Constructing an Overlay Using a Shared Object Set for Streaming Services on a P2P Network

Hyunjoo Kim and Heon Y. Yeom

School of Computer Science and Engineering, Seoul National University
Seoul, 151-742, Korea
{hjkim,yeom}@dcslab.snu.ac.kr

Abstract. In an on-demand streaming service on a P2P network, a client peer can receive parts of a video object from different peers instead of from just one server. For this, the client peer must find enough peers that have the required object. In this paper, we propose a method of constructing an overlay to search for objects efficiently for the on-demand streaming services on a pure P2P network. The proposed overlay is composed of groups, and all peers in a group have common objects. We compared our method with Gnutella by simulation. The results show that our method reduces traffic overheads, hop counts, and the number of messages, at the cost of join/leave overhead.

1 Introduction

Video streaming services are limited in client/server architectures because the number of clients that a server can support and the outbound bandwidth of a server are limited. In CoopNet [1], a client having the required object can provide a service to another client in place of a server. However, CoopNet is based on the client/server architecture, and many authors have tried to address this problem using many ordinary nodes as servers in a P2P network.

There are two kinds of streaming service. In live media services, one source broadcasts video data to many clients, whereas in an on-demand service, stored video data in a storage system are published.

The main approach to improving the quality of live media services is application-layer multicast trees, which have been reported in Narada [2], SpreadIt [3], NICE [4], and ZIGZAG [5]. In these trees, a source becomes a root and clients become tree nodes. On the other hand, in on-demand services, many source peers can provide a streaming service to a client peer because they transmit a stored video object from their own storage systems [6, 7]. In [8] and [9], an overlay network is composed of multicast groups with clients, and the same video object is served to the group members.

Super-peers [10] are installed in Gnutella to manage peers and find objects efficiently. However, because super-peers are intended to be servers rather than normal nodes, they absorb the installation and management overheads, and the overheads increase as the number of nodes increases.

In this paper, we focus on on-demand streaming services from multiple server peers in a pure P2P network. We propose an overlay network that is composed of groups, with one peer in each group selected as a directory server for the management of its group, while the other peers in the group provide streaming services. These roles are determined dynamically and autonomously when peers join the network by comparing their shared objects.

2 Overlay Scheme

In our proposed scheme, all peers in a group have as many common shared objects as possible. A group consists of a *Leader Peer (LP)*, which is the representative of the group, and some *Member Peers (MP)*. A peer becomes an LP if it has some unique objects not in the current network, otherwise it becomes an MP of one or more groups sharing the most common objects with it.

All requests are forwarded to the network through LPs. Because LPs serve as directory servers, when they receive a request they forward it depending on conditions. However, LPs do not provide streaming services to client peers, as the MPs do this. When an LP receives a request, it forwards the request to other LPs only if it cannot be satisfied within the group. A request is flooded only to LPs, not to all peers in the network. The LPs forward the request to any of their own members that have the requested object.

Each LP manages the location information of other LPs for flooding requests. In addition, each LP manages the location information of its MPs for forwarding requests.

2.1 Joining Process

We assume that a join peer knows the location of a peer that manages the process of entering the network (entry peer) and that only LPs can be entry peers. A join peer sends a join request with <*peer id, shared object set*> to the entry peer. When the LP (entry peer) receives the request, it compares its own shared object set with that of the join peer. If the LP has all the objects that the join peer shares, it accepts the join peer as a member. Otherwise, the LP forwards the join request to other LPs to find a more appropriate LP for the join peer. Each LP compares its shared object set after receiving the request and then replies to the join peer with the common shared object set and a join flag. The join peer gathers the object sets and checks whether it has any unique object not in the current network. If it has such objects, it becomes a new LP. If not, it determines which LP is the best to join and then becomes an MP to that LP. The join peer can be a member of one or more groups and it selects LPs to join by the number of common objects. If the join peer has all the objects of any LP, the join peer replaces the LP and makes it its MP.

