Supporting The Web: A Distributed Hyperlink Database System

James E. Pitkow & R. Kipp Jones
Graphics, Visualization, & Usability (GVU) Center
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
pitkow@cc.gatech.edu kjones@harbinger.net

Abstract

In our last paper [Pitkow & Jones 1995], we presented an integrated scheme for an intelligent publishing environment that included a locally maintained hyperlink database. This paper takes our previous work full cycle by extending the scope of the hyperlink database to include the entire Web. While the notion of hyperlink databases has been around since the beginnings of hypertext, the Web provides the opportunity to experiment with the largest open distributed hypertext system. The addition of hyperlink databases to the Web infrastructure positively impacts several areas including: referential integrity, link maintenance, navigation and visualization. This paper presents an architecture and migration path for the deployment of a scalable hyperlink database server called Atlas. Atlas is designed to be scalable, autonomous, and weakly consistent. After introducing the concept and utility of link databases, this paper discusses the Atlas architecture and functionality. We conclude with a discussion of subscriber and publisher policies that exploit the underlying hyperlink infrastructure and intelligent publishing environments.

Keywords

World Wide Web, Internet infrastructure, distributed hyperlink databases, protocol extensions, consistency, integrity.

Introduction

At one point or another, nearly all Web users have clicked upon a hyperlink, only to be presented with a subsequent page containing the message "Error 404. Not found—file doesn't exist." But what exactly has happened here? In most cases, the author of the document containing the hyperlink, incorporated the hyperlink to facilitate the reader's comprehension of the document's content[1]. Regardless of the content and form of the intended anchor (glossary, different media representation, reference material, etc.), the author intentionally included the hyperlink—the anchor extends the content. In the above scenario, in which the user is not able to access the anchor, the full impact and benefits of the provided information can *never* be achieved. This is sub-optimal from both the user's and author's perspective. This is a direct and immediate problem facing all users of the World Wide Web.

How does one go about solving this problem? An initial solution to the problem of "dangling links," more formally known as referential integrity, was developed by Pitkow [Pitkow 1993], called the *html_analyzer*. This simplistic and out-dated program attempted to systematically determine the integrity of external hyperlinks for a given website as well as identify other constraint-based violations. This work was later augmented for use in a distributed environment by Roy Fielding's *Mom Spider* [Fielding 1994], and EIT's *link analyzer* software [McGuire 1994]. While these packages are still helpful in identifying and removing broken links, these solutions have several shortcomings.

Specifically, none of the above packages were designed to be integrated into authoring software and as such require post-hoc execution. Additionally, these programs are executed based on some arbitrary notion of when the state of the Web has sufficiently changed to warrant reexamination, not on a per change basis. Thus, authors and content providers typically run these programs nightly, or on a weekly basis, to maintain some level of local hyperlink consistency[2].

---

1. For the purposes of this paper, we shall use the term *source* to refer to the document containing the hyperlink, and *anchor* to refer to the document pointed to by the hyperlink. Additionally, link and hyperlink will be used interchangeably.

2. Consistency is defined as a characteristic of the state of the Web, where all hyperlinks point to existing objects.

Supporting The Web: A Distributed Hyperlink Database System
Fifth International World Wide Web Conference. Paris, France. May 6-11, 1996.
GVU Technnical Report: GIT-GVU-96-09

J. Pitkow & R. K. Jones

1

While certain content providers will not be concerned about their site's consistency, many will as it increases the utility of the provided content. By current estimates [Netcraft 1996], over 100,000 Web sites exist. Hypothetically, if each site contains 100 external links, and if each site decides to employ one of the above solutions on a nightly basis, then 10 million requests as to the existence of the external anchors need to be issued. This causes a problem of scale, requiring frivolous and, we posit, unnecessary, network traffic and external server load, often across slow network connections.

The issue of referential integrity has other profound and substantial implications on the use and usability of the Web. Traditional closed hypertext systems typically provide the users with navigational aids that enable the users to determine their location in the hypertext. This is often accomplished by utilizing visualizations of the structural connectivity within the hypertext as node link diagrams [Joyce 1991]. While current Web users do not report the inability to visualize locality as a primary problem with using the Web [Pitkow & Kehoe 1995], more empirical evaluations as to the benefits of visualizations are indeed warranted.

Clearly, the Web is not a closed hypertext system. Initial efforts towards providing end-user visualization [Domel 1994; Mukherjea 1995] typically extract the hyperlinks contained within a document and proceed to retrieve the contained anchors in a breadth-first manner. Once these anchors are retrieved, their contents are parsed and the process is repeated until the desired depth for the visualization is achieved. This "gather/extract" method of displaying hyperlink connectivity processes more information than necessary, since the user has not yet requested the retrieved content. Considerable network bandwidth is also unnecessarily consumed.

