Harvest: A Scalable, Customizable Discovery and Access System

C. Mic Bowman
Transarc Corp.

Peter B. Danzig
University of Southern California

Darren R. Hardy
University of Colorado - Boulder

Udi Manber
University of Arizona

Michael F. Schwartz
University of Colorado - Boulder

Duane P. Wessels
University of Colorado - Boulder

March 12, 1995

Technical Report CU-CS-732-94
Department of Computer Science
University of Colorado - Boulder
(Original Date: August 1994; Revised March 1995)

Abstract

Rapid growth in data volume, user base, and data diversity render Internet-accessible information increasingly difficult to use effectively. In this paper we introduce *Harvest*, a system that provides an *integrated* set of customizable tools for gathering information from diverse repositories, building topic-specific content indexes, flexibly searching the indexes, widely replicating them, and caching objects as they are retrieved across the Internet. The system interoperates with WWW clients and with HTTP, FTP, Gopher, and NetNews information resources. We discuss the design and implementation of Harvest and its subsystems, give examples of its uses, and provide measurements indicating that Harvest can significantly reduce server load, network traffic, and space requirements when building indexes, compared with previous systems. We also discuss several popular indexes we have built using Harvest, underscoring the customizability and scalability of the system.

1 Introduction

Over the past few years a progression of Internet publishing tools has appeared. Until 1992, FTP [45] and NetNews [42] were the principal publishing tools. Around 1992, Gopher [39] and WAIS [31] gained popularity because they simplified network interactions and provided better ways to navigate through information. With the introduction of Mosaic [1] in 1993, publishing information on the World Wide Web [2] gained widespread use, because of Mosaic's attractive graphical interface and ease of use for accessing multimedia data reachable via WWW links.

While Internet publishing has become easy and popular, making *effective use* of Internet-accessible information has become more difficult. As the volume of Internet-accessible information continues to grow, it becomes increasingly difficult to locate relevant information. Moreover, current information systems experience serious server and network bottlenecks as a rapidly growing user populace attempts to access networked information. Finally, current systems primarily support text and graphics intended for end user viewing; they provide little support for more complex data, as might be found in a digital library. For a more detailed discussion of these problems, the reader is referred to [9].

In this paper we discuss a system that addresses these problems using a variety of techniques. We call the system *Harvest*, to connote its focus on reaping the growing collection of Internet information. Harvest supports resource discovery through topic-specific content indexing made possible by a very efficient distributed information gathering architecture. It avoids bottlenecks through topology-adaptive index replication and object caching. Finally, Harvest supports structured data through a combination of structure-preserving indexes, flexible search engines, and data type-specific manipulation and integration mechanisms. Because it is highly customizable, Harvest can be used in many different situations.

The remainder of the paper is organized as follows. In Section 2 we discuss related work. In Section 3 we present an overview of the Harvest system. In Sections 4- 9 we discuss each of the subsystems, and provide measurements of these subsystems. In Section 10 we discuss several demonstrations of Harvest, and provide WWW pointers where readers can try these demonstrations. In Section 11 we discuss work in progress, and in Section 12 we summarize Harvest's contributions to the state of resource discovery.

2 Related Work

While impossible to discuss all related work, we touch on some of the better-known efforts here.

Resource Discovery

Because of the difficulty of keeping a large information space organized, the labor intensity of traversing large information systems, and the subjective nature of organization, many resource discovery systems create indexes of network-accessible information.

Many of the early indexing tools fall into one of two categories: file name or menu name indexes of widely distributed information (such as Archie [20], Veronica [21], or WWWW [38]); and full content indexes of individual databases (such as Gifford's Semantic File System [23], WAIS [31], and local Gopher [39] indexes). Name-only indexes are very space efficient, but support limited queries. For example, it is only possible to query Archie for "graphics packages" whose file names happen to reflect their contents. Moreover, global flat indexes become less useful as the information space grows (causing queries to match too much information). One of the advantages of Harvest is that it provides a means of building partial content indexes in a fashion that supports much more powerful searches than file name indexes, with much smaller space requirements than full content indexes.

Recently, a number of efforts have been initiated to create indexes of widely distributed sites (e.g., Gifford's Content Router [48] and Pinkerton's WebCrawler [43]). One of the goals of Harvest is to provide a flexible and efficient system upon which such systems can be built. We discuss this point in Section 4.

