The Architecture of a A Massively Distributed Hypermedia System

Frank Kappe, Gerald Pani

Institute for Information Processing
and Computer Supported New Media (IICM),
Graz University of Technology, Graz, Austria

Abstract

For about 50 years computer science pioneers have dreamt of augmenting their intellect by sharing the collective knowledge of individuals with each other through a global network of machines. However, until recently the technology to implement such a "global information system" was not available at reasonable cost. We believe that today's technology and concepts would allow to create an ambitious information and communication system based on hypermedia principles that would be massively distributed (i.e. over the whole world).

This paper first compares the old visions with systems that are available today. Then we describe the architecture of a global, general-purpose hypermedia system in an evolutionary way, i.e. we show how it can be developed using techniques already explored by existing projects. The resulting global information system is specifically designed to operate in the real-world environment of the internet and makes efficient use of its structure.

1 Grand Visions

Vannevar Bush's 1945 article "As we may think"[3][1] marks the beginning of the long and colorful history of hypertext and hypermedia. Bush, who was then President Roosevelt's director of the Office of Scientific Research and Development, speculated about a device called "memex" that contained not only post-war literature but also sketches, photographs, and personal notes. The machine would let you browse and make "associative trails" (figure 1) between any points of the library, and later on you could traverse these trails. The system was conceived for large amounts of documents (Bush talks about inserting 5000 pages of material every day for hundreds of years) and relied on microfilm and optical technology instead of computers[2]. Of Course, "memex" was never implemented.

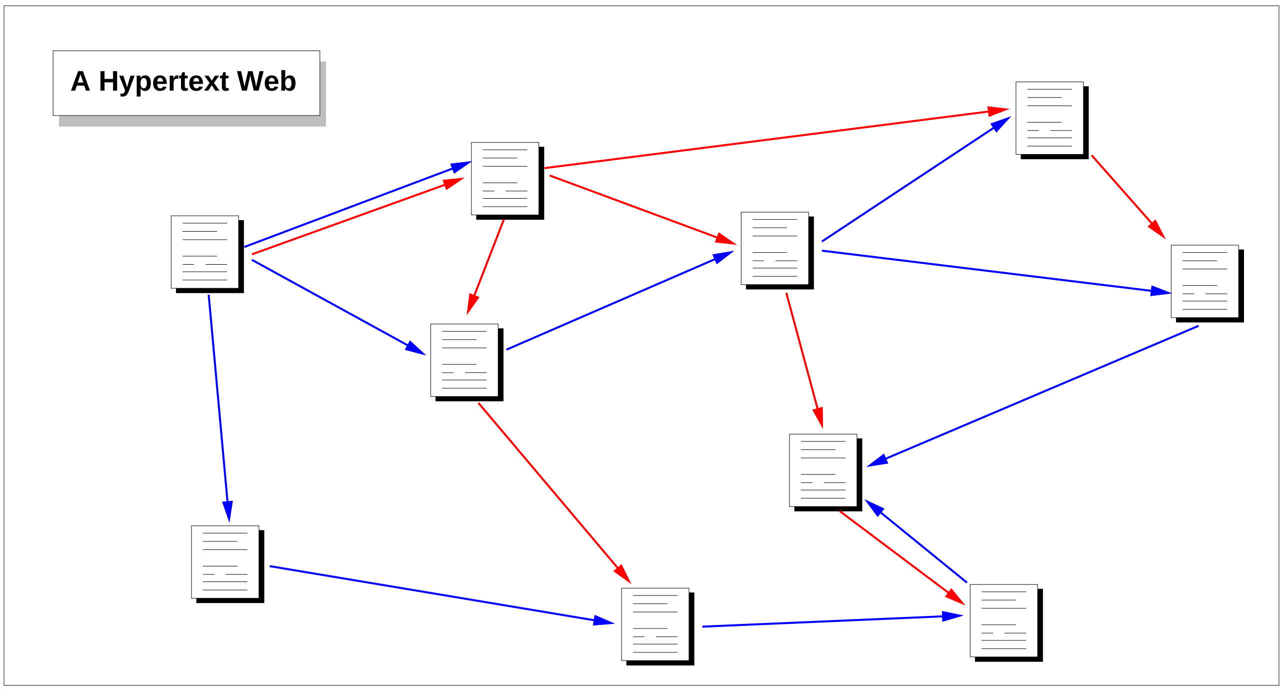


Figure 1: Hypertext: documents connected by links

Douglas Engelbart was the first computer pioneer to be influenced by Bush's concepts of associative links and browsing. In 1962 – at a time when computing resources were so expensive that nobody thought about using them interactively and for nonnumeric tasks such as text processing – Engelbart started work on the Augment project, also known as NLS (for oN-Line System) at the Stanford Research Institute [9]. This system was designed to store memos, research notes, and documentation. It was the the first to provide an on-line help system, integrated mail, and multiple windows. Because speed was important, Engelbart invented the now well-known mouse as input device.

[1] You probably won't be able to get the original. Fortunately, the article was reprinted in [4] and [20].

[2] One other prediction of Vannevar Bush regarded "electronic brains" the size of the Empire State Building with a Niagara-Falls-equivalent cooling system for their tubes and relays – you can not always guess right.

Ted Nelson coined the term "hypertext" in 1965. He envisaged an even more sophisticated system called Xanadu [19, 20], an ever-growing repository of everything that anybody has ever written – the publishing utility of the future – with additional communication support. In order to store this universe of documents (Nelson calls it "docuverse"), Xanadu is designed to run on a network of computers – a distributed hypertext system. However, while some parts of Xanadu have been a product of the Xanadu Operating Company since 1990, Nelson's *grand vision* has never been implemented and probably never will be [22].

With the advent of powerful personal computers and workstations, the term "hypertext" transformed to "hypermedia" to emphasize the fact that other media than text (such as graphics, sound, video, animation) can also be managed and linked together. Since it does not make too much sense to reserve a term for text-only systems, we will use "hypertext" and "hypermedia" interchangeably, and it should be clear that everything that is said about hypertext also applies to hypermedia.

The list of hypertext pioneers would be incomplete without mentioning Sam Fedida, who had similar ideas[10] concerning large-scale, publicly available information systems in Europe, and can be regarded as the father of videotex. Although videotex relies on cheap terminals, ordinary telephone lines, and consequently offers only relatively simple hypermedia features, with its 11 million users world-wide[3], it can be considered to be the first successful implementation of hypertext as a mass medium.

2 Down to Earth

Today we see a number of popular, commercially viable, stand-alone systems and tools (e.g., *HyperCard, SuperCard, Guide, Toolbook*) that conform to "hyper-"principles in that they allow to create linked pieces of information. However, they cannot be considered "real hypertext" systems according to Ted Nelson's definition and the ideas described before, due to the lack of communication features and the limitations on database size. Also, the user interface metaphor of such systems is more targeted towards browsing knowledge in a way predetermined by an author than finding facts[17] in an information system.

There are also a number of research implementations of large-scale information systems from institutions like Brown University (*Intermedia* [18, 26]), the University of Maryland (*Hyperties* [23]), and Xerox PARC (*NoteCards* [12]). More complete surveys on hypermedia systems can be found in [7], [22] and [25]. In general, those systems tend to be restricted to a local or distributed file system, and often are developed for a limited set of platforms.

