Hierarchical Peer-to-Peer Networks

H. T. Kung
Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138, USA

C. H. Wu
Institute of Information Science
Academia Sinica
Taipei, Taiwan

Abstract -- In this paper, we describe a hierarchical architecture that can potentially scale peer-to-peer (P2P) networks to large numbers of peer nodes and contents. Two principles are followed: network routing reflects content clustering, and content placement reflects usage locality. We reason how these principles can lead to scalable P2P networks, and show techniques of implementing them.

I. Introduction

P2P networks are network where peer nodes communicate and transport information directly with each other. Unlike the conventional client-server model over the Internet, a peer node of P2P networks may act as both a client and a server simultaneously to share files or computing powers. It can request, serve, or relay services as needed.

A P2P system can aggregate a dynamic set of hosts in providing services. Resulting from rapid advances in hardware technologies, many machines today, especially user machines, have substantial free storage spaces and idle computing cycles. Thousands or millions of these machines working together could form a very powerful P2P-based virtual machine. A P2P system is then aimed to utilize these resources in a managed manner.

For examples, a file-sharing P2P application like Napster [1] can reduce the retrieval latency of contents and increase their availability with mechanisms to store and replicate contents. A cycle-sharing P2P application like SETI@home [3] can solve computationally demanding problems with mechanisms to distribute jobs to idle computers and collect results from them. Other P2P applications include those which have additional goals such as achieving access and service anonymity [4][6].

A major concern about P2P networks is their scalability. Because a peer node in conventional P2P networks has no knowledge about the global network topology nor content locations, it is difficult to find the target peers or desired contents efficiently. Existing P2P systems rely on either centralized directory servers, or message flooding or depth-first search [2][5][7].

This paper addresses this issue of achieving scalability for P2P networking. We use a hierarchical approach that will allow a P2P network to scale to large numbers of nodes and contents.

The paper is organized as follows. In Section II, we briefly overview current P2P network architectures and their scalability issue. Section III depicts our approach of scaling P2P networks, based on hierarchical content routing and adaptive content placement. In Section IV, we describe a method of constructing the required content hierarchy using content clustering techniques. Section V shows adaptive schemes that place nodes of a content tree onto a physical network based on content access patterns. Three optimization techniques are used: *shortcut, replication* and *migration*. In Section VI, we conclude that the approach of this paper can scale up P2P networks.

II. Current P2P Architectures

In this paper, we focus on the networking part of file-sharing P2P systems. In a P2P network, both the set of nodes and connections between them may change dynamically. In addition, a service request from a source node may invoke multiple intermediate nodes as relays to reach a destination node. A service reply may also work in a similar manner. Thus a node may play multiple roles in serving a request or reply, and may participate simultaneously in the service of multiple requests and replies. This has complex implications in system design in areas such as search queries, message routing, and replication management.

Based on discovery mechanisms of finding the best node containing a requested file and retrieval mechanisms of bringing the file to the requestor, we may classify current file-sharing P2P systems into three basic architectures. First, the Napster system builds a centralized server for search and supports direct file transfer among peer nodes. Second, the Gnutella system supports distributed search by message broadcast and transfers files using HTTP. Finally, the Freenet system supports distributed depth-first search and store-and-forward file transfer. All of these architectures incurred kinds of scalability problem that limit their popularity and usability. We examine each system in detail as follows.

A. Napster

The Napster P2P system, announced in January 1999 by Shawn Fanning, was aimed to share MP3 music files among Internet users [1]. The proprietary client-server protocol and client-client protocol were developed for communication in Napster [2]. The peer nodes (clients) download and serve files, while the centralized servers keep a list of the locations of available files and handle search requests from the clients.

Disk space for files in Napster is provided by the clients. One file may be stored in multiple locations. To download an MP3 music file, a client node first obtains a list of possible locations with the desired file from an index server. Then the client user selects the location for the file he or she would like to download, using the supplementary information provided by the index server. The client node then tries to retrieve the desired file directly from the peer node of the selected location. Since the user can select the nearest location with the desired file, they can download files fast.

The Napster network is a centralized file-sharing P2P system, where the index servers maintain a master list of all the clients, with their IP addresses and MP3 files. In addition, the servers serve every file search request. This kind of centralized P2P architectures have serious scalability problems.

