Peer-to-peer (P2P) web caching has been studied recently as it can exploit local caches of peers for web caching without additional infrastructure. However, dynamic join/leave behaviors or local caching strategies of the peers due to their autonomy in a P2P network may limit the performance of P2P web caching. To overcome these limitations, we propose an effective directory-based P2P web caching system under dynamic participation of peers. We introduce the object lifetime in a P2P network considering the lifetimes of both an object in the local cache of a peer and a peer who owns the object, and utilize this object lifetime information for neighbor selection and storage management in the P2P web caching framework. For the neighbor selection, the proposed system utilizes the object lifetimes in selecting accurately a neighbor who would still retain the requested object and still remain in the P2P network. To improve the storage management, the proposed system uses efficiently the entire cache storage of the P2P network in such a way that the object is stored selectively in the local cache of the peer who requested it, considering the object lifetime. The trace-driven simulation results show that the proposed system has higher accuracy and fewer redirection failures than the conventional directory-based P2P web caching system in the feasible P2P network. 

**key words**: peer-to-peer system, web caching, internet system, world wide web

1. Introduction

Peer-to-peer (P2P) web caching configures the clients to form a P2P network and act as peers. When a client requests a web object, the object is transferred from a neighbor—one of the clients who have already received the object—to the client who requested the object in a P2P manner. Therefore, the cache space can be expanded with the local caches of clients without additional infrastructure. In addition, load balancing of the traffic converged on the server side can be achieved as peers may share the traffic.

However, P2P web caching may not yet be a full replacement model for an existing traditional web caching model, because P2P web caching alone is insufficient in that peers are basically independent entities and thus the local caches of peers are not dedicated caches for public share. There are several challenging issues in P2P web caching due to the autonomy of independent peers.

Since each peer has its own policy, a P2P web caching system should minimize additional loads such as serving the objects to other peers, requesting a peer for staying in a P2P network longer, and so on. It should also reduce the violations of local caching policies of peers, since no peer may want to store the objects that have not been explicitly requested by the peer itself.

A P2P web caching system should take into account dynamic characteristics of peers. Unlike static dedicated caches, peers may join or leave a P2P network dynamically. Therefore, the system should minimize the management overheads and the performance degradation caused by dynamic participation of peers.

In this paper, we present an effective P2P web caching system in a feasible web environment, where peers may join or leave a P2P network dynamically and where compulsory placement of objects conflicting with each peer’s explicit request is not allowed. In addition, we try to improve the caching performance by managing the storage of a P2P network efficiently in the proposed system.

The proposed system is based on the centralized directory-based P2P web caching. Each website that adopts P2P-based web caching plays a role of the central superpeer for the peers who request the objects that belong to the website. That is, the website manages the redirection directory that consists of both its contents of objects and neighbors who have been transferred the objects in the past. With the redirection directory, it redirects a peer’s request for an object to one of the neighbors. Then the object is finally transferred from the neighbor to the peer.

The proposed system is to improve the web caching performance with higher accuracy and fewer redirection failures in a feasible web environment. For this purpose, we introduce the object lifetime for neighbor selection and storage management in the proposed system. We call the proposed system the object lifetime-based P2P web caching system (OLP). Unlike general web caching systems, OLP determines the object lifetime in a P2P network considering not only the lifetime of each object in the local cache of a peer but also the lifetime of a peer who owns the object in the P2P network. OLP utilizes the object lifetimes in selecting accurately a neighbor who would still retain the requested object in its local cache and who would still remain in the P2P network. For the storage management, OLP efficiently utilizes the entire cache storage of the P2P network in such a way that the object is selectively stored in the local cache of the peer who requested it, considering both the object lifetime and the overall request rate for the object.

In this paper, the main contributions of our study are as follows. First, we propose an efficient P2P web caching system in a feasible environment in which peers may join/leave a P2P network dynamically and no peer is forced to store the
objects that have not been requested by peer itself. Second, through the trace-driven simulations with a web log data, we show that OLP is more effective than the conventional directory-based P2P web caching system with higher accuracy and lower lookup cost in a feasible web environment where only a small number of peers in a P2P network are willingly cooperative. Third, we illustrate the potential advantages of considering the object lifetime in a P2P network as a key attribute in neighbor selection and storage management in the P2P web caching framework.

