Closed P2P System for PVR-Based File Sharing

Seungtaek Oh, Jin-Soo Kim, Ki-Sok Kong, and Joonwon Lee

Abstract — *Most data placement schemes of P2P systems are based on open systems where a node can access any data in the system once the node joins the system, which is not suitable for PVR-based file sharing, because PVR is a private device. This paper suggests a closed P2P model where a node is connected only with some specific nodes and data are shared only within those nodes. In this model, since shared files usually contain large multimedia data, the data placement scheme employed should maximize the space available to each node by increasing the degree of sharing and by reducing redundancy in data placement. In this paper, we prove that the problem to maximize the space is NP-hard and suggest several heuristic approaches that can be realized on a peer-to-peer architecture. Performance problems of the above algorithms are explored through simulation studies*.

Index Terms — Closed P2P, peer-to-peer, personal video recorder, storage extension problem

I. INTRODUCTION

The advent of digital TV enables us to handle multimedia data as with computers since a digital TV stores and manages media data in digital format. Because of this digital nature, TV viewers can easily store and copy media contents broadcasted on TV. PVR (personal video recorder) is a device that stores media data on persistent storage like hard disks, whereas a VCR does it on magnetic tapes. Due to random accessibility of hard disk, PVR can record multiple media streams, or record a stream and play back another stream concurrently. In addition, PVR has the functionalities of fast forward, fast rewind and pause of the on-air TV streams. Moreover, because broadcast multimedia data are recorded in digital format, they are easily copied and shared between PVRs that are connected through a network.

The data sharing between PVRs is possible in several ways, but the easiest and natural method is using P2P (peer-to-peer) systems[1, 2, 9]. P2P systems have been proposed to overcome the limitation of client/server model where servers easily become bottleneck. P2P systems are different from previous network model where the roles of server and client are fixed. In P2P systems each user can furnish data as a server and each user can access other user's data as a client. This symmetric nature of a P2P system makes itself the best candidate for an architecture for data sharing between PVRs.

In traditional P2P systems [7], a node connects to one arbitrary node to join a system, and it collects the information about the system from the connected node. This information usually comprises names of connected nodes and locations of each file and access privilege matrix. We call this system as an *open P2P system*, since connection to the system and access to data in the system are open to anyone and the access to data is controlled only through the access control matrix.

However, this type of P2P system is undesirable for PVR-based file sharing because of the following reasons. The amount of data that can be stored in a PVR is dependent on the size of hard disks, and it is usually much larger than that of magnetic video tapes. But, considering the typical size of a movie which is several giga bytes, current storage technology allows each PVR to store only dozens of movies. As the capacity of storage increases, the size of the multimedia objects will grow too, since current multimedia data are usually compressed to be fit within limited storage size sacrificing the quality of media, and HDTV (high definition TV) will deliver much larger contents. Therefore the owner of a PVR will let it be shared by other PVRs only when she can access. Traditional open P2P systems like Gnutella are inappropriate for this purpose because the owner of the PVR equipment would not be happy if many anonymous people access her resources without offering her any benefit.

A *closed P2P system* relies on an agreement between nodes for sharing files, and only the nodes that agree to share files can access the shared files. For example, in a messenger-style P2P model, a node should make an agreement with another node to communicate with the node, and it can communicate only with the nodes in agreement. Even if node A makes an agreement with node B, and node B makes an agreement with node C, node A cannot communicate with node C unless there is an agreement between node A and node C. By the closed P2P system, accessibility to shared data can be limited only within a group that has an agreement.

In a closed P2P system with PVRs, a lack of data placement control may result in redundant file placement and reduced effective total storage, and such a space problem should not be ignored when the sizes of shared objects are large. Therefore, a policy to decide data placement among P2P nodes is necessary. Since the main motivation of file sharing for a PVR owner is to access larger media contents than can be accommodated on its own PVR, the data placement scheme for such P2P system should minimize redundancy in storing shared files. We define *storage extension problem* as a data

Seungtaek Oh is with Mobile Platform Lab. at Software Center, Samsung Electronics, Seoul, Korea. (e-mail: st74.oh@samsung.com)

Ki-sok Kong is with the Department of Computer Engineering at Korea Polytechnic University, Shihung-City, Korea. (e-mail: kskong@kpu.ac.kr)

Jin-Soo Kim, and Joonwon Lee are with the Department of EECS, Korea Advanced Institute Science and Technology, Daejeon, Korea. (e-mail: jinsoo@cs.kaist.ac.kr, joon@cs.kaist.ac.kr)

Contributed Paper
Manuscript received July 15, 2005

placement problem in the closed P2P model. A solution to this problem will maximize the total sum of object sizes that are accessible from interested nodes. In this paper, we present a formal description of the problem and a few heuristic solutions to it.

This paper is structured as follows: we start in Section 2 with the definition of closed P2P model and storage extension problem. In Section 3, we present algorithms for the storage extension problem, and we evaluate the suggested algorithms by simulation in section 4. Section 5 discusses related work, and Section 6 concludes the paper.

