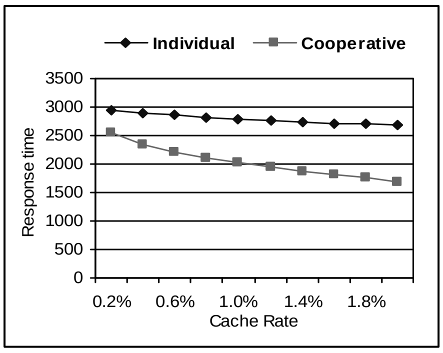


Figure 8: Response times under different cache rates

4.2 Performance under Various Zipf factors

Access skew condition determines the locality of the Web access. The cache hit ratios increase when data access pattern are increasingly skewed. In this simulation, we maintain the cache rate on each proxy at 1%. We change the Zipf factor from 0.4 to 1. As expected, when Zipf factor increases, the hit ratios of the proxy cache increase in both individual proxy server and the P2P cooperative proxy cache system. We do not show data when Zipf factor is less than 0.4 because the Web access is highly skewed according to various studies [27, 50]. As shown in and Figure 10, our P2P proxy cache system outperforms the individual proxy server by very large margin. For instance, when Zipf factor is at 0.6, the individual cache proxy's hit ratio is less than 5% while our P2P proxy cache system yields more than 50% hit ratio as shown in Figure 9.

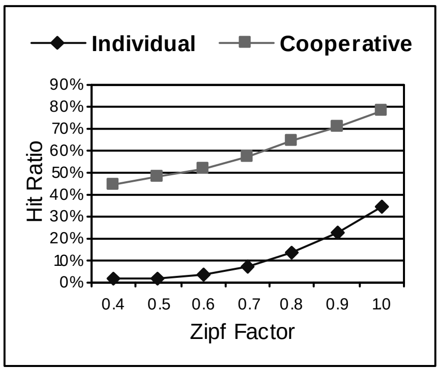


Figure 9: Hit ratios under various Zipf factors.

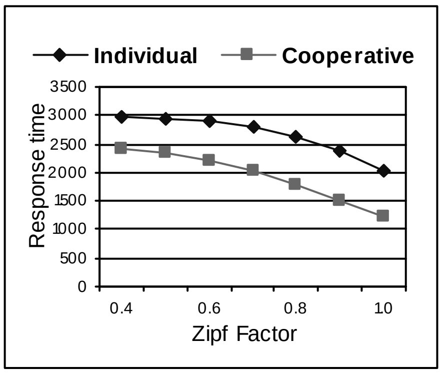


Figure 10: Response times under various Zipf Factors.

5. Concluding Remarks and Future Studies

In this paper, we proposed a novel P2P cooperative proxy cache system using the individual-based cache model. The accumulative results of the individual caching actions by all proxies automatically distribute the data close to the clients. Those caching actions create *artificial life* for the cooperative proxies. The system can be self-configured into a Virtual Proxy Graph using simple rules. Based on demand, data cache and data movement in our cache system create proxy ecology. This unique system design simplifies the management of the proxy cache system. Our simulation results indicate the effectiveness and feasibility of our cache system. We are currently conducting further investigation on the cache coherency and cache replacement options. Dynamic reconfiguration of the cache system is another topic to be studied. We are also implementing a prototype using our proposed proxy caching scheme.

REFERENCES

[1] Peter B. Danzig, Richard S. Hall, and Kurt J. Worrell. A Case for Caching File Objects Inside Internetworks. *In proceedings of ACM SIGCOMM*, pages 239-248, September 1993.

[2] R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich. Web Proxy Caching: The Devil Is in the Details. *ACM Performance Evaluation Review,* 26(3), pages 11-15, December 1998.

[3] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox. Caching Proxies: Limitations and Potentials. In *Proceedings of 4th International World Wide Web Conference*, Boston, Massachusetts, USA, December 1995.

[4] A. Bestavros, R. L. Carter, and M. E. Crovella. Application-level Document Caching in the Internet. *In Proceedings of the Second International Workshop on Services in Distributed and Networked Environments*, Whistler, British Columbia, 1995.

[5] T. M. Kroeger, D. D. E. Long, and J. C. Modul. Exploring the Bounds of Web Latency Reduction From Caching and Prefetching. *In Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems,* Monterey, CA, December 1997.

[6] A. Luotonen and K. Altis, *World Wide Web proxies*, Computer Networks and ISDN systems, First International Conference on WWW, 27(2):147-154, April 1994.

[7] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrel, A Hierarchical Internet object cache, *proceedings of USENIX Annual Technical Conference*, pp. 153-164, San Diego, CA, January, 1996.

[8] R. Malpani, J. Lorch and David Berger. Making World Wide Web Caching Servers Cooperate. *the Fouth International World Wide Web Conference*, Boston, Massachusetts, December 1995.

