PeerCluster: A Cluster-Based Peer-to-Peer System

Xin-Mao Huang, Cheng-Yue Chang, and Ming-Syan Chen, *Fellow, IEEE*

Abstract—This paper proposes a cluster-based peer-to-peer system, called *PeerCluster*, for sharing data over the Internet. In *PeerCluster*, all participant computers are grouped into various interest clusters, each of which contains computers that have the same interests. The intuition behind the system design is that by logically grouping users interested in similar topics together, we can improve query efficiency. To efficiently route and broadcast messages across/within interest clusters, a hypercube topology is employed. In addition, to ensure that the structure of the interest clusters is not altered by arbitrary node insertions/deletions, we have devised corresponding *JOIN* and *LEAVE* protocols. The complexities of these protocols are analyzed. Moreover, we augment *PeerCluster* with a system recovery mechanism to make it robust against unpredictable computer/network failures. Using an event-driven simulation, we evaluate the performance of our approach by varying several system parameters. The experimental results show that *PeerCluster* outperforms previous approaches in terms of *query efficiency*, while still providing the desired functionality of keyword-based search.

Index Terms—Data broadcasting, data sharing, hypercube, peer-to-peer.

✦

1 INTRODUCTION

PEER-TO-PEER computing is attracting increasing attention, and many peer-to-peer systems have emerged recently as platforms for users to search and share information over the Internet. In essence, a peer-to-peer system can be characterized as a distributed network system in which all participant computers have symmetric capabilities and responsibilities. Explicitly, all participant computers in a peer-to-peer system act as both clients and servers to one another, thereby surpassing the conventional client/server model and bringing all participant computers together to yield a large pool of information sources and computing power [11], [28].

As noted in [15], in terms of network structure, current peer-to-peer systems can be classified into three categories, namely, *centralized*, *decentralized/unstructured*, and *decentralized/structured* systems. A well-known example of a *centralized* peer-to-peer system is *Napster* [16], which maintains a constantly updated directory at central locations. To determine which computers in *Napster* hold the desired files, users' queries are sent directly to the centralized directory server for resolution. The advantage of such a centralized system is that the costs (e.g., response time and bandwidth consumption) of resolving a query are minimized, since only two messages (i.e., one query and one response message) and one round-trip delay are incurred. However, centralized systems are prone to a *single point of failure problem*. On the other hand, *decentralized/unstructured* systems (e.g., *Gnutella* [8]) do not have a centralized server or an explicit network topology. Although such systems are fault-tolerant and resilient to computers entering and leaving the system, the search mechanism used in *decentralized/unstructured* systems (i.e., flooding query messages to neighboring computers within a *TTL* (*time-to-live*) [8] framework) is not scalable [20]. In addition, there is a high likelihood that many required files are located outside the search scope of the *TTL* framework; thus, the quality of the search results might be unsatisfactory. This problem has been studied extensively in [9], [25], [15], [6], [29]. In [9], the authors exploited data-mining and text retrieval from previous queries to extract search rules. Based on these rules, queries are routed to peers that are more likely to have an answer. Meanwhile, for the *Gnutella* system, Sripanidkulchai et al. [25] proposed the concept of interest-based shortcuts to connect two peers that have similar interests, where by a query is only flooded to the entire system if none of the shortcuts have the requested content. This method reduces hop-by-hop delays in overlay networks. In [15], a search protocol, called random walker, is proposed. It can reduce the message overhead significantly. In [6], the authors propose the Gia system, which creates a *Gnutella-like* P2P system that can handle much higher aggregate query rates and functions efficiently as the system size increases. The work in [29] presents three techniques for efficient search, while considering such costs as aggregate bandwidth and processing overhead.

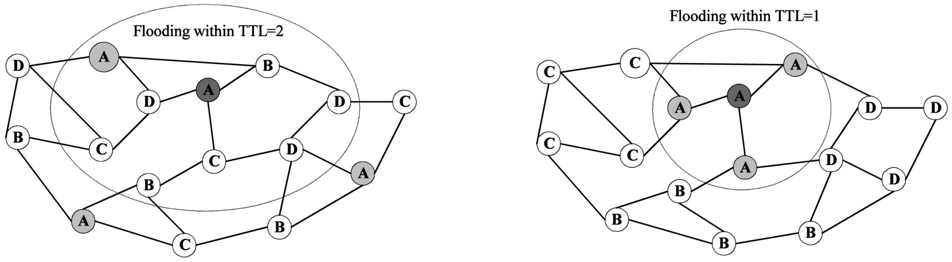
Furthermore, several *decentralized/structured* peer-to-peer systems are proposed in the literature [3], [18], [21], [26], [30]. These works focus on providing one fundamental lookup service, i.e., given a *key*, map the *key* onto a participant computer. Depending on the application using this service, a computer might be responsible for storing a *value* associated with the *key*. In such a system, either the network topology or the distribution of keys is devised in such a way that the specific key distribution strategy facilitates finding the corresponding computer efficiently. In addition, the explicit network topology can limit the

---

- *The authors are with the Department of Electrical Engineering, National Taiwan University, No. 1, Sec. 4, Roosevelt Road, Taipei, Taiwan, R.O.C. E-mail: {bigmoun, cychang}@arbor.ee.ntu.edu.tw, mschen@cc.ee.ntu.edu.tw.*

Manuscript received 14 Sept. 2003; revised 3 July 2004; accepted 10 May 2005; published online 24 Aug. 2006.
Recommended for acceptance by R. Schlichting.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0167-0903.

1045-9219/06/$20.00 © 2006 IEEE Published by the IEEE Computer Society



(a) (b)

Fig. 1. An example to show the benefit of grouping together users who are interested in identical topics.

logical distance between two arbitrary computers to an upper bound. As a result, the performance of these *decentralized/structured* peer-to-peer systems is guaranteed. However, not every application can benefit from this lookup service immediately. For instance, to support the functionality of keyword-based search provided by a Web search engine, one may utilize *Chord* [26] to implement a distributed indexing system, where by a *key* is transformed from the desired keywords, while *values* are lists of machines offering the documents with those keywords. Even so, such a *keywords-to-key* transformation is not straightforward, since a user query consists of various of keywords, and the transformation must guarantee that arbitrary permutation of these keywords can be transformed to the same *key*. In addition, depending on the indexing schemes used, the potential cost of maintaining the distributed indexes could vary dramatically. A new method that provides keyword search functionality for a DHT-based file system or archival storage system has been proposed in [19]. However, these systems require the close cooperation of every participant computer. In other words, one user's data or index of data is located on an other user's computer, depending on the key distribution strategy employed. A conservative user may feel insecure with this feature, because his data/index could be damaged if the other user's computer fails due to poor system design.

In this paper, we propose an innovative peer-to-peer system, called *PeerCluster*, that addresses the above issues from a new perspective. The principle design of *PeerCluster* is based on the phenomenon that one user, at any given time, is usually only interested in a few topics (typically one or two) and tends to issue queries and share collections about those topics he/she is interested in. (Similar phenomena were also explored in previous studies [23], [24].) Therefore, the intuition behind the design of *PeerCluster* is that by logically grouping the computers of users/computers interested in similar topics, we can increase the *query efficiency*. This idea can be best understood by the example in Fig. 1. In Fig. 1a, there are 16 computers in the system, each interested in and sharing its collection about one major topic (labeled A, B, C, or D in each node). Suppose that the dark gray computers would like to make a query about topic A. According to the network topology in Fig. 1a, only one computer sharing its collection of topic A will be reached within $TTL = 2$. However, if we logically group users interested in identical topics together, as shown in Fig. 1b, three computers sharing their collections of topic A will be reached within $TTL = 1$. Note that in the case of Fig. 1b, not only are the search costs reduced, but also the quality of query results is improved. Thus, *PeerCluster* is devised to logically group participant computers into various *interest clusters*. This concept is illustrated in Fig. 2, where three interest clusters are identified, i.e., *movie*, *computer*, and *sport*. To join this system, a computer should initially specify one of the three topics as its major interest. Then, following the *JOIN* protocol, which we describe in Section 3, this joining computer will be logically assigned to the corresponding *interest cluster*. To resolve a query, the computer that receives the query first checks whether the topic of the query agrees with the topic of the interest cluster where this recipient computer resides. If it does, the recipient computer will immediately broadcast the query to other computers in the same interest cluster for resolution (i.e., *intracluster broadcasting*). Otherwise, the query will be routed to a certain computer which resides in the corresponding interest cluster, and then, broadcast to other computers in that cluster for resolution (i.e., *intercluster routing + intracluster broadcasting*).

Intracluster broadcasting and *intercluster routing* are the two major operations used to resolve queries in *PeerCluster*. Considering the performance of these two operations on various distributed network topologies [12], we employ *hypercube* topology to realize the proposed system. As shown in Fig. 3, the system in Fig. 2 can be realized with a five-dimensional hypercube, where the three interest

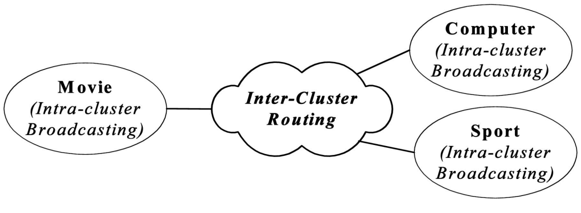

Computer
(Intra-cluster Broadcasting)
Movie
(Intra-cluster Broadcasting)
Inter-Cluster Routing
Sport
(Intra-cluster Broadcasting)

Fig. 2. The design concept of *PeerCluster*.

