Hyper-G: A Distributed Hypermedia System

Frank Kappe[1]

Abstract

Hyper-G is a general-purpose, large-scale, distributed hypermedia system currently developed at the Graz University at Technology. It was designed for handling large amounts of multimedia data, and care has been taken to provide mechanisms for automatic maintenance of a dynamically changing body of information, advanced user interface features, and efficient use of network and computing resources.

While Hyper-G was originally designed to run in fast, local-area networks, this paper shows how Hyper-G is currently being transformed into a global hypermedia information system distributed over Internet that retains Hyper-G's advanced functionality such as automatic link generation and maintenance, navigation facilities, access rights, and distributed searching, and makes efficient use of network bandwidth by extensive caching.

I. Introduction

Hyper-G is the name of an ambitious hypermedia project currently in progress at the Graz University of Technology [1]. Hyper-G is designed as a general-purpose, large-scale, distributed, multi-user, hypermedia information system, similar in scope to Xanadu [2] and Intermedia [3]. Based on previous experience with large-scale information systems (videotex), the aim of the Hyper-G project is to develop a flexible hypermedia framework in order to study and possibly eliminate the problems typically associated with large-scale hypermedia systems.

The need for a large volume of information imposes some design decisions for the implementation of such a system. An important issue for handling large amounts of data is support for automatic structuring as well as maintenance of a dynamically changing body of information. Another aspect of the size of hypermedia systems is that orientation and navigation become more difficult as the size grows. Problems of users of such systems include: getting lost in "hyper-space"; having difficulty gaining an overview; not being able to find information that is known to exist; determining how much information on a given topic exists; how much of it has been seen and how much is left. These issues have been identified as crucial for the acceptance of hypermedia and have been heavily discussed in the literature [4, 5, 6]. Solutions that work well on small systems (e.g., global maps) fail completely when applied to large-scale hypermedia.

After the project started in 1990, a number of user and system requirements have been identified [7], turned into a flexible system design [8], and a prototype system was implemented. While originally meant to be a research project, an unexpectedly high demand for such a system has resulted in real use even of early prototypes of Hyper-G for Campus-Wide Information Systems[2] and similar applications.

Because of the high acceptance of networked information retrieval (NIR) tools like Gopher [9], WAIS [10, 11] and WorldWideWeb [12] that have been developed at the same time as the Hyper-G prototype (each offering some subset of Hyper-G functionality) we have decided to increase the emphasis on a distributed information system in phase 2 of the Hyper-G project (starting 1993), while retaining the original goals of advanced maintenance and navigation functionality. Also, compatibility to these NIR tools will be provided in both directions (i.e. using both their clients and servers in conjunction with Hyper-G). This paper describes the modifications to the original design that are made in response to that new requirement.

II. Basic Hyper-G Architecture

This section describes in the necessary detail the original design of the main Hyper-G components as they are used in the prototype implementation currently available. Given the requirements of a large number of (multimedia) documents and a large number of users it is fairly

---

[1] Dr. Kappe <fkappe@iicm.tu-graz.ac.at> is with the Institute for Information Processing and Computer Supported New Media (IICM), Graz University of Technology, Graz, Austria.

[2] A text-only interface to the campus-wide information system of the Graz University of Technology that is based on Hyper-G can be tried out by remote login to host 'finfo.tu-graz.ac.at'. In addition, the same CWIS may be reached by Gopher (host 'finfo.tu-graz.ac.at', port 70) and WWW (URL http://iicm.tu-graz.ac.at/ROOT) protocols.

obvious to arrive at a client/server design model. Although the design of a client that provides advanced navigation facilities while maintaining user interface consistency over information provided by a multitude of authors is interesting in its own right [13], for the purpose of this paper we will concentrate on the server side.

The Hyper-G server (also called *link server* for historical reasons) is a rather complex object oriented database of *objects*, i.e. descriptions of documents, links, anchors, collections, tours, remote databases, etc. (for an in-depth description of Hyper-G objects and features see [8, 14]) as well as *relations* between such objects (e.g., which anchors are attached to which documents, which source anchors are connected to which destination anchors/documents, which documents belong to which collection, etc.). The link server offers the following functions:

- It assigns object IDs to objects (currently, a 32-bit entity) and assures that no two objects can share the same ID. In addition, it is guaranteed that when an object is modified, it receives a new object ID so that it can be distinguished from the old version. Also, when an object is deleted, its ID cannot be reused.