On the other hand, the technical prerequisites for designing the publication and communication medium of the global village using hyper-principles look bright as never before. The internet

[3] France alone provides 17,000 services through *Télétél*, has an installed base of over 6 million terminals, with 85 million calls (7.4 million hours connect-time) per month (as of January 1992).

connects 992,000 host computers world-wide (July 1992), and grows about 100% per year[16]. Use of the client-server model allows a variety of platforms to interoperate in a distributed fashion. Existing information retrieval systems such as Archie[8], Gopher[1], Prospero[21] and WAIS[13, 24] make use of client-server protocols over wide-area internet but lack hypertext functionality (with the exception of the World-Wide-Web project[2]).

From a systems development view, a distributed system designed to work in a LAN over TCP/IP can be distributed over the whole internet without change. However, transmission bandwidth is not evenly distributed in the network (as is computing power). The internet consists of about 16,300 local area networks (so-called domains) with an average 60 hosts each[16]. While the LANs are usually fairly fast ($\geq$ 10 Mbits/s), the connections between them are typically much slower (sometimes phone lines). In addition, sending a packet to a computer far away may require the packet to pass through a number of gateways and satellite links, which introduces a significant delay for each packet sent, thus reducing interactivity.

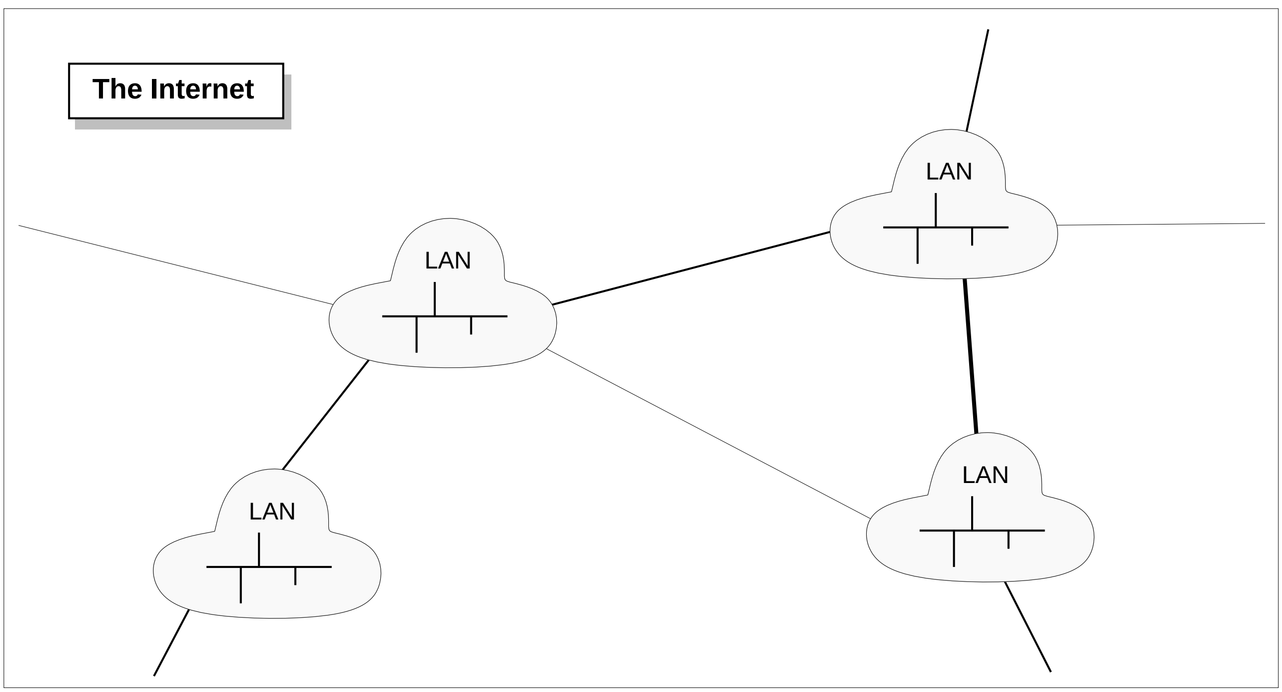


Figure 2: The internet consists of high-speed LANs ("domains") connected by slower links

In this paper we present the architecture of a massively distributed hypermedia system (i.e., it can be distributed over the whole internet) that is specifically designed to operate in this real-world environment of fast LANs connected by slower WANs (Figure 2).

For ease of understanding, let us evolve the global system in five steps. The first step introduces the client/server concept as used by most existing distributed information systems. In a second step we develop a simple distributed hypertext system, like the one implemented by the World-Wide-Web project. The third step describes the architecture of Hyper-G, the hypermedia system currently developed at the Graz University of Technology. In two following steps we will show how Hyper-G can be extended to act as a global hypermedia system, very much like the ones envisaged by the hypertext pioneers.

3 Clients and Servers

Modern multi-user information systems are often implemented as distributed systems with software concurrently executing on a number of information servers on one hand and end-user user interfaces ('clients') on the other. This offers the following advantages when compared to the old 'on-line' paradigm of interactive logins from terminals to a central mainframe:

- In conventional, terminal-based systems every keypress has to be transmitted to and answered by the host computer. This leads to a soft real-time problem, as it is well-known that a response time larger than about 250 milliseconds confuses users[5]. A lower response-time variance seems to be more important than absolute response times [6]. In order to achieve that kind of performance with hundreds of on-line users, expensive mainframes are required. Also, the network is flooded with a number of tiny packets (one for every keypress), which wastes bandwidth (e.g., very inefficient with token ring networks).

  In the client/server approach, local intelligence of the user's computer is used to implement the complete user interface (cursor movement, scrolling, entering keywords, searching...). Only when new data is required, it is fetched from the server, usually in large blocks. Processing of this data (e.g., decompressing, decoding, formatting...) is then again done in the local computer. This model dramatically reduces server and network load. Also, experience shows that users are willing to accept, e.g., to wait a few seconds for a database query, if the client software supplies them with immediate feedback about the acceptance of the query.

- If the system is to be distributed in a wide area (e.g., world-wide), interactive logins can be considered unacceptable. Satellite links provide reasonable transmission rates, but introduce a significant delay (about 500 milliseconds) for every packet. Therefore, transmission and echoing of individual keystrokes takes a long time, while the client/server concept with few but large blocks of data benefits from the high transmission rate.

- Interactive use means reduction of the user interface to the smallest common denominator (e.g. VT100 terminal: no mouse, no graphics, no sound). For users used to work with windows, this looks like the stone age of computing.

  In the client/server model the client (sometimes also called viewer) can use all the features of the user's hardware and software environment (such as windows, icons, pointing devices, bitmapped graphics, sound...). Also, it is possible to choose from a number of different clients, so that users can use their favorite look-and-feel (e.g. MS-Windows, Mac Finder, NeXTStep). In addition, it is easily possible to import data from the client to other (e.g. text processing) software by means of a clipboard.

- A client can connect to multiple servers (even using different protocols) and present information to the user using a consistent user interface. In contrast, with remote logins the user has to adopt to the different user interfaces of different information providers.