II. Storage Extension Problem in Closed P2P

A. Closed P2P model

Napster, one of the open P2P systems, relies on a centralized server for directory service. A node should inquire of the centralized server about the locations of data. The wanted data may reside on several nodes, and the user can get the data from any node among the nodes because all storages of the system are open to any user in the system. In this case, many users may contact to the same node simultaneously, but it may not be a problem because Napster is targeted for sharing small size audio files. However, PVRs share large size video files, and thus it may raise a bottleneck problem when many anonymous users contacts to the same PVR at the same time.

To prevent anonymous users from accessing a node, an agreement is needed. The agreement of closed P2P system is the permission to access the data of a node. Even if node A knows the position of other node B, if the node A has no agreement with node B, node A cannot access the data of node B. If node A wants to make an agreement with node B, node A should get content from node B.

Storages can be shared between nodes that established an agreement, and a user can use the storage of the *neighbor nodes* (nodes that have the agreement with the node). If a node runs short of storage, it can store its data item in the storage of a neighbor node. In general P2P systems, such storage sharing is not desirable, because the storage sharing reduces the availability since any node can be shut down without prior notice. However, the storage sharing of closed P2P systems is suitable for PVR-based P2P systems, because PVRs hardly move and they are usually turned on all the times.

The closed P2P model can be realized on any open P2P system. If a node wants to establish an agreement with a certain node, it may ask the lookup service of the open P2P system for the location, and an agreement can be established easily between the two nodes by exchanging a few messages. Once an agreement is made between them, two nodes can freely access the storage of each other. An agreement can be canceled by a node by sending a cancellation message.

Figure 1 shows closed P2P model where each circle denotes a node and each link shows that connecting nodes have an agreement. If there is a link between node A and node B, node A can access the storage of node B as well as its own storage, and vice versa. As the link represents a logical connection, it is meaningful only between two connecting nodes. We define nodes and links in closed P2P model as follows.

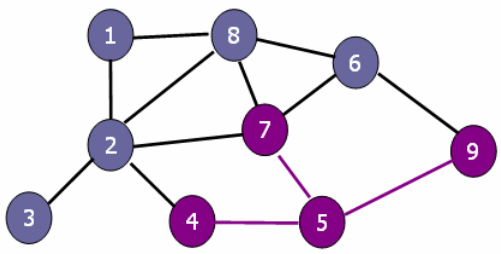

Fig. 1. Virtual storage group

• ***Node***: It means a peer in a P2P system. Each node has some amount of storage, and donates a part of it to share data with other nodes. It has a unique id, n_i, and N is a set of nodes.

$N = \{n_1, n_2, \ldots, n_i, \ldots, n_n\}$

• ***Link***: If two nodes have an agreement, there is a link between them. Storage between agreed nodes can be shared by both nodes. L is a set of links.

$L \subseteq N \times N$

A closed P2P model can be represented as a graph.

$G = (V, E)$ where $V = N$ and $E = L$

• ***Virtual storage group (VSG)***: A node can use its own storage and the storage of all the linked nodes. Virtual storage group VSG_i of a node n_i is a group of nodes that share storage with the node n_i, i.e., nodes that have links to the node n_i. In Figure 1, VSG_5 is $\{n_4, n_5, n_7, n_9\}$.

$VSG_i = \{n_i\} \cup \{ n_j \mid (n_i, n_j) \in L\}$

• ***VSG storage***: It is the sum of all storage in a VSG.

B. Storage extension problem

Data placement is important in a closed P2P system because smart algorithms will enlarge the total storage of the system. In traditional open P2P systems [7, 10, 11, 12], the decision of data placement is made solely by each node independently, and such a lack of control may result in redundant file placement and reduced effective total storage. This kind of space problem should not be ignored when shared objects are large size video files.

In a closed P2P system, some of favorite data may be stored in its VSG storage. However, since there is only limited space in the VSG storage, some data of interest may not be stored anywhere. Following terms need to be defined to describe the problems.

We define

• ***Data (D)***: A set of data.

$D = \{d_1, d_2, \ldots, d_i, \ldots, d_m\}$

• ***Physical storage size (P)***: The size of data that can be stored in a node n_i.

• ***Favorite data (FD)***: We define FD as a set of pairs (n_i, F_i), where F_i is a set of data that are of interest to the node n_i.