- It maps object IDs to objects. In Hyper-G , an object ID is just a unique number (similar to an ISBN number or a mail message id) assigned to every object (and hence every document). In the link server (and only there) more information on the object is stored (such as title, author, creation date, and – in the case of documents – also the data necessary to retrieve the document from a document server). Therefore, should a document be changed (say, moved from server A to server B) only the information in the link server has to be updated.

- Unlike many other hypertext systems (e.g., WorldWideWeb), Hyper-G strictly separates links from documents. This allows to attach links to documents even when the document itself cannot be changed (e.g., because it resides on a CD-ROM or on a remote server and the protocol prevents writing of documents). In general, this is a desirable feature because it enables users to annotate material that is otherwise read-only (e.g., encyclopedias). The link information, together with meta-information about objects (title, author, creation date, etc.) is stored within the link server, while the documents themselves remain on a so-called *document server* (performs similar to a Gopher server).

- A centralized link store enables the system to support *bidirectional links*[3], i.e. answer the question "What other documents refer to the current document?", which is important for two reasons:

  - Whenever a document is deleted or modified, the system is able to tell what other documents refer to the document in question. This means that in such cases references to nonexistent or outdated documents can automatically be identified and (possibly) removed from the web, thus maintaining web integrity. Especially in large multi-user systems, where the person deleting a document can not be expected to know of all the links to that document, this is an essential feature.

  - Advanced user interfaces for hypermedia include graphical browsers that show the "surroundings" of the current document. With bidirectional links, the system is able to show both the links that have the current document as their departure point and those that arrive at the current document.

- Hyper-G extends the primitive node-link model of hypermedia (that has been compared with *GOTO*s of programming languages [15]) by separating organizational from referential links (compare [16]) and replacing organizational links with more high-level structural elements:

  - A *Collection* is a set of other structures, and is used to create a *collection hierarchy* like the one shown in figure 1. When users visit a collection, they are given an overview of all objects belonging to that collection, and can select (and thus visit) any of the objects contained (similar to a Gopher menu).

  - A *Cluster* is similar to a collection, but when visiting a cluster all of its sub-structures are visited (visualized). This structure is used to implement multimedia documents (e.g. play a

---

[3] It should be noted that the link server concept as well as bidirectional links are also implemented in the Intermedia system [3].