[9] Ratan K. Guha and James Z. Wang. Improving Web Accessing Efficiency using P2P Proxies. *in Proceedings of 4th International Workshop on Distributed Computing*, Kolkata, India, December 2002.

[10] James Z. Wang and Ratan K. Guha. Proxy Ecology - Cooperative Proxies with Artificial Life. *In Proceedings of IEEE/WIC IAT-2003*, Halifax, Canada, October 2003.

[11] A. Feldmann, R. Caceres, F. Douglis, G. Glass and M. Rabinovich, Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments, *Proceedings of InfoCom (1)*, pages 107-116, 1999.

[12] Duane Wessels, K Claffy, ICP and the Squid Web Cache, *IEEE Journal on Selected Areas in Communication*, 16(3):345-357, 1998.

[13] Scott Michel, et al., Adaptive Web Caching: Towards a New Caching Architecture, *the 3rd International WWW Caching Workshop*, Manchester, England, June 1998.

[14] P. Rodriguez, C. Spanner and E. W. Biersack, Web Caching Architecture: Hierarchical and Distributed Caching, *the 4th International WWW Caching Workshop*, San Diego, CA, March 1998.

[15] R. Tewari, M. Dahlin, H. Vin and J. Kay, Beyond Hierarchies: Design Consideration for Distributed Caching on the Internet, *Technical Report: TR98-04, Department of Computer Science, University of Texas at Austin*, February 1998.

[16] Joe Touch, The LSAM Proxy Cache – a Multicast Distributed Virtual Cache, *the 3rd International WWW Caching Workshop*, Manchester, England, June 1998.

[17] V. Valloppillil and K. W. Ross, Cache array routing protocol v1.0, *Internet Draft, <draft-vinod-carp-v1-03.txt>*, February 1998.

[18] D. Dovey and J. Harrison, A Distributed Internet Cache. *In 20th Australian Computer Science Conference*, February 1997.

[19] S. Gadde, M. Rabinovich and J. Chase, Reduce, Reuse, Recycle: A Approach to Building Large Internet Caches, *in Workshop on Hot Topics in Operating Systems*, pp. 93-98, May 1997.

[20] Zheng Wang, Cachemesh: A Distributed Cache System for World Wide Web, *the 2rd NLANT Web Caching Workshop*, June 1997.

[21] R. Malpani, J. Lorch and David Berger, Making World Wide Web Caching Servers Cooperate, *the Fouth International World Wide Web Conference*, Boston, MS, December 1995.

[22] L. Fan, P. Cao, J. Almeida and A. Broder, Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol, *IEEE/ACM Transactions on Networking*, 8(3):281-293, 2000.

[23] Alex Rousskov and Duane Wessels, Cache Digests, *Computer Networks and ISDN Systems,* 30(22-23):2155-2168, June 1998.

[24] M. Rabinovich, J. Chase and S. Gadde, Not All Hits Are Created Equal: Cooperative Proxy Cache Over a Wide-Area Network, *Computer Networks and ISDN Systems*, 30(22-23):2253-2259, November 1998.

[25] David Karger et al., Consistent hashing and random trees: Distributed cachine protocols for relieving hot spots on the World Wide Web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pages 654-663* , 1997

[26] David Karger et al., Web Caching with Consistent Hashing, *the 8th international WWW conference*, Toronto, Canada, May 11-14, 1999.

[27] James Z. Wang and Ratan K. Guha, Data Allocation Algorithms for Distributed Video Servers. *In Proceedings of ACM Multimedia*, pages 456-459, Marina del Rey, California, November 2000.

[28] M. Kaiser, K. C. Tsui and J. Liu, Self-organized Autonomous Web Proxies, *in proceedings of AAMAS 2002*, pages 1397-1404, July 2002.

[29] Jia Wang, A Survey of Web Caching Schemes for the Internet, *ACM Computer Communication Review*, 25(9):36-46, 1999.

[30] S. Iyer, A. Rowstron and P. Druschel, SQUIRREL: A decentralized peer-to-peer web cache, *in Proceedings of Principles of Distributed Computing (PODC 2002)*, Monterey, California, July 2002.

[31] L. Xiao, X. Zhang, and Z. Xu, On Reliable and Scalable Peer-to-Peer Web Document Sharing, *Proceedings of 2002 International Parallel and Distributed Processing Symposium, (IPDPS'02)*, Fort Lauderdale, Florida, April 2002.

[32] I. Clarke, O. Sandberg, B. Wiley and T. W. Hong, Freenet: A Distributed Anonymous Information Storage and Retrieval System, *Lecture Notes in Computer Science*, Volume 2009, Pages 46+, 2001.

[33] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications, Proceedings of the 2001 ACM SIGCOMM Conference, Pages 149-160, San Diego, CA, August 2001.

[34] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, pages 329-350, November, 2001.

[35] Beverly Yang and Hector Garcia-Molina, Comparing Hybrid Peer-to-Peer Systems, The VLDB Journal, pages 561-570, September 2001.