The WHOIS and Network Information Look Up Service Working Group in the Internet Engineering Task Force has defined an Internet standard called "WHOIS++" which gathers concise descriptions (called "centroids") of each indexed database [49]. In contrast to our approach, WHOIS++ does not provide an automatic data gathering architecture, nor many of the scaling features (such as caching and replication) that Harvest provides. However, Harvest can be used to build WHOIS++.

Aliweb [33] collects site description templates formatted according to the WHOIS++ specification.

An increasingly popular approach to resource discovery is the use of Web *robots* [32]. These are programs that attempt to locate a large number of WWW documents by recursively enumerating hypertext links starting with some known set of documents. As will be demonstrated in Section 4, Harvest provides a much more efficient architecture for distributed indexing than robots can, because of its optimized gathering architecture, support for incremental updates, and coordinated gathering. Moreover, we believe robot-based indexes will become decreasingly useful over time because, unlike Harvest, they do not focus their content on a specific topic or community.

Many of the ideas and some of the code for Harvest were derived from our previous work on the agrep string search tool [50], the Essence customized information extraction system [28], the Indie distributed indexing system [17], and the Univers attribute-based name service [11].

Caching and Replication

A great deal of caching and replication research has been carried out in the operating systems community over the past 20 years (e.g., the Andrew File System [29] and ISIS [4]). More recently, a number of efforts have begun to build object caches into HTTP servers (e.g., Lagoon [12]), and to support replication [24] in Internet information systems such as Archie.

In contrast to the flat organization of existing Internet caches, Harvest supports a hierarchical arrangement of object caches, modeled after the Domain Naming System's caching architecture. We believe this is the most appropriate arrangement because of a simulation study we performed using NSFNET backbone trace data [16]. We also note that one of the biggest scaling problems facing the Andrew File System is its callback-based invalidation protocol, which would be reduced if caches were arranged hierarchically.

Current replication systems (such as USENET news [42] and the Archie replication system) require replicas to be placed and configured manually. Harvest's approach is to derive the replica configuration automatically, adapting to measured physical network changes. We believe this approach is more appropriate in the large, dynamic Internet environment.

Structured Data Support

At present there is essentially no support for structured data in Internet information systems. When a WWW client such as Mosaic encounters an object with internal structure (such as a relational database or some time series data), it simply asks the user where to save the file on the local disk, for later perusal outside of Mosaic. The database community has done a great deal of work with more structured distributed information systems, but to date has fielded no widespread systems on the Internet. Similarly, a number of object-oriented systems have been developed (e.g., CORBA [25] and OLE [30]), but have yet to be deployed on the Internet at large.

At present, Harvest supports structured data through the use of attribute-value structured indexes. We are currently extending the system to support a more object-oriented paradigm to index and access complex data.

3 System Overview

To motivate the Harvest design, Figure 1 illustrates two important inefficiencies experienced by most of the indexing systems described in Section 2. In this figure and the remainder of the paper, a *Provider* indicates a server running one or more of the standard Internet information services (FTP, Gopher, HTTP, and NetNews). Bold boxes indicate excessive load being placed on servers, primarily because the indexing systems gather information from Providers that fork a separate process to handle each retrieval request. Similarly, bold edges indicate excessive traffic being placed on network links, primarily because the indexing systems retrieve entire objects and then discard most of their contents (for example, retaining only HTML anchors and links in the index). These inefficiencies are compounded by the fact that current indexing systems gather information independently, without coordinating the effort among each other.