B. Freenet

The Freenet network [4] provides a distributed service for storing and retrieving content. Each piece of content stored by Freenet is identified by a unique "file key," which is a hash of the content.

Each Freenet node maintains a "content routing table" and a local data store. The content routing table maps file keys to node addresses at which the files are thought to reside. The local data store is used to hold the actual contents.

Two basic operations are used in the Freenet system: the "Data Request" and the "Data Insert" operations. These operations allow retrieval and storage of content, respectively. Both operations work on

file keys.

Content with an unique file key enters the Freenet using a Data Insert operation. The data is placed onto nodes along the same path as the one taken by an initial "Data Request" message that determined uniqueness of the file key.

Content replies for a Data Request are cached in the local data stores along the reply path to the requestor. If the request recipient does not have the requested content, it initiates a depth-first search of the network in the following way. First, it looks up the file key in its content routing table. It iteratively finds the lexicographically closest match, and passes the request to the matched node. It returns a "Request Failure" message to the sender when a TTL value reaches zero.

C. Gnutella

Instead of the centralized index servers in Napster, the Gnutella network uses a "serverless" protocol for distributed search; peer nodes (called *servents*) broadcast query messages to neighboring servents to find out requested files [7].

When joining a Gnutella network, each new node establishes TCP connections to arbitrary sets of peer nodes. These connections are used to relay Gnutella protocol messages. Existing nodes leave the network by disconnecting them from all their peers. So the Gnutella topology may change at any time.

The two most important message types in the Gnutella protocol are *search queries* and *replies*. Search queries are distributed using a *scoped flood* mechanism. Only new queries are flooded to peers, as identified by query IDs. A time-to-live value limits the propagation distance of queries; a node can reach all nodes that are within a constant number of hops. If the recipients of the query have the requested content, they return search replies. The replies travel only along the reverse path toward the source of the query. Once the source node receives one or more search replies, the user can initiate transfer of the content. The transfer takes place outside of the Gnutella topology, through a direct connection between the source of the query and a node with the content.

Although the Gnutella network can support decentralized P2P queries, many studies have reported that it does not scale well due to broadcast queries [8][9][10]. An analysis [11] has also found that 70% of Gnutella users share no files, and 90% of the users answer no queries.

III. An Approach to Scale P2P Networks

We use two general principles in scaling P2P networks to large numbers of contents and peer nodes. First, we use content hierarchies to guide the routing of content request and reply messages (see Section IV). This routing method based on content hierarchies eliminates the need of message flooding or exhaustive search over P2P networks. Second, we use adaptive content placement schemes, which can replicate and migrate contents based on usage load and locality (see Section V). As described in the rest of the paper, these two principles will work simultaneously.

IV. Hierarchical Content Routing

In this section we describe the framework of hierarchical content routing. We develop the required key concepts on content vectors, content clustering trees, content search trees, physical networks and content placement.

A. Content Vectors

We assume that contents are associated with *content vectors*, which can indicate their similarity or dissimilarity. The content vector of a web page can, for example, be related to frequencies of certain keywords in the page. This process is similar to term weighting in the vector space model for conventional information retrieval systems [12]. Refer to [13] for typical methods of assigning content vectors to given contents.

There are two major categories of similarity metrics for content vectors [14]. These are angle-based metrics using, e.g., the cosine function, and distance-based metrics using, e.g., the inner-product function.

Consider the eight pieces of contents in Figure 1: LA, NY, NCA, PGA, Mozart, Bach, Jack and Helen. Suppose that content vectors are 2-dimensional. Then, as illustrated in Figure 1, the locations of these content vectors on a 2-dimensional space are expected to reflect similarity of contents, i.e., contents of similar nature such as LA and NY are close to each other.

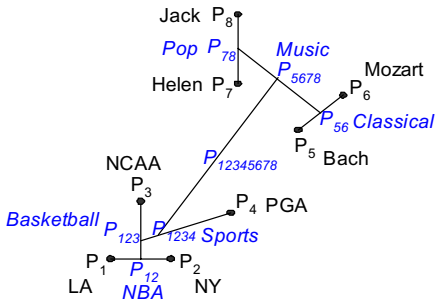

Figure 1: The content clustering tree constructed from eight content vectors.