Join flags represent the relations between shared object sets and are classified into the following (ShrObjSet(p) means the shared object set of a peer p):

Table 1. Join flags example

	shared object set				
LP	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3
join peer	1, 2	2, 3, 4	1, 2, 3, 4	1, 2, 3	4
join flag	ALL	PART	SUB	EQUAL	NONE

ALL: $\{$ShrObjSet(join peer) $\subset$ ShrObjSet(LP)$\}$
PART: $\{$ShrObjSet(join peer) $\cap$ ShrObjSet(LP)$\neq \emptyset\}$ and $\{$ShrObjSet(join peer) - ShrObjSet(LP) $\neq \emptyset\}$ and $\{$ShrObjSet(LP) - ShrObjSet(join peer) $\neq \emptyset\}$
SUB: $\{$ShrObjSet(LP) $\subset$ ShrObjSet(join peer)$\}$
EQUAL: $\{$ShrObjSet(LP) == ShrObjSet(join peer)$\}$
NONE: $\{$ShrObjSet(join peer) $\cap$ ShrObjSet(LP) == $\emptyset\}$

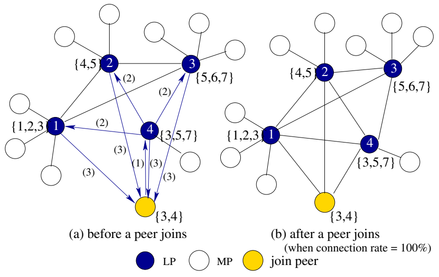
Table 1 shows examples of join flags.

Figure 1-(1) shows an example of a join process. A join peer has a shared object set $\{3,4\}$ and (1) sends a join request to an entry peer, LP4. LP4 compares its shared object set with that of the join peer's. LP4 does not have all objects of the join peer, hence, (2) it forwards the join request to the other LPs, that is, LP1, LP2 and LP3. The LPs receiving the request compare their own shared object sets with that of the join peer and then (3) reply to the join peer with the common object sets and join flags. The replies are $<\{3\}$, PART$>$ for LP1, $<\{4\}$, PART$>$ for LP2, $<\emptyset$, NONE$>$ for LP3, and $<\{3\}$, PART$>$ for LP4.

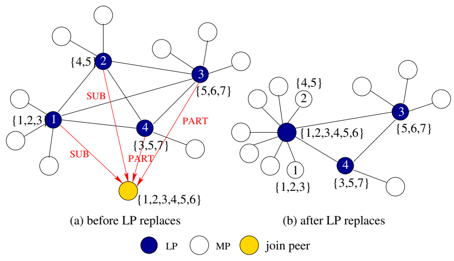
A join peer decides which group to join based on the replies from the LPs. If some LPs send EQUAL or ALL for a join flag, the join peer can be an MP of all of those LPs.

If all LPs reply with the NONE flag, it means that no LP has objects in common with the join peer. Hence, the join peer becomes a new LP and makes a new group. In the new group, there are unique objects not in the other groups, so service requests for those unique objects can be served from only this group. Later, if another peer joins the network and has those objects plus some new unique objects, it will become a new LP and those objects of the previous LP can be published in this group too.

If some LPs reply with NONE, they have no common objects with the join peer and if some reply with PART, they have some common objects. However, the join peer does not know whether or not it can be an LP, because it cannot know whether or not it has unique objects in the current network. Hence, the join peer gathers the replies from the LPs and checks if it has any unique objects. If so, it becomes an LP, otherwise it becomes an MP of one or more of the LPs that sent PART flags. The join peer can connect to only the LP that has the most objects in common with it, or to some LPs that send PART flags. If a join peer connects to more LPs, it will receive more hits on requested objects as a client. *Connection rate* is the number of actual connections divided by the number of