The rest of this paper is organized as follows. Section 2 presents related work. In Sect. 3, the proposed web caching system is described in detail. Section 4 provides the performance. In Sect. 5, we discuss some other challenging issues and limitations. Finally, in Sect. 6, the conclusion and future work are given.

2. Related Work

There are mainly three types of P2P web caching systems according to neighbor discovery methods; they are the directory-based method, the flooding-based method, and the distributed hash table (DHT)-based method.

In the web caching system with the directory-based method [2], [4], [5], [14], either the central server or the superpeer manages a redirection directory and redirects the current request of a client for a certain object to its neighbor after directory lookups. Since this method is so simple and no complex P2P overlay is required, it is relatively easy to implement. The cost for directory lookups is also very low. However, the system has low reliability due to the risk of a single point of failure. In addition, highly dynamic participation of peers causes the management overheads of keeping the up-to-date redirection directory. Without a tight up-to-date redirection directory, the lookup cost will be increased because redirection failures may occur more frequently.

Pseudoserving [2] is one of the pioneer P2P web caching systems. It was designed for mitigating the traffic bottleneck of a server. Pseudoserving has introduced the concept of a contract—the duty that a peer must stay in a P2P network for a certain period of time—in order to resolve difficulties in the peer management due to dynamic participation of peers. However, such contracts impose nothing but additional duties on peers. CoopNet [4], [5] is a directory-based system and has been designed for mitigating the traffic bottleneck caused by the flash-crowd problem. Moreover, CoopNet tries to minimize the traffic of redirection messages from a server to each client. But CoopNet does not consider the local caching strategies of peers and the risk of the redirection failures. Browser-aware proxy server [14] is another directory-based system. In the Browsers-aware proxy server, in order to prevent the redirection failures caused by dynamic participation and by the local cache replacement strategy of each peer and to perform the replica management efficiently, a proxy server manages the whole information of peers including the indices of all the objects in the local cache of each peer. However, the needs of frequent updates to manage the up-to-date full index result in heavy communication overheads between each peer and an upper proxy server as the number of peers increases.

The web caching system with the flooding-based method [3], [11]–[13] is similar to the pure P2P system. Searching in this system finds the neighbors through controlled flooding of lookup messages. This system is reliable because it does not depend on only a few peers in the search process. Spreading lookup messages to adjacent peers gives the impression that the searches of this system are done on an unstructured arbitrary overlay of peers. However, the network traffic and the latency overheads caused by flooding messages could become serious especially for searching rare objects.

POOF [11] is a flooding-based system designed to deliver the objects in flash crowd conditions. Randomized, scoped, flooding searches are used to locate the object that cannot be retrieved from the overwhelmed server. POOF can achieve the low latency even when membership to the overlay changes dynamically with time and when there exist members that limit their participation in the system. However, POOF admitted that the potential network traffic overheads caused by searching the objects that are not popular would be heavy. In case of BuddyWeb [3], [13], a flooding-based search technique with the similarity function is applied to the system. BuddyWeb uses similarity to select adjacent peers who have high possibility of having the required object in order to reduce the lookup cost. However, the computation overheads due to calculating the similarity of each pair of peers with the content information in the peers will be heavy when there are many peers in a P2P network.

The DHT-based search [1], [10] is suitable for web caching focused on availability, because it can find a neighbor for the requested object within a bounded number of hops. However, there are some drawbacks in this system. First, since there is only one neighbor for a certain object in this system, DHT-based search is only suitable for finding a rare object and imposes heavy serving loads to a matched neighbor for a hot spot object. Second, it is not clear whether the amount of transfer overheads makes sense in order to maintain a highly structured P2P overlay in the web environment with dynamic participation of peers [11]. Third, the DHT-based system cannot guarantee the local storage management policies of peers since the system requires peers to cache the objects that they have not requested by themselves.