For example, imagine that the user is attempting to visualize NCSA's "What's New Page" and all the referenced anchors. By the above gather/extract method, if a visualization were to attempt to display two levels of connectivity, the content for hundreds of sites would need to be downloaded and processed. Not only does this falsely inflate hit counts, but it suffers from the problems of increased network traffic and inability to scale.

Besides facilitating end-user navigation and a global sense of connectivity, visualizations are useful for Webmasters and content providers, as it enables an entire site to be visually managed. While some commercial systems are beginning to appear that provide visual maintenance, these systems are limited to the local website and
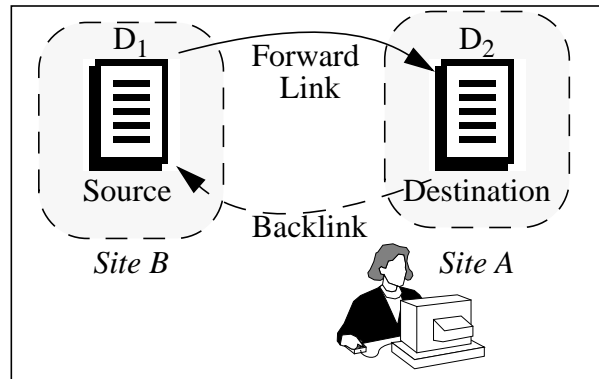
FIGURE 1. The basic bidirectional nature of Web based hyperlinks. Notice the *backlink* is not explicitly maintained in today's Web

rely upon the undesirable hyperlink integrity and consistency solutions mentioned above.

Many content providers also desire to know which Web pages point to their resource. Currently, this information can be painfully gathered via logging of the "Referrer" HTTP header, but only if this header was sent by the client. With distributed hyperlink databases, not only can one visualize forward links, but *backlinks* as well, i.e, which pages link to your pages. Stated differently, given an infrastructure which provides a means for communicating information between servers whose resources are linked, the source of the external links to local resources can be known. Backlinks typically provide as much information as the resource itself, since links are usually semantically related.

For example, suppose the user is visiting a page on computer graphics at *Site A*, as in Figure 1. The user has no navigational knowledge of the forward link from the source document on *Site B* to the destination document on *Site A*, though the sites contain related material. We conjecture that for any given Web page, a user should have the ability to view such related pages. In a bidirectional hyperlink system, this information is readily accessible, but the Web uses a unidirectional hyperlink model, which requires additional facilities to ascertain this information.

To summarize, existing attempts at managing referential integrity and visualizations of the Web leave room for improvement. Fortunately, the Web provides a unique and fertile testbed for experimentation with alternative solutions, one of which is our prototype called Atlas. The remainder of this paper will discuss Atlas, which attempts to solve the problem of referential integrity and facilitates efficient and complete visualizations.

Supporting The Web: A Distributed Hyperlink Database System

J. Pitkow & R. K. Jones

Fifth International World Wide Web Conference. Paris, France. May 6-11, 1996.

2

GVU Technnical Report: GIT-GVU-96-09

Atlas

Atlas is a distributed hyperlink database system that works in conjunction with traditional Web servers. Atlas includes a server-to-server protocol for the autonomous maintenance of referential integrity as well as a client to server protocol to handle queries regarding the topology and connectivity of a site. The underlying architecture of Atlas and the associated API are designed to enable replacement with other technologies as appropriate. The protocol is a minimal HTTP-like protocol called ATLASP, that can be extended to include advanced security and authentication. Additionally, a high degree of distribution transparency is maintained by replicating necessary information to the relevant servers.

The following design decisions were used throughout the design of the Atlas architecture:

1. Completely autonomous Atlas servers;
2. Weak referential consistency;
3. Atlas servers can instigate communication.

The motivation for completely autonomous servers stems from the unreliable nature of the Internet services and the lifecycle of Internet software. By autonomous, we mean that server can operate correctly in the absence of other servers, i.e., in an isolated environment. For example, just about all of us have experienced the inability to connect to a desired Web server. Whether a result of a local or remote failure, the server is unreachable, and therefore can not process requests. Any robust Internet service, e.g., mail delivery via SMTP, must have mechanisms built in to handle such failures. Because Atlas is an autonomous service, it does not rely upon remote services for continuous operations.

Deployment of new technologies on the Internet have the problem of gaining a critical number of users—too few and the technology is not widespread enough to be useful. Too many, and the software becomes difficult to modify. Initially, most software suffers from too few users and generally requires a sufficient core of users to become successful. Provided that the software is useful, this core set of users can continue to grow while modifications and adjustments are made relatively easily and inexpensively, e.g., the Apache group.