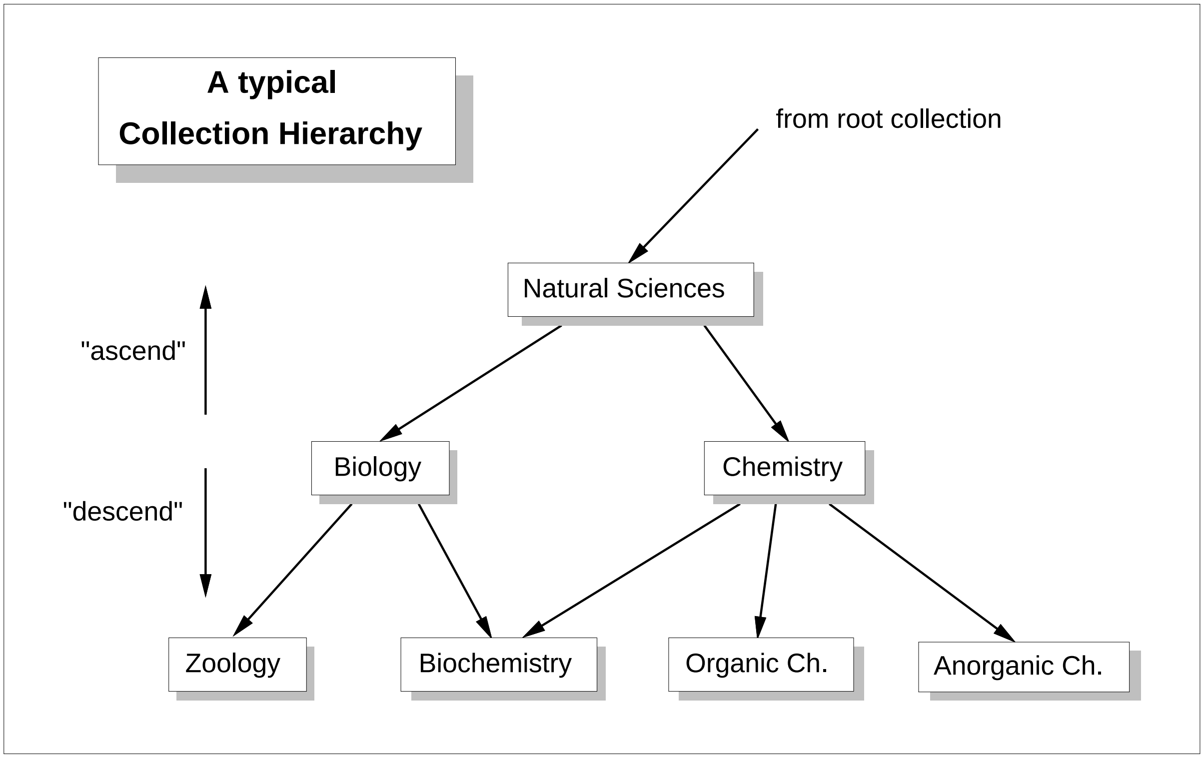

Figure 1: The Collection Hierarchy

sound while an image and a text document are shown), and to support multilingual documents and version control.

  - A *Tour* is a collection that visits all of its substructures in a certain order when the tour is visited. It is used to implement 'linear hypertexts' with automatically generated 'next' and 'previous' user interface elements.

The link server is aware of these structural elements and uses them to maintain database consistency (e.g. making sure that every document is a member of at least one collection, deletion of a document from a tour automatically joins its neighbors, etc.)

- As every Hyper-G object contains some meta-information (title, author, creation date, additional keywords, ...) the link server can perform complex boolean and fulltext queries (e.g., "Give me documents with title containing *UNIX*, created by user *fkappe* after *93/06/01*") either over the whole database or a certain subset of the *collection hierarchy*. Unlike Gopher or WorldWideWeb, Hyper-G does not require the information provider to set up a certain search engine for every suitable collection of documents. Rather, every document and every collection created is automatically searchable and it is the user's decision where a search seems feasible.

- A sophisticated, hierarchical *access control* scheme built into the link server allows to restrict access to individual documents and collections to certain groups of users. The link server also supports modification of the database (including rearrangement of the collection hierarchy and editing of text documents) by clients, for which access control is also a precondition. There are plans to support accounting functions as well, but they have not been implemented yet.

- The link server is also the ideal place to gather detailed *statistics* on the usage of the system.

The Hyper-G clients connect to the link server and use it to search and browse through the information space. Whenever a document (text, image, sound...) is needed, it is fetched from a so-called *document cache server* (see section IV). While in the original design there is only one link server, we will now describe how the link server itself can be distributed.

III. The Distributed Link Server

Why do we need a distributed link server at all? We might also choose to have a number of independent servers running at different sites, with documents within one server referring to objects in a different server in order to allow cross-references between servers (similar to Gopher and WorldWideWeb). However, this simple approach suffers from a number of shortcomings:

- We would lose Hyper-G 's ability to automatically maintain consistency of the information space. To illustrate this, consider the following example: A site has made available an interesting collection of data. Immediately other people would include references to it either by including it into their local hierarchy (like Gopher) or by creating hypertext links to it (like WorldWideWeb). When the original site chooses to remove, modify, or just move the data to a different site, it is difficult if not impossible to find out who has created references to the original information, to contact them, and to have that references updated/removed. In any case, it would require a significant (manual) maintenance effort.

- It would not be possible to perform (e.g., fulltext) searches over a certain subset of the collection hierarchy, if that hierarchy is distributed over a number of servers.

- The possibilities for caching are somewhat limited in this setup, causing a lot of network traffic.

- The approach leads to an effect that Ted Nelson has called "balkanization" [17]: Information tends to be organized by geographical location rather than by content. This is due to the difficulties of maintaining a collection of related information that is spread over a number of servers.

Figure 2 shows the architecture of Hyper-G (a more detailed analysis of which can be found in [18]). Clients are not required to connect to a number of Hyper-G servers. Rather, clients talk to the same server all the time. Should information from a remote server be needed, the local server fetches it and delivers it to the client. This approach offers the following advantages:

- It keeps clients simple and allows for a connection-oriented protocol.