B. Content Clustering Tree

Based on their content vectors, we can cluster a set of given content to produce a *content clustering tree*, by using a number of clustering methods [15][16]. Figure 1 depicts the content clustering tree resulting from such a method.

The method works as follows. It finds the pair of points, P_1 and P_2, which are the closest among all pairs of points, and then replaces these two points with a new point P_{12}, which is the midpoint of the segment (P_1, P_2). The method repeats this process until all points are connected. The position of each midpoint reflects the "weights" of the two endpoints, i.e., the numbers of contents under them. For example, midpoint P_{123} is closer to P_{12} than P_3 by a factor of two, since the tree rooted at P_{12} has twice as many content source nodes as that rooted at P_3.

C. Content Search Tree

Given a content clustering tree, we can form a *content search tree* that supports content search. For example, Figure 2 (b) depicts the content search tree derived from the content clustering tree of Figure 2 (a). The internal nodes of a content search trees, denoted by circles and Hs, are called *content nodes* or *content hubs*. The leaves of a content search tree, denoted by dots, are called *content sources*. Contents are stored in content sources. A content node may have one or more children, which are either content nodes or content sources.

We consider the content search tree of Figure 2 (b) to illustrate the working of content search. Suppose that the target content for the search is at the location indicated by the star in Figure 2 (a). The search will start at the root H0.

To determine the search direction at a content node, we make use of the Voronoi diagram [17] determined by geographic locations of its

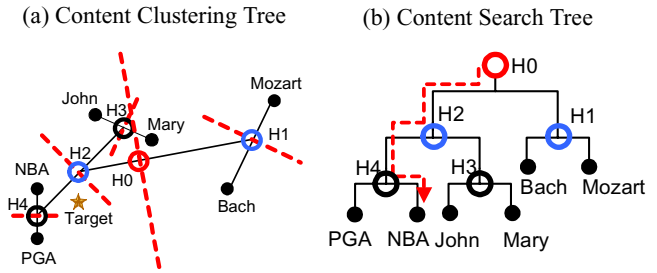

Figure 2: (a) Content clustering tree with dividing hyperplanes of Voronoi diagrams; and (b) content search tree derived from the content clustering tree.

children. (The Voronoi diagram is weighted reflecting the "weights" of the children, as described in the end of the preceding section.) That is, the search will go to the child whose site, defined by the weighted Voronoi diagram, contains the search target. Figure 2 (a) shows portions of those dividing hyperplanes which bound the sites defined by Voronoi diagrams. Thus, as depicted in Figure 2 (b), the search for content at the star location will go to H2 from H0, to H4 from H2, and finally to NBA from H4.

D. Physical Networks

In this paper, *physical networks* refer to P2P networks. Nodes of a physical network, called *physical nodes*, are peer nodes of the associated P2P network. By the same token, links of a physical network, called *physical links,* are peer links of the associated P2P network. We assumes that physical links are bi-directional.

There are three points worth of mentioning about physical networks. First, physical links are actually not "physical", since these links in P2P networks are TCP/IP connections over IP networks [2][5][7]. Thus, unlike T1 or OC-3 circuits, physical links in our context can be easily established and terminated as needed. We will use this feature in shortcut and replication optimizations described below.

Second, nodes of physical networks may physically reside in different regions of the globe, and served by different ISPs. It is therefore desirable that contents frequently requested from a given physical node be in the same region or served by the same ISP. This motivates content replication and migration optimizations discussed below.

Third, the *physical address* of a physical node is its network address. Since our physical networks are P2P networks over IP networks, physical addresses in our context refer to IP addresses.

Figure 3 depicts a physical network with its physical nodes partitioned into two regions. These regions could correspond to, for example, Europe and North America.

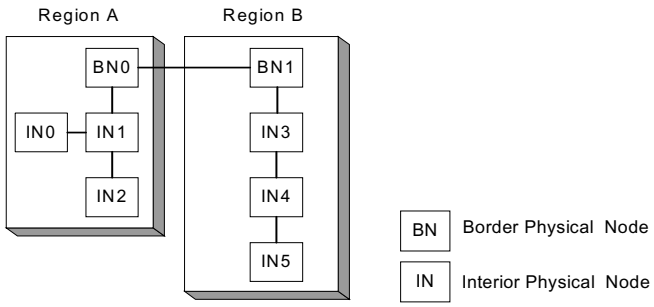

Figure 3: An example of physical network, consisting of physical nodes located in two regions.