$FD = \{(n_1, F_1), (n_2, F_2), \ldots, (n_i, F_i), \ldots, (n_n, F_n)\}$ where $F_i \subseteq D$

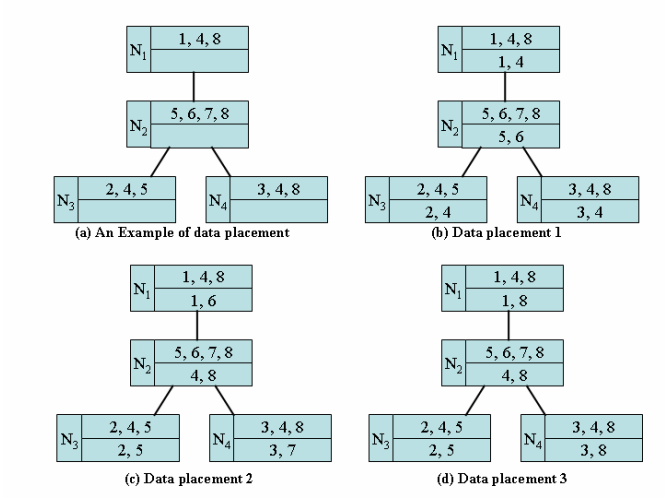

Fig. 2. Examples of data placement

In our model of P2P storage system with PVR, each node has a set of data of interest. These kinds of favorite data can be easily specified by the present PVR system. In TiVo [14], a famous PVR system, a user can specify a favorite data, and besides favorite data can be predicted by TiVo. TiVo can analyze the pattern of user's favorite data and regard the series of favorite object or the similar kinds of favorite object as favorite data.

• ***Stored data (SD)***: *SD* is a set of pairs (n_i, S_i) where each S_i represent a set of data that stored in the storage of node n_i. Note that S_i needs not be a subset of F_i; it is possible for a node to store uninterested data in its own storage, permitting the data to be accessed by the nodes in its virtual storage group.

$SD = \{(n_1, S_1), (n_2, S_2), \ldots, (n_i, S_i), \ldots, (n_n, S_n)\}$ where $S_i \subseteq D$ and $|S_i| \le P$.

• ***Stored data of virtual storage group* (*SVSG$_i$*)**: A set of data stored in VSG_i.

$SVSG_i = \{d \mid d \in S_i$ where $n_i \in VSG_i$ and $(n_i, S_i) \in SD\}$

• ***Virtual storage size (VSS)*:** A node can access the whole VSG storage, and it may contain some data of interest to other nodes since the storage is shared among the group. Only the space of the VSG storage that contains favorite data of a node is meaningful to the node. We call the size of meaningful space the virtual storage size (VSS).

$$VSS_i = | SVSG_i \cap F_i |$$

• ***Total virtual storage size (TVSS)*:** The total sum of VSS of all the nodes in a system. It is natural for any data placement algorithm to maximize this metric since the larger VSS means that more favorite data can be stored.

$$TVSS = \sum_{i=1}^{n} VSSi$$

Data placement schemes affect the total virtual storage size and the available storage of each node. Figure 2 illustrates several examples of data placement schemes, where each square box denotes a node; node number in the left, favorite data in top right, and stored data in bottom right. It is assumed that the sizes of all data are same and the physical storage sizes of all nodes are twice the size of data. Figure 2(a) shows the initial setting where no data has been placed on the storage. Figure 2(b) and (c) illustrate two different data placement schemes. Table 1 shows the virtual storage sizes of these two placement schemes. This example shows that the total virtual storage size depends on the effectiveness of the data placement scheme. In Figure 2(c), the total virtual storage size is 13, and all nodes can access all their favorite data. Though the physical storage size of n_2 is 2, this data placement increases its virtual storage size to 4. It can be seen that the total virtual storage can be increased by forcing some node to store unwanted data.

Problem 2.1. **Storage extension problem**: Consider a closed P2P model represented as a graph *G* with a node set *N* and a link set *L*.

$G = (V, E)$ where $V = N$ and $E = L \subseteq V \times V$

When *G*, *D*, *P*, and *FD* are given, the solution of storage extension problem is to find an *SD* that maximizes the total virtual storage size (TVSS).

C. NP-hardness of storage extension problem

In this subsection, we show the storage extension problem is an NP-hard problem. We can prove it using the minimum dominating set problem [5,6].

1) Minimum dominating set problem

Consider a graph with a vertex set *V* and an edge set *E* such that:

$G = (V, E)$ where $E \subseteq V \times V$

A dominating set V' for *G* is defined as follows.

$V' \subseteq V$ such that for all $u \in V - V'$, there is a $v \in V'$ for which $(u, v) \in E$.

A dominating set with the minimum number of vertices is called a minimum dominating set, and its cardinality is termed the domination number of *G*. It is well-known that this problem is NP-complete for general undirected graphs [5].

2) Proof of NP-hardness of the storage extension problem

The minimum dominating set problem can be reduced to the storage extension problem in polynomial time as follows.

For a given minimum dominating set problem, we can consider a corresponding storage extension problem as follows:

$G = (N, L)$ where $N = V$, $L = E$

$D = \{d_1, d_2, \ldots, d_i, \ldots, d_n, d_{n+1}\}$

Sizeof$(d_i) = 1$

$F_i = \{d_i, d_{n+1}\}$

$P = 1$

Assume that we have a solution for the storage extension problem.

$SD = \{(n_1, S_1), (n_2, S_2), \ldots, (n_i, S_i), \ldots, (n_n, S_n)\}$

In the solution, we can define three distinct sets of vertices.