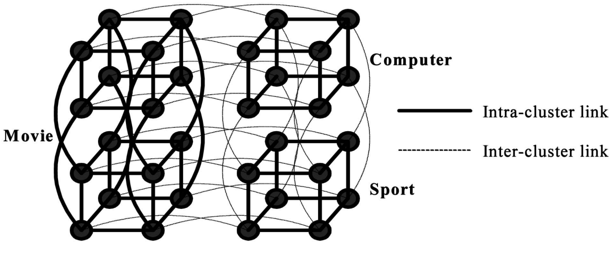


Fig. 3. One possible hypercube topology for the example in Fig. 2.

clusters, *movie*, *computer*, and *sport*, form one four-dimensional and two three-dimensional subhypercubes. Note that the nodes and edges in Fig. 3 are both virtual entities. That is, a node in a hypercube simply represents a relative address in the hypercube, and an edge between two nodes represents the neighborhood relationship between these two hypercube addresses only. One hypercube address in *PeerCluster* is always mapped to one participant computer. However, one participant computer in *PeerCluster* could be associated with multiple hypercube addresses, if the number of participant computers is smaller than the number of hypercube addresses. In addition, interest clusters in *PeerCluster* do not have to be the same size (e.g., in Fig. 3 the size of the movie cluster is larger than the sport cluster). Actually, the cluster size will be proportional to the popularity of each interest cluster.

The approach in [22] also exploits the hypercube structure to construct a P2P system, named HyperCup, where an ontology concept is used to connect different topic hypercubes. Explicitly, a hypercube is an item of ontology. Thus, in HyperCup, all hypercubes form a hierarchical structure in which a hypercube can only connect with other hypercubes that are one layer above or below it, or to its sibling hypercubes. On the other hand, the clustering technique in [17] integrates peers into locations already populated by peers with similar characteristics. In [17], a superpeer in a cluster is a peer that has better resources, such as adequate bandwidth, fast processing speed, or larger disk space. Other peers located in the same cluster provide their data to the superpeer. Note that *PeerCluster* is different from the systems in [22] and [17] in that it has two kinds of link: interlink and intralink. Explicitly, each node in *PeerCluster* holds an interlink and an intralink, and each cluster holds the hypercube structure itself. Conceptually, the *PeerCluster* system can be viewed as one entire hypercube, whereas the HyperCup system is comprised of many smaller hypercubes, and exploits the ontology hierarchy concept to connect them. Also, in [22] and [17], only some of the peers hold interlinks.

Hypercube topology has been extensively studied in the context of computer architecture and parallel computing. Due to its structural regularity and good potential for parallel execution of various algorithms, the topology has been applied to many innovative designs, such as multiprocessor computers [5] and distributed communication networks [14], [26], [30]. *PeerCluster* distinguishes itself from prior works in the way it utilizes the hypercube topology, and is explicitly unique in two respects. First, *PeerCluster* partitions the hypercube structure into multiple parts to form several interest clusters. Second, as we show later, the hypercube topology provides a dynamically configurable address scheme for *PeerCluster*. Specifically, to guarantee that the structures of the interest clusters in *PeerCluster* will not be altered by arbitrary node insertions/deletions, we have devised *JOIN* and *LEAVE* protocols. This hypercube structure enables us to employ the *Huffman coding technique* [10] to determine the address prefix for each interest cluster, which greatly facilitates our design on the corresponding *JOIN*, *LEAVE*, and *SEARCH* operations. The complexities of these protocols are analyzed in Section 3. In addition, we augment *PeerCluster* with a system recovery mechanism to make it robust against unpredictable computer/network failures. Note that the concept of *PeerCluster* can be applied to improve the operation of some discussion boards by establishing proper clusters [2]. In existing P2P systems, users have no knowledge about new files that have been published recently. In some discussion boards, the purpose of creating groups is to publish files that have the same characteristics. By treating such groups as predefined clusters in *PeerCluster* and the number of articles as the popularity, we can analyze these discussion boards to establish the clusters and facilitate the corresponding query and file search. Note that the structure of *PeerCluster* allows peers the flexibility of using all-to-all broadcast schemes to exchange file indexes [7]. Using an event-driven simulation, we evaluate the performance of *PeerCluster* by varying several system parameters. The experimental results show that *PeerCluster* performs well in *query efficiency*, while providing the desired functionality for keyword-based search.

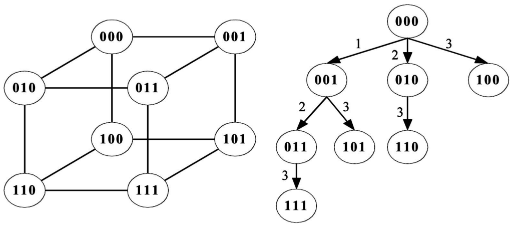


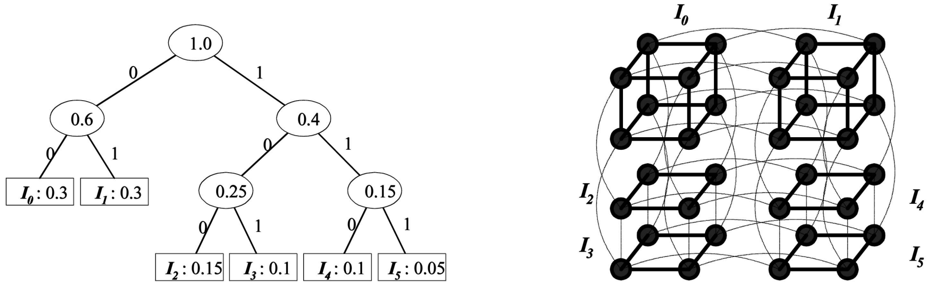
Fig. 4. An example of broadcasting a message from node 000 to all other nodes in Q_3.

The remainder of this paper is organized as follows: Preliminaries are given in Section 2. The design of *PeerCluster* is presented in Section 3. In Section 4, the performance of *PeerCluster* is empirically evaluated. Finally, in Section 5, we present our conclusions.

2 PRELIMINARIES

Based on the properties of hypercubes [13], we describe the procedures for broadcasting and routing a message in the hypercube Q_n. Let Q_{subq} be the subhypercube of Q_n, where all hypercube addresses in Q_{subq} differ from each other in the least significant $subq$ bits only. To broadcast a message from one node to all other nodes in Q_{subq}, each node in Q_{subq} has to follow the broadcasting procedure, *Proc_Broadcast*, in a recursively doubling manner. *Proc_Broadcast* takes four arguments, $subq$, msg, $node\_addr$, and $step$, as the inputs. $subq$ specifies the subhypercube Q_{subq}, where the message msg is broadcast; and $node\_addr$ is the hypercube address of the node executing the procedure. *Proc_Broadcast* is executed immediately after a node receives the message sent by another node in line 3 of *Proc_Broadcast*. An example of broadcasting a message from node 000 to all other nodes in Q_3 is shown in Fig. 4. The values on the directed edges on the right of Fig. 4 represent the values of the argument $step$ passed to the other nodes. For example, after node 001 receives the message sent by node 000 with the parameters $subq = 3$ and $step = 1$, *Proc_Broadcast*(3, *msg*, 001, 1) is performed immediately.

```
$Proc\_Broadcast(subq, msg, node\_addr, step)$
01  for ($i = step$ to $subq - 1$){
02      $dest\_addr = node\_addr \oplus 2^i$;
```



(a) (b)

Fig. 5. Determining the address prefix of each interest cluster with the *Huffman coding technique*. (a) Huffman tree. (b) The corresponding hypercube.

```
    send(subq, msg, dest_addr, $i + 1$);
}
```

```
Proc_Route(msg, dest_addr, node_addr)
if (dest_addr != node_addr) {
    $i =$ Compare(dest_addr, node_addr);
    send(msg, dest_addr, node_addr $\oplus 2^i$);
}
```

To route a message from one node to another in Q_n, we use the routing procedure called *Proc_Route*, which takes three arguments as inputs: msg, $dest\_addr$, and $node\_addr$. Specifically, msg is the message to be routed to the destination node, $dest\_addr$ is the hypercube address of the destination node, and $node\_addr$ is the hypercube address of the current node that is executing the procedure. The function **Compare(**$dest\_addr$, $node\_addr$**)** in the *Proc_Route* returns the position of the first different bit between $dest\_addr$ and $node\_addr$ in ascending order from the least-significant bit (i.e., position 0) to the most-significant bit, e.g., **Compare**$(1010, 0100) = 1$. According to *Proc_Route*, the routing path from node 001 to 110 is thus $001 \longrightarrow 000 \longrightarrow 010 \longrightarrow 110$.

Note that the notion of *PeerCluster* can be easily generalized in terms of generalized cubes [4] when the communication capability of a node is increased.

3 CLUSTER-BASED PEER-TO-PEER SYSTEM

We now present the system design of *PeerCluster*. The system design is given in Section 3.1. The protocols for computers joining/leaving and searching the system are described in Section 3.2. In Section 3.3, we address the scalability issue of PeerCluster. Section 3.4 extends *PeerCluster* to allow a participant computer to join multiple interest clusters. We present the system recovery mechanism of *PeerCluster* in Appendix B.