Squirrel [1] and Backslash [10] have DHT-based discovery methods. Squirrel uses the search technique based on Pastry [8]. Backslash uses a similar search technique to the technique in CAN [7]. Although these DHT-based P2P web caching systems are able to locate the objects within a bounded number of hops, they have some drawbacks as depicted above.

In this paper, we focus on proposing an effective P2P web caching system in a feasible environment, where dy-
namic participation of peers due to peers’ autonomy is re-
spected and where the lookup cost for an object is reason-
able. For such a target environment, the DHT-based search
is not suitable because its management overheads for main-
taining P2P overlay are too high under dynamic participa-
tion of peers. The flooding-based search is not also suitable,
because the lookup cost for rare objects is high. Therefore,
the proposed system in this paper is based on the directory-
based P2P web caching. We improve the conventional
directory-based P2P web caching scheme with higher ac-
curacy and fewer redirection failures under dynamic partici-
pation of peers, utilizing an object lifetime-based approach.

3. An Object Lifetime-Based P2P Web Caching System

3.1 Basic Operations

OLP configures a P2P network with the web server and
clients who visit the server. Then the clients act as peers
until they leave the P2P network by leaving the server.

In OLP, the server maintains a redirection directory file.
When the server receives an object request of a client, the
server picks several neighbors for the object from the redi-
rection directory in the neighbor selection process and sends
the client the redirection message with the list of addresses
of those neighbors, called the neighbor list. If one of the
neighbors in the neighbor list is in the online status—is ac-
tive at the server—and still has the object, it serves the ob-
ject to the client who requested it. Otherwise, the server
transfers the object to the client who requested it directly. In
case when the client is served by a neighbor, after the object
is shown to the client, the local agent of the client decides
whether it stores the object in its local cache or not by the
storage management process. In detail, the basic steps of
OLP after a client requests an object are given below. Fig-
ure 1 also illustrates the basic steps.

1. If the request is hit in the local cache of the client, the
   object is shown to the client and then all the steps are
   terminated.
2. The server refers to its redirection directory and per-
   forms the neighbor selection process in order to find N
   neighbors who received the object in the past. Then the
   server sends the redirection message including the neigh-
   bor list to the client. If there is no neigh-
   bor for the requested object in the redirection directory,
   the object is transferred from the server to the client di-
   rectly and the server updates the redirection directory
   before all the steps are terminated.
3. After the client receives the redirection response from
   the server, the client sends the lookup message to each
   neighbor in the neighbor list.
4. If the request is hit in any one of the neighbors’ local
   caches, the client is served by the neighbor in a P2P
   manner. Otherwise, the object will be transferred from
   the server to the client directly and the server updates
   the redirection directory before all the steps are termi-
   nated.
5. After the object is transferred from the neighbor and is
   shown to the client, the local agent of the client decides
   whether it should keep the object in its local cache or
   not through the storage management process.
6. The client sends the update message to the server. The
   server updates the redirection directory, if necessary.

We introduce the concept of the object lifetime in a
dynamic P2P network and utilize the object lifetime infor-
mation for the neighbor selection and replica management
processes. The subsequent subsections describe these key
issues of OLP in detail.

3.2 The Crucial Factors

3.2.1 The Object Lifetime in a P2P Network

In general web caching, the lifetime of an object in the local
cache of a client is over when the object is deleted from the
local cache by the local replacement policy [6]. In P2P web
 caching, for the lifetime of an object in the local cache but also the lifetime of a peer who own
the object, because a peer is supposed to join and leave a P2P network dynamically. Therefore, we consider the
object lifetime suitable for the P2P network and utilize it in the
OLP framework. Let \( LT_A \) be the lifetime of an object stored
in peer A in the P2P network. Then \( LT_A \) can be predicted
as the minimum time between the average activity period of peer A and the average object replacement time of peer A in
Eq. (1). In Eq. (1), \( p_A \) is the total number of leave activity of peer A whose \( i \)th join time was \( T_{ij} \) and \( i \)th leave time was \( T_{ij} \),
\( q_A \) is the total number of removing objects in the local cache
of peer A who stored the \( j \)th object at time \( T_{ij} \) and removed
the \( j \)th object at time \( T_{ij} \).