N_1 = a set of n_i, where $(n_i, \{d_{n+1}\}) \in SD$

N_2 = a set of n_i, such that for all $n_i \notin N_1$, there is $n_j \in N_1$ where $(n_i, n_j) \in E$

$N_3 = N - N_1 - N_2$

Definition 2.1. *Virtual Storage Contribution Factor (VSCF)*

The virtual storage of n_i is described as $SVSGi \bigcap Fi$. $VSCF_j$ denotes that the number of nodes that are interested in d_j and are able to access the data from their own virtual storage group.

$VSCF_j = |\{n_i \mid n_i \in N, d_j \in F_i, \text{ and } d_j \in SVSG_i\}|$

In Figure 2(b), $VSCF_4$ is 3, because d_4 is the favorite data of 3 nodes (n_1, n_3, n_4) and it can be accessed by the nodes, and $VSCF_8$ is 0, because d_8 is the favorite data of 3 nodes (n_1, n_2, n_4) but no node can access d_8. TVSS is decreased by $VSCF_j$, if we remove the data d_j from S_i for all node i. In Figure 2(b), if d_4 is removed from the stored data of all nodes, TVSS is decreased by 3.

Lemma 2.1. In the solution of the storage extension problem, $VSCF_j$ is 1, for all d_j such that $(n_i, \{d_j\}) \in SD$, and $n_i \in N_2 \cup N_3$.

Proof. Since $n_i \in N_2 \cup N_3$, d_j is one of $\{d_1, d_2, \ldots, d_n\}$. There are no two vertices that have the same d_j by their favorite data. Hence, $VSCF_j$ cannot be greater than 1. $VSCF_j$ cannot be smaller than 1, either. Consider a solution of the storage extension problem. In the solution we assume there is a node n_i where $(n_i, \{d_j\}) \in SD$, $n_i \in N_2 \cup N_3$, and $VSCF_j$ is 0. We also assume that n_i has n links. It means that VSG_i has $n+1$ nodes and the nodes have $n+2$ favorite data (d_k where $n_k \in VSG_i$, and d_{n+1}). Because the total physical storage size of the neighbors of n_i is n, we can find at least two favorite data which are not stored in n_i or any neighbor of n_i. If we store the data in n_i, we can increase the total virtual storage size. Thus we can conclude that the given solution can not be a solution of storage extension problem, which means $VSCF_j$ can not be 0 for some d_j such that $(n_i, \{d_j\}) \in$ SD, and $n_i \in N_2 \cup N_3$. □

Lemma 2.2. $N_1 \cup N_3$ is a dominating set

Proof. By definition, all $n_i \in N_2$ have an edge to $n_j \in N_1$. So $N_1 \cup N_3$ is a dominating set. □

Lemma 2.3. The total virtual storage size of the solution is $|N| + |N_2|$

Proof. When a solution for the storage extension problem is given, we can make another solution, SD'.

$SD' = \{(n_1, S_1'), (n_2, S_2'), \ldots, (n_i, S_i'), \ldots, (n_n, S_n')\}$, where

$S_i' = S_i$, if $n_i \in N_1 \cup N_2$

$S_i' = \{d_{n+1}\}$, if $n_i \in N_3$

SD' is a new solution that has no element of N_3 by replacing the stored data of N_3 with d_{n+1}. The replacement does not increase the total virtual storage size; otherwise the original solution SD can not be the solution of the storage extension problem. The replacement does not decrease the total virtual storage size, either. $VSCF_j$ is 1 by Lemma 2.1, and there is no edge from the vertices of N_3 to vertices of N_1. In SD', because all N_3 are removed, only N_1 dominates all vertices. Hence the $VSCF_{n+1}$ becomes $|N|$, and $VSCF_j$ is 1, where $n_i \in N_2$ and $(n_i, \{d_j\}) \in SD$. Accordingly, the total virtual storage size is $|N| + |N_2|$. □

Theorem 2.1. The solution of minimum dominating set is $N_1 \cup N_3$.

Proof. According to Lemma 2.2, $N_1 \cup N_3$ is the dominating set, and according to Lemma 2.3, the total virtual storage size is $|N| + |N_2| = 2|N| - (|N_1| + |N_3|)$. The given solution has the maximum total virtual storage size; it means that it has the minimum value of $|N_1| + |N_3|$. So, $N_1 \cup N_3$ is a minimum dominating set. □

III. Data Placement Algorithms

Since the storage extension problem is NP-hard, we need to devise effective heuristic algorithms. There are many factors that affect the real performance of each algorithm such as network bandwidth, node performance, changes in data preference, data size, and so on. To simplify the problem without loss of any applicability of the algorithm, we made the following assumptions.

- Same storage size: All nodes have the same storage size.
- Same data size: All data sizes are equal. Therefore, each node can store the same number of data.
- Sufficient network bandwidth: There is no network bottleneck and a node can always perform services to other linked nodes.
- Off-line algorithm: We assume that all nodes and their agreement and favorite data are known in advance. In this paper, we only consider the initial data placement assuming that there is no change in node configurations or data preference.