3.1 Design of PeerCluster

As noted in Section 2, a system realized with an n-dimensional hypercube is able to admit 2^n computers at most. To divide the system capacity (i.e., 2^n hypercube addresses) among various interest clusters, we employ the *Huffman coding technique* to determine the address prefix of each interest cluster in proportion to its popularity.

Assume that the entire system is an n-dimensional hypercube, and there are k different interest topics in the system. Let I_j denote the jth interest topic, where $0 \leq j \leq k-1$. The popularity of I_j is denoted by $pop[I_j]$, where $0 < pop[I_j] < 1$ and $\sum_{j=0}^{k-1} pop[I_j] = 1$. Using the *Huffman coding technique*, we first construct a *Huffman tree* from these $pop[I_j]$. Then, based on the constructed *Huffman tree*, the address prefix (denoted by $prefix[I_j]$) for each I_j is determined by the path from the root to the leaf accordingly. For example, suppose $n = 5$, $k = 6$, and $pop[I_0], \ldots, pop[I_5]$ are equal to 0.3, 0.3, 0.15, 0.1, 0.1, and 0.05, respectively. The *Huffman tree* thus constructed is shown in Fig. 5a. Based on this *Huffman tree*, the address prefixes for the interest clusters $I_0, \ldots, I_5$ can be determined as 00, 01, 100, 101, 110, and 111, respectively. Once the address prefix for each interest cluster is determined, the size of the corresponding interest cluster can be computed as $2^{n-length(prefix[I_j])}$. As shown in Fig. 5b, the sizes of the interest clusters $I_0, \ldots, I_5$ are thus 2^3, 2^3, 2^2, 2^2, 2^2, and 2^2, respectively.

After determining the address prefix for each interest cluster, we design the routing table for each participant computer, as shown in Fig. 6. The routing table consists of two kinds of information: the hypercube address(es) owned by the participant computer, and the mapping from its neighboring hypercube addresses to its neighboring computers' IP addresses. First, as explained in Section 1, one participant computer can be associated with multiple hypercube addresses in *PeerCluster*. In this case, the participant computer is assigned one hypercube address as its *primary* address; and the other addresses are its *alias*

***ID* : 00000 (blackcircle.net)**	
Alias addresses	
00001, 00010, 00011, 00100, 00101, 00110, 00111	
Neighbors	
01000: blackstar.net	10000: blacksquare.net
01001: blackstar.net	10001: blacksquare.net
01010: blackstar.net	10010: blacksquare.net
01011: blackstar.net	10011: blacksquare.net
01100: blackstar.net	10100: whitecircle.net
01101: blackstar.net	10101: whitecircle.net
01110: blackstar.net	10110: whitecircle.net
01111: blackstar.net	10111: whitecircle.net

Fig. 6. The routing table of a participant computer whose primary hypercube address is 00000.

addresses. In Fig. 6, the hypercube address 00000 is chosen as computer blackcircle.net's primary address and 00001, 00010, 00011, 00100, 00101, 00110, and 00111 as its alias addresses. Second, to broadcast or route messages in *PeerCluster*, a participant computer must keep track of the mapping from its neighboring hypercube addresses to its neighboring computers' IP addresses. Let $addr(A)$ denote the set of hypercube addresses (including both primary and alias addresses) owned by participant computer A. The set of A's neighboring hypercube addresses, denoted by $NH(A)$, is defined as $\bigcup_{a_i \in addr(A)} Ne(a_i) - addr(A)$, where $Ne(a_i)$ is the set of the hypercube addresses adjacent to the address a_i. In Fig. 6,

$$\begin{aligned} A &= \{00000, 00001, 00010, 00011, 00100, 00101, 00110, 00111\}, \\ Ne(00000) &= \{00001, 00010, 00100, 01000, 10000\}, \text{ and} \\ NH(A) &= \{01000, 01001, 01010, 01011, 01100, 01101, \\ &\qquad 01110, 01111, 10000, 10001, 10010, 10011, \\ &\qquad 10100, 10101, 10110, 10111\}. \end{aligned}$$

Note that, initially, there exist k computers in *PeerCluster*, and they are deployed to maintain the system's operation. Computer j ($0 \leq j \leq k-1$) owns all hypercube addresses of the $j\text{th}$ interest cluster. The primary address of computer j is address 0 of its interest cluster. In the example in Fig. 5b, initially there are six computers in the system, and their primary addresses are 00000, 01000, 10000, 10100, 11000, and 11100. In *PeerCluster*, these k existing computers are not necessarily physical units. To reduce the deployment cost, we can use fewer physical computers ($< k$) to deploy the *PeerCluster* system. However, the trade-off will be system reliability.

For address assignment, we utilize the assigned tree to record the number of free addresses in every cluster. The tree has the following properties:

1. The root address is the lowest address. In other words, with the exception of prefix bits, all bits are 0.
2. The parent address and the child address differ from each other by one bit only.
3. The child address is larger than the parent address.
4. The present address manages the assignment of the child address.
5. Every address records the number of free addresses of all its children. The initial number of free addresses of children is the total number of the subtrees.
6. When a parent address wants to assign a free address to a joining request, it checks the number of free addresses of its children starting from the lowest low address and moving to higher addresses.

For example, Fig. 7 is an assigned tree of I_1 in Fig. 5. The prefix bit is 01. Thus, the root address is 01000. It records the number of free addresses of its children, 01001, 01010, and 01100. The numbers of the free addresses are 1, 2, and 4. When address 01000 receives a message to assign an address, it first checks the number of free addresses of 01001. If the value is not equal to 0, the root address assigns the address of 01001 and subtracts 1 from the total number of free addresses of 01001. If the number of free addresses of 01001 is equal to 0 and the number of 01010 is not 0, the root address sends a message to 01010 to assign the address and subtracts 1 from the number of free address of 01010.

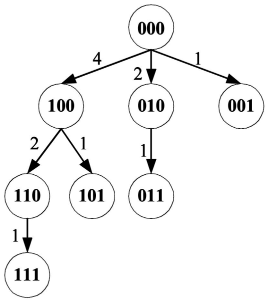


Fig. 7. The assigned tree of I_1 in Fig. 5.

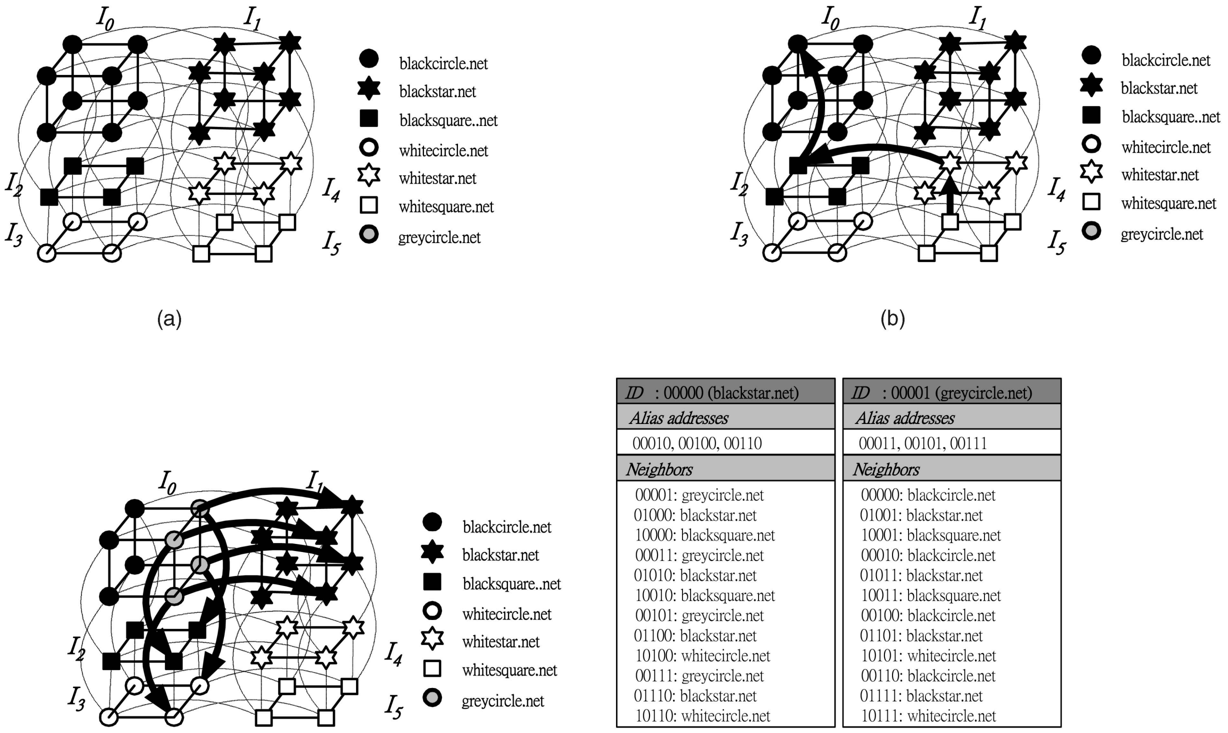
3.2 Description of Protocols

In a peer-to-peer system, computers are expected to join or leave at any time. To guarantee that the structures of the interest clusters will not be altered by arbitrary node insertions/deletions, we have devised *JOIN* and *LEAVE* protocols for *PeerCluster*. In addition, we have designed a *SEARCH* protocol to resolve queries efficiently.

