Proxy Ecology - Cooperative Proxies with Artificial Life

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

Ratan K. Guha
Department of Computer Science
University of Central Florida
Orlando, FL 32816
407-823-2956
guha@cs.ucf.edu

Abstract

In this paper, we propose a novel P2P cooperative proxy cache system using an individual-based model. We borrow the idea from an ecological system as well as economic systems to manage the cooperative proxies through data and information exchange among individual proxies. The data flow among proxy nodes creates artificial life for the cooperative proxies. The proxy servers with artificial life can automatically configure themselves into a Virtual Proxy Graph. The aggregate effect of caching actions by individual peer proxies automatically distributes the Web document closer to the clients and balances the workload. Our simulation results show that the proposed proxy caching scheme tremendously improves the system performance. In addition, the individual-based design model ensures the simplicity and scalability of the cache system.

1. Introduction

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 [2]. The proxy server tends to be overloaded as the client number increases and hence causes a lot of cache missing. To solve the problem, organizations normally employ many servers to serve as the 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.

There are many different cache architectures proposed in cooperative Web caching. Hierarchical cooperative caching was first introduced in the Harvest project [2]. The Harvest cache system organizes proxy caches in a hierarchy and uses a cache resolution protocol called Internet Cache Protocol (ICP) [3] to search the cached document. Adaptive Web Caching [4] 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 [5]. 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 [5, 6, 7, 8, 9, 10]. In distributed Web caching systems, all cache proxies are viewed as the same level within the caching system. Those distributed proxy cache systems use a centralized directory server to maintain the global cache status, or employ sophisticated directory lookup or search schemes to discover the global cache information.

The ideal properties [11] for a proxy caching system include *Minimized latencies, Robustness, Availability, Transparency, Efficiency, Flexibility, Stability, Load balancing, Scalability, and Simplicity*. 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 scalable cooperative proxy cache system. In this proxy cache system, the cooperation among proxy servers is handled naturally by simulating an ecological

system - flocking. The 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.

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 examine our observations and prove the effectiveness of the P2P proxy cache system. The simulation results are presented in section 4. We have our conclusion and discuss future studies in section 5.

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. Configuring and managing those distributed proxies into a cooperative proxy cache system using existing caching schemes require intensive manual configuration based on the network architecture and hence are very complicated. In a distributed proxy cache system, proxies serve the requests from their clients as well as the requests from the peer proxies. They fetch the requested documents either from the Web servers or from cooperative proxies to minimize the request 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, routing requests and caching data in a heterogeneous distributed proxy cache system are complicated. The complexity of management grows exponentially with the growth of the cache system. Further more, using complex caching scheme adds more complexity to the system and in turn hinders the manageability and scalability of the proxy cache system.

2.1. An Individual-Based Model

In real world, there are two proven mechanisms that can successfully self-manage a massive distributed 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 by demand policy. 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. 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 world.

Researchers had been searching a simple model to simulate the nature flock in computer animation until the paper "*Boids*" [12] published at SIGRAPH in 1987. Since then the *Boids* model has become an oft-cited example of principles of Artificial Life. 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 the Web requests over a longer period time; in a shorter time frame, the 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. 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 proxies exchange data and information using some simple rules. The aggregate effect of caching actions by individual proxies automatically distributes the Web documents closer to clients and also automatically balances the workload.

2.2. Virtual Proxy Graph

Using an individual-based cache model, a proxy needs to find its neighbors with which it exchanges data and information. Unlike some other existing schemes in which the architecture of the cache system are manually configured, the individual proxies in our 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 the data and information with their neighbors. The neighbor proxies help each other in searching for cached Web documents and balancing workload. This is similar to Boids model where individual boid maneuvers based on the positions and velocities its nearby flockmates.

The configuration of the VPG follows some simple rules. First, neighbors must be close to each other. This rule makes sure the network distance between two neighbors is short so that the 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 larger network bandwidth will link with more neighbors. On the other hand, a proxy that has limited network bandwidth should not link with many neighbor proxies. Assigning the number of links to a proxy based on its computing resources is the key for automatic load balance. We discuss the automatic VPG configuration process in following paragraph.

When a proxy wants to join the P2P proxy cache system, the following steps are needed:

1. ***Token request:*** A proxy 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 does not get any response from any other network nodes or no other proxy has been found holding the token, this requesting proxy will create a token and assume it is 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 requesting proxy gets the token, it starts configuring itself into the proxy cache system. It first broadcasts a request-for-join message to the existing cache proxies. An existing cache proxy replies the request-for-join message with the acknowledge-to-join message that contain its characteristic information, including storage capacity, network bandwidth, the maximum distance allowed to its neighbors, and the number of its neighbors or the IP addresses of its neighbors. Based on the information gathered from the other existing proxies, the requesting proxy decides which proxies to link with to form a new VPG.

3. ***Configuration:*** Many factors affect the number of links that the new proxy can establish. Besides its own storage capacity, network bandwidth and user tolerance to request latencies, the characteristics of all other exiting proxies also contribute to the final decision. In general, the neighbors should be close to each others in the network in order to get fast responses for the requests sent to the 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 in this paragraph. First, the new proxy determines the maximum distance allowed for other proxies to be its neighbors. It is decided by its user tolerance to the 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 with. Using information 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 also must be less than its own maximum allowable neighbor distance. After selecting the links, the new proxy needs to recalculate 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 about the selection. 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 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 structure of 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 non-neighbor proxies, storing the VPG locally at every proxy is only an enhancement option.

Configuring cache proxies into a VPG has several advantages. With the VPG, proxies only exchange the data and messages with their neighbors. So management at each proxy is simpler than those in other 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 changes in network environment but also robust to failure of some proxies. With the VPG, the aggregate effect of data movement among neighbor proxies creates a life-like group behavior that automatically distributes the data closer to clients with less complexity. The VPG configuration process also provides a well-balanced distributed proxy cache structure independent of 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 1 shows a traditional cache line.