The simplest algorithm that we can think of is the greedy one which considers only its own favorite data and tries to allocate them on its own storage. Because of the greediness, this one will fail to maximize the total virtual storage size. We call this one *owner-based algorithm*. Another class of algorithms considers the preferences of all the nodes in the VSG and tries to maximize the total virtual storage size. We call them *group-based algorithms*. Below, we describe one owner-based algorithm and four group-based algorithms; random, most-popular, most-popular with neighbor consideration, maximum popular value.

- Owner-based random (OR)

When the storage of a node is not sufficient to store all the favorite data of the node, data are randomly selected. It is the simplest algorithm that does not consider the virtual storage group.

- Group-based random (GR)

A node randomly selects data among the favorite data of the nodes in its VSG. Since this one relies on randomness without considering global optimization, we expect that its effectiveness would not be satisfactory.

- Most-popular (MP)

This algorithm selects data among the favorite data of all the nodes in its VSG according to the popularity of the data. *Popular value* (V_{ij}) is the number of nodes that would be able to access data d_j if the data is stored in node n_i. Table 2 shows an example of all popular values of node n_2, when the favorite data are given as in Figure 2(a). Data to be stored in node n_2 are selected according to these popular values to maximize the happiness of the nodes that are interested in the data. Figure 2(d) shows the resulting data placement when we employ the

TABLE II
ALL POPULAR VALUES OF NODE N_2 IN FIGURE 2(A)

Data number	1	2	3	4	5	6	7	8
Popular value (V_{2j})	1	1	1	3	2	1	1	3
Benefit node	n_1	n_3	n_4	n_2, n_2, n_2,	n_2, n_3	n_2	n_2	n_2, n_2, n_2,

most-popular algorithm.

• Most-popular with neighbor consideration (MPNC)

The optimization considered in the previous algorithm is local in that each node tries to maximize the total sum of popular values of data stored in that node only without considering the data placed in other nodes. In Figure 2(d), data 8 is stored in n_1, n_2, n_4. However, it is enough for data 8 to be stored only in n_2, because n_1 and n_4 have a link with n_2. MPNC prevents such unnecessary duplication by not allowing linked nodes to have the same data. In other words, if two linked nodes have the same favorite data, the data are stored only in one of them. Naturally, the tie breaking rule chooses the node which has the higher popular value than the other node. This algorithm needs communications between neighboring nodes, but it does not require global information about all the nodes in VSG.

• Maximum popular value first (MPVF)

Even though the MPNC algorithm considers neighboring nodes, its optimization is still local as it does not consider data placed in nodes of more than one-hops away. As a result, it is sometimes possible that the effect of data placement decreases the virtual storage size of the nodes at long distance. To eliminate this effect, it is necessary to take into account all the nodes in designing data placement algorithms. For MPVF, we need a centralized server like Napster, and we assume that a centralized server collects all the needed information from every node, and decides which data should be placed in which node.

The centralized server prepares a *global popular value table* that contains the popular values of all nodes and data, and then makes a decision on data placement according to this table. It repeats the selection of the maximum popular value as shown in Figure 3. V_{ij} in the table denotes a popular value of data j for node n_i. The selection of V_{ij} by this algorithm means that data j would be stored in node i. In the first step, V_{24}, the maximum popular value, is selected. The selection of V_{24} means that data 4 are stored in node n_2. Nodes linked to node n_2 (node n_1, n_3, and n_4 in this example) need not store data 4 any more, and thus, V_{14}, V_{34}, and V_{44} are set to 0. Next, we select V_{24} which is the maximum among unselected popular values, and set V_{18}, V_{38}, and V_{48} to 0. Because one node store only two data, node n_2 can not store data any more. It means that the popular value of column 2 can not be selected any more, and all popular values of column 2 should be set to 0. This process is repeated until all nodes have no more room to store any data or there is no more data to store.

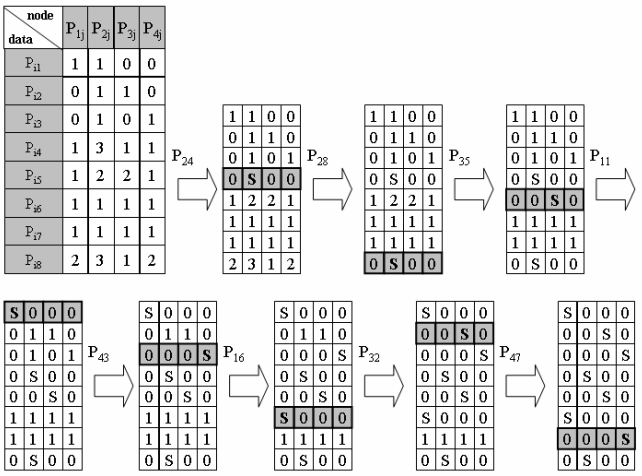

Fig. 3. The process of MPVF in Figure 2(a)

IV. PERFORMANCE

The goal of algorithms presented in section 3.1 is to maximize the total virtual storage size. We use simulation to evaluate the effectiveness of the suggested five algorithms. The parameters considered for this simulation studies are as follows.