3.2.1 *The* JOIN *Protocol*

Let A denote the joining computer and I_A denote the interest cluster that A would like to join. The *JOIN* protocol is comprised of four phases:

1. **Find an arbitrary computer in the system**. To join *PeerCluster*, A must know at least one IP address of an online computer to initially communicate within the system. We assume that A is able to learn the IP address of some online computer from an external mechanism similar to *Gnutella* [27].
2. **Using the computer found in Phase 1, find a participant computer in the interest cluster with the same major interest**. Let B denote the computer found in Phase 1 and I_B denote the interest cluster where B resides. A initially sends a *Join Request* (*JRQ*) message containing the information of I_A to B. Upon receipt of the *JRQ* message, B will first check whether $I_B = I_A$. If $I_B = I_A$, then B is a suitable computer needed for Phase 2. However, if $I_B \neq I_A$, B routes the *JRQ* message to some computer in I_A that owns the hypercube address with $prefix[I_A]$ and the postfix of B's primary address. For example, suppose $prefix[I_A] = 00$, $prefix[I_B] = 100$ and B's primary hypercube address is 10000. Since $I_B \neq I_A$, B will route the *JRQ* message to the computer that owns the hypercube address 00000, i.e., computer we find in Phase 2.
3. **Using the computer found in Phase 2, find a participant computer that owns an alias address in its interest cluster**. Let C denote the computer found in Phase 2 and I_C denote the interest cluster where C resides. If C owns an alias address, then C is the computer we want in Phase 3. However, if C does not own an alias address, C conducts a *Depth First Search* (*DFS*) in the broadcasting tree rooted from its primary address to all other hypercube addresses in



(a)

(b)

ID : 00000 (blackstar.net)
Alias addresses
00010, 00100, 00110
Neighbors
00001: greycircle.net
01000: blackstar.net
10000: blacksquare.net
00011: greycircle.net
01010: blackstar.net
10010: blacksquare.net
00101: greycircle.net
01100: blackstar.net
10100: whitecircle.net
00111: greycircle.net
01110: blackstar.net
10110: whitecircle.net

ID : 00001 (greycircle.net)
Alias addresses
00011, 00101, 00111
Neighbors
00000: blackcircle.net
01001: blackstar.net
10001: blacksquare.net
00010: blackcircle.net
01011: blackstar.net
10011: blacksquare.net
00100: blackcircle.net
01101: blackstar.net
10101: whitecircle.net
00110: blackcircle.net
01111: blackstar.net
10111: whitecircle.net

(c)

Fig. 8. An example of the *JOIN* protocol.

I_C until it finds a suitable computer. If such a computer exists, C will route the *JRQ* message to that computer, and that computer will be the one we find in Phase 3. Otherwise, it means that I_C is full. Then, C will execute the cluster extension functionality, which we describe in Section 3.3.

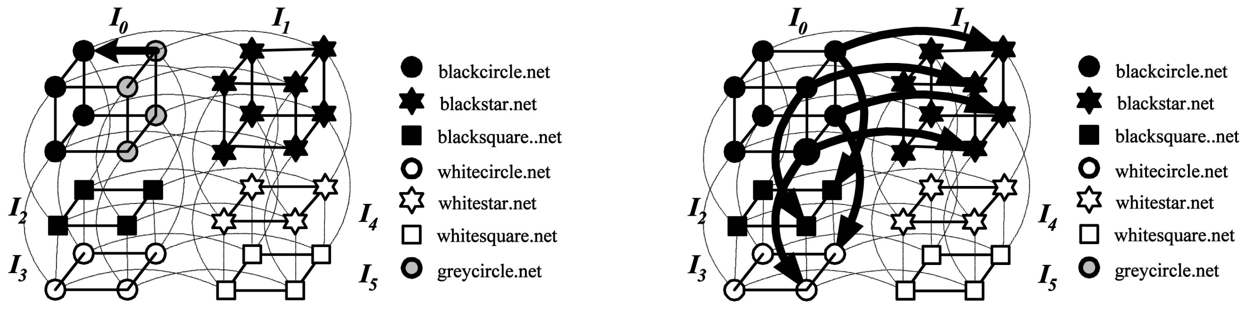
4. **Acquire the hypercube address(es) from the computer found in Phase 3 and update the related routing tables**. Let D denote the computer found in Phase 3. Assume that $addr(D) = \{d_0, \ldots, d_m\}$, where d_0 is D's primary address and $d_1, \ldots, d_m$ are D's alias addresses ($d_1 < \ldots < d_m$). Let T_D denote D's broadcasting tree rooted from d_0 to all other addresses in its interest cluster. In the *Join Confirm* (*JCF*) message returned to A, D will assign d_1 as A's primary address and the addresses that are the descendants of d_1 in T_D as A's alias addresses. In addition to the address, the *JCF* message includes the routing information for A from which A can construct the routing table for itself. Finally, A will notify its neighboring computers of the mapping from its hypercube addresses to its IP address. In addition, D will also send the Decrease Free Address (*DFA*) message to its ascendant addresses defined in the assigned tree until the root address. When an address receives the *DFA* message from its child address, it subtracts 1 from the remaining free address value of the subtree that contains the child address.

Example 3.1. As shown in Fig. 8a, there are initially six participant computers in the system. The routing table of the computer blackcircle.net is shown in Fig. 6. Suppose that a new computer, greycircle.net, wants to join the interest cluster I_0. The computer found in Phase 1 is whitesquare.net. The joining computer, greycircle.net, will then send a *JRQ* message to whitesquare.net. However, since whitesquare.net (whose primary hypercube address is 11100) is not in I_0, it will route, in Phase 2, the *JRQ* message to 00000 via the path $11100 \longrightarrow 11000 \longrightarrow 10000 \longrightarrow 00000$, as shown in Fig. 8b. Note that blackcircle.net, which owns the hypercube address 00000, has an extra alias address in Phase 3. It then sends a *JCF* message back to greycircle.net in Phase 4, assigning the hypercube address 00001 as greycircle.net's primary address and the addresses 00011, 00101, and 00111 as its alias addresses. Also, based on the routing information included in the *JCF* message, greycircle.net can construct a routing table for itself, as shown in the Fig. 8c. Finally, greycircle.net notifies its neighboring computers of its IP address. The routing table of blackcircle.net is thus revised, as shown in Fig. 8c.

3.2.2 *The* LEAVE *Protocol*

The *LEAVE* protocol is comprised of two phases:

1. **Find the computer that owns the smallest hypercube address among the leaving computer's neighboring addresses**. Let A denote the leaving computer and $a_{\min}$ denote the smallest hypercube address in $Ne(A)$. The IP address that $a_{\min}$ is mapped to in the routing table is the computer with the smallest hypercube address.



(a) (b)

Fig. 9. An example of the *LEAVE* protocol.

2. **Give the computer found in Phase 1 the leaving computer's addresses and update the related routing tables**. Let B denote the computer found in Phase 1. Then, A will send a *Leave Request* (*LRQ*) message to B, giving all A's hypercube addresses and routing information. Based on the *LRQ* message, B will update its routing table accordingly and notify its neighboring computers of the mapping from its newly incorporated hypercube addresses to its IP address. In addition, B will add 1 to the remaining free address value of the subtree containing A. B will also send the Increase Free Address (*IFA*) message to its ascendant addresses defined in the assigned tree. Finally, computer B returns a *Leave Confirm* (*LCF*) message to A and completes the *LEAVE* process.

Example 3.2. Given the routing tables of blackcircle.net and greycircle.net in Fig. 8c, suppose that greycircle.net in Fig. 9a is going to leave the system. The *LEAVE* protocol first searches the routing table to find the smallest neighboring hypercube address, i.e., 00000. Then, according to the IP address that the hypercube address 00000 is mapped to, greycircle.net sends an *LRQ* message to computer 140.112.17.53, as shown in Fig. 9a. Based on the information included in the *LRQ* message, blackcircle.net takes over the hypercube addresses 00001, 00011, 00101, and 00111 as its alias addresses, updates its routing table, as shown in Fig. 6. It also notifies its neighboring computers of the mapping from its newly incorporated hypercube addresses to its IP address, as shown in Fig. 9b. Finally, blackcircle.net returns an *LCF* message to greycircle.net and terminates the *LEAVE* process.

3.2.3 *The* SEARCH *Protocol*

To resolve a query efficiently in *PeerCluster*, we have devised a *SEARCH* protocol, which consists of three phases:

1. **Route the query message to a computer in the corresponding interest cluster**. Let A denote the computer that starts the query search and I_A denote the interest cluster of A. Let I_Q denote the interest topic that the query relates to. If $I_A = I_Q$, then A is the computer needed in Phase 1. However, if $I_A \neq I_Q$, A will route the query message to some computer in I_Q that owns the hypercube address with $prefix[I_Q]$ and has the postfix of A's primary address.
2. **Using the computer found in Phase 1, broadcast the query message to other computers in its interest cluster**. Let B denote the computer found in Phase 1 and I_B denote the interest cluster of B. Upon receipt of the query message routed from A, B will broadcast the query message to other computers in I_B by the *Proc_Broadcast* procedure with the parameter $subq = SL$ (the search limit specified by the user).
3. **Resolve the query and return the query results to the computer that issued the query search**. In this phase, each computer that receives the query message will look up its local database to resolve the query and return the results to A.

Example 3.3. Suppose blackcircle.net in Fig. 10 wants to issue a query for topic I_5. Since blackcircle.net does not reside in the interest cluster I_5, it will route the query message to the hypercube address 11100 via the path 00000⟶00100⟶01100⟶11100 according to the *Proc_Route* procedure described in Section 2. Upon receiving the query message sent from blackcircle.net, whitesquare.net will broadcast the query message to other computers in I_5. However, being the only computer in I_5, whitesquare.net will look up its local database immediately to resolve the query and return the query results to blackcircle.net.