(1) Join process



(2) LP replacement

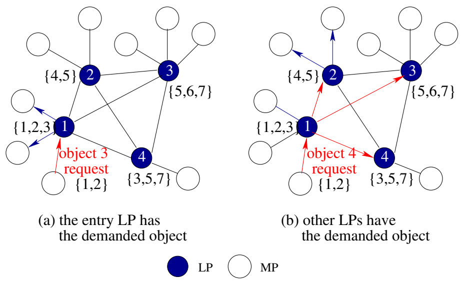

{4,5}
{5,6,7}
{1,2,3}
{3,5,7}
object 3
request
{1,2}
object 4
request
(a) the entry LP has the demanded object
(b) other LPs have the demanded object
LP
MP
(3) Object request

Fig. 1. Overlay scheme example

all possible connections between a join peer and LPs, and indicates the number of LPs with which a join peer is connected.

For example, assume that a join peer has a shared object set {2, 3, 4, 5 } and LP1, LP2, and LP3 have {1, 2, 3, 4}, {4, 5, 6}, and {3, 6, 7} respectively. LP1, LP2, and LP3 send the replies <{2, 3, 4}, PART>, <{4, 5}, PART>, and <{3}, PART> flags to the join peer. The join peer can connect to all the LPs (LP1, LP2, LP3) or some of them, or only one. If the join peer connects to only one LP, it must be LP1, because LP1 has the most common shared objects. The connection rate is a percentage value of the number of common objects divided by the maximum number of common objects. In this example, the maximum number of common objects is three (by LP1). Hence, if the join peer connects to LP1, the connection rate will be 0%, for LP1 and LP2 it will be 50%, and for all LPs it will be 100%. In Fig. 1-(1), the join peer connects to LP1, LP2, and LP4, which send PART join flags, for a connection rate of 100%.

If one or more LPs send SUB join flags, it means that the join peer has all the objects of those LPs. Hence, the join peer replaces those LPs and they and their MPs become members of the new LP. At all times, a new LP has some unique objects not previously in the network. Figure 1-(2) shows the process by which some LPs are replaced by a join peer. The join peer has a shared object set {1, 2, 3, 4, 5, 6}. When the join peer sends a join request, LP1, LP2, LP3, and LP4 return <{1, 2, 3}, SUB>, <{4, 5}, SUB>, <{5, 6}, PART>, and

<{3, 5}, PART>, respectively. The join peer becomes a new LP controlling the group members of LP1 and LP2 because the join peer has all the objects of LP1 and LP2.

2.2 Search Process

All requests are forwarded through LPs. Both LPs and MPs can request services. An LP requests object services from other LPs when it does not have the objects in its group. The request of an MP can be satisfied within its own group when its LP has the requested object. If the LP has the requested object, then some members also have the object. Hence, an MP makes a request to its LP first. If the LP has the object, the LP forwards the request to its other MPs. Otherwise, the LP forwards the request to the other LPs.

Figure 1-(3)(a) shows an example in which a request is satisfied within a group. When an MP requests object 3 from its leader LP1, LP1 checks whether it has the requested object. It does, so it forwards the request to its other MPs. The MPs in the group that have the requested object reply to the client and can be server peers. Then the client MP requests different segments of the object from each server peer using the assignment methods in [7]. If the client does not find enough server peers, LP1 forwards the request to the other LPs to search for more server peers.

When an MP requests an object from its LP but the LP does not have that object, the LP forwards the request to the other LPs. Those LPs that have the requested object forward the request to their own MPs. In Fig.1-(3)(b), a client peer requests object 4 from its leader LP1. LP1 forwards the request to the other LPs because it does not have that object. Among those LPs, LP2 has object 4 and can provide the service. Hence, only LP2 forwards the request to its MPs and the MPs having the object reply to the client. The client selects the server peers from those MPs.

2.3 Leaving Process

When an MP leaves the P2P network, it sends a "leave" message to the LP of its group and the LP deletes the MP's information from the MP list. When an LP leaves the P2P network, a member of its group replaces it and the new LP notifies the other group members of its takeover. The MPs of the group change their information about the LP. Without takeover, all the MPs in a group would be disconnected from the network by the departure of their LP. To address this problem, the LP selects a candidate LP to replace it.