- It enables caching of remote information in the local link server.

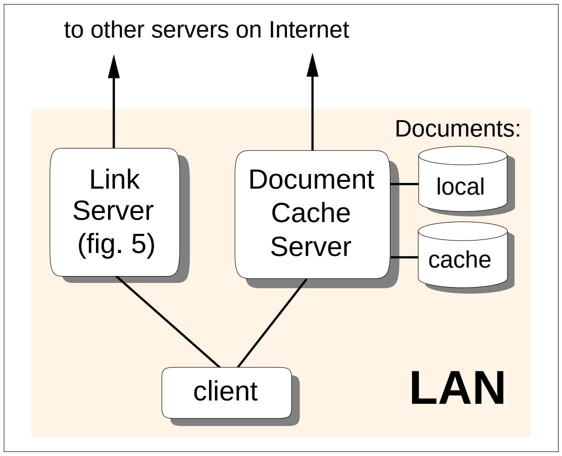

Figure 2: Hyper-G Architecture

- It eases maintenance of users and access rights in the local link server (user has to identify to one server only).

- It enables the link server to gather statistics and user profiles on a per-session basis.

The distributed link server is able to guarantee web consistency and to perform distributed searches, both across server boundaries. For the client, the existence of servers other that the local server is not visible; the local server performs like a "super-server" that knows about all the information stored in all other servers. In a way, it behaves similar to a domain name server that can be queried with the *gethostbyname()* system call on UNIX machines, knows about some local names, asks other servers for remote names, and caches the results for higher performance, all invisible for the calling process.

In order to see how this can be done, we have to dig into the internals of the link server a little bit more. We will first concentrate on the functions that are necessary to maintain relationships between documents and collections (e.g., "what collections does this document belong to?", "what documents have links pointing to this document?", etc.) over different servers, and discuss the problem of distributed searching later.

Information describing relationships between Hyper-G objects is not contained in the objects themselves, but rather in relations that are administrated by the link server. These relations are the only information that has to be shared between link servers in order to answer the questions raised above. For ease of explanation, we will now concentrate on one such relation, the *document-collection relation*.