As depicted in Figure 3, there are two types of physical nodes in a region. The first type is *border physical nodes,* which have physical links connecting to border physical nodes of other regions. The second type is *interior physical nodes,* which are the rest of physical nodes. Note that for the content migration optimization described below we will make explicit use of the notion of border physical nodes.

E. Placing a Content Search Tree on a Physical Network

At any given time, a content search tree is placed on a physical network. That is, each content node (and its content sources) reside at one or more physical nodes. Figure 4 (b) illustrates such a placement. Content nodes H0 and H1 are placed at interior physical node IN3 in region B, whereas H2 and H4 are at IN4 and IN5, respectively. Content node H3 is placed at IN2 in region A. Figure 8 (b) depicts an example where a content node (H0) resides in multiple physical nodes (IN3 and IN6).

A content node keeps track of physical addresses of its children. Thus, when a search needs to be directed to a child, the search can be correctly sent to the current physical address of the child. In addition, a content node keeps track of the physical address of its parent. In Section V, we discuss methods for optimizing the placement of content search trees on physical networks

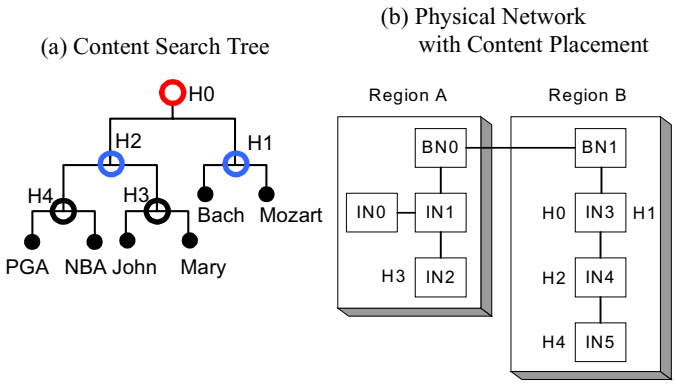

Figure 4: (a) Content search tree, and (b) physical network with content placement.

F. Hierarchical Content Routing

Having developed the required concepts and terminologies, we now describe *hierarchical content routing (HCR)*. HCR means routing messages on a physical network according to a content search tree and its current placement on the physical network. Consider, for example, the content search tree and its current placement on a physical network, shown in Figure 4. Figure 5 (a) illustrates the routing path under HCR for a message originated at IN2 and destined at a content target near NBA. Figure 5 (b) shows the corresponding search path on the content search tree.

More precisely, under HCR, a *request message* searching for a target content will first be routed to the root of a content search tree, such as H0 in Figure 5 (a), that is likely to have contents similar to the target content. (Physical nodes may cache the physical addresses of the roots of a number of content search trees of interest, or use directories or search engines to obtain these physical addresses.) After the request message reaches the root of the content search tree, its future routing decisions will be based on those on the content search tree, as depicted by Figure 5 (b). The actual message routing will be carried out on the physical network, as shown in Figure 5 (a), using the current placement of the content search tree on the physical network.

The *reply message*, which contains the requested content, will follow the path that the request message took, in the reserve direction.

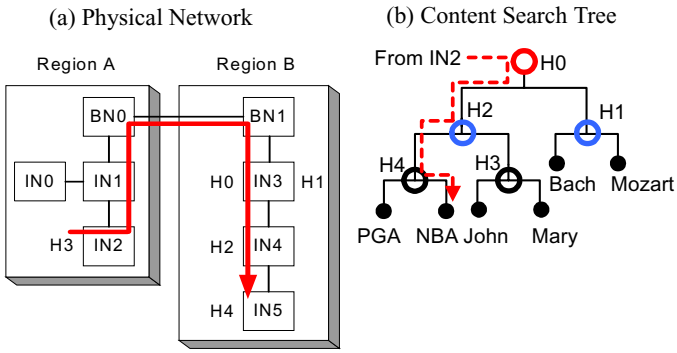

Figure 5: (a) Routing path for request messages under hierarchical content routing (HCR) on a physical network, and (b) the corresponding search path on the content search tree.