\[
LT_A = \min \left( \frac{\sum_{i=1}^{p_A} (T_{ij} - T_{ij})}{p_A}, \frac{\sum_{j=1}^{q_A} (T_{ij} - T_{ij})}{q_A} \right)
\]

Consequently, we can also predict the disappearance
time of an object stored at time \( t \) where \( t < T_{ij}^{p_A+1} \). Let \( DT_A^t \)
be the disappearance time of an object stored in peer A at
time \( t \). Then \( DT_A^t \) can be predicted as given in Eq. (2).

\[
DT_A^t = \min \left( T_{ij}^{p_A+1} + \frac{\sum_{i=1}^{p_A} (T_{ij} - T_{ij})}{p_A} \cdot t + \frac{\sum_{j=1}^{q_A} (T_{ij} - T_{ij})}{q_A} \right)
\]
3.2.2 The Object Request Rate

Server $S$ manages the request rate for each object, and uses the request rates for storage management in OLP. The next request time of an object can be predicted with the request rate of the object. The predicted next request time $RT_x'$ of object $x$ requested at time $t'$ can be computed as follows.

$$RT_x' = t' + \frac{1}{\text{the request rate of } x} \tag{3}$$

3.3 Neighbor Selection

In a feasible web environment where highly dynamic participation and the local caching strategies of peers are allowed, redirecting the request to one of the neighbors from the redirection directory may not always be correct. A wrong redirection may occur due to the following two factors. First, a peer may delete some objects based on its local replacement policy. Second, it is possible for a peer itself to leave the P2P network. Therefore, in a feasible environment where allows the autonomy of peers, the redirection failures may occur. These failures could degrade the caching performance due to additional traffic and delays in the lookup process.

In OLP, the server contains the list of a limited number of recent neighbors for the objects that the server owns, and does not monitor the indices of all the objects in the local cache of each peer. It only manages the online/offline status of each peer. Moreover, OLP can be regarded as an even more lightweight system that the server does not monitor every join/leave activity of each peer. Therefore, OLP should be resilient in an environment where the server is not aware of each peer’s online/offline status in real-time. For the server’s management of monitoring dynamic participation, OLP has two different modes. In the tight mode, the server is aware of dynamic join/leave activity of each peer in real-time and does not select peers who are in the offline status as neighbors. In the loose mode, the server does not have the duty of awareness of dynamic participation of each peer.

If there is more than one neighbor for the same object in the redirection directory, the neighbor selection process is needed before redirection. To avoid the redirection failure, OLP focuses on selecting neighbors who would still keep the required object in its local cache and who would still be in the online status. For this purpose, OLP uses the object lifetime for neighbor selection. In order to determine the neighbor list of size $N$, OLP selects the first $N$ neighbors who have the longest $DT$’s from the redirection directory. Receiving the redirection message with the neighbor list, the client sends the request to each neighbor in the neighbor list.

3.4 Storage Management

One of the key issues in the cooperation of distributed peers is how efficient the storage management is. In P2P web caching, since each cache is not dedicated and is local only to the peer, any cooperation strategies conflicting with the intention of the local policy result in a violation of the local policy of the peer. Therefore, we assert that the system that forces a peer to store the objects that have never explicitly requested by the peer itself is not desirable. Only for the object that a client explicitly requested, we try to improve the caching performance through storing “valuable” objects in the local cache of each peer and minimizing object replacement caused by storing “valueless” objects. By a valuable object we mean that if an object is stored, it is likely to exist within the time period enough to be served to other peers.