An LP selects the candidate LP that has the most of the common shared objects. The LP knows the shared object sets of all its MPs because peers send their shared object sets when they join the network. Hence, an LP updates the information about possible candidate LPs whenever a new join peer has more common shared objects than the current candidate LP. However, LPs do not manage the detailed information about the shared object sets of MPs.

Table 2. Simulation parameters

Parameter	Value
Number of peers	5000
Number of total objects in P2P network	1000
Max. number of objects of a peer	50–800
Interval between requests by a peer	86400 sec. (exponential distr.)
Lifetime of peers	1800–14400 sec. (exponential distr.)
Idle time between connections	21600–345600 sec. (exponential distr.)
Connection rate	0–40%
Number of neighbors in Gnutella	10
Time-to-live (TTL) in Gnutella	7
Simulation time	864000 sec. (10 days)

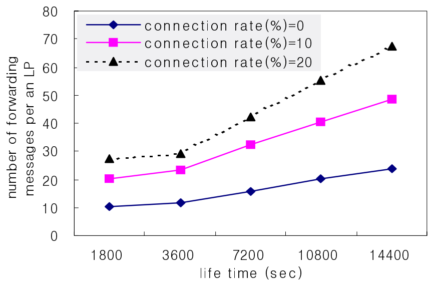
When an LP leaves the network, it sends a "take over" message to its candidate LP. The candidate LP takes over and then notifies the other LPs and its MPs of the change of LP. Then the MPs replace their current LP with the new LP. After the change of LP, some MPs may not be members of the group any longer. For example, assume that the current LP has the shared object set {1, 2, 3, 4, 5}, its candidate LP has {3, 4, 5}, and one of its MPs has {1, 2}. The candidate LP and the MP can be in the same group with the current LP because they have the common object sets {3, 4, 5} and {1, 2}, respectively. However, after the candidate LP takes over, the MP cannot be in the same group because it has no common objects with the new LP. To address this case, the candidate LP sends a state-change message to its MPs after takeover. All MPs compare their own object sets with the LP's and inappropriate MPs in the group rejoin the network.

3 Simulation

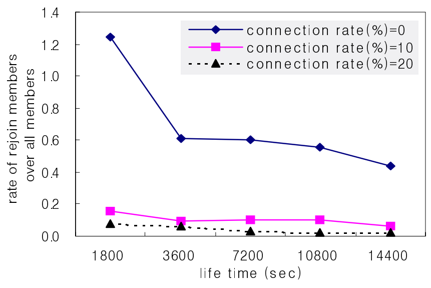
In this section, we present and discuss simulation results. The parameters used for the simulation are shown in Table 2. We used CSim for the simulation. Peers connect to (join) the network and request services or provide them during their lifetimes and then disconnect from the network. All peers repeat this processes during the simulation time.

When the connection rate is large, a join peer becomes the member of more groups. Hence, the number of MPs of an LP increases when the connection rate increases. Similarly, the hit rate and the number of redundant forwarding messages also increase.

It is not necessary for a client to find all the peers having the requested object for an on-demand streaming service. In the proposed scheme, a client can find the appropriate number of peers for a service by adjusting the connection rate. Unnecessary messages for a search can be reduced when the connection rate is small. In Fig. 2, the performance is compared as the lifetime varies for 0%, 10%, and 20% connection rates. Figure 2-(a) shows that the forwarding overhead for an LP is small for small connection rates because the group size is small. However, Fig.2-(b) shows that a 0% connection rate can cause problems,



(a)



(b)

Fig. 2. The effects of connection rate

as all MPs in a group must rejoin the network when the group's LP leaves the network. This makes the rejoin overhead high. Hence, the selection of the proper connection rate can improve the performance by reducing overheads for leaving nodes.