That is, the reply message from a content source will first be routed to the root of the content tree using the content source's tree path to the root. Then the reply message will be sent from the root of the content tree to the content requestor using its physical address.

G. Inserting Contents

Content insertion will first perform a search for the content to be inserted. Then it will restructure the content clustering tree and the content search tree in response to the new content, by updating the content nodes along the insert path in the bottom-up order. This restructuring may create new content nodes or destroy some useless old content nodes.

As an example, consider the insertion of the NCAA content into the tree of Figure 2. As depicted in Figure 6 (a), the search for NCAA will find content NBA. Since H4 is the nearest content node that knows the location of NBA, the physical node containing H4 will allocate a space for the new NCAA content and create a new content node, H5. Then H4 will update itself as H4' to reflect the weight change due to the new content and propagate the result to its parent nodes in an insert ACK message. While receiving the insert ACK message, each intermediate node along the insert path such as H2 will update itself. Finally, the root content node H0 will also update itself as H0'. The updated content search tree is shown in Figure 6 (b).

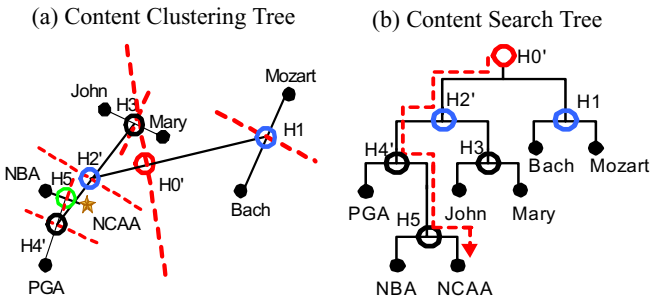

Figure 6: (a) Content clustering tree after inserting NCAA content into the tree of Figure 2 (a); and (b) the corresponding content search tree.

In implementation, updating a content node means updating the corresponding centroid vector. Using the numbers of represented contents as weights, we can simply compute the new centroid vector by averaging the old centroid vector and the newly inserted content vector. The updated vector of a child content node can also be reported to its parent node in the insert ACK message to prevent additional message communication. Both of above methods can reduce the content insertion cost.

Note that, even if we do not restructure the tree immediately, we are still able to find the new inserted content. This is because the old search path used the content insertion is still valid. So we can accumulate a bunch of update requests and perform the updates in a batch in order to reduce the update frequency of a content node. For the robustness of the tree, an updated content node may analyze the new subtree and cluster its child contents and nodes if necessary.

To illustrate this, consider this example of inserting NCAA again. Before inserting NCAA, we have the old search path H0-H2-H4 to find the location for the new content. After creating H5 and before updating H4, H2, and H0, we still can find H5 exactly through the extended old search path H0-H2-H4-H5. The difference between the extended old search path and the new search path is their capability of searching similar contents.

V. Adaptive Content Placement

Hierarchical content routing described above in Section IV supports a number of performance optimizations. This section describes three such optimization strategies.

A. Optimization 1: Shortcut

Shortcut links can be created for a content search tree to provide direct links between content nodes that have heavy traffic between them. Figure 7 illustrates the use of shortcut links. Using the shortcut links from H0 to H3 and H4, and the existing link from H0 to H1, the content node H0 will directly forward a search to one of its children, H1, H3 and H4, without going through an intermediate content node H2. The search direction at H0 will be determined by which of the three sites of Figure 7 (b) that has the target content, rather that the two sites of Figure 7 (a).