3.2.4 *Discussion*

In our scenario, a user picks a cluster to join. If the cluster selection is not ideal, the system will still work, but at a low

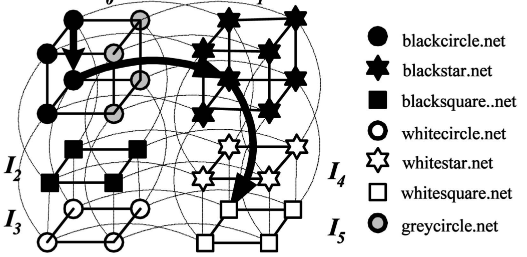


Fig. 10. An example of the *SEARCH* protocol.

upload rate. When there are many misclassified peers in a system, its performance will be degraded. We have therefore designed a penalty mechanism that will restrict a peer's download rate if it upload rate is too low. In addition, the system will recommend that the user should join another, more suitable, cluster.

As mentioned earlier, the structure of *PeerCluster* allows peers the flexibility of using all-to-all broadcast schemes to exchange file indexes [7]. In this case, the peer delivers the newly added or removed file indexes to other peers periodically via an all-to-all broadcast mechanism. Consequently, a peer knows what new files have been published in the system recently. In contrast, a peer in a traditional P2P system is usually unaware of recently published files, and a keyword-based search mechanism is used to query corresponding files.

3.3 Scalability

In this section, we address the scalability issue of *PeerCluster*. In Section 3.3.1, we describe a protocol, referred to as *Cluster Extension*, that intelligently extends the capacity of an interest cluster according to the address utilization rate of its neighboring clusters. In Section 3.3.2, we discuss how to enlarge the system when there is no additional space to extend the capacity of the interest cluster using the *Cluster Extension* protocol.

3.3.1 Cluster Capacity Extension

To extend the capacity of an interest cluster, we devised the following *Cluster Extension* protocol. Assume that a computer, C_x, wants to join interest cluster I_A, which is already full. Let computer C_A be a member of I_A and receive the JRQ message from C_x. To accommodate the incoming computer, the interest cluster I_A extends its capacity in three phases:

1. *Query the address utilization rates of neighboring clusters.* Let I_A have r neighboring clusters, $I_1 \dots I_r$ and computers $C_1 \dots C_r$ maintain the lowest internal addresses in clusters $I_1 \dots I_r$, respectively. To query the address utilization rates of neighboring clusters, C_A sends a *Space Request Query* (*SRQ*) message to all its neighboring clusters. Let I_B be one of I_A's neighbors and C_B maintain the lowest internal address in I_B. When C_B receives an *SRQ* message from C_A, it calculates its address utilization rate as

$$Rate_{utilization}(I_B) = \frac{Num_{assigned}(I_B)}{\frac{1}{2}Num(I_B)},$$

   where $Num_{assigned}(I_B)$ is the number of assigned addresses in I_B and $Num(I_B)$ is the size of I_B. $Num_{assigned}(I_B) = Num(I_B) - Num_{free}(I_B)$, where $Num_{free}(I_B)$ is the remaining free address value of I_B. The result is then sent to I_A with a *Space Confirm* (*SC*) message.
2. *Choose a neighboring cluster for borrowed space.* From the neighboring clusters, C_A chooses the one with the smallest address utilization rate, but that rate must be lower than a predefined threshold, γ. If one of I_A's neighbors satisfies these conditions, the system goes to the next phase. However, if none of I_A's neighbors satisfy these conditions, it is necessary to extend the system's capacity.
3. *Split-combine the borrowed space.* Let C_A choose I_B from all its neighboring clusters for the borrowed space. C_A then sends a Split Space Request Query (*SSRQ*) message to C_B. Upon receipt of the message, C_B separates the space owned by I_B into two subspaces according the highest bit of internal address. The subspace whose highest bit of internal address is one, is called the high space; the other is called low space. The high space is the borrowed space. Computers already assigned to the high space will be moved to the low space. They will send a *JRQ* message via their neighbors to rejoin to the system. The computer assigned to the borrowed space does not execute the move function immediately. This is executed when the address that is still being used by computers of other clusters is assigned by the new owner of the borrowed space. Finally, C_B sends a Split Space Confirm (*SSC*) message back to C_A, which then chooses an address from the borrowed space and sends it to C_X.

For example, let a computer, A, belong to I_A. Suppose A receives a *JRQ* message and knows that I_A is full. We assume that I_B and I_C are the neighboring clusters of I_A, and that computers B and C are the lowest internal addresses in I_B and I_C, respectively. Computer A will send an *SRQ* message to computers B and C. In addition, we assume that the sizes of clusters I_B and I_C are both 32, and that the number of peers in them is 13 and 10, respectively. The utilization rate of I_B, $Rate_{utilization}(I_B)$, is calculated by B and the value is 0.8125. $Rate_{utilization}(I_C)$ is calculated by C and the value is 0.625. Computers B and C return the utilization rates to A, and I_A chooses I_C to extend its cluster address. Then, computer A sends the *SSRQ* message to computer C. Thus, computer C will empty the high space and loan it I_A.

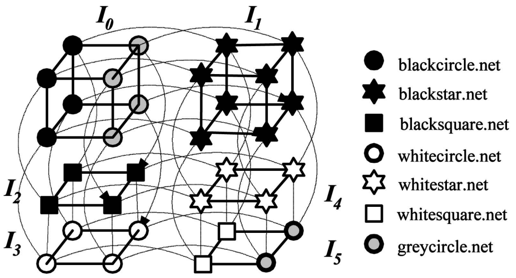
In Phase 2, the threshold γ indicates the degree of popularity of the cluster. In our experiments, γ is 0.95. When the utilization rate is larger than γ, the cluster is probably growing and will soon need a larger space. Note that the value of the threshold depends on the applications and is determined empirically. When a cluster grows quickly, the value will be lowered to leave more room for growth.

Though the address space released from a loose cluster can be reassigned to another cluster, the cluster address space of the latter will be partitioned. Therefore, when the system restores, the cluster address space can be compacted for the system to regenerate a new cluster address space region.

By exploiting the utilization rate and the popularity threshold, the cluster can gather the appropriate free space to extend itself. However, if the cluster cannot find the appropriate space from neighboring clusters, system extension will be performed.

3.3.2 System Extension

When a cluster cannot find a appropriate space to extend itself, or the system is full, the system's size will be scaled



***ID*: 00001 (greycircle.net)**
Alias addresses
00011, 00101, 00111
Neighbors
00000: blackcircle.net 01001: blackstar.net 10001: blacksquare.net 00010: blackcircle.net 01011: blackstar.net 10011: blacksquare.net 00100: blackcircle.net 01101: blackstar.net 10101: whitecircle.net 00110: blackcircle.net 01111: blackstar.net 10111: whitecircle.net

***ID*: 11101 (greycircle.net)**
Alias addresses
11111
Neighbors
11100: whitesquare.net 11001: whitestar.net 10101: whitecircle.net 01101: blackstar.net 11110: whitesquare.net 11011: whitestar.net 10111: whitecircle.net 01111: blackstar.net

Fig. 11. Support of multiple interests.

up by adding bits to its address. Consequently, all clusters double in size.

Note that *PeerCluster* is a structural system that employs extra effort to maintain its architecture, such as an alias list and a neighbor list. Therefore, when the address bits are extended by one bit, the entries in the neighbor list increase by one bit in every address and the capacity of the assigned tree doubles. To reduce the influence of the system extension, we use the following method to notify all participant computers that the system capacity has been extended. Every participant computer maintains a timestamp, called the extension time, which records the time of the last address extension and is added to the message. When a receiver receives the message, it will compare its own extension time to that of the sender. If they are not the same and the sender's extension time is more recent than the receiver's, the receiver extends the address bits. This method avoids broadcasting and reduces the influence of the system extension.

3.4 Support of Multiple Interests

Next, we consider the case when a participant computer chooses multiple topics as its interests. Recall that the interest cluster in which each computer participates is distinguished by its hypercube address(es). Therefore, to allow one participant computer to join multiple interest clusters, we extend the participant computer of *PeerCluster* to hold multiple sets of hypercube addresses in accordance with the interest clusters. For example, to allow the greycircle.net in Fig. 11 to join interest clusters I_0 and I_5, we maintain two sets of hypercube addresses (i.e., $\{00001, 00011, 00101, 00111\}$ and $\{11101, 11111\}$) and the corresponding routing tables in greycircle.net, as shown in Fig. 11.

The *JOIN*, *LEAVE*, and *SEARCH* protocols remain unchanged. To join the interest clusters $I_1, \dots I_j$, a participant computer, A, just sends multiple *JRQ* messages with the corresponding information of $I_1, \dots I_j$ to the system. Following the protocol in Section 3.2, A will be assigned multiple sets of hypercube addresses and then allowed to join the interest clusters $I_1, \dots I_j$. The conditions for a participant computer to search or leave the system are analogous.

4 PERFORMANCE ANALYSIS

To assess the performance of *PeerCluster*, we derive the theoretical bounds of the number of messages sent in the *JOIN*, *LEAVE*, and *SEARCH* protocols in Appendix A. A complexity comparison between *PeerCluster* and existing *decentralized/structured* systems is provided. In Section 4.1, we describe a simulation model to compare the *query efficiency* of *Gnutella* and *PeerCluster*, using the random walker method of *Gnutella* [15] and the random walker method of *PeerCluster*. The experimental results are given in Section 4.2.