1. n is the number of nodes.

$$n = |\mathrm{N}|$$

2. l is the average number of links for each node in the whole P2P system.

$$l = 2\frac{|L|}{|N|}$$

Given n and l, the possibility that there is a link between two arbitrary nodes is $\frac{l}{n-1}$.

3. d is the total number of data.

$$d = |\mathrm{D}|$$

4. f is the number of favorite data for each node.

$$f = |F_i|$$

Given d and f, the possibility that arbitrary data are the favorite data of an arbitrary node is $\frac{f}{d}$.

5. P is the physical storage size of a node in terms of the number of data that can be stored.

Figure 4 shows the total virtual storage size varying the number of nodes from 10 to 10,000 and fixing other parameters such that l = 3, d = 50, f = 15, and P = 5. The results are the average value of 10 simulations. In each simulation, all nodes set up new links and new data preferences. The TVSS are normalized according to OR. We can see that the number of nodes does not affect the performance of algorithms unless the number of nodes is very small. If the number of nodes is larger than 200, we can find the similar performance gaps among five algorithms.

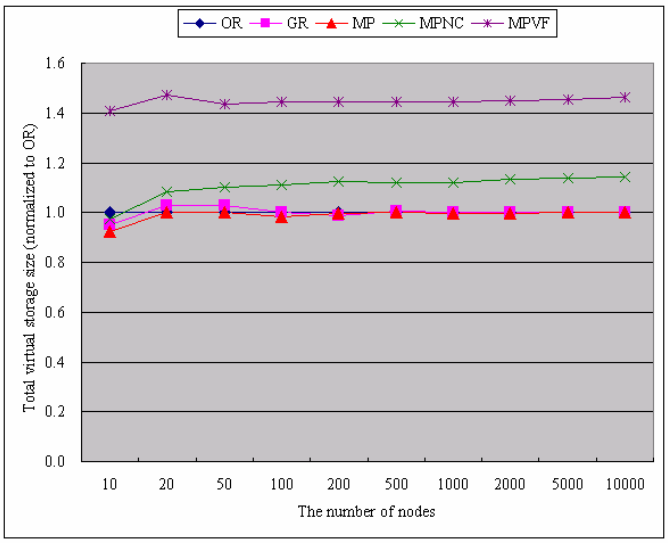

Fig. 4. The normalized TVSS varying the number of nodes

Figure 5 shows the results of varying the physical storage size from 1 to 11 and fixing other parameters such that n = 1,000, l = 3, d = 50, and f = 15. Though we explored almost all possible design space, we present only meaningful results here.

The performance of OR, GR, and MP algorithms are similar. Though we expected that the MP algorithm would outperform others significantly, its blindness to the linked nodes entails somewhat amount of undesirable duplications of stored data, and this duplication downgrades the performance compared to other random algorithms.

MPNC prevents two linked nodes to store the same data, and thus eliminates some duplication. However, even if two nodes have a link, it is sometimes desirable to have the same data due to the other nodes linked to them. MPNC does not consider this situation, and this limitation degrades the performance more as the physical size becomes larger. It is surprising that MPNC is sometimes inferior even to other simpler algorithms.

The performance of MPVF is always the best. It was expected from the beginning since this algorithm considers global situation and the placement is done one by one reflecting the effect of the previous placement. In Figure 5, we can see that the increment of the total virtual storage size slows down in the range of the physical storage size of 9, and it means that almost all the nodes can access all the favorite data. The virtual storage size should be 15,000, if all nodes can access all favorite data of them.

In Figure 6, we test different data access pattern, Zipf-like distribution [14], which is well-known as a pattern of web access. Zipf-like distribution states that popularity of the i'th most popular data is proportional to $1/i^{\alpha}$, $\alpha \approx 1.0$. We simulate the total virtual storage size varying the physical storage size, as we did in Figure 5. The simulation result is similar to Figure 5, but the total virtual storage sizes are larger than those of Figure 5. The reason is because a few popular data are the favorite data of many nodes and the sharing of the data can easily increase the total virtual storage size. MPNC does not

