Improving Web Access Efficiency Using P2P Proxies

Ratan K Guha

SEECS- Computer Science
University of Central Florida
Orlando, FL 32816, USA
407-823-2956
guha@cs.ucf.edu

James Z. Wang

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

Abstract. Recent studies have shown that caching data at the proxy server can improve the user response time for web document retrieval. However, a single cache proxy has a limited cache space, thus the amount of data that can be cached is limited. In this paper, we organize the distributed proxy servers into a group of peer-to-peer cache proxies. By exploiting the aggregate cache space and computing power, we can reduce the average user response time and improve our quality of services. Unlike some previous works that achieve the similar results by replacing the single proxy server with a cluster of servers, we simply link the existing distributed proxy servers using a set of rules for connection, data cache and data routing to build a self-organized scalable distributed P2P proxy caching system. Our simulation has proven the feasibility and effectiveness of our cache system. In addition, our P2P proxy cache system is configured using individual based model and is easy to implement in a large scale distributed environment.

1 Introduction

Although the Internet backbone capacity increases as 60% per year, the demand for bandwidth is likely to outstrip supply for foreseeable future as more and more people blend the WWW into their daily life. The WWW contains a wide range of information, such as news, education, sports, entertainment, shopping. Due to the bandwidth limitation, the performance of Web surfing is suffering from network congestion and server overloading. Especially when some special event happens, the web servers who have the related information always experience unordinary number of HTTP requests on those information. Recent studies have shown that caching popular objects at locations close to the clients is an effective solution to improve 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.

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 [1]. Although using caching proxy server 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. As the number of clients increases, the proxy server tends to overloaded and hence causes a lot of cache missing. In a heterogeneous bandwidth environment, a single naive proxy cache system might actually degrade the Web performance and introduce instability to the network. [2] To solve the problem, organizations normally use many servers to serve as the cooperative proxies.

There are many different cache architectures proposed in cooperative web caching. Hierarchical Web caching cooperation was first introduced in the Harvest project [3]. The Harvest cache system organizes caches in a hierarchy and uses a cache resolution protocol called Internet Cache Protocol (ICP) [4] to search the cached document. Adaptive Web Caching [5] 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 [6,7]. 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. In distributed web caching systems [7,8,9,10,11,12,13,14,15], 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. Internal Cache Protocol (ICP) [4] 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 [8] is also a multicast based distributed web cache architecture providing automated multicast push of web pages to self configuring interest groups. Cache Array Routing Protocol (CARP) [9] divides the URL-space among an array of loosely coupled caches and lets each store only the documents whose URLs are hashed into it. Provey and Harrison proposed a hierarchical metadata-hierarchy [10], 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 [7] proposed a similar distributed internet cache using a scalable hierarchy of location hints combined with caching of these hints near clients. CRISP [11] cache adopts a centralized global mapping directory for caches. Cachemesh [12] 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 [13] 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 [14] and Cache Digest [15] keep local directories to locate cached documents in the

other caches. The cooperative servers exchange summary or digests of the documents in their caches.

In hierarchical caching architectures, not only organizing and maintaining a cache hierarchy require a significant amount of administrative coordination between the institutions, 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 look up or search schemes. But increasing global hit ratio does not always imply reduction of request latencies in distributed environment. In those caching schemes, not all cache hits are good for user request latencies [16]. Those protocols also create hot spots when some web servers become very hot sometimes. To solve the hot spot problem, Karger et al proposed a consistent hashing technique to construct per-server distribution trees to balance the work load among the proxy servers [17]. However implementing consistent hashing and random tree requires a lot of changes of current internet infrastructure [18]. Some problems associated with distributed caching schemes that use multicast or directory servers to locate the documents include high connection times, higher bandwidth usages and administrative issues [6]. On the other hand, because most those schemes concentrate on reduce the miss rates, load balance among the proxy servers are not thoroughly discussed. In reality, balancing the workloads is very important in quality and fairness of services. Load balancing are not only affected by the user request patterns, but also by the characteristics of the servers, such as storage capacity and network bandwidth of the cooperative servers [19].