In OLP, when a peer receives the requested object from a certain neighbor, it determines whether the received object should be stored into its local cache or not. If there is unused space in the local cache, the received object will be stored. Even when there is no unused space, if $DT$ of the peer is longer than both $DT$ of the neighbor and $RT$ of the object, then the received object is regarded as valuable object and hence stored. Figure 2 shows a pseudo code of the storage management process in OLP after peer $A$ requests a web object $x$ at time $t$ and receives $x$ at time $t'$.

3.5 Redirection Directory File and Redirection Message

In OLP, the server manages a redirection directory file that consists of the list of $<$Object_ID, Request_Rate, the list of $<$Neighbor_IP_Address, DT_of_Neighbor>$. The number of neighbors for the same object is limited at a few tens. When the server receives a request for an object from a peer, it selects $N$ neighbors from the redirection directory and sends the redirection message including the neighbor list that contains the neighbors’ IP addresses and their $DT$’s. The redirection message also includes the request rate for the object.

3.6 Agent in Each Peer

In OLP a simple agent in each peer manages the statistical information such as the recent online/offline time, the aver-
age active time, and the average object replacement time of its local cache. The agent predicts the object lifetime with this information. When the peer requests an object and receives it from the neighbor, the agent selectively stores the newly transferred object according to the storage management process. If object replacement is needed due to storing the newly received object in the local cache, the peer performs object replacement with its own replacement policy. In OLP, the LRU (Least Recently Used) scheme is adopted for the policy.

In case of successful storing the object, the agent sends the update message including both the IP address and its own DT to the server in order to be registered in the redirection directory. If cache misses occur at some neighbors during lookuping the local caches of neighbors in the neighbor list, the agent can send the update message which contains the IP addresses of those neighbors to the server, because those neighbors have already removed the object by their local replacement policies.

4. Performance Evaluation

4.1 The Simulation Environment

We have used the WorldCup'98 dataset [15] for the trace-driven simulations. The WorldCup'98 dataset consists of busy 1,352,804,107 requests made to the 1998 World Cup web site during the period between April 30, 1998 and July 26, 1998. Each day’s dataset has tens of thousands of clients, containing millions of requests. Each log of requests contains a timestamp, the client ID, the object ID, the object size, and so on. We have selected logs of 10 days randomly and have chosen 300 clients randomly for one hour of each day in our simulations. This means that we assume only 300 clients are willing to cooperate with OLP. The simulation results are provided as the average values over all such data for evaluating the performance.

In our simulation, the server has a connection type of T1, and peers have various connection types from a 14.4 kbps modem to a T3 line according to the distribution of reported bandwidths of peers in [9]. For the consistency of each object, even though the ID of the requested object is the same as that of a cached object, if their sizes are different, then the requested object is regarded as an updated object. We assume that there is no proxy server between a peer and the origin server because we wish to investigate the pure effect of P2P web caching.

Each peer has the maximum load of serving an object to only one other peer simultaneously for fair comparison of P2P caching performance. The LRU scheme is assumed for the local replacement strategy of each peer.

In order to simulate dynamic participation of peers with the web log, we assume that a peer is regarded as joining the P2P network at the time it requests a web object for the first time and that a peer leaves the P2P network after δ seconds since the latest request has been made. Therefore, smaller δ values indicate that the participation of the peers is more dy-

![Fig. 3](image-url)  
(a) Activity threshold = 60  
(b) Activity threshold = 600

**Fig. 3**  
The distribution of average activity period of peers.

The threshold δ varies from 60 seconds to 600 seconds at an activity interval of 60 seconds. Figure 3 shows some distributions of simulated average activity period of peers. Even though we set the activity threshold δ as an arbitrary value, we observed that the distribution of the average activity period had a consistent tendency; the number of peers is decreased as the activity period grows higher. It means that the number of peers who more steadily stay and act in the P2P network without leaving is relatively smaller than the number of peers who work temporarily in the P2P network. All the simulations have been performed under dynamic participation of peers.

The size N of the neighbor list in the redirection message is 1, 3, or 5 in the simulations. A client who receives the redirection message sends the request to each neighbor in the neighbor list sequentially until the requested object is found. If the redirection failures occur for all the neighbors in the neighbor list, the object is transferred from the server to the client.