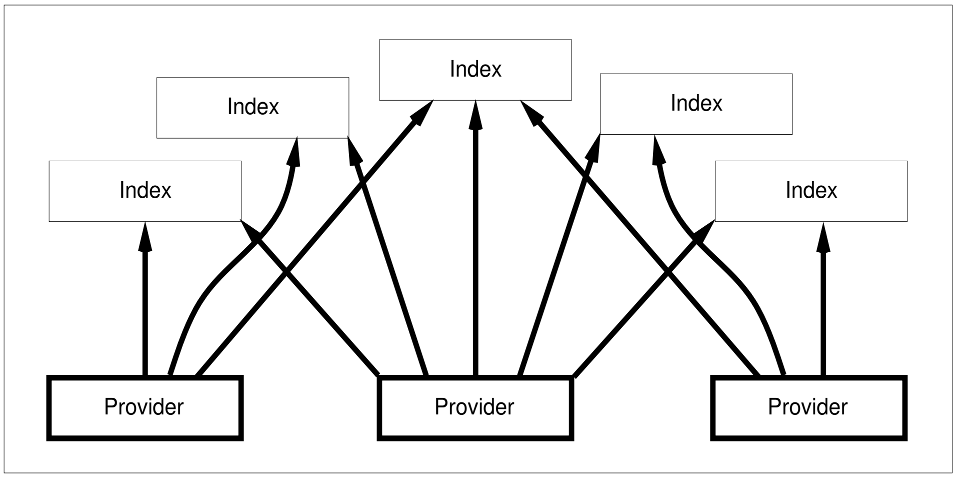

Figure 1: Inefficiencies Caused by Uncoordinated Gathering of Information from Object Retrieval Systems

Figure 2 illustrates the Harvest approach to information gathering. A *Gatherer* collects and extracts indexing information from one or more Providers. For example, a gatherer can collect PostScript files and extract text from them, it can collect UNIX tar files and extract their contents, or it can collect NetNews articles and extract abstracts and author names.

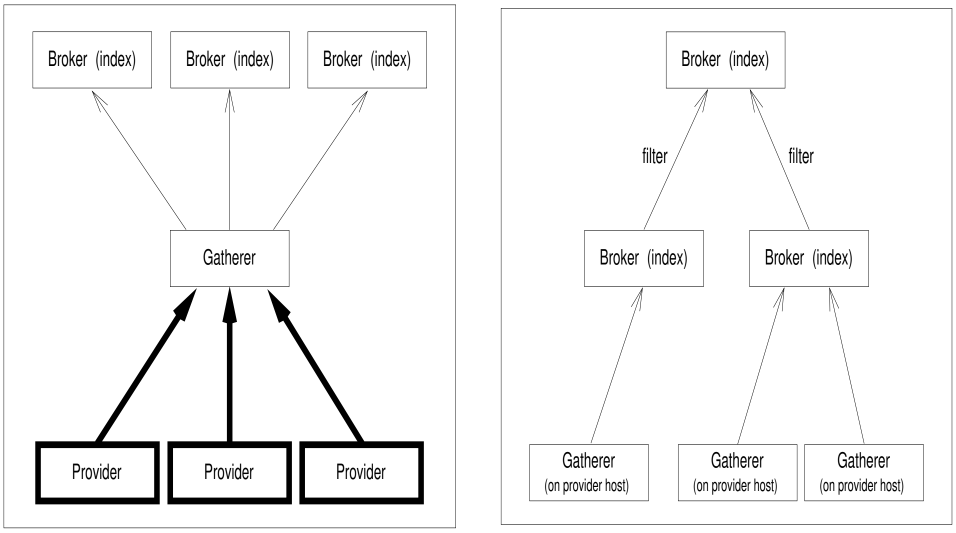

Figure 2: Harvest Information Gathering Approach

A *Broker* provides the indexing and the query interface to the gathered information. Brokers retrieve information from one or more Gatherers or other Brokers, and incrementally update their indexes.

Gatherers and Brokers can be arranged in various ways to achieve much more flexible and efficient use of network bandwidth and server capacity than the approach illustrated in Figure 1. The left half of Figure 2 shows a Gatherer accessing a Provider from across the network (using the native FTP, Gopher, HTTP, or NetNews protocols). This arrangement mimics the configuration shown in Figure 1, and is primarily useful for interoperating with sites that do not run the Harvest software. This arrangement experiences the same inefficiencies as discussed above, although the gatherered information can be shared by many Brokers in this arrangement—and thereby reduce network and server load.

In the right half of Figure 2, the Gatherer runs on the Provider site hosts. As suggested by the lack of bold boxes and lines, this configuration can save a great deal of server load and network traffic. We discuss the techniques used by the Gatherer to achieve these efficiency gains in Section 4.

A Broker can collect information from many Gatherers (to build an index of widely distributed data) and a Gatherer can feed information to many Brokers (thereby saving repeated gathering costs). By retrieving information from other Brokers (illustrated in the rightmost part of Figure 2), Brokers can also cascade indexed *views* from one another—using the Broker's query interface to filter or refine the information from one Broker to the next.