The essential features for an ideal proxy cache system include minimized access latencies, robustness, transparency, scalability, efficiency, flexibility, stability, load balancing, and simplicity. It is important to design and implement a Web caching scheme to satisfy all those properties. However, scalability and simplicity are the biggest problems in the current Web cache schemes as discussed earlier. To solve the aforementioned problems, we propose a novel caching scheme using P2P concept. In this proxy caching system, the cooperation between proxy servers is handled naturally by simulating an ecological system. The load balance is achieved by data caching and data replication based on an economical model. The design of this Web cache system is unique that it satisfies reasonably well all desirable properties mentioned above. The rest of the paper is organized as follows. In section 2, we discuss the fundamental of the proposed proxy caching system and explain the design details. In section 3, we design a simulation model to examine our observations and to check the feasibility of the caching system. The simulation results are presented in section 4. Finally we have our conclusion and future study in section 5.

2 Cache Ecology – A Novel Proxy Caching Scheme

A distributed proxy cache system consists of many proxy servers inter-connected through network. Each server or a cluster of servers normally serve a group of clients

within an institution. Figure 1 depicts a typical distributed proxy caching system on the internet.

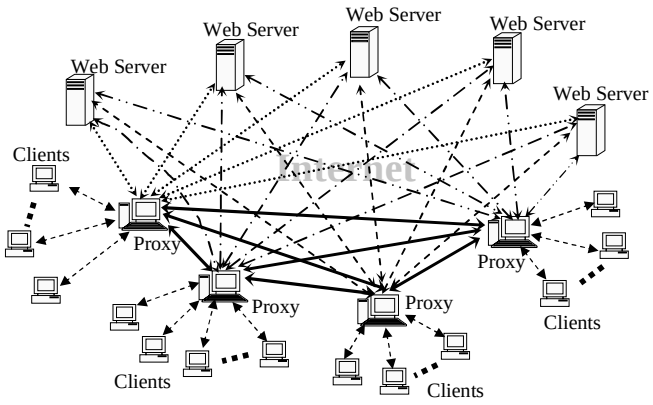

Figure 1: A typical distributed proxy cache system.

Proxy servers serve the requests from their clients as well as the requests from the peer servers. On the other hand, they download the requested documents either from the web servers or from its neighbor proxies to minimize the latencies for their clients. There are many factors contributing to the complexity of implementing a large scale distributed proxy caching system. Institutions have their own business agenda and economic policies hence the common interests of all institutions are different. Thus the available caching resources, such as storage capacity and network bandwidth, vary dramatically in different proxy servers. Hence, the routing of requests and caching data in a distributed proxy system, as shown in Figure 1, are complicated and the complexity of management grow exponentially with the growth of the network. Further more, using complex caching scheme adds more complexity to the system and in turn hinders the scalability of the system.

2.1 An Individual-based Design Model

In real world, there are two proven mechanisms that can successfully manage a massive cooperative system. 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 and demand model. The other 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. Researchers have tried to find a simple model to simulate the nature flock in computer animation until the paper “boids”1 published at SIGRAPH in 1987 [20]. Since then the boids model has become an oft-cited example of principles of Artificial Life. Flocking is a particularly evocative example of emergence: where complex global

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

behavior can arise from the interaction of simple local rules. In the boids model, interaction between simple behaviors of individuals produce 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. The result is life-like group behavior.

A distributed proxy caching system can be viewed as a flock of individual cache nodes 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 proxy system possesses this property due to the unpredictability of Web requests over a longer period time. In a shorter time frame, the Web requests seem to very predictable because of the flock like behavior of human interests. Thus modeling the distributed proxy caching 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 [21]. The boids model is an example of an individual-based model2, 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-organized proxy ecology in which individual cache nodes exchange data and information using some simple rules. The aggregate effect of caching actions by individual cache nodes automatically distributes the data to nearest clients and also automatically balances the workload.