[36] E. Adar and B. Huberman, Free riding on gnutella, *Technical Report*, Xerox PARC, August 2000.

[37] S. Saroiu, P. K. Gummadi and S. D. Gribble, A Measurement Study of Peer-to-Peer File Sharing Systems, Proceedings of Multimedia Computing and Networking 2002 (MMCN '02), San Jose, CA, USA, January 2002.

[38] Craig W. Reynolds, Flocks, Herds, and Schools: A Distributed Behavioral Model, in *Computer Graphics*, 21(4) (*SIGGRAPH '87 Conference Proceedings*) pages 25-34, 1987.

[39] L. Kleinrock, Distributed System, invited paper for *ACM/IEEE-CS Joint Special Issue: Communications of the ACM*, Vol. 28, No. 11, pp. 1200-1213, November 1985.

[40] Jim Pitkow and Mimi Recker. A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns. *the 2nd International World Wide Web Conference*, Chicago, Illinois, USA, October 1994

[41] S. Williams, M. Abrams, C.R. Standridge, G. Abdulla, and E.A. Fox. Removal Policies in Network Caches for World-Wide Web Documents. *In Proceedings of the ACM SIGCOMM '96 Conference*, August 1996.

[42] Pei Cao and Sandy Irani. Cost-Aware WWW Proxy Caching Algorithms. *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA, 1997.

[43] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox. Caching Proxies: Limitations and Potentials. In *Proceedings of 4th International World Wide Web Conference*, Boston, Massachusetts, USA, December 1995.

[44] C. C. Aggarwal, J. L. Wolf and P. S. Yu. Caching on the World Wide Web. *Knowledge and Data Engineering*, Volume 11, Number 1, Pages 95-107, 1999.

[45] Luigi Rizzo and Lorenzo Vicisano. Replacement policies for a proxy cache. *IEEE/ACM Transactions on Networking*, Volume 8, Number 2, Pages 158-170, 2000.

[46] P. Scheuermann, J. Shim, and R. Vingralek. A case for delay-conscious caching of Web documents. *Computer Networks and ISDN Systems*, volume 29, number 8-13, pages 997-1005, 1997.

[47] Konstantinos Psounis and Balaji Prabhakar. A Randomized Web-Cache Replacement Scheme. in Proceedings of IEEE INFOCOM 2001, pages 1407-1415, April 2001.

[48] David Starobinski and David N. C. Tse. Probabilistic methods for web caching. *Performance Evaluation*, Volume 46, Number 2-3, Pages 125-137, 2001.

[49]T. P. Kelly, Y. M. Chan, S. Jamin and J. K. MacKie-Mason. Biased Replacement Policies for Web Caches: Differential Quality-of-Service and Aggregate User Value. *Proceedings of the 4th International Web Caching Workshop*, 1999.

[50] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. *in proceedings of IEEE INFOCOM'99*, pp. 126-134, March 1999.

[51] Beomseok Nam and Kern Koh. Periodic Polling for Web Cache Consistency. *WebNet (1)*, Pages 800-804, 1999.

[52] Chengjie Liu and Pei Cao. Maintaining Strong Cache Consistency in the World-Wide Web. *In Proceedings of the 17th International Conference on Distributed Systems*, May 1997.

[53] J. Yin, L. Alvisi, M. Dahlin and A. Iyengar. Engineering server-driven consistency for large scale dynamic Web services. *In Proceedings of the 10th International World*

Wide Web Conference, pages 45-57, Hong Kong, May 2001.

[54] Vincent Cate. Alex -- A Global File System. *Proceedings of the USENIX File System Workshop*, pages 1-11, Ann Arbor, Michigan, 1992.

[55] James Gwertzman and Margo Seltzer. World Wide Web Cache Consistency. *In Proceedings of the 1996 USENIX Annual Technical Conference*, San Diego, California, January 1996.

[56] M. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. Ousterhout. Measurement of a distributed file system. *In proceedings of 13th ACM Symposium on Operating System Principles*, pages 198-211, October 1991.

[57] B. Krishnamurthy and C. E. Wills. Study of Piggyback Cache Validation for Proxy Caches in the World Wide Web. *USENIX Symposium on Internet Technologies and Systems*, 1997.

[58] Victor Sosa and Leandro Navarro. Influence of the Document Validation/Replication Methods on Cooperative Web Caching Architectures. *Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS'02), Western Multiconference (WMC'02),* Pages. 238-245, San Antonio, Texas, January 2002.

[59] F. Cristian. Automatic Reconfiguration in the Presence of Failures. *Proceedings of the International Workshop on Configurable Distributed Systems*, IEE, London, pp. 4-17, March 1992.

[60] Y. Saito, B. N. Bershad and H. M. Levy. Manageability, Availability and Performance in Porcupine: A Highly Scalable, Cluster-based Mail Service. 17th ACM *Symposium on Operating Systems Principles*, Pages 1-15, December 1999