4.1 Simulation Model

To further explore the superiority of *PeerCluster* in terms of *query efficiency*, we compare it with *Gnutella* by simulation. We present the simulation model in this section.

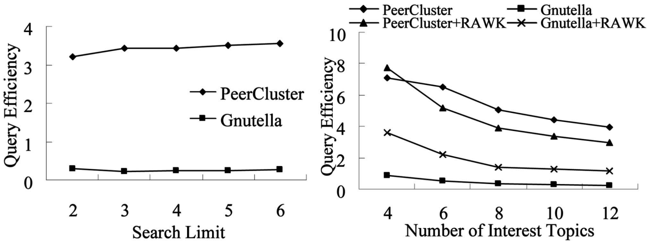
As described in Section 1, *PeerCluster* is designed to exploit the scenario where, at any given time, a user is usually only interested in a few topics and tends to issue queries and share collections about those topics. To investigate this phenomenon, we exploited the data of a Web directory search engine to form the interest groups and their keyword set. This data set was collected from the directory names and their subdirectory names in the Open Directory Project (ODP) [1]. In addition, we also gathered the number of items contained in the directory and the subdirectory. In our experiment, a directory name is an interest group, and its subdirectory names are the keywords of the interest group. Moreover, the popularity of the interest groups and the keywords depends on the number of items in the directory. Consequently, the replication of a keyword is relative to its popularity. The more popular the keyword is, the more replicas there are in the system.

The simulation model consists of N participant computers, each of which is interested in one major topic selected from the $|I|$ interest topics according to its popularity. The number of files kept by one participant computer is given by a normal distribution with a mean file number of 340 and a variance of 50. Of the files kept by a participant computer, 80 percent relate to the topic that is the user's major interest, whereas 20 percent are distributed over other interest topics according to their popularity. The number of queries issued by a computer follows a Poisson distribution with a mean query rate of 9.86×10^{-3} (query/second) [29]. Similarly, the query topic follows the 80/20 distribution, i.e., 80 percent of the queries are about the computer's major interest, and the other 20 percent are about other interest topics.

In the model that simulates the *Gnutella* network topology, the number of links per node is set to 4, which is the default setting of the *Gnutella* system [20]. For both *Gnutella* and *PeerCluster*, a search limit, SL, is used to limit the search range of a query. In *Gnutella*, SL plays the same role as TTL used in flooding query messages. In *PeerCluster*,

TABLE 1
Parameters Used in the Simulation Model

Parameter	*Dist./Default value*
# of interests	$\vert I\vert = 8$
Interest popularity	ODP directory popularity
# of participant computers	$N = 20000$
# of shared files per	Normal dist. ($\mu = 340$)
Query rate	9.86×10^{-3}
Query topic	80/20
Search limit	$SL = 4$
# of walkers for random walker method	32
Search limit for random walker method	20



(a) (b)

Fig. 12. (a) Query efficiency under various Search Limits. (b) Query efficiency under various skews of interest popularity.

SL is used as described in Section 3.2. Table 1 provides a summary of the parameters used in the simulation model.

4.2 Experimental Results

The performance metric used in our experiments is *query efficiency*, which is defined as the ratio of the number of files contained in the query results to the number of query messages sent to resolve a query. We compare the *query efficiency* of *Gnutella* and *PeerCluster*, using the random walker method of *Gnutella* and the random walker method of *PeerCluster* by varying the system parameters SL, $|I|$. The experimental results are given in Sections 4.2.1, 4.2.2, and 4.2.3. In Section 4.2.4, we empirically measure the numbers of messages sent in the *JOIN* and *LEAVE* protocols under various levels of address utilization. In Section 4.2.5, we empirically measure the address utilization in the JOIN and LEAVE protocols under various system sizes.

4.2.1 Search Limit

In the first experiment, we investigate the performance of *PeerCluster* by varying the SL parameter. The simulation results in Fig. 12a show that *PeerCluster* significantly outperforms *Gnutella* in terms of *query efficiency*. Specifically, the improvement of *query efficiency* over *Gnutella* increases from 10.76 times to 16.17 times as the value of SL increases from 2 to 6. This can be explained as follows: As the value of SL increases, the number of messages sent to resolve a query in *Gnutella* grows exponentially on the base 4 (i.e., the number of links per node), whereas in *PeerCluster* the base is 2. Although *Gnutella* incurs a higher cost than *PeerCluster* to resolve a query, it does not obtain an equivalent return of query results. The reason is that, in *Gnutella*, not all computers reached by the query messages share the same interest topic. In contrast, by grouping participant computers into various interest clusters in *PeerCluster*, we can obtain an equivalent, or even better, return of query results. It can be seen from Fig. 12a that the *query efficiency* of *PeerCluster* increases as the value of SL increases, whereas the *query efficiency* of *Gnutella* decreases.

4.2.2 Number of Interest Topics

The second experiment examines the performance of *PeerCluster* by varying the number of interest topics. The simulation results are shown in Fig. 12b, where, as the number of interest topics increases, the *query efficiency* of both *Gnutella* and *PeerCluster* drops. Note that when the number of interest topics increases, the computers that can be reached by *Gnutella* within the same search limit share an increasing number of different interest topics. Therefore, within the same search limit in *Gnutella*, there are fewer files that match the query. On the other hand, when interest topics increase in *PeerCluster*, the number of the intercluster connections also increases. Thus, within the same search limit in *PeerCluster*, there are more query messages sent. The *query efficiency* of *PeerCluster* outperforms that of *Gnutella* by margins from 8.08 times to 15.65 times as the value of an interest topic varies from 4 to 12.

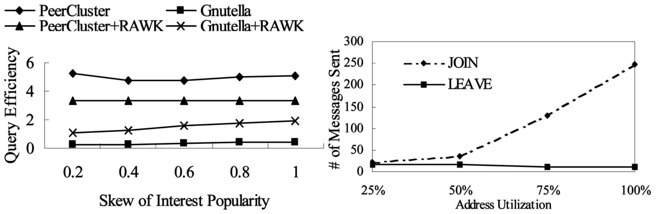
We also compare *PeerCluster* with *Gnutella* system when both are implemented with the random walker method (RAWK). From the results shown in Fig. 12b, it can be seen that with the RAWK method, the *query efficiency* of *PeerCluster* is still better than that of *Gnutella*.

Note that, according to the Fig. 12b, the *query efficiency* of the RAWK search method is outperformed by the blind search in *PeerCluster*. However, if we increase the TTL value of the RAWK method, its *query efficiency* improves. Indeed, the blind search and the RAWK methods have different advantages: the response time of blind search is faster, whereas the search radius of RAWK is larger. Hence, RAWK is more suitable for infrequent keyword searches, and its advantage is not prominent for popular keywords.

4.2.3 Skew of Interest Popularity

To model data skew, we assume that there are $|I|$ interest topics in the simulation model. The popularity of these $|I|$ topics follows a Zipf-like distribution with the parameter θ from 0.2 to 1 (i.e., the popularity each interest topic I_j is set to $\frac{j^{-\theta}}{\sum_{k=1}^{|I|} k^{-\theta}}$).

We now assess the performance of *PeerCluster* by varying the skew of the interest popularity. As presented in Fig. 13a, with skewed interest popularity, the *query efficiency* of *PeerCluster* is better than that of *Gnutella*. The reason is very similar to the one in the second experiment. When the interest popularity becomes more skewed, the computers that can be reached by *Gnutella* within the same search limit share fewer different interest topics. Therefore, within the same search limit in *Gnutella*, there are more files matching the query, and in turn, the *query efficiency* increases. Nevertheless, *PeerCluster* still outperforms *Gnutella* regardless of the skew of interest popularity. Similarly, the *query efficiency* of *Gnutella* using the RAWK search protocol is also affected by the interest popularity. In contrast, *PeerCluster* using RAWK is stable.



(a) (b)

Fig. 13. (a) Query efficiency under various skews of interest popularity. (b) Numbers of messages sent under various address utilizations.

4.2.4 Number of Messages Sent

In the fourth experiment, we measure the numbers of messages sent in the *JOIN* and *LEAVE* protocols under various levels of address utilization. As shown in Fig. 13b, the number of messages sent in *JOIN* remains almost the same when address utilization is less than 50 percent, whereas it increases linearly when the utilization increases from 50 percent to 100 percent. The reason is that, when the address utilization is under 50 percent, each participant computer owns two or more hypercube addresses. Thus, a joining computer can acquire a hypercube address easily. However, as address utilization increases from 50 percent to 100 percent, the number of participant computers owning multiple hypercube addresses decreases linearly. Therefore, the number of messages sent in Phase 3 of the *JOIN* protocol increases linearly. Also, the number of messages sent in the *LEAVE* protocol decreases slightly as address utilization decreases, which agrees with Theorem 3.

5 CONCLUSION AND FUTURE WORK

In this paper, we have proposed a cluster-based peer-to-peer system, called *PeerCluster*, for sharing data over the Internet. To guarantee that the structure of the interest clusters will not be altered by arbitrary node insertions/deletions, we devised *JOIN* and *LEAVE* protocols for *PeerCluster*. The complexities of these protocols were derived. In addition, we augmented *PeerCluster* with a system recovery mechanism to make it robust against unpredictable computer/network failures. Using an event-driven simulation, we evaluated the performance of *PeerCluster* by varying several system parameters. The experimental results showed that *PeerCluster* achieves superior *query efficiency*, while providing the desired functionality of keyword-based search.