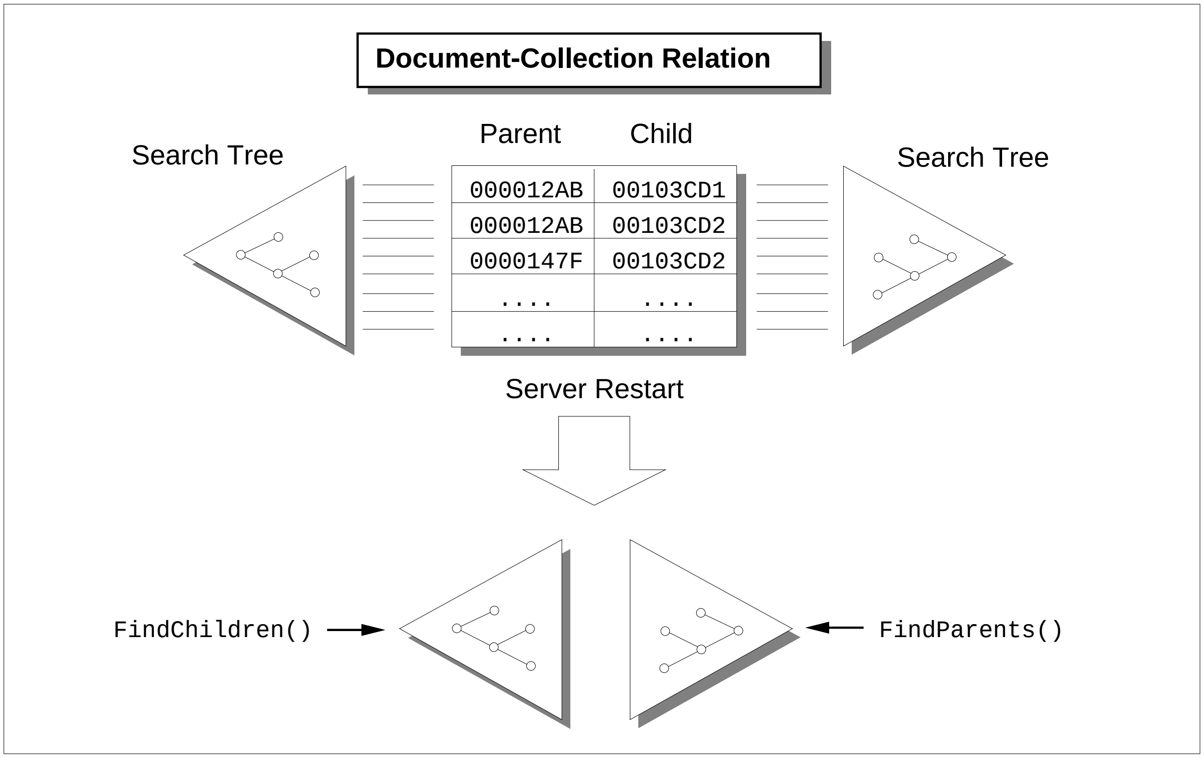

Figure 3: Implementation of the Document-Collection Relation

In general, we assume that Hyper-G data is read mostly and modified rather infrequently, so that we are willing to spend more time on insertion/deletion than on search and retrieval of objects. Being an object-oriented database, the link server internally performs functions like (in C++ notation):

```
doc->FindParents()
col->FindChildren()
```

to find the parents (i.e. the collection(s) it belongs to) of a document and the children (i.e. the subdocuments) of a collection, respectively. A more complex function like

```
doc->Delete()
```

uses the `FindParents()` function to find all the parents of the document to be deleted and instruct them to remove that document from the list of their children before actually deleting the document. Internally, `FindParents()` and `FindChildren()` are implemented using the document-collection relation as shown in figure 3.

The relation (the tabular array in the middle of figure 3) is supplemented by two search trees that allow rapid access (O(log n)) to the relation from both sides. The left tree is used during the `FindChildren()` operation as the leaves store the Object IDs of the children of a given parent collection, while the right tree is used for the `FindParents()` operation as its leaves store the parent collection(s) of a given document. When inserting or deleting a relation, the relation table as well as both trees have to be updated. As indicated in the lower part of figure 3, the trees are only auxiliary data structures that can be generated from the relation table on server restart. After that, the original table is accessed only when modifying the relation (which we assume to be a rather infrequent operation). Most of the time the two trees will suffice.

There exists a similar arrangement of relations for dealing with links (*anchor-document* relation and *source_anchor-destination_anchor* relation) that work the same way and are used to find source and destination documents (anchors) of links, as well as all links (anchors) attached to a certain document. However, in order to show how relations are dealt with in the distributed environment, it is sufficient to discuss the distribution of the *document-collection* relation.

III.A. Distributed Relations

When evolving the single link server into a distributed link server, we first have to extend our notion of Object IDs. To achieve a unique identifier for every object in the Hyper-G world, we use a very simple scheme: A unique server ID is assigned to every Hyper-G server, and each server generates unique 32-bits IDs for its domain. Thus, the concatenated 64-bit ID is unique