2.2 Self-Configuration of Proxy Network

Self-configured network architecture is the most important part of our proxy cache system. When a proxy server decides to join the proxy network, it broadcasts a request-for-join message to the existing cache nodes; the existing cache nodes reply the request-for-join message with the characteristics of their own. Those characteristics include storage capacity, network bandwidth, the number of linked neighbors or the IP addresses of the neighbors who link to the exiting node. Then the requester decides which existing nodes it wants to link to. Many factors determine the number of links that the requester can establish. Besides its own storage capacity, network bandwidth and user tolerance to response latencies, the characteristics of the exiting nodes, such as distances to the requester, storage capacities, network bandwidths and average number of links per nodes, all contribute to the final decision. After the decision is made, the new added node informs the other nodes who it wants to link with to update their neighbor database. It also builds a neighbor table for later lookup. An entry in the neighbor table includes an ID which is normally an integer and the IP address to the neighbor. If the IP addresses of the neighbor nodes for all the existing nodes have been sent to the requester, the requester can actually build a

2 http://www.red3d.com/cwr/ibm.html

proxy graph and store it for later reference. If a proxy graph is required to store in every proxy servers, the newly added node has to send the IP addresses of its neighbors to all the other nodes in the system so that the other nodes can update the proxy graph on their sites. Because our individual based model does not require the individual proxy aware of all the nodes other than its neighbors, storing the proxy graph is only an enhancement option. However, with the proxy graph stored locally, we can improve the performance in searching the cached documents. Once the links have been virtually established, the new added node is restricted to only exchange information and data with its neighbors. This constraint guarantees the simplicity of the management on each individual cache nodes. Figure 3 demonstrates a proxy graph with 7 nodes.

2.3 Data caching and data flow

In our proxy cache system, the cached data may be transmitted to current node from its neighbors instead of directly from the original Web server. To cope with the new data type, we extend the traditional cache line to include data ID, Metadata and state bytes. An ID of the cache line can be the URL of the cached document. Metadata should include some simple descriptions of the document. State bytes contain information for cache coherency control as well as some other statistical data, such as number of replicas of the document in the proxy system known to this node, shortest distance to a replica. Figure 2 depicts a Web cache line in our proxy cache system.

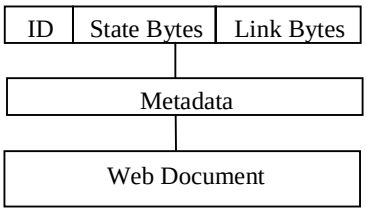

Figure 2: A web cache line.

The cached document might be replicated from other nodes in the proxy caching system or from links of other nodes that had previously accessed this document. Additional fields are needed in cache line to provide the link information. A distance-to-replica field must pair with the links to tell the distance from this node to the nearest replica along the link.

The heart of our proxy cache system is to link the data across the proxies by flowing metadata or data among the neighbors. We explain the rules for each individual proxy node to handle the data caching and data flow instead of the algorithm. We generate a flock effect as seen in boids model by controlling the caching and flow of data on demand as in an economic system. Figure 3 shows how cached documents flow among the proxies in a 7-node proxy graph.

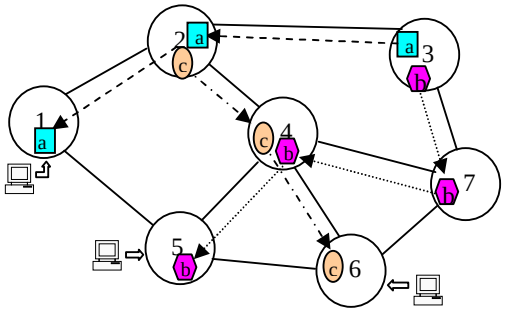

Figure 3: Data flow among the proxies

Assume document *a, b* and *c* are originally cached in nodes 3, 7 and 2 respectively. When client at proxy node 1 requests for document a, node 1 does not have the document cached in its site; it asks its neighbors for a cached copy. In turn, the neighbors ask their neighbors and finally find document *a* in node 3. Node 3 sends document *a* to node 2. Node 2 then sends the data to node 1 where the request was originated. Most importantly those proxy nodes could replicate the document for serving later requests along the way that data routed through. Actually those nodes that sit on the path between the query originator and data source do not need to cache the Web document if it is not necessary by its own local policy. However it needs to cache the ID, state bytes, links and metadata. By caching the information, the later requests for the same Web document from its neighbors or itself can be quickly routed to the cached replicas. The number of replicas passes along the nodes and increases when a node on the path replicates the document. When a proxy decides to replicate the document, it updates the shortest distance to a replica in state bytes and sends a message to its neighbors for them to update their state bytes and link bytes accordingly. Then the neighbors notify their neighbors if their shortest distances to a replica are modified. All updates on one node are based on the information passed in by its neighbors and the information stored locally. Similarly when client at proxy node 5 request for document *b*, document b is passed to and possibly replicated along path 3-7-4-5. When client at node 6 requests document c, it is replicated and linked on path 2-4-6. 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.