For the performance comparison, we have simulated a general centralized directory-based P2P web caching system (GCD) that selects neighbors who are most recently registered in the redirection directory, OLP that improves GCD with the object lifetime-based neighbor selection (OLP_PS), OLP that improves GCD with the object lifetime-based storage management (OLP_SM), and the full OLP that executes both the object lifetime-based neighbor selection and the object lifetime-based storage management (OLP_PS+SM).

4.2 The Performance Metrics

To evaluate the caching performance, we use two evaluation metrics: the hit ratio and the byte hit ratio. The hit ratio is the proportion of the number of requests that hit in either the local caches or in the neighbors’ caches to the total number of requests. The byte hit ratio is the proportion of the number of bytes that hit in the local caches or in the neighbors’ caches to the total number of bytes requested. To evaluate the lookup cost, we use the number of lookup messages for cache hit as the evaluation metric. It means how quickly the neighbor who really has the object can be found if the client sends the lookup message to each of the neighbors in the neighbor list sequentially.
4.3 The Simulation Results

4.3.1 The Effect of Dynamic Participation of Peers

Figure 4 shows the hit ratios and the byte hit ratios of GCD under static and dynamic partitions for various activity intervals. Both hit ratios and byte hit ratios of GCD under dynamic participation of peers are smaller than those of GCD under static participation over the entire range of activity intervals. These results show that higher dynamic participation reduces the caching performance. This is because highly dynamic participation leads to a smaller number of peers who stay in the P2P network at a particular moment. Therefore, the opportunity of finding a neighbor becomes slim and the possibility of the redirection failures becomes higher consequently.

4.3.2 Evaluation of Accuracy

We now inspect how the local cache size of a peer influences the caching performance. In Fig. 5, the caching performances increase as the local cache size of each peer gets larger in all the systems. Each value in this figure is the average value over the results from simulations with various activity thresholds $\delta$. OLP_PS and OLP_SM show lower hit ratios and byte hit ratios compared with other systems. Both OLP_PS and OLP_SM outperform GCD with similar degrees of the performance gap.

We have evaluated each system’s sensitivity of the hit ratios and the byte ratios to the activity threshold $\delta$ under a fixed size $S$ of a peer’s cache. Figure 6 shows that OLP_PS+SM consistently outperforms other schemes under various degrees of dynamic participation of peers. OLP_PS and OLP_SM show relatively similar performances each other. Figure 6 demonstrates that OLP_PS+SM successfully reduces the negative impact of dynamic participation.

To evaluate the impact of scaling the size of the neighbor list, we compared the hit ratio and the byte hit ratio of each system with scaling the size $N$ of the neighbor list in Fig. 7. The values of the hit ratio and the byte hit ratio have been averaged over the results from the simulations with various $\delta$’s. It is clear that the performance increases as $N$ grows larger, since a larger number of lookup trials leads to a higher possibility of locating the requested object. When $N = 1$, the performance gap between OLP_PS and GCD is much larger than that between OLP_SM and GCD because accurate neighbor selection is a critical factor of cache hit with only one chance of lookup trial. But the performance gap between GCD and each of other systems tends to be reduced gradually as $N$ grows larger, because the higher chances to cache hit with large $N$ dilute the pure performance gain of accurate neighbor selection.
4.3.3 Evaluation of the Lookup Cost

Because of dynamic participation and the local replacement policy of each peer, the number of hops to locate the object is not always two even in a centralized directory-based P2P web caching system. To evaluate the lookup cost, we have looked into the number of lookup messages for cache hit with various sizes $N$ of the neighbor list in Fig. 8. Each value in this figure is the average value over the results from the simulations with various activity thresholds $\delta$. OLP,PS reduces the required number of messages because it makes the more accurate neighbor list through the neighbor selection that utilizes the object lifetime. OLP,SM also reduces the number of messages, because each peer tends to store the objects that have long object lifetime according to the proposed storage management scheme. Furthermore, OLP,PS+SM gains a synergy effect of both the neighbor selection and the storage management schemes and shows the best performance.