Because of above benefits, existing distributed information systems such as Gopher, World-Wide-Web and WAIS implement the client/server concept.

4 A Simple Distributed Hypertext System

Let us now consider a simple distributed hypertext system with clients and servers. For implementation of such a system two important design decisions have to be resolved:

1. How are documents stored and addressed?

2. How are links stored and followed?

As we want to have a distributed system, we opt to store documents on a number of computers connected by a network. On any such computer there exists a "document server", i.e. a process that sends documents upon request by the client. Retrieval of documents requires the definition of a common protocol shared by all clients and servers. The protocol can be very simple: Basically, the client sends a certain "selector string" containing an identification of the document – the "document identifier" – to the server (a certain port number on a certain host), and then the server sends back the document over the same connection (figure 3). This is the approach that systems like Gopher (though not a hypertext system) and World-Wide-Web take.

An important issue is the addressing of documents, i.e. the format of the "document identifier" mentioned before. The identifier has to serve two purposes:

1. To uniquely identify the document, similar to how an ISBN number identifies a book. Obviously, a certain mechanism has to guarantee that no two documents should share the same identifier, unless the documents are exact copies of each other.

2. To address the document, i.e. tell the client how to get it, including specification of the host and port number to contact and selector string to send.

Observe that the two requirements are to some extent contradictory: Because of the need for addressing, two identical copies of a document stored on two different machines will get two different document identifiers. Also, if a document "moves" from machine to machine, its document identifier changes. Consequently, there has been a sometimes controversial discussion on the topic of "Universal Document Identifiers (UDIs)" within the networking community.

Concerning the implementation of links, the easiest solution – and the one adopted by most hypertext systems – is to store links directly within documents. E.g., within a text document,

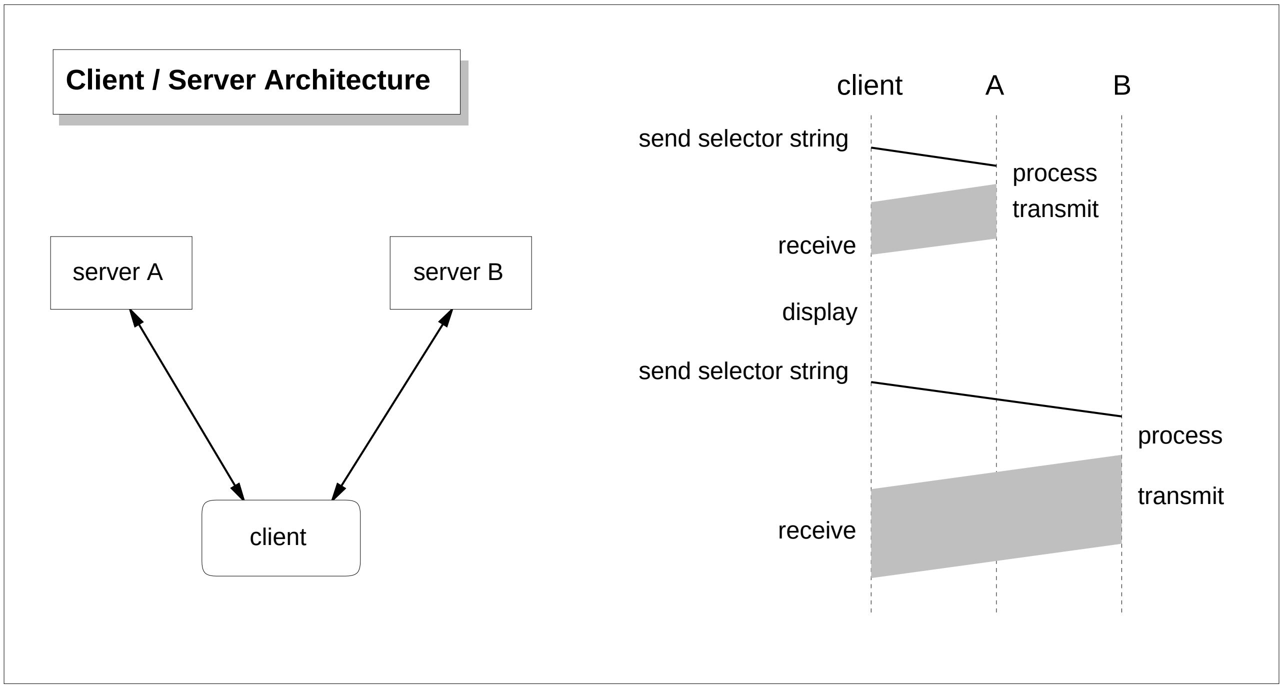


Figure 3: A simple client/server protocol that can be used for distributed hypertext

a link (more precisely: the source anchor of a link) is surrounded by special escape characters, or – in the case of SGML documents – by certain SGML tags (figure 4). Between the markers there is a text that is to be displayed in some highlighted manner (i.e. what the user sees) and the document identifier of the target document (invisible for the user). When the user selects the links (e.g. by clicking on the highlighted phrase) the client retrieves the destination document based on interpretation of the invisible part of the link.

The rather simple scheme as presented by now allows for world-wide distributed hypertext using the internet. Documents are easy to create, and both server and client software is easy to write. Moreover, its feasibility has already been demonstrated by the World-Wide-Web project[2].

However, the concept appears to be too simple in some respects, including:

- It is not possible to search the whole web for keywords.
- The fact that links are stored within documents means that changing a link requires change of the document. This is clearly a disadvantage in the following situations:
  - The document cannot be changed, e.g. because it resides on a CD-ROM, or because it is stored on a remote computer and the protocol prevents writing of documents.
  - The user has no permission to change the document. E.g., the user wants to attach an annotation to an encyclopedia entry. Obviously we do not want users to change encyclopedia entries but we do want to allow them to attach personal links (annotations).