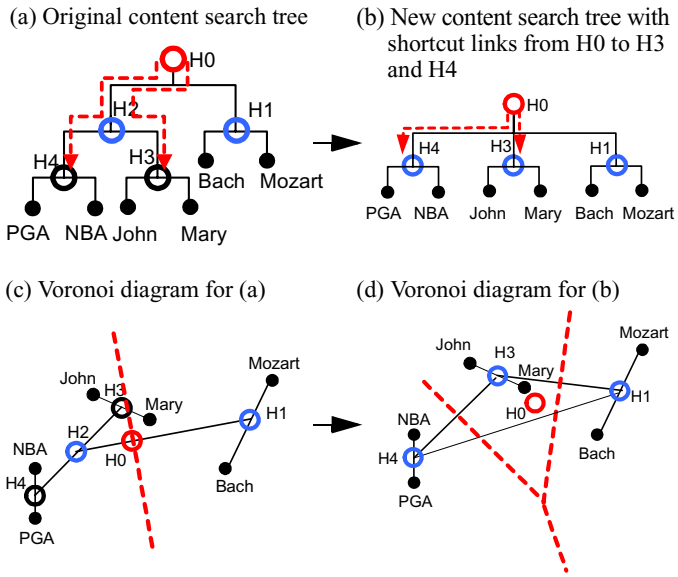

Figure 7: Shortcut optimization. (a) The physical node where H0 is placed recognizes that there is heavy traffic from H0 to both H3 and H4, and there is no traffic destined to H2. (b) The physical node creates shortcut links from H0 to both H3 and H4, without involving H2. The physical node will now direct search to one of the three sites of (d) rather than the two sites of (c).

B. Optimization 2: Replication

Under the replication optimization, replicates of a heavily used content node can be created and placed on multiple physical nodes. As depicted by Figure 8 (a), when the physical node IN3 recognizes that

its load is too high, it acquires another physical node IN6 to share the load. IN6 will have physical links connecting to IN3's neighbors, BN1 and IN4. In addition, IN3 will inform its upstream physical node BN1 to split the work between IN3 and IN6. Figure 8 (b) depicts the scenario where search going to H1 and H2 will use IN3 and IN6, respectively.

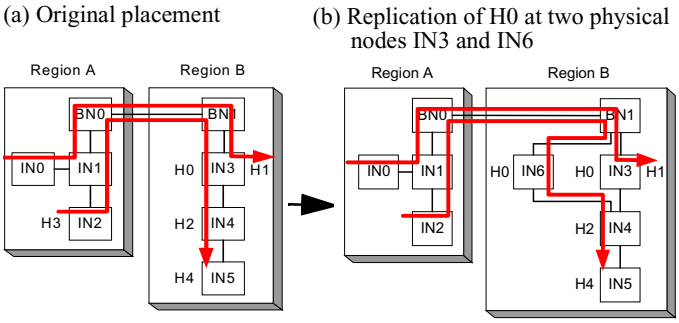

Figure 8: Replication optimization. (a) The physical node IN3 containing H0 recognizes that it has heavy traffic going to both H1 and H2. (b) IN3 replicates H0 into two copies, one for IN3 and another for IN6, and initiates the establishment of physical links between IN6 and BN1 and between IN6 and IN4 if the links do not already exist.

C. *Optimization 3: Migration*

Under the migration optimization, content nodes and sources frequently accessed by a region will migrate to the region. Consider for example the scenario of Figure 9 (a). Suppose that physical node IN1 in region A recognizes that there are a large number of requests going to H0 at IN3 in region B. Then IN1 will initiate a process of migrating a copy of H0 from IN3 to IN1, as shown in Figure 9 (b). Similarly, after IN2 recognizes that it has heavy traffic with H4 in region B, H4 migrates from IN5 to IN2 as shown in Figure 5 (c). After these migrations, access from IN2 to H4 will be local within region A as shown in Figure 5 (d).

VI. Conclusions

We have formulated a new approach to making P2P networks scalable. The approach is based on two principles: use content hierarchies to route content request and reply messages, and place content to reflect usage load and locality. By design, these principles eliminate costly message flooding or broadcast which has prevented existing P2P systems from being scalable. We have presented three performance optimization techniques: shortcut, replication and migration. P2P networks, where peer links can be freely established and storage can be provided at any peer nodes, naturally support these optimizations.