In effect, this Gatherer/Broker design provides an efficient *network pipe* facility, allowing information to be gatherered, transmitted, and shared by many different types of indexes, with a great deal of flexibility in how information is filtered and interconnected between providers and

Brokers. Efficiency and flexibility are important because we want to enable the construction of many different topic-specific Brokers. For example, one Broker might index scientific papers for microbiologists, while another Broker indexes PC software archives. By focusing index contents per topic and per community, a Broker can avoid many of the vocabulary [22] and scaling problems of unfocused global indexes (such as Archie and WWWW).

Harvest includes a distinguished Broker called the *Harvest Server Registry (HSR)*, which allows users to register information about each Harvest Gatherer, Broker, Cache, and Replicator in the Internet. The HSR is useful when searching for an appropriate Broker, and when constructing new Gatherers and Brokers, to avoid duplication of effort.

Gatherers and Brokers communicate using an attribute-value stream protocol called the Summary Object Interchange Format (SOIF) [27], intended to be easily parsed yet sufficiently expressive to handle many kinds of objects. It delineates streams of object summaries, and allows for multiple levels of nested detail. SOIF is based on a combination of the Internet Anonymous FTP Archives (IAFA) IETF Working Group templates [19] and BibTeX [34]. Each template contains a type, a Uniform Resource Locator (URL) [3], and a list of byte-count delimited attribute-value pairs. We define a set of mandatory and recommended attributes for Harvest system components. For example, attributes for a Broker describe the server's administrator, location, software version, and the type of objects it contains. An example SOIF record is illustrated in Figure 3. [1]

Figure 4 illustrates the Harvest architecture in more detail, showing the Gatherer/Broker interconnection arrangements and SOIF protocol, the internal components of the Broker (including the Index/Search Engine and Query manager, which will be discussed in Section 5), and the caching, replication, and client interface subsystems.

The Harvest *Replicator* can be used to replicate servers, to enhance user-base scalability. For example, the HSR will likely become heavily replicated, since it acts a point of first contact for searches and new server deployment efforts. The Replication subsystem can also be used to divide the gathering process among many servers (e.g., letting one server index each U.S. regional network), distributing the partial updates among the replicas.

The Harvest *Object Cache* reduces network load, server load, and response latency when accessing located information objects. In the future it will also be used for caching *access methods*, when we incorporate an object-oriented access mechanism into Harvest. At present Harvest simply retrieves and displays objects using the standard HTML mechanisms, but we are extending the system to allow providers to associate access methods with objects (see Section 11). Given this support, it will be possible to perform much more flexible display and access operations.

We discuss each of the Harvest subsystems below.

4 The Gatherer Subsystem

As indexing the Web becomes increasingly popular, two types of problems arise: data collection inefficiencies and duplication of implementation effort. Harvest addresses these problems by providing an efficient and flexible system upon which to build many different indexes. Rather than building one indexer from scratch to index WWW pages, another to index Computer Science technical reports, etc., Harvest can be configured in various ways to produce all of these indexes, at great savings of Provider-site server load and network traffic.

[1] While the attributes in this example are limited to ASCII values, SOIF allows arbitrary binary data.

```
@FILE { http://harvest.cs.colorado.edu/harvest/user-manual/node99.html
update-time{9}: 793962520
description{27}:          About this document ...
time-to-live{8}:        14515200
refresh-rate{7}:        2419200
gatherer-name{57}:      Networked Information Discovery and Retrieval
gatherer-host{21}:      bruno.cs.colorado.edu
gatherer-version{3}:    1.0
type{4}:        HTML
file-size{4}:   2551
md5{32}:        bea4c43ce6b976b3403c24f6a353f610
author{42}:     Darren Hardy
Wed Feb 15 13:27:56 MST 1995
keywords{68}:   about document drakos harvest html index latex manual
nikos this user
url-references{274}:    user-manual.html
node98.html
node3.html
...
partial-text{601}:         About this document ...
...
}
```

Figure 3: Example SOIF Template

Data Collection Efficiency

Koster offers a number of guidelines for constructing Web "robots" [32]. For example, he suggests using breadth-first rather than depth-first traversal, to minimize rapid-fire requests to a single server. These guidelines make a good deal of sense for uncoordinated information gathering. In contrast, Harvest focuses on coordinating and optimizing the gathering process.