In Fig. 7, the performance gap of accuracy between GCD and each of other systems gets narrower as $N$ increases. On the other hand, Fig. 8 shows that the performance gap of the lookup cost between GCD and each of other systems becomes wider. It means that even when GCD succeeds in redirections as many as OLP with the large size of the neighbor list, OLP finds neighbors much more quickly than GCD.

4.3.4 The Effects of Server’s Up-to-Date Awareness of Dynamic Participation of Peers

Since the trends for the tight mode are quite similar to those in the loose mode, we omit the results for the tight mode in all the above simulations due to the space constraints of the paper. Figure 9 shows the effect of server’s up-to-date awareness of dynamic participation of peers. GCD and OLP,PS+SM both in the tight mode and the loose mode are compared under the condition of highly dynamic participation with activity threshold $\delta = 60$. Observe that the performance in the tight mode is higher than that in the loose mode for each system, because the server’s up-to-date awareness of peers’ online/offline status leads to more accurate redirection. In Fig. 9, OLP,PS+SM in the loose mode has the same or slightly higher performance compared to GCD in the tight mode. This indicates that the proposed object lifetime-based schemes for the neighbor selection and the storage management help the efficient management of P2P web caching even without up-to-date information of dynamic join/leave activities of peers.

5. Discussion

The network speed between a pair of peers may differ in a network, several neighbor selection methods have been proposed such as selecting the peer who is in the nearby area. But OLP focuses on minimizing cache misses at a neighbor. Hence OLP selects the neighbor, who would still hold the required object in its local cache and still be in the online status. For more efficient neighbor selection in OLP, the locality of peers may play an important role.

Selective storing in the storage management of OLP makes more cache hit in the entire P2P network in our simulations. But if it makes less cache hit in the local cache of each peer than unconditional storing does, practical lookup latency may increase. Therefore, we investigated the local hit ratio and the local byte hit ratio for the storage management. The local cache hit of the proposed selective storing is almost the same as that of unconditional storing with the performance gap that is less than 0.001. We analyzed that this is because the local hit for the existing objects that are not replaced thanks to selective storing of new objects counterbalances the negative impact of selective storing.

Since the size of a redirection message or an update message in OLP ranges from a few tens to some hundreds in bytes, the size of a message is quite smaller than that of a real web page practically. Therefore, it helps a server avoid the flash-crowd problem. But if there is an extreme flash-crowd, the connection overhead itself may overwhelm the server. It is one of general challenging problems in the centralized directory-based P2P web caching. To resolve this problem, it will be helpful to distribute the directory on the hot objects into multiple peers or to combine the directory-based method with other search technique such as...
the flooding-based method to check the directory.

6. Conclusion

In this paper, we presented an effective directory-based P2P web caching system in a feasible environment where the clients have different join/leave patterns and where the compulsory placement of objects conflicting with each peer’s explicit request is not allowed. The lifetime of an object in a P2P network is utilized for neighbor selection and replica management for the proposed web caching system. The trace-driven simulation results indicate that both the neighbor selection and the storage management schemes considering the object lifetime are effective under various conditions and show that the proposed system outperforms the conventional directory-based P2P web caching system with higher accuracy and fewer redirection failures. We are currently trying to combine the object lifetime-based approach with a different search technique such as the flooding-based search in P2P web caching.

References


Young-Suk Ryu received the B.S. and M.S. degree in Computer Science from Yonsei University, Seoul, Korea, in 1999 and 2001. He is currently a Ph.D. candidate at the Department of Computer Science, Yonsei University, and his research interests include peer-to-peer computing and Grid computing.

Sung-Bong Yang serves as a faculty member at the Department of Computer Science, Yonsei University, Seoul, Korea, and his research interests include mobile system, peer-to-peer computing, and 3D computer graphics. He received the Ph.D. degree in Computer Science from the University of Oklahoma.