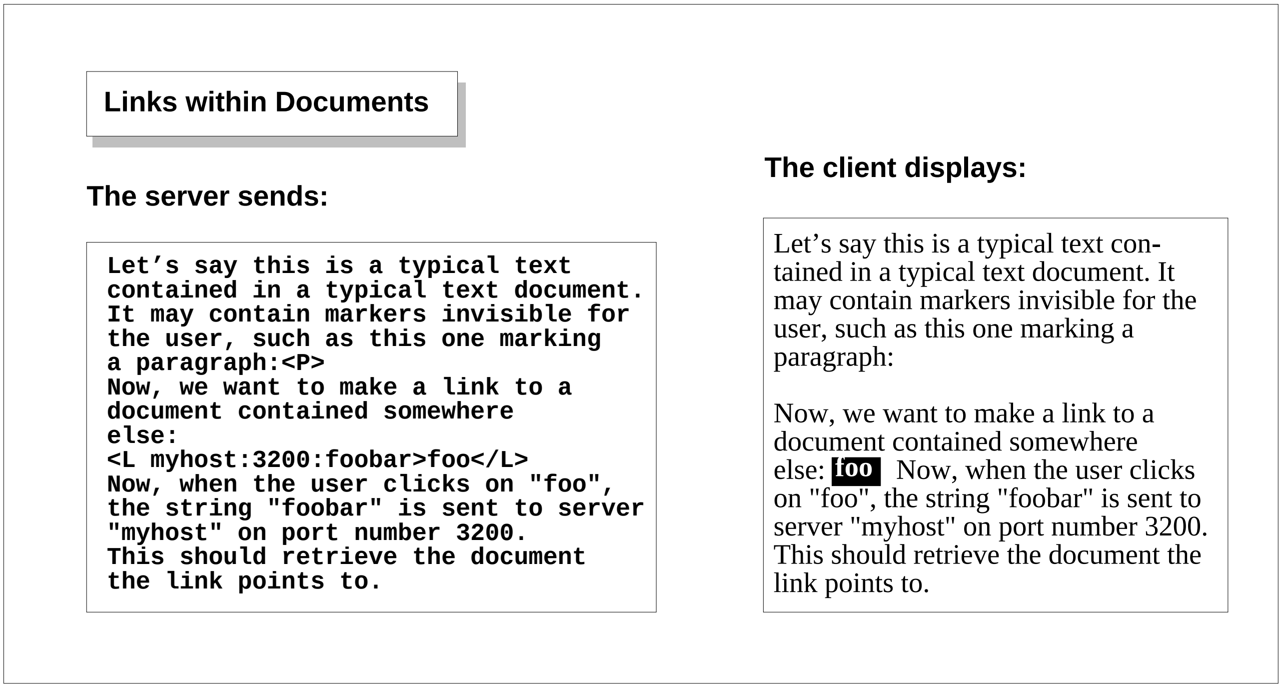


Figure 4: Links can be stored within documents (SGML example)

- In a multi-user environment, we would like to have the ability to create private links, i.e. attach access rights to links. In the simple system, access rights can only be attached to a whole document, including all the links of the document.

- Links cannot be traced backwards (unidirectional links), i.e. there is no way of finding all documents that link to a given document. This is a serious disadvantage for two reasons:

  – Whenever a document is deleted or modified, the system is not able to tell what other documents refer to the document in question. This means that in such cases references to nonexistent or outdated documents remain in the web, making it impossible to guarantee 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 completely unacceptable.

  – The system can only show the links that have the current document as their departure point but not the ones that arrive at the current document. In other words, the system cannot answer the question: 'What other documents refer to the current document?'. They are also unable to give the user an overview of the structure of the information network around the current document (like in Figure 1), although such an overview (“graphical browser”) could decrease the disorientation of users that is often reported in conjunction with hypertext systems.

- When used in a wide-area network such as the internet, traffic will be high and performance will be moderate due to the large number of document retrievals from far away.

5 A Local Area Hypermedia Architecture

Because of the shortcomings of the simple system outlined above, we will now describe a more powerful, albeit more complex architecture for a distributed hypermedia system that is implemented in Hyper-G, the hypermedia system currently developed at the Graz University of Technology. Hyper-G is designed to run in a fast local area (e.g. campus-wide) network. As a first application, Hyper-G is used as the basis of the university information system of the Graz University of Technology[15]. In the forthcoming sections we will then show how this concept can be transformed into a global distributed hypermedia system.

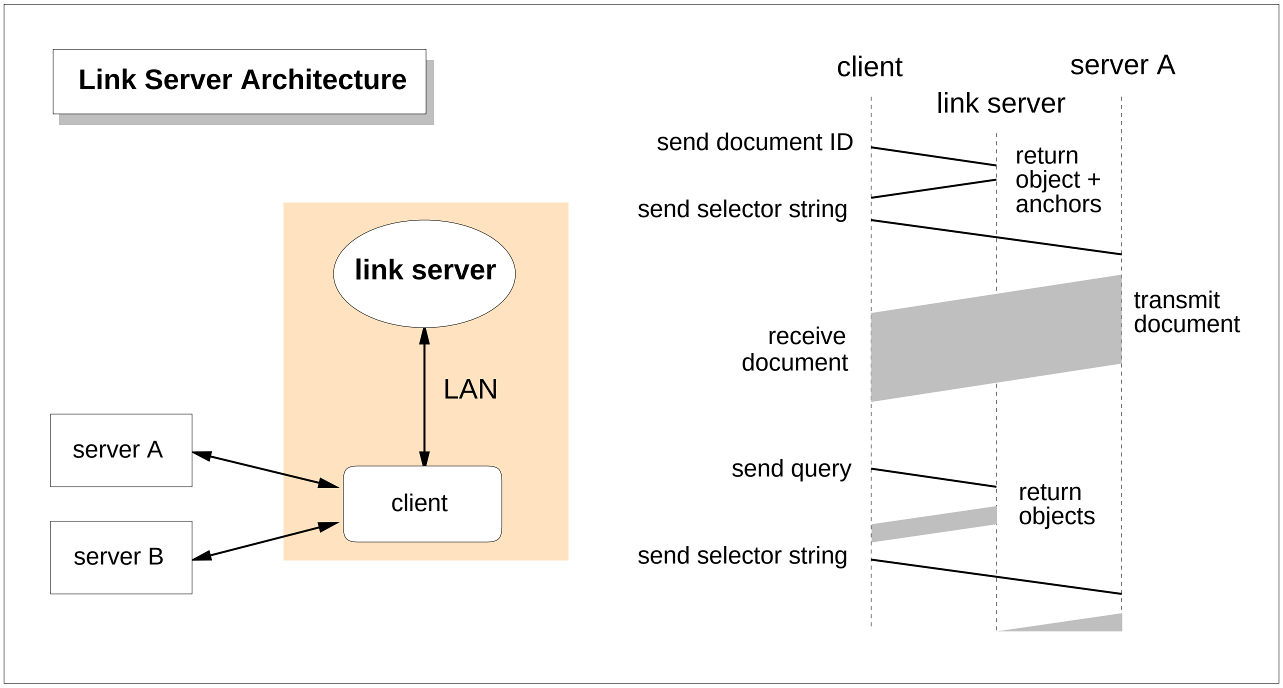


Figure 5: Link server protocol (see text for explanation)

The basic idea is to strictly separate links from documents. Links are stored in a so-called *link server*[4], while documents remain in document servers (for now). The link server is an 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 [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 serves the following purposes:

- It assigns object IDs to objects 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.

[4]It should be noted that the link server concept as well as bidirectional links are also implemented in the Intermedia[11] system.

- 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.

- It maps anchors to documents. In contrast to the simple system described before, Hyper-G documents do not directly contain link information. Instead, the link server (and only the link server) knows what the source anchors (i.e. the "hot spots") in the document are and transmits this information to the client together with the document object (see figure 5; top right). The client then uses the document object to retrieve the document (without links) from the document server, merges document and anchors, and displays them. Should the user click on an anchor, the client transmits the anchor ID (again, just a number – no indication of target document) to the link server, which in turn finds the corresponding destination document, returns it to the client, and the story continues.

  This allows to modify link information independently from documents, and enables private links (if the user has no permission to follow the link, it just isn't forwarded to the client).

- A centralized link store enables the system to trace links backwards, which is a precondition for automatic link maintenance (e.g. detection of links pointing to deleted documents; insertion of preliminary links to non-yet-existing documents and creating them automatically when the destination document is eventually inserted) and advanced user interfaces (such as graphical browsers).