2.4 Searching the cached replicas

With the cache strategy described in the previous subsection, when a new request for a document comes in to certain proxy, the proxy node first checks if the document is cached in its site or if the link information is cached in its site. If the document is already cached, the request from the client is satisfied. If the proxy node contains only the link information, the proxy sends a request to its neighbor that the link points to. The nodes linked by cached link information relay the request to eventually reach the node that has a cached replica. The cached replica is then transmitted to the requesting node along the path that the request was replayed. However, if the proxy

node contains neither the requested document nor the link information, then this proxy node has to issue a query to its neighbors. The query message includes the ID of the requested document, a timestamp, maximum waiting time. The timestamp tells the time when the query was issued. The maximum waiting time indicates how long the requester will wait for response. The proxy node who initiated the query waits for the query results until the waiting time expire. If a node receives a query that passed the maximum waiting time, it discards the query message. Once a node finds a cached document or link information matching the ID given in query message, it sends a response to its neighbor where the query message comes from. The neighbors then route the response all the way back to the query originator. A response must include a field to indicate the distance to the cached replica. This field is updated along the way that the response propagates back to the requester. For instance, assume a client at node 7 request for document *c*, a query is sent to its neighbors, including nodes 3, 4, and 6. Some time later, node 7 gets responses back from nodes 3, 4, and 6. Node 7 then decides the route to obtain the document based on the distance information provided in all responses. If a node, after it issues a query to its neighbors, does not get responses within the maximum waiting time, the node will contact the original Web server for the document.

3 Simulation model

There are many metrics that need to be studied to evaluate a proxy cache system. Those metrics include hit ratio, average user response time, load balance, management complexity, etc. In this paper, we focus on study the feasibility of our proxy ecology scheme. Without loss of generality, we configure a proxy 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 proxy graph so that all proxy servers have 4 neighbors each, excepting that those servers on the edge and the corner 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 proxy servers. 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. First we assume the distance from a client to its proxy server is 100ms. If the requested document is cached in the proxy server, the user response time is 200ms ignoring the other overheads. We assume the distance between the neighbor proxies is 200ms, which is double the distance from client to its proxy server. So if a requested document is found in a neighbor node, the user response time is 600ms. 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 1500ms.

Thus the user response time for a cache miss is 3000ms ignoring all other overheads. We further assume each proxy can cache 1% of total web documents. Based on our observation [19] and some other studies [22], web requests follow a Zipf-like distribution. 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 Web documents in the system, and $0 \le z \le 1$ is the zipf-factor [19]. We assume z = 0.7 in our simulation. We use the aforementioned parameters as our default simulation parameters unless explicitly specified otherwise. In this simulation, each node performs as a proxy server. When the proxy server receives a request from its client, it sends the document to the client if the document is cached in the proxy. Otherwise, it routes request to the Web server as well as to its neighbor proxies. The first response from either the original web server or from the neighbor nodes will be used. Because user response time is the most important metric for customer service from proxy server's point of view, simultaneously sending requests to its neighbors and to the original web server yields better response times. We use a simple LRU algorithm as our cache replacement algorithm in simulation.

4 Simulation results

We study the system performance on various cache rates. Our simulator simulates the 64 proxy nodes in either individual mode or cooperative mode. We collect the results of our simulations running in those two modes. As most of the Web cache system, we use hit ratio and average response time as system performance metrics. We issue 200,000 web requests to the cooperative proxy cache simulator. We start to collect statistic data after the first 50,000 web requests have been processed to avoid the inaccurate results due to the startup of the simulation. We observe the user response times and cache hit ratios at all nodes. The simulation results are presented in Figure 4 and Figure 5.