In the future, as an extension of this study, we will explore dynamic interest groups and incorporate a feature hierarchy. In PeerCluster, the interest group is predefined and therefore fixed. However, as fashions change, so do interest groups. Therefore, the dynamic interest groups should be designed to be more adaptive to changing interests.

APPENDIX A

THEORETICAL PROPERTIES OF PEERCLUSTER

Let N denote the system capacity of *PeerCluster*, n denote the number of online computers currently in the system, and $|I|$ denote the number of interest topics. First, we derive the number of a participant computer's neighboring computers and the number of messages sent in intercluster routing in Lemma 1 and Lemma 2, respectively.

Lemma 1. *The mean number of neighboring computers that one participant computer in PeerCluster has is equal to* $\lg N$.

Proof of Lemma 1. Since the system capacity is N and there are currently n online computers in the system, then the mean number of hypercube addresses owned by one participant computer is equal to N/n. According to the properties of hypercube topology described in Section 2, one hypercube address in $Q_{\lg N}$ has $\lg N$ neighboring hypercube addresses; therefore, the mean number of neighboring computers that one participant computer in *PeerCluster* has equals $\lg N = \frac{N/n*\lg N}{N/n}$. □

Lemma 2. *Intercluster routing takes* $O(\frac{n}{N} \lg |I|)$ *messages.*

Proof of Lemma 2. From Section 3.2, we can see that the length of the path for intercluster routing depends on the address prefixes of the source and destination interest clusters. However, as described in Section 3.1, the address prefix of each interest cluster is determined by the Huffman tree constructed based on the interest popularity. To determine the length of the path for intercluster routing, we consider two extreme cases of the distribution of interest popularity: 1) If the popularity of all interest clusters is equal, the Huffman tree will be balanced with the height equal to $\lg |I|$. Then, we can obtain the mean length of the path for intercluster routing as follows:

$$\frac{1}{|I|} \times \frac{\sum_{i=1}^{\lg |I|} i \cdot Cr(\lg |I|, i)}{|I|-1} \times |I| = \frac{|I|}{2(|I|-1)} \lg |I| \simeq O(\lg |I|), \tag{1}$$

where Cr is the *Combination Formula*. 2) If the popularity of the interest clusters is extremely skewed, then the Huffman tree will have the height $|I|-1$. Then, we can determine the average length of the path for intercluster routing as follows:

$$\begin{aligned} &\frac{1}{2^{|I|-1}} + \sum_{i=0}^{|I|-2} \frac{2^i}{2^{|I|-1}} \times \frac{2^{|I|-1}-1}{2^{|I|-1}-2^i} \\ &= \frac{1}{2^{|I|-1}} + \frac{2^{|I|-1}-1}{2^{|I|-1}} \sum_{i=0}^{|I|-2} \frac{1}{2^{|I|-i-1}-1}. \end{aligned} \tag{2}$$

Note that when $|I| \geq 8$, the value of (1) will be larger than that of (2). Therefore, we have $O(\lg |I|)$ for the mean length of the path for intercluster routing. Since the mean number of hypercube addresses owned by each participant computer is N/n, the number of messages sent for intercluster routing is $O(\frac{n}{N} \lg |I|)$. □

With Lemma 1 and Lemma 2, we can derive the complexities of the number of messages sent in the *JOIN*, *LEAVE*, and *SEARCH* protocols in Theorems 1, 2, and 3, respectively.

Theorem 1. *It takes* $O(\frac{n}{N} \lg |I| + \lg N)$ *messages for one computer to join PeerCluster if the address utilization of PeerCluster is kept under 50 percent.*

Proof of Theorem 1. Recall that the *JOIN* protocol consists of four phases. Phase 1 takes $O(1)$ messages. Phase 2,

TABLE 2
Comparison between PeerCluster and Other Decentralized/Structure Systems

	PeerCluster	Chord	Pastry	CAN
Neighbors	$\lg N$	$\lg N$	$(2^b-1)\lg_{2^b} N$	$2d$
Search type	Keyword	Lookup/Keyword	Lookup/Keyword	Lookup/Keyword
Join	$O(\frac{n}{N}\lg\lvert I\rvert+\lg N)$	$\lg^2 N + key\_trans.$	$\lg_{2^b} N+key\_trans.$	$\mathrm{O(d)}+key\_trans.$
Leave	$O(\lg N)$	$\lg^2 N + key\_trans.$	$\lg_{2^b} N+key\_trans.$	$\mathrm{O(d)}+key\_trans.$
Search	$O(\frac{n}{N}(\lg\lvert I\rvert + 2^{SL}))$	$\lg N$	$\lg N$	$dn^{\frac{1}{d}}$

based on Lemma 2, takes $O(\frac{n}{N}\lg|I|)$ messages. Phase 4, according to Lemma 1, takes the mean of $\lg N$ messages. The number of messages sent in Phase 3 depends on the utilization level of the hypercube addresses. Let ϵ denote the number of messages sent in Phase 3. In the worst case, ϵ will be the size of the largest interest cluster when the joining computer would like to join the largest interest cluster and search all over that cluster. However, as shown in Section 4.2.4, if we keep the address utilization under 50 percent, we are able to approximate ε to a constant. Therefore, we have the complexity $O(\frac{n}{N}\lg|I| + \lg N)$ for the number of messages sent in the *JOIN* protocol. □

Theorem 2. *The mean number of messages sent in the LEAVE protocol is equal to* $\lg N$.

Proof of Theorem 2. Theorem 2 follows from Lemma 1 and the fact that the mean number of messages sent in Phase 2 of the *LEAVE* protocol is equal to $\lg N$. □

Theorem 3. *Let SL denote the search limit specified by the user. It takes* $O(\frac{n}{N}(\lg|I| + 2^{SL}))$ *messages to resolve a query in PeerCluster.*

Proof of Theorem 3. In Phase 1 of the *SEARCH* protocol, the intercluster routing takes $O(\frac{n}{N}\lg|I|)$ messages according to Lemma 2. In Phase 2, since the query messages are only broadcast to the computers in the subhypercube Q_{SL}, the messages sent for intracluster broadcasting are $O(\frac{n}{N} \cdot 2^{SL})$. Overall, it takes $O(\frac{n}{N}(\lg|I| + 2^{SL}))$ messages to resolve a query in *PeerCluster*. □

To provide more insight into *PeerCluster*, we compare *PeerCluster* with several existing *decentralized/structured* peer-to-peer systems (i.e., *Chord* [26], *Pastry* [21], and *CAN* [18]) in Table 2, where b is a configuration parameter in *Pastry* with a typical value of 4 and d is a parameter in *CAN*, specifying the dimensions of Cartesian coordinate space. It is observed that *PeerCluster* is the only system that supports the functionality of keyword-based search, whereas other systems only support the fundamental lookup service. In addition, *PeerCluster* performs better than other systems in terms of the *Join* and *Leave* operations. This advantage becomes more apparent as the number of *keys* that need to be transferred during the *Join* and *Leave* operations in other systems increases.

APPENDIX B

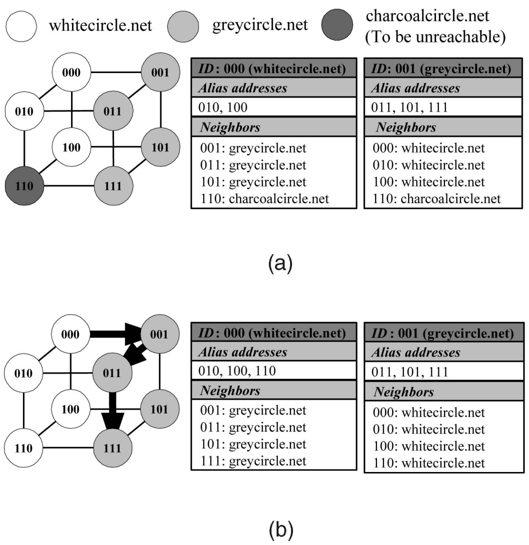
SYSTEM RECOVERY OF PEERCLUSTER

Under normal conditions, the integrity of the interest cluster structures can be guaranteed by the *JOIN* and *LEAVE* protocols described in Section 3.2. However, due to unpredictable network/computer failures, one participant computer could suddenly become unreachable. To ensure that *PeerCluster* is robust against such failures, we devised a system recovery mechanism.

The system recovery mechanism requires that every hypercube address be supervised by another hypercube address, called the supervisor address. Let a_i denote one hypercube address. Then, the supervisor address a_j of a_i is defined as the smallest hypercube address in $Ne(a_i)$. For example, the supervisor address of 00011 is 00001, since 00001 is the smallest hypercube address in $Ne(00011) = \{00010, 00001, 00111, 01011, 10011\}$. By this definition, every hypercube address will have one supervisor address. To detect failures, each computer that owns a supervisor address is required to periodically *ping* the computer that owns the supervised address to check that it is alive. Once a failure is detected, the following system recovery procedure should be followed:

1. **Incorporate the unreachable address**. Let a_u denote the unreachable hypercube address and A denote the computer that owns the supervisor address of a_u. Once a_u is detected as unreachable, A will incorporate the unreachable address a_u into A (i.e., $addr(A) \cup \{a_u\}$).
2. **Update the related routing tables**. After the incorporation of the unreachable address a_u, the new set of neighboring hypercube addresses of A will become $\bigcup_{a_i \in addr(A)\cup\{a_u\}} Ne(a_i) - (addr(A) \cup \{a_u\})$. To maintain the correctness of the related routing tables, computer A will notify its neighboring computers of the mappings from its new neighboring addresses to its IP address accordingly and, at the same time, bring back their IP addresses to update its own routing table.