- It allows complex queries over the whole database (e.g., "Give me documents with title containing *UNIX*, created by *fkappe* after *92/06/01*") or part of the database. As shown on the lower right side of figure 5, the client transmits the query to the link server, which answers with a number of objects (e.g., documents). The client software would then let the user choose from the list of hits, and eventually a document would be retrieved from a document server. As Hyper-G documents are organized in a hierarchy of so-called *collections*, search can be limited to a certain subset of the collection hierarchy.

- As a multi-user database, it also allows immediate notification of clients as a result of changes in the database relevant for that client, among other things...

In order to prepare the forthcoming sections, it will be necessary to take a look at the internals of the link server and see how the link server accomplishes this. Figure 6 shows an example of how links are represented in the database by means of a number of objects and relations between objects:

- The *Document* object holds information about a document, including document type, author, title, as well as access information (e.g., host, port, protocol). All objects are identified by a unique object ID – currently just 32-bit numbers – and can efficiently be searched for by object ID, title, author, etc. Care has been taken to store objects

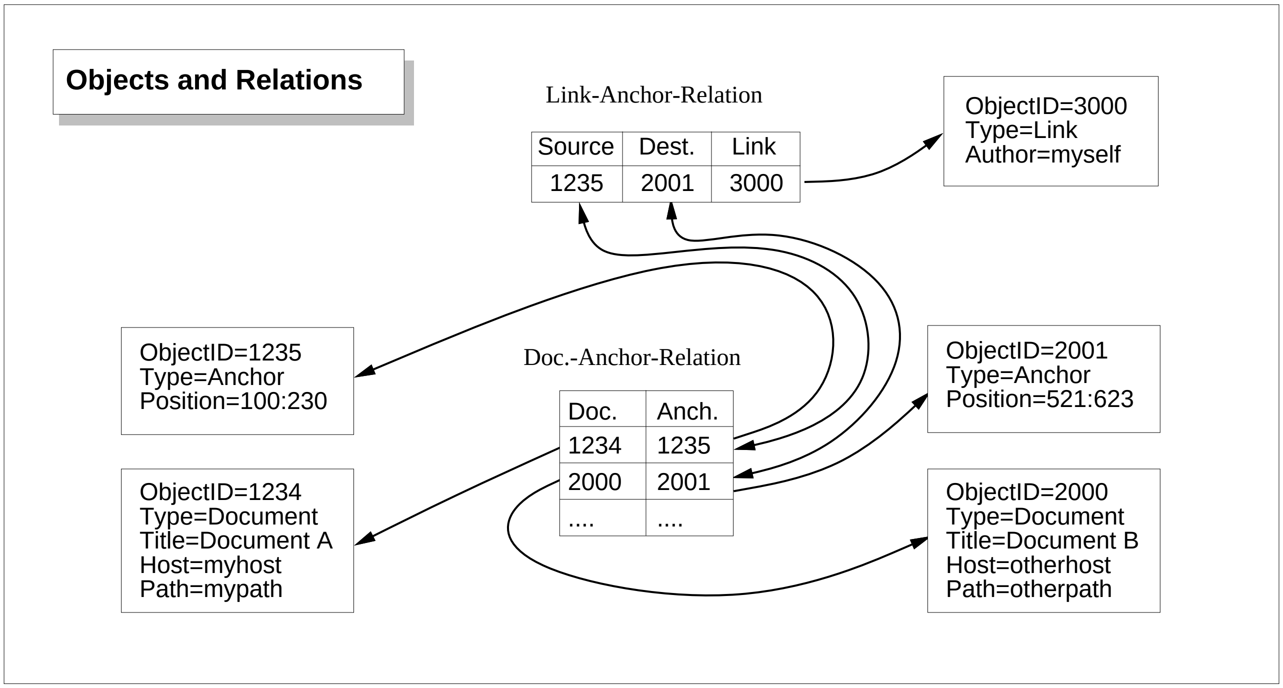


Figure 6: Some of the link server's internal objects and relations

in a flexible manner, so that the list of object attributes can be easily extended. Also, no references to other objects are made within objects. Instead, all such references are handled by relations.

- The *Anchor* object represents an anchor, i.e. a certain position within a document. Among other attributes such as author and access rights anchors keep a *Position* attribute that specifies the position of the anchor within a document. The format of the position attribute depends on the document type the anchor is attached to (e.g. text, image, sound...) and is ignored by the link server, but subject to interpretation by the client.

- The *Link* object is used to store certain attributes of the link (such as link type, author, creation date, access rights...) and is used to filter and to search for certain links (e.g. depending on link type).

- The *Document-Anchor Relation* specifies which anchors belong to which documents by means of their object ID. As it can be searched in either direction it may be used to find all the anchors of a given document, or to find the document a certain anchor belongs to.

- The *Link-Anchor Relation* joins two anchors (the *source* and *destination* anchor) as well as a link object to form a link. Sometimes (in fact in most cases) the destination anchor will be the whole document. In order to save some space and execution time, in this special case no destination anchor object exists and the destination anchor field of the link-anchor relation will directly store the destination document. Note that this relation defines which anchors serve as source and which as destination anchors. The anchors themselves may be used in either direction. As this relation may also be searched in any direction, links in Hyper-G are bidirectional (i.e. the link server may also find all objects that link to a certain document).

Let us now consider what happens when the client retrieves document "1234" to display it (looking at figures 5 and 6):

1. The client sends a request for document "1234" to the link server. The link server answers by returning document object "1234" as well as anchor objects attached to this document (found in the document-anchor relation), including anchor object "1235".

2. The client uses access information (*Host, Port, Protocol, Path*) contained within document object "1234" to retrieve and display the document, and the *Position* information contained within the anchor objects to highlight the anchors. Let us assume that the user now clicks on anchor "1235".

3. The client sends anchor ID "1235" in a request for the corresponding destination document. The link server finds "1235" as a source anchor in the link-anchor-relation, possibly checks the attached link object "3000" to see if the user is authorized to access that link, and uses the attached destination anchor "2001" to find the corresponding destination document "2000" in the document-anchor relation. As in step 1, the link server returns document object "2000" as well as the anchors attached to it, and the story continues with step 2...

As has been said before, the link server may also be used to query the database. In this case, the object database is searched for matching document objects, which are then sent to the client. The client will typically display a list of the matches and the user will select one. From then on, the sequence of events is as described above.

While the concept of link servers offers the advantages already described, it also suffers from a few shortcomings:

- Both the servers and the clients are more complex than the simple scheme described in the previous section.

- Obviously, there will be some traffic between the link server and the client, although only relatively short packets have to be transmitted. Therefore, we have to assume that both reside in the same LAN to achieve reasonable performance.

- There has to be exactly one link server, while the number of document servers and clients is potentially unlimited. However, document servers may also be not part of the LAN, but distributed all over the internet.

- If the document servers are far away (in "bandwidth metric"), there is still high long-distance traffic to retrieve documents and performance will be moderate.

As a conclusion, the system as presented by now is well adapted for use in LANs, but not a truly distributed system suited for the internet. In the following two sections we will show how this disadvantage can be eliminated.