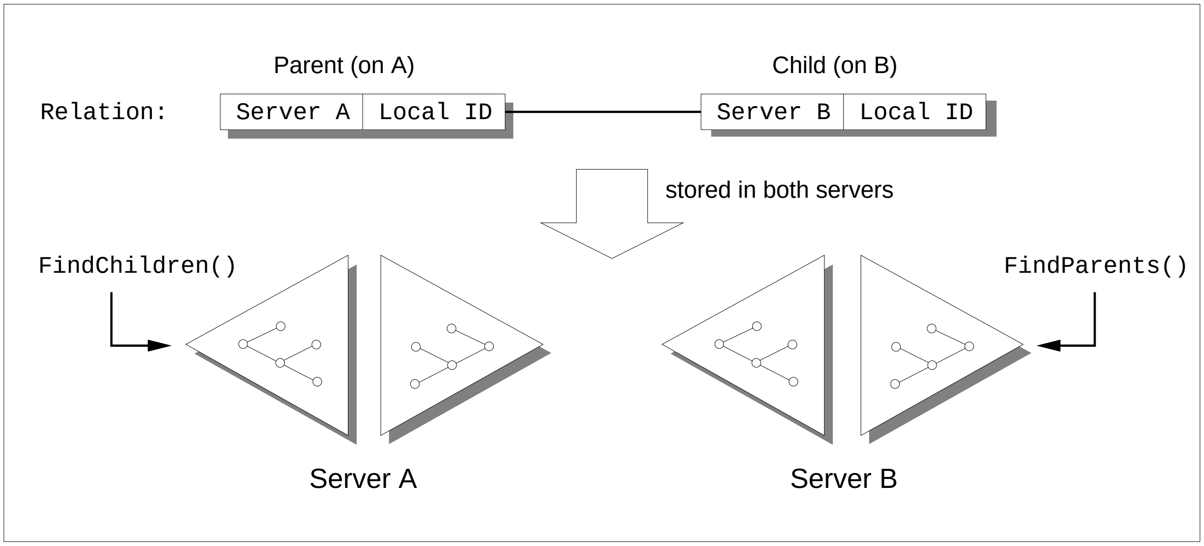

Figure 4: Distributed Document-Collection Relation

over the whole world of Hyper-G servers.

When we look at the distributed document-collection relation, we may distinguish two cases:

1. Both the parent and the child object reside on the same server (as indicated by having the same server part of the ID). In this likely yet trivial case, an entry in the document-collection relation of the server where both objects reside is made, and only on that server. Searching, insertion and deletion are performed as usual.

2. The parent collection resides on server A and the child on server B (figure 4). In that case, the relation is entered into the relation table on both servers, as well as into the left tree of server A and into the right tree of server B. When searching for the children of the object on server A, the left tree of server A is searched (leaves of left tree hold all children). When searching for the parents of the object on server B the right tree of server B is searched (leaves of right tree hold all parents).

As a rule of thumb, searching is always done on the server that stores the object one wants to know something about. It is always obvious where to look for information, and searching a relation can be completed by a single server. When a relation is going to be modified, however, the two servers have to be synchronized in modification of their relations, which requires a special server-to-server protocol and may take some time (this is consistent with our "read-mostly" assumption). Observe that at most two servers are involved in such an operation (no broadcasts necessary).

It is important to note that the ability to find the children as well as the parents of every object in the web (as well as finding the document with links pointing to the current document) enables the system to maintain database consistency over server boundaries, and to present to the user a graphical overview of both the collection hierarchy and the hyperlinks around the current document, regardless of server boundaries.

Because of performance considerations the 64-bit IDs are not really used in the relations. Instead, we stick with 32-bit local IDs and dummy objects created for remote objects. These dummy objects also serve as a local cache for remote objects.

III.B. Distributed Searching

To see how distributed searching can be performed, we again have to take a closer look at the implementation of the link server. As shown in figure 5, the link server internally consists of a low-level database (called *dbserver* for historical reasons) and a number of high-level database engines (called *hgserver*), one per user. The *dbserver* knows about objects and relations on a primitive level and is used to perform fast, atomic functions (as well as critical tasks like object locking), while the *hgservers* know about the user's context, have a more high-level view of the database (e.g. know what has to be done when an object is to be deleted and translate this to a series of calls to *dbserver*) and are able to spend more time on completion of the client's request as they run in parallel. The *ftserver* is specifically dedicated to full text retrieval, document clustering and automatic link generation, can run on a different machine because of high memory con-