Atlas was designed to be a useful Internet service even if it never reaches critical installation mass. At the local Web site level, the hyperlink database embedded within Atlas adds many value-added services like local maintenance, visualization, and referential integrity. Still, the full potential of Atlas is *only* realized when the server-to-server communication takes place amongst many sites.

Clearly, given the size and indeterminable set of Web servers and hyperlinked media, the Web will *never* reach a fully consistent state. Furthermore, even with a complete directory of all existing Web servers and the pages they contain, the Web can not reach full consistency, due to the dynamic and distributed nature of the Web. The next best alternative then, is to attempt to achieve some sort of weak consistency between anchors and their referring objects. The notion of weak consistency is that while all hyperlinks may not currently point to existing objects, they will eventually converge towards a consistent state.

A primary directive in our effort has been to create a system which helps bring the state of the Web towards weak consistency. As a loosely coupled set of distributed servers, this convergence towards consistency requires *anti-entropy* processes. For the purposes of this paper, an anti-entropy process is a process that extends the global knowledge of the system by sharing local knowledge. Atlas' anti-entropy process periodically verifies the consistency of local information and then propagates this information to those servers which need to know. This requires the Atlas servers to be able to instigate communication. For example in Figure 1, suppose Site B removes the forward link to Site A, Atlas will communicate this modification information to Site A. For all changes to all forward links on Site B, the Atlas server on Site B will notify all affected servers of these changes. This notification process occurs even if certain affected servers are currently unavailable.

Architecture

"Atlas" refers to the hyperlink database, the Atlas server, the ATLASP protocol, and supporting utilities. Currently, the Atlas server runs in standalone mode independent, but tightly coupled to an existing Web server. To increase the interoperability and to facilitate experimentation with different components of the system, each functional unit was modularized and an API defined. The hyperlink database used for the initial implementation is Mini SQL (mSQL) [Hughes 1995], though any database can be plugged in with the inclusion of a set of backend routines. The majority of the code was written in C for UNIX platforms. Currently, we are in the processes of incorporating components of
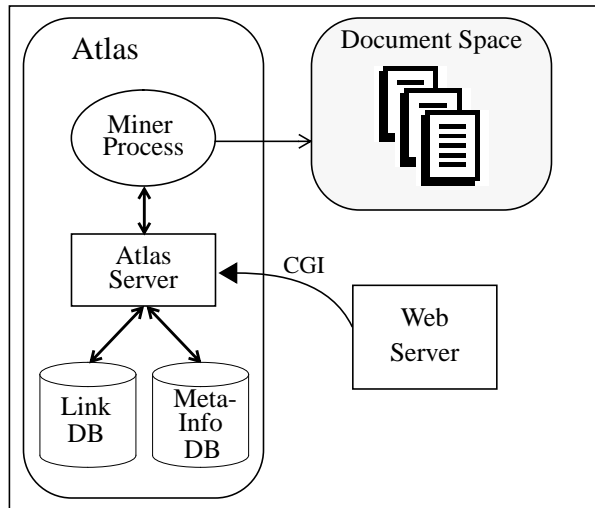
FIGURE 2. General Atlas architecture depicting local repository processes only.

Atlas into the repository architecture being developed by NCSA.

Figure 2 depicts the overall architecture of Atlas. The left side of the figure contains some of the core repository elements. In an ideal publishing situation, the authoring tools used to create content for the repository are an integral part of the architecture, where the meta-information for the newly formed content is automatically extracted and stored, hyperlinks are extracted and stored, and the integrity of the content, meta-information and hyperlinks is verified [Pitkow & Jones 1995]. Atlas was designed to perform updates and notifications in real-time, as part of an integrated, front-end publishing environment. In the absence of such environment, Atlas performs these functions on a configurable, periodic basis. Thus, we are guaranteed that within a certain time frame, the meta-information and hyperlink databases will be locally consistent.

Initial population of the meta-information and hyperlink databases are handled using a 'miner' which examines the local content. The miner walks the local filesystem, which permits detection of retrievable files that may not be linked in from the Web server's root document. The filesystem provides much of an object's meta-information, e.g., author, last modification time, file size, etc. Additional meta-information is gathered when the content is an HTML object, by looking for specific tags like title and headers. Hyperlinks are extracted for inclusion into the link database from HTML objects as well. More extensive exploitation of content specific meta-information can be extracted with tools like Essence [Hardy & Schwartz, 1994] and structured using the guidelines set

forth by the "Dublin Dozen" [Weibel 1995], though the current Atlas prototype does not.

Additional hooks are in place to allow population of hyperlinks by examining the inflow of HTTP "Referrer" field headers. This ability to use outside information to populate the database greatly adds to the usefulness of the Atlas system initially, but will become less important as the saturation of Atlas servers increases.