6 Document Cache Servers

Let us first concentrate on the problem of slow access to documents over wide-area internet. A well-known technique routinely used under such circumstances is that of caching. Introduction of a document cache server leads us to the architecture shown in figure 7:

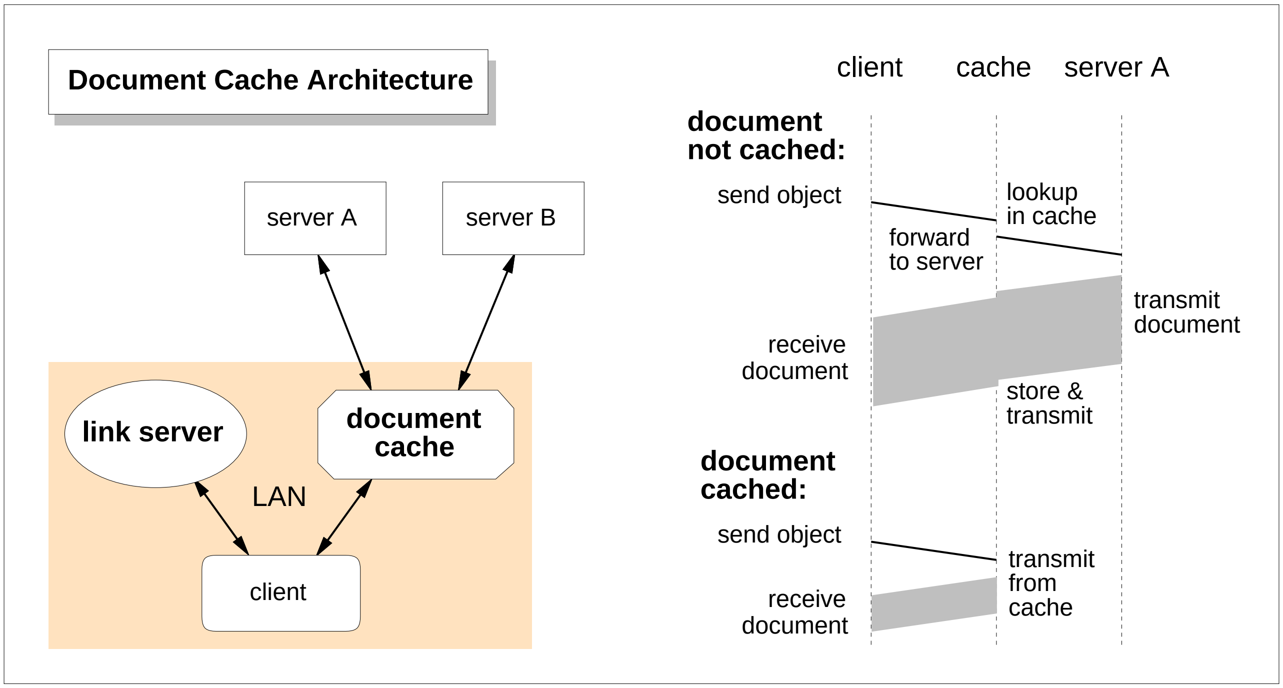


Figure 7: Adding a document cache server per LAN boosts performance

Clients do no longer connect directly to document servers. 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 stored the document (upper right of figure 7). 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 (lower right).

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 should remove 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.

There may be more than one cache server per LAN. It may even be reasonable to install cache servers on a per-user basis, as a certain user tends to remain within a certain subset of documents of the docuverse during the navigation process. Consequently, the cache server can also be built into the client.

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 hits 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).

However, with alien protocols the problems of cache consistency reappear. Therefore, it is suggested that document caches remove such documents early (e.g., after one day).

As a summary, introduction of cache servers speeds up retrieval of documents by clients, at least for documents that are retrieved very often – typically those near the user's entry points into the web and those of common interest to a group of users. It therefore makes sense to install at least one cache server per internet domain (compare figure 2). Because the problem of cache update is solved very elegantly by the link server, there is no problem with installing an unlimited number of cache servers all over the world. No process other the connecting client has to know of the existence of the document cache.

7 Distributing the Link Server

Our design as presented by now relies on exactly one link server for all of internet. While this guarantees database consistency, it may mean mediocre performance for the participants far away (in terms of bandwidth) from the link server. In order to solve this problem, we propose to distribute the link server over the whole network.

Experience with other distributed information systems (WorldWideWeb, Gopher, WAIS) suggests that the anticipated applications of such systems will consist of certain datasets that are prepared by certain information providers. Within such a dataset (for example, an encyclopedia, a user's manual, an enzyme database, a database of chemical compounds) there will be a rather large amount of connectivity (i.e. lots of links between documents) and a certain structure (i.e. membership relationships). However, connections to other material (especially that which is offered by other information providers) will probably be relatively sparse.

Because of this observation (loosely connected data sets) and the structure of the internet (see figure 2) of loosely connected LANs we propose the following architecture of a global internet-based hypermedia system (figure 8):

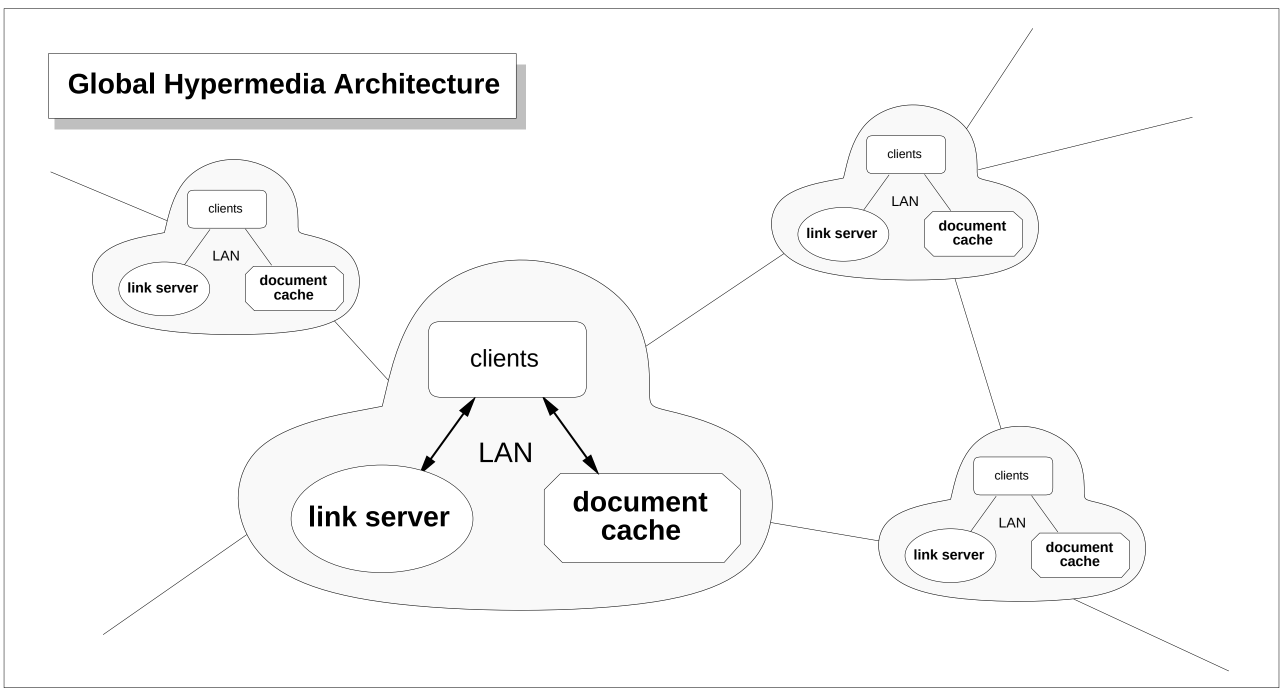


Figure 8: A global internet-based hypermedia architecture

For every information provider, there exists a link server as described in section 5 in the information provider's domain (LAN), and also a document cache server as described in section 6.The document cache server not only caches remote documents but also permanently stores local documents. In other words, it looks like a document server (like the ones labeled "server A" and "server B" in figure 7) for document caches of other LANs.