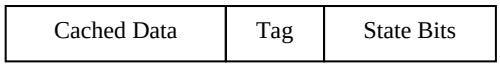

Figure 1: A traditional cache line

Unlike in traditional hierarchical cache where cached data is normally a data page or an instruction from lower level storage devices, data cached in a caching proxy are Web documents that may migrate to the current proxy from its neighbors instead of directly from the original Web servers. 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 2 depicts a Web cache line in our proxy cache system.

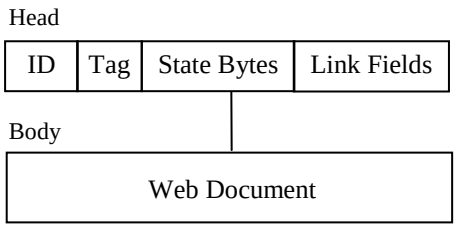

Figure 2: A web cache line.

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, Tag, 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 the current proxy in the cache system and shortest distance to the nearby replicas. Because the cached Web document might be replicated from other proxies in the 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 Web 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 caching proxies automatically distributes Web documents closer to clients interested in the cached Web documents. To achieve this goal, we must find a set of well designed rules that individual proxies would use to route the query messages and cache the Web documents.

Efficiently searching the cached Web documents is the most important function in a cooperative 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 based on an individual-based model, the search algorithm should only based on limited global cache information provided by data cache and dada exchange. The search algorithm runs on all proxies simultaneously and each proxy only queries its neighbors for Web documents. When a new request for a Web document comes in to a

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 a Web cache line for the requested Web document, it checks the shortest distance to a replica stored in the Head. If the distance to the nearest replica is greater than its expected response time by requesting from the Web server directly, it issues an HTTP request to the Web server.

   - If the shortest distance to a replica is less than the expected response time by requesting directly from the Web server, the proxy searches the link fields to find a link to the neighbor that can lead to the nearest replica. Then a query is sent to that neighbor to which the link points. 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 proxies.

   - 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 a cached replica. Then the query message is routed to the 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 broadcast 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's 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. If no match is found, the query message is relayed to its neighbors.

   - A response message must include a field to indicate the distance to the cached Web 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 (increased) 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.

   - Before the waiting time is expired, the requesting proxy gathers the response messages. It creates a Web cache line based on the first response message and updates the appropriate head fields according to information in the succeeding response messages. The requesting proxy should find a link to the nearest replica in this Web cache line, which now contains statistic information about the replicas. 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 if the IP address of the requesting proxy is included in the query message.

3. Simulation Model

There are many factors affect the performance of a proxy cache 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 request 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 initial part of the document arrives at client's workstation. We compare the request 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 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 the response time is 600ms if a requested document is found in a neighbor node. 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 request response time for a cache miss is 3000 ms, ignoring all other overheads.

Some studies [13, 14] have found that Web requests follow a Zipf-like distribution [15]. 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 [15]. A larger z value corresponds to a more skew condition, i.e., some Web documents are accessed considerably more frequently than other Web documents. When $z = 0$, the Web access is uniformly distributed, i.e., all Web documents have the same access frequency. We simulate our proxy caching scheme on this automatically configured VPG.

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 [14], 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 performance of our proxy cache system. Our simulator can run in two different modes, individual mode and cooperative mode. The individual 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 are issued to the 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 request response times and cache hit ratios at all proxies.

Access skew condition affects the locality of the Web access. The cache hit ratios should increase when data access pattern are increasingly skewed. In this simulation, we maintain the cache rate on each proxy at 1% of the total Web documents. We change the Zipf factor from 0.4 to 1. The simulation results are depicted in Figure 3 and Figure 4. 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 [13, 14]. As shown in Figure 3 and Figure 4, 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 3. At the same time, the Web

request response time using individual proxy cache is 32% higher than using our P2P proxy cache system, as shown in Figure 4.

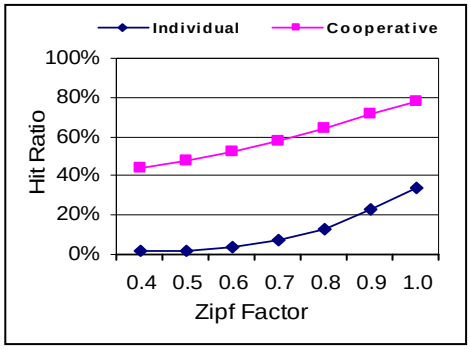

Figure 3: Hit ratios under various Zipf factors.

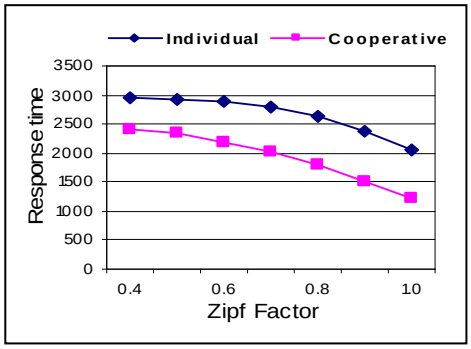

Figure 4: Response times under various Zipf Factors.

5. Concluding Remarks and Future Studies

In this paper, we propose a novel P2P cooperative proxy cache system using an individual-based cache model. The accumulative results of the individual caching actions by all proxies automatically distribute the data closer 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 caching and data replication in our cache system create proxy ecology. This unique system design simplifies the management of the cooperative 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 implementing a prototype using our proposed proxy caching scheme.

6. References

[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. 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.

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

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

[5] 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.

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

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

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

[9] 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.

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

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

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

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

[14] 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.

[15] 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.