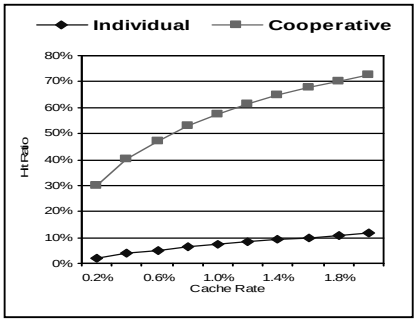

Figure 4: Hit ratios under various cache rates.

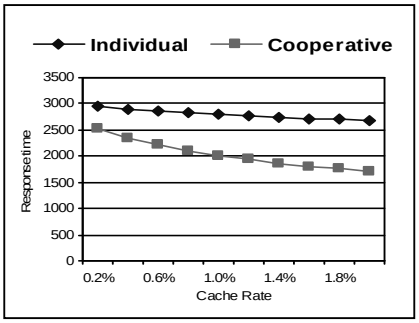

Figure 5: Response times in various cache rates.

As expected, increasing the cache rate increases the cache hit ratios and in turn improves the system average response time for requests. This is true in both individual proxy cache system and cooperative proxy cache system. As shown in Figure 4, Our P2P cooperative proxy system increases cache hit ratios tremendously over the individual proxy cache. In our simulation, the average hit ratio for individual proxies is only 7%. On the other hand, the P2P cooperative cache system yields 57% hit ratio. In terms of user response times, Figure 5 shows our P2P cooperative cache system outperforms individual proxy cache by large margin. For instance, when each individual proxy caches only 1% of the total web documents, the average response time for individual proxy is 38% more than that for our cooperative proxy cache system.

5 Conclusion

In this paper, we proposed a novel P2P cooperative proxy system using an individual based model. Each proxy node needs only to communicate with its neighbors. The accumulative results of the individual actions by all proxy nodes automatically distribute the data close to the clients. The system can be self-configured into a proxy graph using simple rules. Based on the demand, data cache and data movement in our proxy cache system create an ecological proxy system. This unique system design simplifies the management of a large-scale cooperative proxy cache system. Our simulation results indicate the effectiveness and feasibility of our cache system.

Acknowledgment

This work has been partially supported by ARO under grant DAAD19-01-1-0502. The views and conclusions herein are those of the authors and do not represent the official policies of the funding agency or the University of Central Florida, or Clemson University.

Reference:

[1] 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.

[2] 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.

[3] 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.

[4] Duane Wessels, K Claffy, ICP and the Squid Web Cache, IEEE Journal on Selected Areas in Communication, 16(3):345-357, 1998.
[5] Scott Michel, et al., Adaptive Web Caching: Towards a New Caching Architecture, the 3rd International WWW Caching Workshop, Manchester, England, June 1998.
[6] 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.
[7] 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.
[8] Joe Touch, The LSAM Proxy Cache – a Multicast Distributed Virtual Cache, the 3rd International WWW Caching Workshop, Manchester, England, June 1998.
[9] V. Valloppillil and K. W. Ross, Cache array routing protocol v1.0, Internet Draft, <draft-vinod-carp-v1-03.txt>, February 1998.
[10] D. Dovey and J. Harrison, A Distributed Internet Cache. In 20th Australian Computer Science Conference, February 1997.
[11] 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.
[12] Zheng Wang, Cachemesh: A Distributed Cache System for World Wide Web, the 2rd NLANT Web Caching Workshop, June 1997.
[13] 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.
[14] 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.
[15] Alex Rousskov and Duane Wessels, Cache Digests, Computer Networks and ISDN Systems, 30(22-23):2155-2168, June 1998.
[16] 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.
[17] 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
[18] David Karger et al., Web Caching with Consistent Hashing, the 8th international WWW conference, Toronto, Canada, May 11-14, 1999.
[19] 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.
[20] Craig W. Reynolds, Flocks, Herds, and Schools: A Distributed Behavioral Model, in Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings) pages 25-34, 1987.
[21] 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.
[22] 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.