The link server manages the information provider's documents and links. As the information provider's domain will usually be the one most interested into its own information, it makes sense to keep access to this "local data" fast. Within its "local address space" the link server has full control of objects and relations. In particular, it controls the assignment of (local) object IDs.

Distributing the link server means distributing the link server's relations. In other words, there may be relations (e.g., in conjunction with links) that involve objects of different link servers. As a first step into this direction, we promote local object IDs to global object IDs in a global address space. Although more complex addressing rules have been proposed for similar systems, the simple scheme shown in figure 9 will suffice for our purpose.

`63 . . 48 47 . . 32`	`31 . . 16 15 . . 0`
`32-bit server ID`	`32-bit local ID`

Figure 9: A simple scheme of 64-bit global document IDs

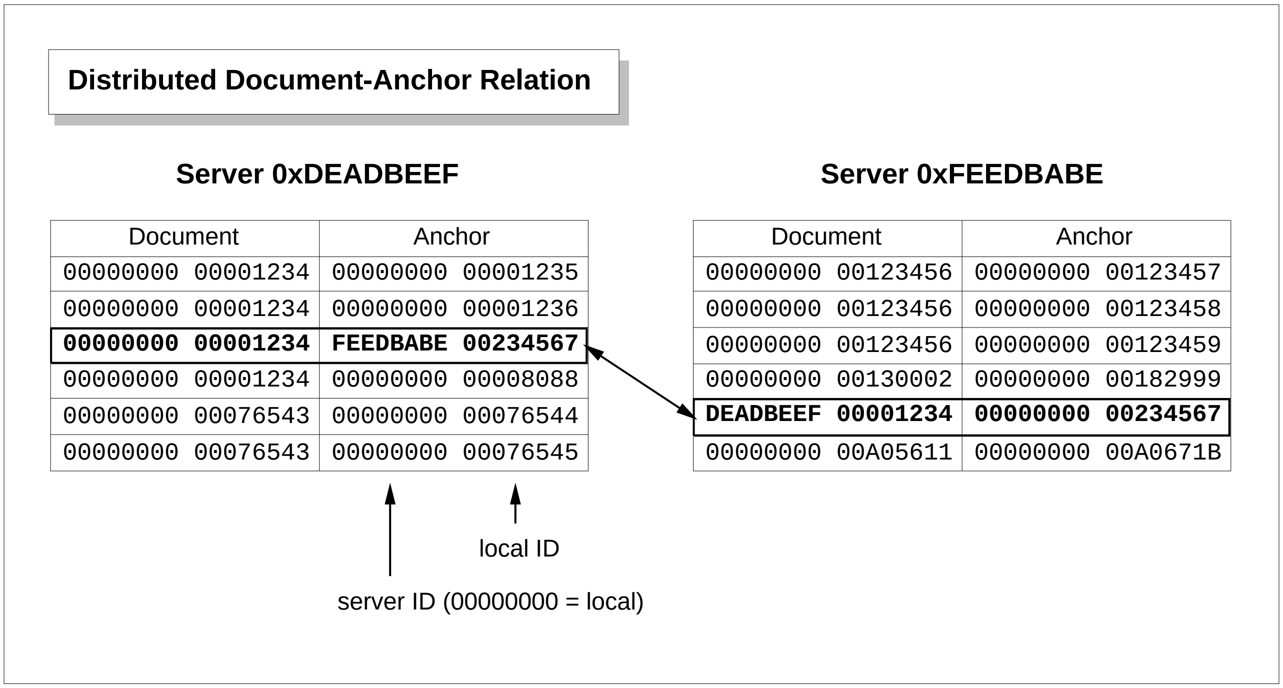
Basically, the 32-bit local ID is concatenated with another 32-bit server ID. While the server ID could be automatically calculated from the internet address, this would generate problems when data is moved between servers or internet addresses change. Therefore, we propose to register link server IDs with some registration authority to be defined. The simple 64-bit global object ID allows for up to 4 billion link servers with up to 4 billion objects each, and can be managed efficiently (e.g., compared) by state-of-the-art 64-bit processors.

We will now take a more detailed look at relations between local and remote objects. As a simple example, we will consider the case where a local anchor (source or destination) is attached to a remote document (so that we then could attach a local link to the local anchor and thus to the remote document; compare figure 6). The only relation that is affected in this example (figure 10) is the document-anchor relation.

For relations that involve only local objects (the server ID part of the global ID is set to zero for local objects), the link server behaves exactly as the Hyper-G link server described in section 5. If, however, one of the relation's objects is a remote object (server ID part not zero), the relation is duplicated in all servers affected.

In our example, the link server with server ID `0xDEADBEEF` stores a document object with ID `0x00001234`, with three local anchors attached (figure 10). In addition, the document-anchor relation lists a remote anchor of server `0xFEEDBABE` with ID `0x00234567` (in `0xFEEDBABE`'s name space). On the other side, the document-anchor relation of `0xFEEDBABE` refers to `0xDEADBEEF`'s object `0x00001234` as a remote document.

Some relations can be searched in either direction. For example, in the case of the document-anchor relation one could ask the questions "what document does anchor `0xFEEDBABE00234567` belong to?" or "what anchors are attached to document `0xDEADBEEF00001234`?". As a rule,



Server 0xDEADBEEF

Document	Anchor
00000000 00001234	00000000 00001235
00000000 00001234	00000000 00001236
00000000 00001234	**FEEDBABE 00234567**
00000000 00001234	00000000 00008088
00000000 00076543	00000000 00076544
00000000 00076543	00000000 00076545

Server 0xFEEDBABE

Document	Anchor
00000000 00123456	00000000 00123457
00000000 00123456	00000000 00123458
00000000 00123456	00000000 00123459
00000000 00130002	00000000 00182999
DEADBEEF 00001234	**00000000 00234567**
00000000 00A05611	00000000 00A0671B

Figure 10: Example relation between local and remote objects (document-anchor relation)

the question must always be directed to the link server that holds the “input” object. In our example, the first question has to be answered by server `0xFEEDBABE`, the second by `0xDEADBEEF`.