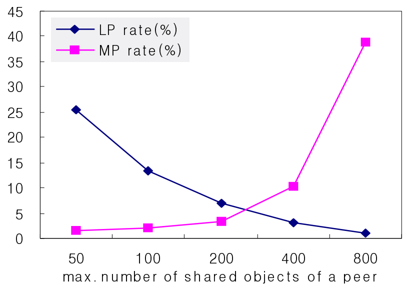
In Fig. 3-(a) the LP rate is the number of LPs divided by the number of live nodes and the MP rate is the number of members in a group divided by the number of live nodes. If peers have very large stable storages and they can share enormous numbers of objects, the probability that a peer has unique objects in the network becomes low. Hence, the number of LPs decreases and the number of MPs increases. If we assume the size of an object is 600 MB, the realistic number of objects of a peer is about 100–200 and the LP rate is about 5–15%. Figure 3-(a) also shows that each LP manages less than 5% of the live peers as MPs when the maximum number of shared objects is less than 200.

Figure 3-(b) shows the amount of information an LP must maintain. An LP manages the location of all LPs and its MPs. As the maximum number of shared objects in each peer increases, each LP maintains less LP information and more MP information. In this figure, the number of LPs and MPs is balanced when the maximum number of shared objects is 200. Even though we cannot control the number of LPs and MPs, there is a balanced value depending on the maximum number of shared objects.

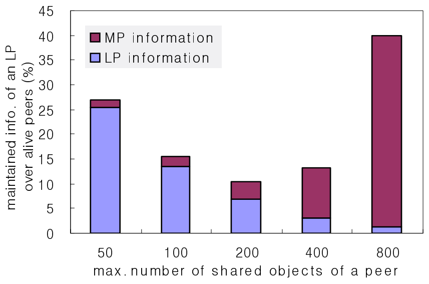
We compared the performance with Gnutella. We do not show the results because of the space limitation, but we found that the proposed scheme reduces traffic overhead, hop counts, and the number of messages compared to Gnutella, at the cost of overhead as peers join and leave the network.

4 Conclusion

In this paper, we proposed an overlay scheme using shared objects for an on-demand streaming service in a pure P2P network. The proposed scheme is composed of groups and all peers in a group have common objects. A leader peer



(a)



(b)

Fig. 3. The overheads of the proposed overlay

(LP) in a group serves as a directory server while the member peers (MPs) provide on-demand streaming services. The role of peers is decided autonomously when a peer joins the network, and service requests are managed by some peers without any server installation and management cost. The environment for the on-demand streaming service can be tuned by careful selection of the connection rate.

References

[1] V.N. Padmanabhan, H.J. Wang, P.A. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in ACM/IEEE NOSSDAV, 2002. 784

[2] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang, "A case for end system multicast," in ACM SIGMETRICS, 2000. 784

[3] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming Live Media over a Peer-to-Peer Network," Stanford Database Group Technical Report (2001-30), Aug. 2001. 784

[4] S. Banerjee, Bobby Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in ACM SIGCOMM, 2002. 784

[5] Duc A. Tran, Kien A. Hua, and Tai Do, "ZIGZAG: An efficient peer-to-peer scheme for media streaming," in IEEE INFOCOM, 2003 784

[6] W.T. Leung and J.Y.B. Lee, "A server-less architecture for building scalable, reliable and cost-effective video-on-demand systems," in Internet2 Workshop on Collaborative Computing in Higher Education: Peer-to-Peer and Beyond, 2002 784

[7] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On Peer-to-Peer Media Streaming," in IEEE ICDCS, 2002. 784, 788

[8] Kien A. Hua, Duc A. Tran, and Roy Villafane, "Overlay multicast for video on demand on the internet," in ACM Symposium on Applied Computing, 2003. 784

[9] Duc A. Tran, Kien A. Hua, and Simon Sheu, "A new caching architecture for efficient video services on the internet," in IEEE Symposium on Applications and the Internet, 2003. 784

[10] B. Yang and H. G. Molina, "Designing a super-peer network," in IEEE ICDE, 2002. 784