Link Representation

Traditional Dexter-based hypermedia systems have an explicit notion of links and anchors as separate objects. Specifically, anchors are defined as the "glue" connecting network structures to the contents of particular components and links are used to connect two or more anchors [Halasz & Schwartz 1990]. In systems like Hyper-G [Kappe et al. 1992], the separation of links and anchors from the objects enables the addition of links to arbitrary media types such as MPEG, PostScript, and 3D graphical models. While this nominally requires data fusion by the viewer, it nicely separates the function from content.

Several links types are identified within the Hyper-G model [Kappe 1994]. Links are classified as either:

- *core links* which connect objects on a local server, or
- *surface links* which cross server boundaries.

Updates and inserts of core links do not require external communication as they only affect the local subsystem. Updates to surface links however require notification to the server connected by the link. These servers that share links are considered to be related.

Atlas supports link types as currently implemented on the Web in that links are the relationship between a source and a destination document. Extending current practice, Atlas defines an additional set of attributes, or meta-information, which include the following[3]:

- **Source URL**: the document from which the link originates;
- **Destination URL**: the destination document (anchor) of the link.

---

3. It is interesting to observe that with correct storage of link and meta-information, the state of the Web at any time can be recreated for archival as well as historical analysis.

- **Link ID**: a function of the local server name and a unique identifier within the local link name space. The server which contains the source of the link is responsible for proper naming of the link, and the link ID will be replicated on the destination server;

- **Creator**: the author of the link;

- **Creation Key**: currently unused, this field was added to authenticate the link's origin for enhanced security capabilities;

- **Creation Time**: the time the link was created;

- **Update Time**: the link's last modified time;

- **Permission**s: contains coding of who can modify the link. Currently defaults to world readable and owner server write permissions.

- **Label**: defines a string which can be used by visualization software to help provide hints to the user as to the purpose and use of a particular link. For simplicity, this can be thought of as 'link type.'

- **Locality**: specifies whether the link is a surface or core link.

As mentioned previously, the *miner* may not be able to determine all of the above attributes completely. The set of attributes is extendable and is expected to change as the complexity of the Web evolves. It does however provide a starting point for experimentation. Note that a surface link's meta-information is replicated on both the source and the destination link database, thus providing a high degree of distribution transparency.

Database

The underlying database is mSQL, though the selection of a particular database is not important to the overall functionality of Atlas. mSQL supports a subset of SQL and is freely distributable to the Internet community. It was also our intent to not limit the range of deployment of Atlas by relying on a particular brand of potentially proprietary database technology. The database used for link information can also store object meta-information and other system related data, as with NCSA's repository architecture, though this is not implemented in our prototype.

An API exists to facilitate the "dropping in" of other databases. This also allows experimentation and comparison amongst different flavors of databases. The database API supports the six following functions:

- **Insert**: add a link to the database;

- **Delete**: remove a link from the database;

- **Update**: update a link in the database;

- **Into**: return source URLs for all links pointing a destination;

- **Outof**: return destination URLs for all links emanating from a source;

- **Query**: enables link information to be retrieved from an SQL query;

Insert, Delete, and Update all have the capability of modifying the database. Into, Outof, and Query return a list of links meeting the criteria. Support for other DBMS's requires only the creation of a module that understands these six routines.

Atlas Server

The Atlas server consists of a a stand-alone daemon which listens for connections via either a TCP socket for remote communication or a UNIX socket for local communication. This provides multiple interfaces to perform link maintenance and execute queries for local and distributed processes. This enables the *miner* to periodically communicate changes in the local information space to Atlas. The Atlas communication also enables visual site maintenance tools to extract and modify the current state of the site's meta-information and connectivity information.

ATLASP

The protocol spoken by Atlas servers is ATLASP and sits on top of the reliable TCP/IP transport layer. ATLASP is connection oriented which allows for a number of commands to be processed during one connection (also referred to as batch mode processing). Batch mode commands are more desirable over single command transactions for the following reasons:

- All messages from one sever to another can be handled at once, incurring a single TCP/IP setup/tear down & slow start cost.

- Since real-time publishing environment software is not prevalent, updates are determined in batch mode via the *miner*. Thus breaking each change into a single command is inefficient.

- Batch mode processing can be scheduled to occur during off-peak CPU and network loads.

While batch mode processing contains more transactions, the entire communication is designed and expected to be brief. The results and error messages are returned to the initiator of the connection using an encoding similar to HTTP. This implies that the instiga-

Supporting The Web: A Distributed Hyperlink Database System

Fifth International World Wide Web Conference. Paris, France. May 6-11, 1996.

GVU Technnical Report: GIT-GVU-96-09

J. Pitkow & R. K. Jones

5

tor must buffer messages until a return acknowledgment has been received. The use of the buffer will increase reliability by allowing for the repeating of messages which have not been acknowledged on a periodic basis. This property also facilitates a verification mechanism by which the receiver of an update message can verify the intended update is legitimate. Details of this mechanism are discussed below.