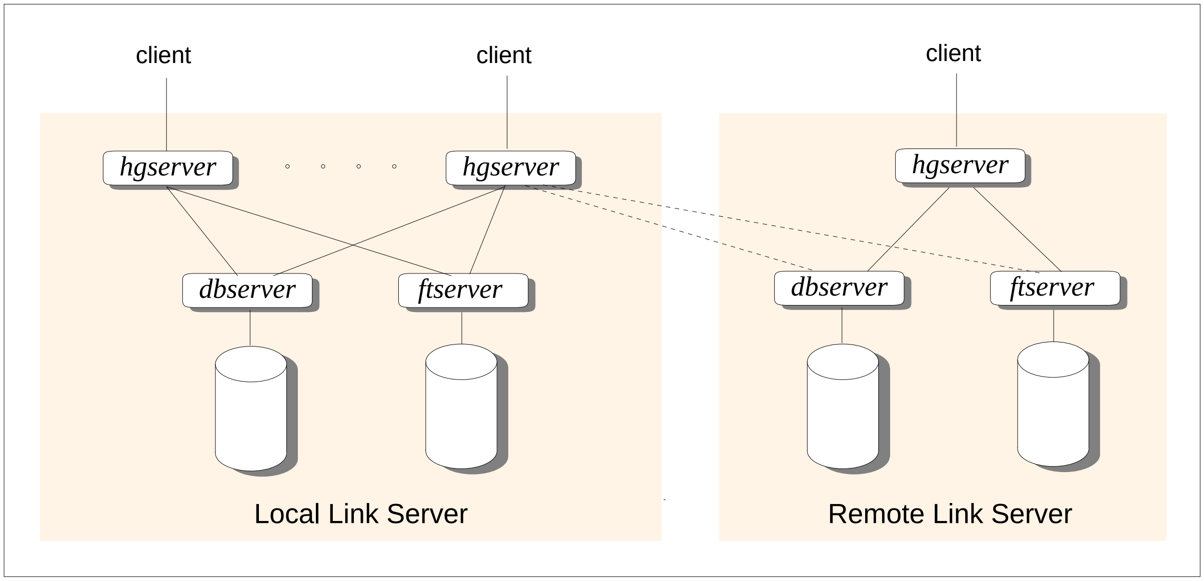

Figure 5: Internals of Link Server – Distributed Searching

sumption, and also performs operations in parallel.

Searching the Hyper-G database involves basically three steps:

1. Find the IDs of the documents that match the query. This step performs rather fast (O(log n) where n is the number of documents in the server) and returns an array of object IDs. It is performed by *ftserver* (when the contents of a text document is searched) and *dbserver* (deals with all other searches).

2. The list returned by the previous step is matched against the set of activated collections (i.e. the ones that the user switched on for this search operation). This is done by the user's *hgserver*, in cooperation with *dbserver* and can be slow if many hits are returned by the previous step. The object IDs that were found to reside in activated collections are returned to the client.

3. In a last step, the client may request the full objects for the set of object IDs found (minus some that are already cached in the client), so that the title and other information on the object can be displayed. This operation is again rather slow but can be performed by *hgserver* alone.

As shown in figure 5, a user's *hgserver* can also connect to *dbservers* and *ftservers* of remote link servers. A bidirectional asynchronous protocol between *hgserver* and the other link server components allows to broadcast a call to a number of *dbservers* and *ftservers* simultaneously and collect the answers that are returned during a certain timeout period.

This feature makes is possible to perform a distributed search over any number of servers in parallel. With a constant timeout period (say, 15 seconds) it does not take longer to query all Hyper-G servers in the world than only two of them. Of course, too short a timeout period will result in loss of some of the answers, as will failure of a network connection.

IV. The Document Cache Server

In order to speed up access to documents (text, images, movies, sound...) and to conserve network bandwidth, Hyper-G uses a *document cache* server to store documents (shown in figure 2).

Clients do not connect directly to the document server that stores the desired document. Rather, all document requests are routed through the document cache. The client sends the document object (including ID, host, port, protocol, path...) to the document cache. If the document requested has not yet been cached, the request is forwarded to the document server that stores the document. The document cache retrieves the document from the document server and simultaneously retransmits it to the client and stores it in the cache. If the document is found in the cache, it is transmitted directly from the cache server.

The document cache server should be configured to control a certain amount of mass storage (e.g. a few hundred megabytes of a hard disk) and use it as cache memory for incoming documents. When that space gets filled, the server removes the document that has not been accessed for the longest time ("least-recently-used" strategy). It is intended that the document cache server resides in the same LAN as the client, so that transmission from cache to client is reasonably fast.

A problem typically associated with caching in distributed environments is that of modification of what is cached. E.g., when a document gets modified, we have to make sure that the user sees the new version of the document when it is retrieved the next time and not an old copy that remained in the cache. Other systems do this by assigning expiration dates to objects, or notifying all caches whenever objects get changed or deleted, which in turn requires to maintain a list of all caches and what they have cached, and so forth.

Fortunately, however, the problem does not arise at all in our design. Remember, the link server guarantees that new object IDs are assigned to modifications of objects and that IDs of deleted objects cannot be reused. When a document gets modified the new version receives a new document ID that is passed to the client when the user visits that document. The client will pass the new object to the cache server. Therefore, it is impossible that the cache server finds the new object in its cache and will automatically reload the new document from the document server. An old copy of the document residing in the cache cannot be accessed any more and will therefore eventually be deleted because of the "least-recently-used" strategy of the cache.