Relations can be designed in such a way that at most two servers are involved (i.e. at most one of the relation’s objects is a remote object), so that only one other server will have to be informed when such a relation is modified. Observe that in this design no information ever needs to be broadcast to a number of (or all) link servers. Therefore, consumed internet bandwidth does not depend on the number of servers (as opposed to other systems where identical copies of databases have to be maintained on a number of servers). Also, the point-to-point communication needed to modify remote relations can be implemented reasonably fast and reliable.

In many cases, when reference is made to a remote object, that object has to be retrieved from the remote server. To increase performance, remote objects could be cached in the local server (e.g. object `0xFEEDBABE00234567` could be stored in server `0xDEADBEEF` also). Similar to the document cache, database integrity is guaranteed be the link servers: Should the anchor be modified at a later time, the link server that owns it would create a new local ID for it, and modify all relations involved accordingly (i.e. equivalent to deleting the old entries and creating new ones). The old ID can no longer be accessed and would automatically disappear from the cache after some time (least-recently-used cache strategy). Again, no broadcasts are needed to inform the other servers.

One issue that has not yet been addressed is that of searching in a distributed system. Obviously, it would be a time-consuming task to perform all queries over the full database (i.e. the whole network of link servers). However, the collection approach of Hyper-G (restrict searches to a subset of the collection hierarchy) naturally scales up for the global system. In the global hypermedia system, collections usually do not span link server boundaries, so restriction of the search to a set of collections would also restrict it to a set of link servers.

Full text queries, on the other hand, require installation of a full text indexer (e.g. a WAIS server) per LAN that accesses the document cache server and indexes the text documents contained therein. The full text indexer would return a number of global document IDs in response to a query, which would be interpreted by the link server.

8 Summary

Global hypermedia systems have been on the minds of visionaries for decades now. We believe that the time has eventually come to really implement a massively distributed hypermedia system based on the internet using the client-server architecture.

The system we propose can be evolved out of the existing Hyper-G system in an evolutionary way. It features a distributed link server, makes efficient use of internet bandwidth by extensive caching, no broadcasts, and relatively small packets between link servers.

Because it is derived from the Hyper-G system, it inherits all the features of Hyper-G, including multimedia document types, collections, guided tours, advanced searching capabilities, bidirectional links, automatic link maintenance and generation, multilingual documents, access rights, multiple user interface metaphors, four user identification modes, etc. A detailed description of Hyper-G can be found in [14] and [15].

References

[1] ALBERTI B., ANKLESARIA F., LINDNER P., MCCAHILL M., and TORREY D.: "The Internet Gopher Protocol: A Distributed Document Search and Retrieval Protocol". March 1992. Anonymous ftp from boombox.micro.umn.edu in directory pub/gopher/gopher_protocol.

[2] BERNERS-LEE T., CAILLIAU R., GROFF J., and POLLERMANN B.: "WorldWideWeb: The Information Universe". *Electronic Networking: Research, Applications and Policy*, 1(2):52–58, Spring 1992.

[3] BUSH V.: "As We May Think". *The Atlantic Monthly*, 176(1):101–108, July 1945.

[4] BUSH V.: "As We May Think". In LAMBERT S. and ROPIEQUET S. (editors), *CD ROM – The New Papyrus*, pages 3–20, Microsoft Press, 1986.

[5] CARD S. K., MORGAN T. P., and NEWELL A.: *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, Hillsdale, N.J., London, 1983.

[6] CARSON G. S.: "Graphics, Networking, and Distributed Computing". In *Proc. Workshop on Graphics and Communications (ESPRIT Project 2463 - ARGOSI), Breuberg, Germany, Oct. 1990*, pages 39–47, Springer, 1990.

[7] CONKLIN J.: "Hypertext: An Introduction and Survey". *IEEE Computer*, 20(9):17–41, September 1987.

[8] DEUTSCH P.: "Resource Discovery in an Internet Environment – The Archie Approach". *Electronic Networking: Research, Applications and Policy*, 1(2):45–51, Spring 1992.

[9] ENGELBART D. C.: "A Conceptual Framework for the Augmentation of Man's Intellect". In HOWERTON P. D. and WEEKS D. C. (editors), *Vistas in Information Handling, Vol. 1*, pages 1–29, Spartan Books, Washington D.C., 1963.

[10] FEDIDA S.: "An Interactive Information Service for the General Public". In *Proc. European Computing Conference on Communication Networks*, pages 261–282, 1975.

[11] HAAN B. J., KAHN P., RILEY V. A., COOMBS J. H., and MEYROWITZ N. K.: "IRIS Hypermedia Services". *Communications of the ACM*, 35(1):36–51, January 1992.

[12] HALASZ F. G.: "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems". *Communications of the ACM*, 31(7):836–852, July 1988.

[13] KAHLE B., MORRIS H., DAVIS F., TIENE K., HART C., and PALMER R.: "Wide Area Information Servers: An Executive Information System for Unstructured Files". *Electronic Networking: Research, Applications and Policy*, 1(2):59–68, Spring 1992.

[14] Kappe F.: *Aspects of a Modern Multi-Media Information System.* PhD thesis, Technical University Graz, 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.

[15] Kappe F., Maurer H., and Sherbakov N.: *Hyper-G – A Universal Hypermedia System.* IIG Report 333, IIG, Graz University of Technology, Austria, March 1992. Also available by anonymous ftp from iicm.tu-graz.ac.at in directory pub/Hyper-G/doc.

[16] Lottor M.: "RFC 1296: Internet Growth (1981-1991)". January 1992. This report, as well as up-to-date data is available by anonymous ftp from ftp.nisc.sri.com in directory pub/zone.

[17] Marchionini G. and Shneiderman B.: "Finding Facts vs. Browsing Knowledge". *IEEE Computer*, 21(1):70–80, January 1988.

[18] Meyrowitz N. K.: "InterMedia: The Architecture and Construction of an Object Oriented Hypertext/Hypermedia System and Applications Framework". In *Proc. OOPSLA 86, Portland, Oregon*, pages 186–201, September/October 1986.

[19] Nelson T. H.: "A File Structure for the Complex, the Changing, and the Indeterminate". In *Proc.* 20^{th} *ACM National Conference*, pages 84–100, 1965.

[20] Nelson T. H.: *Literary Machines, Ed. 87.1. (1987).* The Distributors, South Bend, IN, 1981.

[21] Neumann B. C.: "Prospero: A Tool for Organizing Internet Resources". *Electronic Networking: Research, Applications and Policy*, 1(2):30–37, Spring 1992.

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

[23] Shneiderman B.: "User Interface Design for the Hyperties Electronic Encyclopedia". In *Proc. of Hypertext '87, TR88-013*, pages 199–205, University of North Carolina, Dept. of Computer Science, March 1988.

[24] Stein R. M.: "Browsing through Terabytes – Wide-area information servers open a new frontier in personal and corporate information services". *Byte*, 16(5):157–164, May 1991.

[25] Tomek I., Khan S., Muldner T., Nassar M., Novak G., and Proszynski P.: "Hypermedia – Introduction and Survey". *Journal of Micro Computer Applications*, 14(2):63–100, 1991.

[26] Yankelovich N., Haan B. J., Meyrowitz N. K., and Drucker S. M.: "Intermedia: The Concept and the Construction of a Seamless Information Environment". *IEEE Computer*, 21(1):81–96, January 1988.