ATLASP is based upon a minimal core set of commands and controlling constraints. The required set and complete specification of ATLASP commands are presented in [Jones 1996], though we will briefly overview the look and feel of the protocol. These commands are similar in naming and function to those previously specified for the API.

ATLASP permits varying degrees of secure server-to-server communication for automatic link maintenance with minimal network impact. Atlas accomplishes this with the LINK, UNLINK, UPDATE, and VERIFY methods. A small set of optional commands and constraints are supported which allow users to modify security information. The client-to-server communication has requirements for small light-weight transactions, which are supported by the INTO, OUTOF, EXISTS, and QUERY methods.

Table 1 presents examples of various commands and their associated responses. Each Atlas message is similar in syntax to the LINK command. Note that the protocol is quite similar to HTTP, though some of the capabilities of HTTP are not necessary for Atlas. Many of the similarities stem from the use of human readable, easily parsed content, using standard CRLF delimiters and *name:value* pairs for much of the information. A method for negotiation is included as well as the capability to maintain a connection for multiple transactions. In addition, a content/encryption negotiation process is specified in ATLASP, and is currently being implemented.

Upon receipt of the LINK command, the Atlas server parses the message and places the link attributes into the hyperlink database. Likewise, when an UPDATE message is received, Atlas extracts the new link information and issues the corresponding Update call via the API to the database. The VERIFY command is used to acknowledge the completion of a transaction (discussed below). In the scenario where an Atlas server receives a DELETE message regarding the removal of a hyperlink to a local object, the Atlas server removes the surface hyperlink from the database, but leaves the object meta-information intact. Using Figure 1, this scenario trans-

lates to the forward link from D_1 on Server B being removed. Server B sends a DELETE message to Server A, which removes the link for its local database.

TABLE 1.

LINK Command:
ATLASP/0.5 **Content-Length**:185 **LINK** **BeginLink** **LinkID**:atlas://www.yahoo.com/2812 **SrcURI**:http://www.yahoo.com/Statistics/ **DestURI**:http://www.cc.gatech.edu/gvu/user_surveys/   [etc.] **Locality**:Surface **EndLink**
Typical response to LINK Command:
ATLASP/0.5 **Content-Length**:64 202 Message Accepted
INTO Query:
ATLASP/0.5 **Content-Length**:214 **INTO** **BeginLink** **DestURI**:http://www.cc.gatech.edu/CoC.html **Locality:**Any **EndLink**
Typical response to INTO Query:

ATLASP/0.5
Content-Length:302
202 Message Follows
BeginLink
LinkID:atlas://www.cc.gatech.edu/2057
SrcURI:http://www.cc.gatech.edu/people/index.html
  [etc.]
Locality:Core
EndLink
BeginLink
LinkID:atlas://faser.cs.olemiss.edu/39283
SrcURI:http://faser.cs.olemiss.edu/jordan/
  [etc.]
Locality:Surface
EndLink

The other DELETE situation occurs when the D_2 on Server A is removed. Server A issues a DELETE message to Server B along with all other servers with hyperlinks to D_2. Upon receipt of this message, Server B automatically removes the hyperlink for the its local

database and optionally removes the reference to D_2 from D_1. Notification mechanisms similar to those mentioned in our earlier work can be invoked at this point, i.e., email can be sent to the author of the source object.

For external server communication of updates, the protocol allows for periodic asynchronous server-to-server communications. These mechanisms propagate messages to the appropriate servers relaying all information that has changed since the last communication (see Figure 3). Updates which have not been acknowledged by other servers are queued and re-issued using an exponential back-off time table.
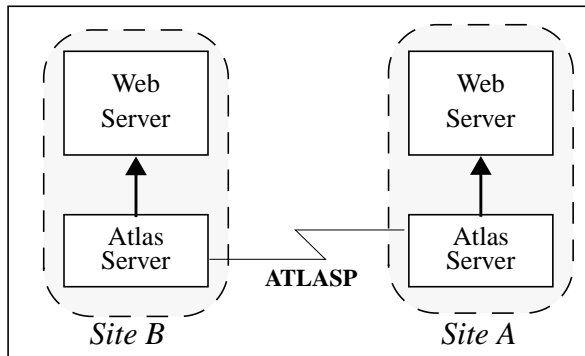
FIGURE 3. Atlas servers communicate amongst related servers. Site A and B are related by at least one link in this diagram.