The biggest inefficiency in current Web indexing tools arises from collecting indexing information from systems designed to support object retrieval. For example, to build an index of HTML documents, each document must be retrieved from a server and scanned for hypertext links. This causes a great deal of load, because each object retrieval requires creating a TCP connection, forking a UNIX process, changing directories several levels deep, transmitting the object, and terminating the connection. An index of FTP file names can be built more efficiently using recursive directory listing operations, but this still requires an expensive file system traversal. More recently there have also been efforts to reduce the costs of HTTP retrievals by creating servers that do not fork [41].

The Gatherer dramatically reduces these inefficiencies through the use of Provider site-resident software optimized for indexing. This software scans the objects periodically and maintains a

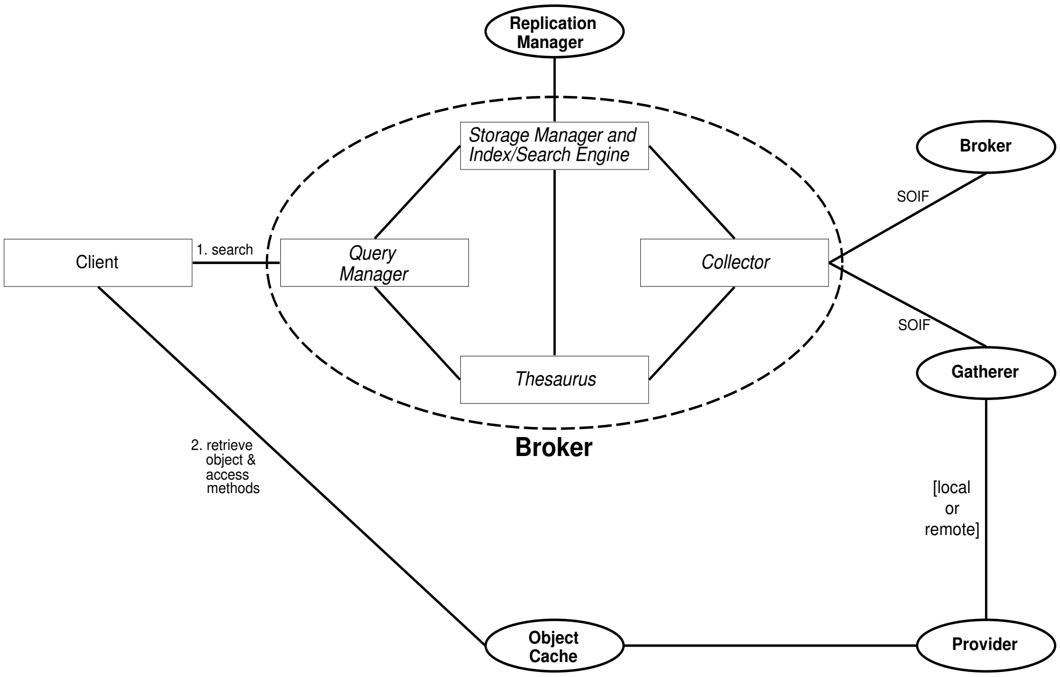

Figure 4: Harvest System Architecture

cache of indexing information, so that separate traversals are not required for each request. [2] More importantly, it allows a Provider's indexing information to be retrieved in a single stream, rather than requiring separate requests for each object. This results in a huge reduction in server load. The combination of traversal caching and response streaming can reduce server load significantly.

The Gatherer uses four techniques to reduce network traffic substantially. First, it uses the Essence system [28] to extract *content summaries* before passing the data to a remote Broker. Essence uses type-specific procedures to extract the most relevant parts of documents as content summaries — for example, extracting author and title information from LaTeX documents, and routine names from executable files. Hence, where Web robots might retrieve a set of HTML documents and extract anchors and URLs from each, Essence extracts content summaries at the Provider site, significantly reducing the amount of data to be transmitted. Second, the Gatherer transmits indexing data in compressed form. Third, the Gatherer supports incremental updates by allowing Brokers to request only summaries that have changed since a specified time. For access methods that do not support time stamps, the Gatherer extracts and compares "MD5" [46] cryptographic checksums of each object to determine if the object has changed since the last time the Gatherer generated a summary for it. Finally, the Gatherer maintains a cache of recently retrieved objects, to reduce the load of starting the system back up after a crash. [3].

We present measurements of these savings later in this section.

[2] Some FTP sites provide a similar optimization by placing a file containing the results of a recent recursive directory listing in a high-level directory. The Gatherer uses these files when they are available.