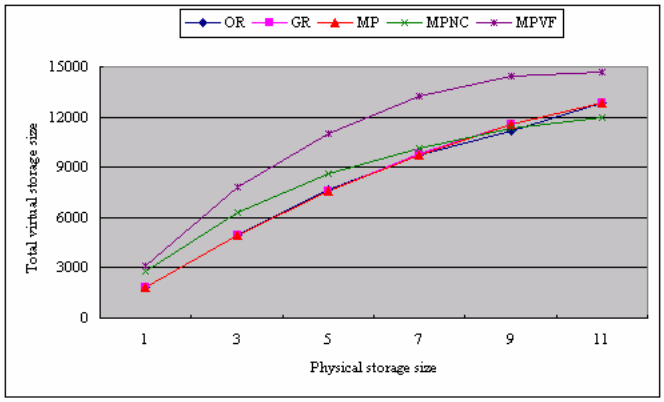

Fig. 5. The effect of varying the physical storage size

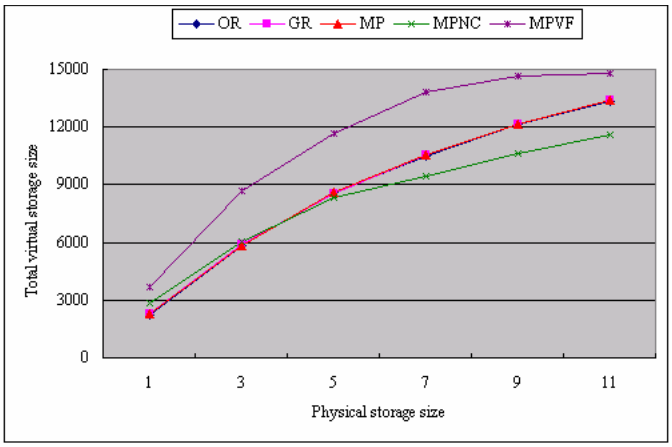

Fig. 6. The performance with Zipf-like distribution

perform well with Zipf-like distribution, because MPNC discourages sharing of popular data.

To acquire the global knowledge, MPVF incurs high communication overhead. The iterative nature of the algorithm increases the complexity. Moreover, MPVF needs the centralized server and it has a weakness of the scalability. If we can make MPVF decentralized, we can use the algorithm in the P2P system with a large number of PVRs. We can make the decentralized MPVF algorithm by regarding all the nodes as the server. In the algorithm, each node should execute MPVF independently and decide which data items would be stored in the node. For the decentralized algorithm, all the nodes should know all the favorite data of all the nodes at a time. The overhead of the algorithm is larger than original centralized MPVF.

Instead of that, we can consider a decentralized MPVF which is loosely synchronized. In this algorithm, each node sends the lists of the favorite data to the neighbor nodes periodically. Though the lists should include all favorite data known to a node, the size of sending message may be small because the node sends only the difference from the former notification. Since the lists need not be sent out immediately whenever there is a change, they may be attached to other data packet to reduce communication overhead.

In this decentralized MPVF, each node executes the MPVF algorithm independently to decide which data to be stored on

its own storage and predicts what will be stored in the neighbor nodes. The prediction may be wrong, because the notification of the favorite data is stale. A node can easily recognize this wrong situation by checking the stored data of neighbor nodes. For that, the list of the stored data should be sent with the lists of favorite data. However, the list need not include the stored data of other node, but it should include only the stored data of the node itself. If the stored data of neighbor nodes differ from the prediction of the stored data, the node should execute the algorithm with the new list of favorite data. The cost of this algorithm would be prohibitive if global information and data preferences change very often. Fortunately, in PVR, such changes are very rare.

V. Related Work

P2P systems began with the unstructured P2P systems where a node can freely select and store its data, such as Napster [10], Gnutella [7], Freenet [11], and KaZaA [12]. Because there is no rule to store data, unstructured P2P systems need the lookup algorithms. Napster has the centralized server which manages the information of the nodes and the data in the system and the nodes query the server to find the location of data item. Though two nodes transfer the data directly, Napster still has the drawbacks of the client/server systems, because the centralized server should have the information of all nodes and data. Gnutella and Freenet use the distributed lookup where the centralized server is not needed. Flooding is used for lookup in Gnutella, and Freenet use the routing based on lexical closeness of keys. Because these lookups do not require the centralized server, they can overcome the drawbacks of the client/server model. However, the lookups of Gnutella and Freenet are not guaranteed to find an existing object and use much network bandwidth. The lookup of KaZaA is the hybrid of the centralized lookup and the distributed lookup. In KaZaA, there exist index servers (super nodes) like Napster, but the centralization can be reduced by distributing the indexes of the nodes to many super nodes.

However, unstructured P2P systems do not completely overcome the limitation of scalability, and structured P2P systems are studied for solving the problem. Lookup is very fast in the structured P2P systems, because the data items are stored in the fixed nodes by the rule of the P2P system, but we need the efficient placement of nodes and data items for the efficient lookup. CAN maps the nodes in a d-dimensional Cartesian coordinated space [4]. Because each node maintains the routing table only for $2d$ nodes, the overhead of routing table is small; however, the length of routing path increases rapidly, as the number of nodes in the system becomes larger. Chord maps the nodes in a one-dimensional space with m-bits address and routes the message based on numerical difference of the destination address [3]. Tapestry [13] and Pastry [8] use the longest prefix/suffix address routing where the routing table is constructed by the routing levels and each routing level has the pointers of the neighbor nodes which match the prefix/suffix for that levels. Tapestry and Pastry also use the network locality for the efficient routing. One of the most important objects of the data placements of structured P2P systems is the fast lookup. However, the efficient data placement also guarantees to find the existing objects and sometimes make the systems fault tolerant.