Scalability of Notification

The binary hyperlink relationship that exists between source and destination objects warrants the use of *point-to-point* communication of updates. Alternative communication models, like the proposed *p-flood* algorithm [Kappe 1995], typically provide an infrastructure in which *all* servers in the *entire* system receive *all* notifications. An underlying assumption in the p-flood model is that all servers are aware of all other servers. This, in affect, causes a flat name space to be imposed upon the entire systems. While this is certainly possible in small scale hypertext systems, since no implementation of p-flood currently exists, it remains to be proven that such an architecture can scale to the size and extremely volatile nature Internet communications and services.

Initial simulations of the p-flood algorithm conducted in the Spring of 1995 [Kappe 1995] show estimated network caused by the link maintenance to be in the range of 0.005% of the total Internet traffic. These simulation results were based on an initial value of 1,000 servers. The network traffic is a function of the number of servers, the density of surface links, and the number of

objects in the system. The use of percent of total Internet traffic as an empirical metric with which to gauge the efficiency of a distributed hyperlink system, we believe, is misleading. A more meaningful measure is to determine the amount of information the system uses and compare this to the absolute minimal amount of information necessary to communicate all updates (the lower bound). The fundamental difference in these metrics is that the former measures the cost to the current infrastructure while the latter measures absolute efficiency.

Interestingly, in the year since the simulation was conducted the number of servers has increased two orders of magnitudes to over 100,000 servers [Netcraft 1996], containing over 16 million objects. We know of no existing server registration service that can manage such a large flat name space in the heterogenous, distributed, environment of the Internet. Existing registration services like DNS are hierarchical in nature. The reliance upon a registration service for distributed link maintenance only adds to the complexity and subtracts from the widespread integration of the solution. Furthermore, both the number of servers and documents are projected to continue to grow at an alarming rate, which further decreases the likelihood of p-flood to scale gracefully.

Atlas, while bound by the same situation, does not rely on notifying servers other than those with a "need to know." No global flat names space needs to be maintained. Atlas servers do not receive notification of changes to unrelated servers. For example, the Georgia Tech server does not need to know that Company A placed a link to Company B and will not be notified. With point-to-point communication, as the number of servers grows, the percentage of servers with a "need to know" will tend to decrease on a per server basis. In tandem with the batch capabilities of Atlas, point-to-point communication provides a highly scalable infrastructure for link maintenance. Part of the Atlas research agenda includes periodic monitoring of the effect of this communication model has on Web servers and the Internet.

Security

Many protocols which support the Internet's infrastructure lack sufficient security features and are easily taken advantage of by unscrupulous users. Security, even on the level of infrastructure and support, should be recognized as an integral part of network based protocols. Yet a distinction needs to be made between the security requirements for infrastructure activity and those for the transmission of personal objects. Typically, the cost of having an intruder discover that a link exists between

Supporting The Web: A Distributed Hyperlink Database System

Fifth International World Wide Web Conference. Paris, France. May 6-11, 1996.

GVU Technnical Report: GIT-GVU-96-09

J. Pitkow & R. K. Jones

7

two objects is substantially less than having an intruder discover the contents of the actual objects. Atlas was developed with this in mind and its current implementation supports a subset of the security capabilities of the protocol.

The first level of defense within ATLASP is the use of a call-back feature for all server-to-server update requests. These requests include inserts, deletes, and updates. It is assumed that any update request is coming from a server which also supports ATLASP. The second level of defense is the use of 40 bit public key encryption technologies. The decision to use only 40 bit keys enables non-restricted distribution of executable Atlas servers, while providing sufficient protection against traffic analysis and intruders. Larger keys can be used for enhanced protection. Thus, private Webs can be constructed utilizing Atlas and have a reasonable chance of remaining private. More importantly, the integrity of the structural information contained within each Atlas server is preserved. This second level of security is currently under development.

Client Communications

As initially motivated, the benefits of Atlas are twofold: link maintenance and providing client side navigation and visualization capabilities. Atlas supplies the efficient fuel for visualizations in that the expensive gather/extract loop is bypassed. We envision the growth of visualization software to occur in the browser, as helper applications, and in authoring software. ATLASP affords INTO and OUTOF queries. These messages enable "What objects point to this URL" and "What objects does this URL point to?" to be asked (respectively) of any Atlas server. This information can clarify the user's sense of location and opportunities by providing a visual representation of the information space. As a consequence, the "lost in hyperspace" problem often
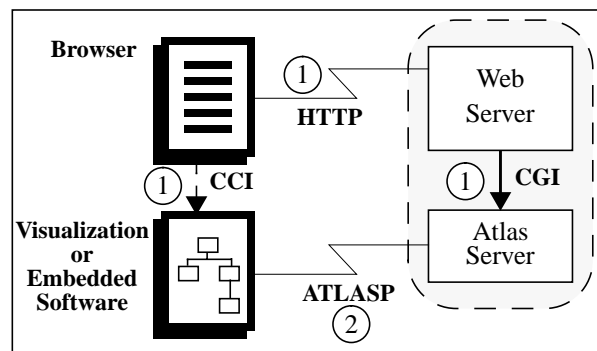
FIGURE 4. Two methods of client-server communication via HTTP (1) or ATLASP (2).

encountered and thought to constrain complex navigation, can be minimized using Atlas.