Example B.1. As shown in Fig. 14a, there are currently three participant computers in the system. The routing tables of the computers blackcircle.net and 163.17.233.15 are shown on the right of the figure. Assume that the computer 128.15.114.21 suddenly becomes unreachable due to network failure. Since blackcircle.net owns the hypercube address 010, which is the supervisor address of 110, it will detect this failure and initiate the system recovery procedure immediately. In the first step, blackcircle.net will incorporate the hypercube address 110 into its alias addresses. Thereafter, in the second step, blackcircle.net will check every neighboring hypercube address of the incorporated address 110 (i.e., 010, 100, and 111) and find that 111 is the new neighboring



(a)

(b)

Fig. 14. An example scenario of the system recovery mechanism.

address. Finally, blackcircle.net will send its IP address to the computer that owns address 111 via the path 000⟶001⟶011⟶111 and bring back the corresponding IP address of 111 to update its own routing table, as shown in Fig. 14b.

In the recovery mechanism, a failed hypercube address is managed by its supervisor address. Each computer that owns a supervisor address is required to periodically ping the computer that owns the supervised address to it is working. Note that information exchange is frequent between peers in P2P systems. This behavior supports the probing scheme. Therefore, the probing scheme does not have to be performed frequently.

In addition, a failed hypercube address is recovered by its supervisor address, and its neighbors will map the failed hypercube address to the supervisor address's IP address of the failed address. Note that the system recovery procedure recovers one hypercube address at a time. If a computer owns multiple hypercube addresses becomes unreachable, or multiple participant computers fail simultaneously, multiple runs of the system recovery procedure will be needed.

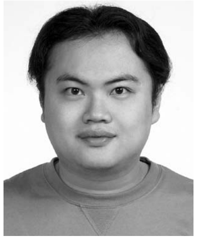
ACKNOWLEDGMENTS

The work was supported in part by the National Science Council of Taiwan, R.O.C., under Contracts NSC93-2752-E-002-006-PAE.

REFERENCES

[1] Open Directory Project (ODP), http://dmoz.org/, 1998.
[2] The Discussion Board of eDonkey, http://www.cyndi.idv.tw/forum/index.php, 2001.
[3] K. Aberer, "P-Grid: A Self-Organizing Access Structure for P2P Information Systems," *Proc. Int'l Conf. Cooperative Information Systems,* 2001.
[4] L. Bhuyan and D.P. Agrawal, "Generalized Hypercube and Hyperbus Structures for a Computer Network," vol. 33, pp. 323-333, 1984.
[5] T.F. Chan and Y. Saad, "Multigrid Algorithms on the Hypercube Multiprocessor," *IEEE Trans. Computers,* vol. 35, no. 11, pp. 969-977, Nov. 1986.
[6] Y. Chawathe, S. Ratnasamy, B.L.N. Lanham, and S. Shenker, "Making Gnutella-Link P2P Systems Scalable," *Proc. SIGCOMM '03,* 2003.
[7] M.S. Chen, P.S. Yu, and K.L. Wu, "Optimal NODUP All-to-All Broadcasting Schemes in Distributed Computing Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, pp. 1275-1285, 1994.
[8] Clip2.com, The Gnutella Protocol Specification V0.4, http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf, Mar. 2001.
[9] E. Cohen, A. Fiat, and H. Kaplan, "Associative Search in Peer to Peer Networks: Harnessing Latent Semantics," *Proc. IEEE INFOCOM '03,* 2003.
[10] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms.* MIT Press/McGraw-Hill Book Company, 1990.
[11] A. Crespo, "Routing Indices for Peer-to-Peer Systems," *Proc. 22nd Int'l Conf. Distributed Computing Systems (ICDCS),* 2002.
[12] N. Gunther, "Hypernets—Good (G)news for Gnutella," http://www.perfdynamics.com/Papers/Gnews.html, 2002.
[13] F. Harary, *Graph Theory.* Mass.: Addison-Wesley, 1969.
[14] J. Liebeherr and T.K. Beam, "HyperCast: A Protocol for Maintaining Multicast Group Members in a Logical Hypercube Topology," *Proc. First Int'l Workshop Networked Group Comm. (NGC '99),* 1999.
[15] Q. Liv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Network," *Proc. ACM SIGMETRIC '02,* 2002.
[16] Napster Inc., Napster Website, http://www.napster.com, 2006.
[17] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Lser, "Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-to-Peer Networks," *Proc. 12th Int'l World Wide Web Conf. (WWW '03),* 2003.
[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Proc. SIGCOMM '01,* 2001.
[19] P. Reynolds and A. Vahdat, "Efficient Peer-to-Peer Keyword Searching," *Proc. ACM/IFIP/USENIX Middleware Conf.,* 2003.
[20] J. Ritter, "Why Gnutella Can't Scale? No, Really," http://www.darkridge.com/jpr5/doc/gnutella.html, Feb. 2001.
[21] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Proc. 18th IFIP/ACM Int'l Conf. Distributed Systems Platforms (Middleware '01),* 2001.
[22] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl, "A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services," *Proc. Second Int'l Conf. Peer-to-Peer Computing,* pp. 104-111, 2002.
[23] K. Sripanidkulchai, "The Popularity of Gnutella Queries and Its Implications on Scalability," http://www.cs.cmu.edu/kunwadee/research/p2p/gnutella.html, Feb. 2001.
[24] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location and Retrieval in Peer-to-Peer Systems by Exploiting Locality in Interests," *Proc. ACM SIGCOMM '01,* 2001.
[25] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," *Proc. IEEE INFOCOM '03,* 2003.
[26] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. SIGCOMM '01,* 2001.
[27] G. Vrana, *Peering through the Peer-to-Peer Fog,* EDN Access (www.ednmag.com), 2001.
[28] B. Yang and H. Garcia-Molina, "Comparing Hybrid Peer-to-Peer Systems," *Proc. Very Large Data Bases Conf. (VLDB),* 2001.
[29] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Systems," *Proc. 22nd Int'l Conf. Distributed Computing Systems (ICDCS),* 2002.
[30] B.Y. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An Infrastructure for Fault-Tolerant Wide Area Location and Routing," Technical Report UCB/CSD-01-1141, Univ. of California at Berkeley, 2001.

Xin-Mao Huang received the MS degree from the Electrical Engineering Department at the National Taiwan University, Taipei, Taiwan, R.O.C., in 1998. He is currently a PhD student in the same department. His research interests include data mining, distributed systems, and multimedia applications.



Cheng-Yue Chang received the MS and PhD degrees in electrical engineering at the National Taiwan University in Taiwan. He is now an advanced researcher at Arcadyan Incorporation. His research interests include wireless home networking, audio/video streaming, and content management.

Ming-Syan Chen received the BS degree in electrical engineering from National Taiwan University, Taipei, Taiwan, and the MS and PhD degrees in computer, information, and control engineering from the University of Michigan, Ann Arbor, MI, in 1985 and 1988, respectively. Dr. Chen is currently the chairman of the Graduate Institute of Communication Engineering, and also a professor in both the Electrical Engineering Department and the Computer Science and Information Engineering Department, National Taiwan University. He was a research staff member at the IBM T.J. Watson Research Center, Yorktown Heights, New York, from 1988 to 1996. His research interests include database systems, data mining, mobile computing systems, and multimedia networking, and he has published more than 200 papers in his research areas. In addition to serving as a program committee member in many conferences, Dr. Chen served as an associate editor of *IEEE Transactions on Knowledge and Data Engineering* (TKDE) from 1997 to 2001, is currently on the editorial board of the *Very Large Data Base Journal* (VLDB), *Knowledge and Information Systems Journal* (KAIS), the *Journal of Information Science and Engineering*, and the *International Journal of Electrical Engineering*, and was a Distinguished Visitor of the IEEE Computer Society for Asia-Pacific from 1998 to 2000, and is also serving from 2005 to 2007 (invited twice). He served as the international vice chair for INFOCOM 2005, program chair of PAKDD-02, program cochair of MDM-03, program vice-chair of IEEE ICDE-06, IEEE ICDCS-05, ICPP-03, and VLDB-2002, and many other program chairs and cochairs. He was a keynote speaker on Web data mining in the International Computer Congress in Hong Kong, 1999, a tutorial speaker on Web data mining in DASFAA-1999 and on parallel databases in the 11th IEEE ICDE in 1995, and also a guest coeditor for *IEEE TKDE* on a special issue for data mining in December 1996. He holds, or has applied for, 18 US patents and seven ROC patents in the areas of data mining, Web applications, interactive video playout, video server design, and concurrency and coherency control protocols. He is a recipient of the NSC (National Science Council) Distinguished Research Award, Pan Wen Yuan Distinguished Research Award, and K.-T. Li Research Penetration Award for his research work, and also the Outstanding Innovation Award from IBM Corporate for his contribution to a major database product. He also received numerous awards for his research, teaching, inventions, and patent applications. Dr. Chen is a fellow of the IEEE and the IEEE Computer Society, and a member of the ACM.