The cache server may also be used as a protocol and format converter. In order to access information stored within other databases (e.g. WAIS, Gopher, WorldWideWeb) clients may connect to the document cache server who will retrieve and cache the document for the client. In addition, the client may request the document in a specific representation (say, a certain image format and/or quality), and the cache server would convert it for the client and cache the result (and possibly also the original representation). This approach makes clients more simple as it relieves them from knowing about the oddities of other information retrieval protocols and file formats. Software maintenance becomes easier because only the cache server's code has to be modified in order to support new protocols and file formats (remember, there may be a large number of clients tailored to different platforms and user types).

Acknowledgments

Partial Support of the Hyper-G project by the Austrian Ministry of Science, Joanneum Research, and the European Space Agency is gratefully acknowledged.

References

[1] F. Kappe and H. Maurer, "Hyper-G: a large universal hypermedia system and some spin-offs," *ACM Computer Graphics, experimental special online issue*, May 1993. Available by anonymous ftp from siggraph.org in directory publications/May_93_online/Kappe.Maurer.

[2] T. H. Nelson, *Literary Machines, Edition 87.1.* South Bend, IN: The Distributors, 1987.

[3] B. J. Haan, P. Kahn, V. A. Riley, J. H. Coombs, and N. K. Meyrowitz, "IRIS hypermedia services," *Communications of the ACM*, vol. 35, pp. 36–51, Jan. 1992.

[4] D. E. Egan, J. R. Remde, and T. K. Landauer, "Lost in hyperspace: cognitive mapping and navigation in a hypertext environment," in *Hypertext: Theory into Practice*, (R. McAleese, ed.), pp. 105–125, Blackwell Scientific Publications Ltd., 1989.

[5] J. Nielsen, "The art of navigating through hypertext," *Communications of the ACM*, vol. 33, pp. 296–310, March 1990.

[6] J. Nielsen, *Hypertext & Hypermedia.* San Diego, CA: Academic Press, 1990.

[7] F. Kappe, H. Maurer, and I. Tomek, "Hyper-G – specification of requirements," IIG Report 284, IIG, Graz University of Technology, Austria, Apr. 1991. Also appeared in: Proc. CIS '91. Also available by anonymous ftp from iicm.tu-graz.ac.at in directory pub/Hyper-G/doc.

[8] F. Kappe, *Aspects of a Modern Multi-Media Information System.* PhD thesis, Graz University of Technology, Austria, June 1991. Also available as IIG Report 308; IIG, Graz University of Technology (Jun 1991), and by anonymous ftp from iicm.tu-graz.ac.at in directory pub/Hyper-G/doc.

[9] B. Alberti, F. Anklesaria, P. Lindner, M. McCahill, and D. Torrey, "The internet gopher protocol: a distributed document search and retrieval protocol,"

March 1992. Available by anonymous ftp from boombox.micro.umn.edu in directory pub/gopher/gopher_protocol.

[10] R. M. Stein, "Browsing through terabytes – wide-area information servers open a new frontier in personal and corporate information services," *Byte*, vol. 16, pp. 157–164, May 1991.

[11] B. Kahle, H. Morris, F. Davis, K. Tiene, C. Hart, and R. Palmer, "Wide area information servers: an executive information system for unstructured files," *Electronic Networking: Research, Applications and Policy*, vol. 2, pp. 59–68, Spring 1992.

[12] T. Berners-Lee, R. Cailliau, J. Groff, and B. Pollermann, "WorldWideWeb: the information universe," *Electronic Networking: Research, Applications and Policy*, vol. 2, pp. 52–58, Spring 1992.

[13] K. Andrews and F. Kappe, "Strait-jacketing authors: user interface consistency in large-scale hypermedia systems," in *Hypermedia '93, Zurich, Switzerland*, (H. P. Frei and P. Schäuble, eds.), (Berlin), pp. 130–137, Springer, March 1993.

[14] F. Kappe, H. Maurer, and N. Sherbakov, "Hyper-G – a universal hypermedia system," *Journal of Educational Multimedia and Hypermedia*, vol. 2, no. 1, pp. 39–66, 1993.

[15] L. DeYoung, "Linking considered harmful," in *Hypertext: Concepts, Systems and Applications; Proc. ECHT'90*, (A. Rizk, N. Streitz, and J. Andr