Figure 4 depicts two mechanisms for client communications. In the preferred method, the client visualization software talks ATLASP directly to an Atlas server. The other method uses HTTP and invokes a CGI process which provides the gateway interface to the Atlas server. Non ATLASP visualization software can immediately leverage off of Atlas services. An extension to HTTP could be provided via the newly proposed Protocol Extension Protocol [Khare 1995] to allow a tighter integration of Atlas and Web servers.

Additionally, a general query capability is built into both the protocol and the server. Queries of the form "Give me all of the documents with links to a server in Australia" are readily answerable. The queries are expressed in SQL, which opens the door for a wider range of uses and functionality than initially anticipated. To further expand the ability of other programs to access the information contained in Atlas, a Java applet and a client-side library are being developed.

Publishers, Subscribers, and Policies

Part of the beauty of the Web is that it minimizes by orders of magnitude the costs involved with large scale and individual publishing while expanding by orders of magnitude the potential audience. Authors have the ability to incorporate content developed by other authors by simply placing a link to the supporting content. The potential synergy that this property creates in the Web must not be understated—the impact is indeed revolutionary.

If we assume for a moment the widespread use of distributed hyperlink database technologies, certain interesting abilities emerge that have implications on generalized use of the Web as well as more controlled and structured environments like digital libraries and corporate intranets. The emergent properties center around 1) the initial act of linking to existing content and 2) when the linked content is deleted. Digital libraries and corporate intranets are a prime examples.

Clearly, certain material may not be intended to be reused by other authors in other contexts, yet this constraint should not necessarily forgo the publishing of the material on the Web. Since we are assuming an infrastructure where notifications of hyperlinks are sent to the server which contains the destination document, pol-

Supporting The Web: A Distributed Hyperlink Database System

Fifth International World Wide Web Conference. Paris, France. May 6-11, 1996.

GVU Technnical Report: GIT-GVU-96-09

J. Pitkow & R. K. Jones

8

icies can be implemented to control appropriate linking. In this model, the *publisher* initially creates the content (C) with authoring software. The software extracts meta-information and hyperlinks and presents this to the publisher for verification. It is during the verification process that the publisher can select the desired policy for others who may wish to link to the content. Once the publisher selects the policy, it is entered into the repository's meta-information database.

After a while, another publisher sees the content (C) and concludes that the content would be useful in an existing document (D). Using local authoring tools, the publisher submits a request to *subscribe* to the content (C). The server on which (C) resides handles the request and has a few alternatives with which to reply. As is the current policy of the Web, all users can be allowed to subscribe to the content (C). Similar to the authorization mechanism in HTTP, certain users can subscribe to the content (C), while others are restricted. In the most extreme case, no users could be allowed to subscribe to the content (C).

More time passes and the original publisher wishes to remove the object from the local repository. In the simple case, no links exist to the published object and the object can simply be removed and the object's meta-information updated. However, when other resources subscribe to the published object, the situation becomes a bit more complex. We submit that the publisher ought to be able to control this notification process, since the potential cost of notifying all subscribers could be quite significant, e.g., if GNN's "What's New Page" were to be deleted.

TABLE 2. Subscription and notification policies.

Subscription Policy	Notification Policy
Allowed by all users	Notification or No Notification
Allowed by certain users	Notification or No Notification
Not allowed	Not Applicable

The control of the notification can be embedded into the initial reply to subscribe as shown in Table 2. By no means is this to be taken as an exhaustive list of possible subscription and notification policies. Indeed, it is expected that the set of valid policies will evolve with real world use. For example, how do you mask the identity of subscribers?

The default policy quite naturally ought to be to allow for all users to subscribe to content, as is current practice. Since current authoring tools are not integrated into an intelligent publishing environment that enable control of policies, it is quite natural to expect authors to override undesirable subscription policies. In more controlled and enforceable environments like digital libraries and corporate intranets, this counter-productive behavior is not likely to occur. Furthermore, breaches of subscriptions can be detected, reported, and repaired by software, e.g., the miner.

Regardless of the utility of controlling subscriptions, notifications of deletions to content is of interest to both publishers and subscribers.

Conclusion

Atlas has been developed to solve many problems daunting the continued utility and manageability of the Web. The primary contributions of this method involve the automatic maintenance of link information across distributed servers, the gradual deployment of the servers with convergence towards consistency for the Web, and the infrastructure for supporting efficient end user visualizations.