Data replication studies are somewhat similar to our study in point that efficient data placement improves the performance of the P2P system. However traditional P2P model is not suitable for the PVR-based P2P model. Because unstructured P2P systems do not control the data placement, the redundant file placement reduces the effective total storage. In structured P2P systems, a user can not decide the position of the data. Instead the data are stored in the fixed nodes by the rule of the P2P system, where the data items can be stored far from the user and they can be accessed by the anonymous nodes. Therefore, structured P2P systems are not suitable for PVR-based P2P systems, either.

In closed P2P model, a node can communicate and share its data only with some nodes that have an agreement. This model is very useful and already used in real environment, but no previous P2P systems are based on the model. In this model, we can study the storage extension problem which is not considered in the previous model, and we present a formal description of the problem and a few heuristic solutions to it.

VI. Conclusion

This paper suggested the PVR-based P2P model for private communications and sharing where a node links to a small number of nodes and communicates only with them. We found that a smart data replacement would increase the virtual storage size of nodes and that finding optimal solution is NP-hard. We suggested and evaluated five heuristic algorithms for this problem. The simulation results show that the MPVF algorithm increases the total virtual storage size most because it considers interests of all the nodes in the P2P system.

Further work includes expanding our model to adopt real world constraints. For example, various sizes of data can make the algorithm complex, and the limitation of network bandwidth can draw a distinction between data stored in its own disk and those stored in neighboring nodes. Real time requirements like VOD would make the model much more complex.

References

[1] A. Oram, “Peer-to-Peer: Harnessing the Power of Disruptive Technologies”, O’Reilly, 2001.

[2] A. J. Ganesh, A. M. Kermarrec, and L. Massoulie, “Peer-to-peer membership management for gossip-based protocols”, *IEEE Transactions on Computers*, vol. 52, no. 2, pp. 139-149, Feb. 2003.

[3] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications”, *IEEE/ACM Transactions on networking*, vol. 11, no. 1, pp. 17-32, Feb. 2003.

[4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A Scalable Content-Addressable Network”, *In Proceedings of ACM SIGCOMM 2001*, Aug. 2001.

[5] M.R. Garey and D.S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-completeness", Freeman, San Francisco, 1979.
[6] D. S. Johnson, "Approximation algorithms for combinatorial problems", *J. Comput. System Sci.* pp. 256-278, 1974.
[7] Gnutella, http://gnutella.wego.com/.
[8] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems", *In Meddleware*, pp. 329-350, 2001.
[9] M. J. Freedman and R. Morris, "Tarzan: a Peer-to-peer anonymizing network layer", *In ACM-CCS*, pp. 193-206, 2002.
[10] Napster, http://www.napster.com/, 2001.
[11] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system", *In Workshop on Design Issues in Anonymity and Unobservability*, pp. 311-320, Jul. 2000.
[12] KaZaA, http://www.kazaa.org/, 2002.
[13] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A Resilient Global-scale Overlay for Service Deployment", *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 44-53, Jan. 2004
[14] TiVo, http://www.tivo.com/.
[15] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," *In Proceedings of INFOCOM*, Apr. 1999

Seungtaek Oh was born in Seoul, Korea in 1974. He received his B.S. and M.S. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 1997 and 1999. He is working toward his Ph.D. degree at Korea Advanced Institute of Science and Technology. He is currently with Mobile Platform Lab. at Software Center, Samsung Electronics, Seoul.

Jin-Soo Kim is currently an assistant professor of the department of electrical engineering and computer science at Korea Advanced Institute of Science and Technology (KAIST). Before joining KAIST, he was a senior member of research staff at Electronics and Telecommunications Research Institute (ETRI) from 1999 to 2002. He was with the IBM T. J. Watson Research Center as an academic visitor from 1998 to 1999. He received his B.S., M.S., and Ph.D. degrees in Computer Engineering from Seoul National University in 1991, 1993, and 1999, respectively. His research interests include peer-to-peer and grid computing, embedded systems, and operating systems.

Ki-sok Kong is an associate professor in the Department of Computer Engineering at Korea Polytechnic University, Shihung-City, Korea. He received the B.E. degree in control and instrumentation engineering and the M.E. degree in computer engineering from Seoul National University, Seoul, Korea, in 1984 and 1986, respectively. He earned his Ph.D. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST). From 1986 to 1989, he was with the Computer Division of Samsung Electronics Co., Ltd., Korea, and from 1989 to 1992, he was with TriGem Computer, Inc., Korea, and from 1992 to 1994, he was with EOS Technologies, Inc., Korea. From 1999 to 2000, he was with Electronics and Telecommunication Research Institute (ETRI), Korea and from 2000 to 2001, he was with IMDB, Inc., Korea. His research interests include operating systems, embedded system software and ubiquitous computing.

Joonwon Lee is a Professor in the Division of Computer Science of the Department of Electrical Engineering & Computer Science at Korea Advanced Institute of Science and Technology (KAIST). He earned his B.S. degree in Statistics and Computer Science at Seoul National University and M.S. and Ph.D. degrees in Computer Science at the College of Computing, Georgia Tech. At IBM, He worked in a multiprocessor design team. His research interests are Operating Systems, Computer Architecture, and Parallel Processing.