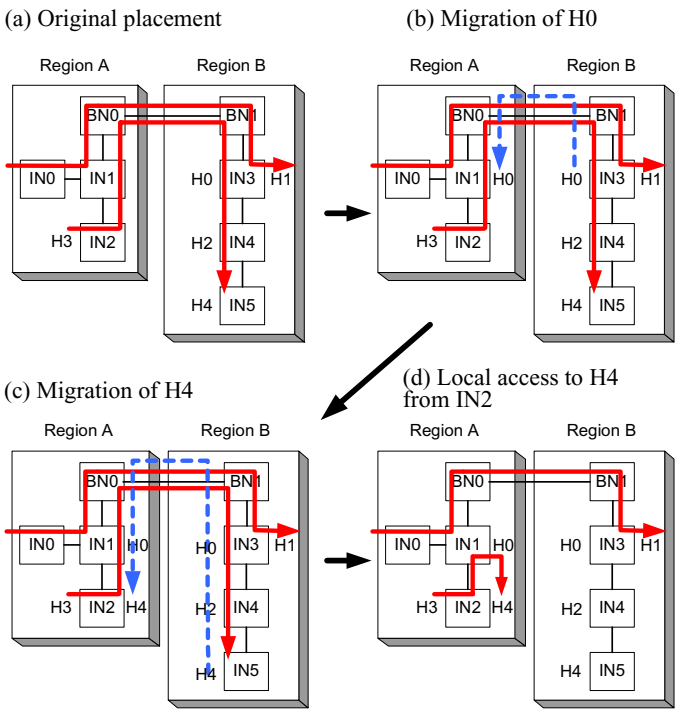

Figure 9: Migration optimization. (a) The physical node IN1 in region A recognizes that it has heavy traffic with H0. (b) A copy of H0 migrates from IN3 to IN1. In the meantime, IN2 recognizes that it has heavy traffic with H4 in region B. (c) A copy of H4 migrates from IN5 to IN2. (d) Access to H4 from IN2 will now be local within region A.

References

[1] Napster Homepage, http://www.napster.com/.

[2] "Napster Messages," http://opennap.sourceforge.net/napster.txt.

[3] Korpela, E., Werthimer, D., Anderson, D., Cobb, J., and Lebofsky, M., "SETI@home - Massively Distributed Computing for SETI," Computing in Science and Engineering, 3(1):78-83, January/February 2001.

[4] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W., "Freenet: A Distributed Anonymous Information Storage and Retrieval System," In Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability, International Computer Science Institute, Berkeley, CA, 2000.

[5] "Freenet Protocol 1.0 Specification," http://freenetproject.org/index.php?page=protocol.

[6] Gnutella Homepage, http://gnutella.wego.com/.

[7] Clip2, "The Gnutella Protocol Specification v0.4," http://dss.clip2.com/GnutellaProtocol04.pdf.

[8] Ritter, J., "Why Gnutella Can't Scale. No, Really," http://www.darkridge.com/~jpr5/.

[9] Sripanidkulchai, K., "The Popularity of Gnutella Queries and Its Implications on Scalability," http://www.cs.cmu.edu/~kunwadee/research/p2p/paper.html.

[10] Jovanovic, M., "Scalability Issues in Large Peer-to-Peer Networks - A Case Study of Gnutella," University of Cininnati Technical Report 2001. Available at http://www.ececs.uc.edu/~mjovanov/Research/paper.html.

[11] Adar, E., and Huberman, B. A., "Free Riding on Gnutella," Technical report, Xerox PARC, August 10, 2000.

[12] Salton, G., Wong, A., and Yang, C. S., "A Vector Space Model for Automatic Indexing," Communications of the ACM, 18(11):613-620, November 1975.

[13] Salton, G., and Buckley, C., "Term-Weighting Approaches in Automatic Text Retrieval," Information Processing and Management, 24(5):513-523, 1988.

[14] Jones, W., and Furnas, G., "Pictures of Relevance: A Geometric Analysis of Similarity Measures," Journal of The American Society for Information Science, 38(6):420-442, November 1987.

[15] Jain, A., and Dubes, R., "Algorithms for Clustering Data," Published by Prentice Hall, 1988.

[16] Han, E.-H., and Karypis, G., "Centroid-Based Document Classification: Analysis and Experimental Results," In Proc. of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), September 2000.

[17] Okabe, A., Boots, B., Sugihara, K., Chiu, S. N., and Okabe, M., "Spatial Tessellations: Concepts and Applications of Voronoi Diagrams," Second Edition, Published by John Wiley and Sons, July 2000.