Much of this work has been done with input from NCSA in an effort to provide one piece of the proposed 'repository' architecture for the next generation of World Wide Web technology. An important piece of this is the maintenance of the repository information, and the ability to automate as much of this work as possible. This work has a direct benefit for the publishing process. Updates are immediately propagated to the appropriate servers as users publish their new or changed information. The results of this project will be placed in the public domain in the hopes that it will provide a basis from which to go forth on. Additional documentation on the protocol as well as the API's will be provided in the near future.

References

[Domel 1994] Domel, P. WebMap—A Graphical Hypertext Navigation Tool. *Proceedings of the Second International World Wide Web Conference.* Chicago: 1994. <URL:http://www.tm.informatik.uni-frankfurt.de/Publications/Doemel/WWWFall94/www-fall94.html>

[Fielding 1994] Fielding, R. Maintaining Distributed Hypertext Infostructures: Welcome to MOMspider's

Supporting The Web: A Distributed Hyperlink Database System

J. Pitkow & R. K. Jones

Fifth International World Wide Web Conference. Paris, France. May 6-11, 1996.

9

GVU Technnical Report: GIT-GVU-96-09

Web. *Computer Networks and ISDN systems*, Volume 27, issue 2, 1994.

[Jones 1996] Jones K. Atlas Protocol, Request For Comments. Intenet RFC, to be issued in July 1996.

[Joyce 1991] Joyce, M. Storyspace as a Hypertext System for Writers and Readers of Varying Ability. *Hypertext 1991 Proceedings*, Pittsburgh: Association for Computing Machinery, 1991. pp. 281-388.

[Halasz & Schwartz 1990] Halasz, F., Schwartz, M.The Dexter Hypertext Reference Model. *NIST Hypertext Standardization Workshop*, pp. 95-133, January 16-18, 1990.

[Hardy & Schwartz 1994] Hardy, D. & Schwartz M. Customized Information Extraction as a Basis for Resource Discovery. Technical Report CU-CS-707-94, Department of Computer Science, University of Colorado, Boulder, March 1994 (revised February 1995).

[Hughes 1995] Hughes, D. mSQL—A Lightweight Mini SQL Database Engine. <URL:ftp://Bond.edu.au/pub/Bond_Uni/Minerva/msql/>. 1995.

[Kappe 1995] Kappe, F. Maintaining Link Consistency in Distributed Hyperwebs. *Proceedings of INET 1995*. <URL:http://inet.nttam.com/HMP/PAPER/073/absr.html>.

[Kappe 1994] Kappe, F. A Scalable Architecture for Maintaining Referential Integrity in Distributed Information Systems. *IICM*, Graz University of Technology, 1994.

[Kappe et al. 1992] Kappe F., Maurer H., Sherbakov N. Hyper-G - A Universal Hypermedia System. *IIG Report 333*, IIG, Graz University of Technology, Austria, March 1992. <URL:ftp://iicm.tu-graz.ac.at/pub/Hyper-G/doc/>.

[Khare 1995] Khare, R. PEP: An Extension Mechanism for HTTP. W3 Consortium Working Draft, December 1995.

[McGuire 1994] McGuire, J. The EIT Link Verifier Robot. <URL:http://wsk.eit.com/wsk/dist/doc/admin/Webtest/verify_links. html>. 1994.

[Mukherjea 1995] Mukherjea, S., Foley, J. D. Visualizing the World Wide Web with the Navigation View Builder. *Computer Networks and ISDN Systems*, Volume 27, issue 6, 1995.

[Netcraft 1996] Netcraft HTTP Server Survey <URL:http://www.netcraft.co.uk/Survey/>. 1996.

[Pitkow 1993] Pitkow, J. The Html Analyzer Documentation. <URL:http://www.cc.gatech.edu/pitkow/html_analyzer/README.html>. 1993.

[Pitkow & Jones 1995] Pitkow, J., Jones, K. Towards an Intelligent Publishing Environment. *Computer Networks and ISDN Systems*, Volume 27, issue 6, 1995.

[Pitkow & Kehoe 1995] Pitkow, J. & Kehoe, C. Results for the Third World Wide Web User Survey. *The World Wide Web Journal*, Volume 1, issue 1, 1995.

[Weibel 1995] Weibel, S., Godby, J., Miller, E., Daniel, R.: "OCLC/NCSA Metadata Workshop Report," <URL:http://www.oclc.org:5046/conferences/metadata/dublin_core_report.html>. March 1995.

Acknowledgments

Supporting The Web: A Distributed Hyperlink Database System

Fifth International World Wide Web Conference. Paris, France. May 6-11, 1996.

GVU Technnical Report: GIT-GVU-96-09

J. Pitkow & R. K. Jones

10