[3] The Gatherer uses its own cache rather than the normal Harvest Cache because gathering isn't likely to exhibit much locality, and would hence affect the Cache's performance.

Flexibility of Information Gathering and Extraction

In addition to reducing the amount of indexing data that must be transmitted across the network, Essence's summarizing mechanism permits a great deal of flexibility in how information is extracted. Site administrators can customize how object types are recognized (typically by examining file naming conventions and contents); which objects get indexed (e.g., excluding executable code for which there are corresponding source files); and how information is extracted from each type of object (e.g., extracting title and author information from bibliographic databases, and copyright strings from executable programs). Essence can also extract files with arbitrarily deep presentation-layer nesting (e.g., compressed "tar" files). An example of Essence processing is illustrated in Figure 5.

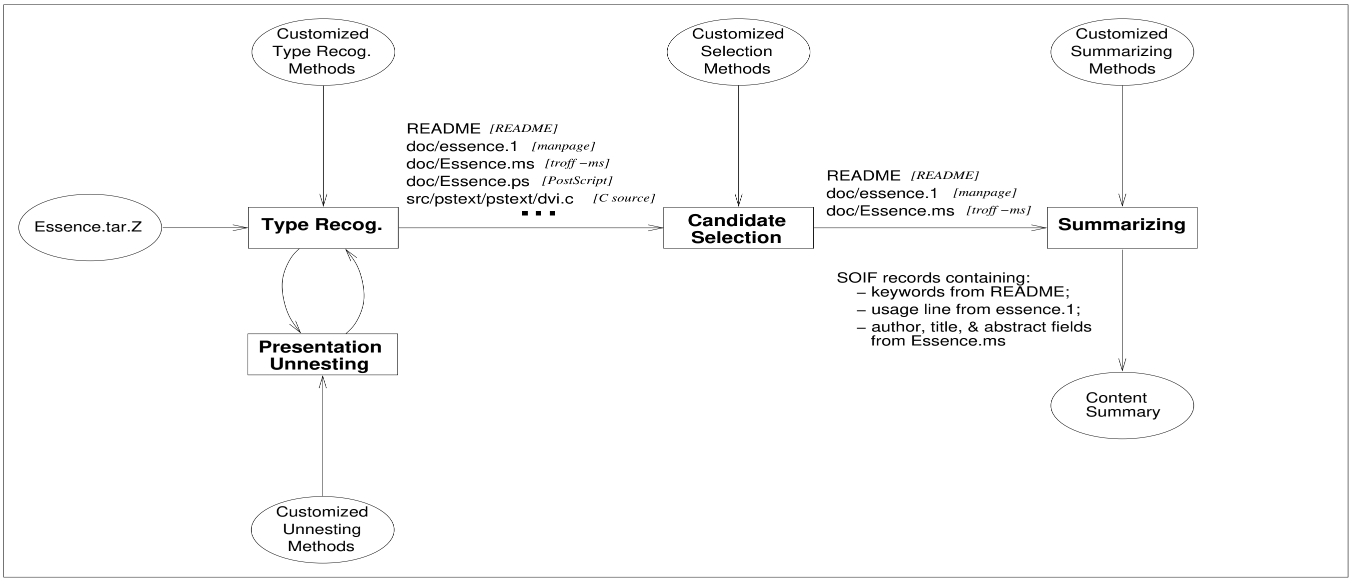

Figure 5: Example of Essence Customized Information Extraction

The Gatherer allows objects to be specified either individually (which we call "LeafNodes") or through type-specific recursive enumeration (which we call "RootNodes"). For example, FTP objects are enumerated using a recursive directory listing, while HTTP objects are enumerated by recursively extracting the URL links that point to other HTML objects. The RootNode enumeration mechanism provides several means by which users can control the enumeration, including site and URL stop lists (both based on regular expressions), enumeration depth limits, and enumerated URL count limits. The default limits are conservative (for example disallowing HTTP enumerations beyond the scope of a single server), to prevent a misconfigured Gatherer from traversing large parts of the Web.

The combination of Gatherer enumeration support and content extraction flexibility allows the site administrator to specify a Gatherer's configuration very easily. Given this specification, the system automatically enumerates, gathers, unnests, and summarizes a wide variety of information resources, generating content summaries that range in detail from full-text to highly abridged and specialized "signatures." The Gatherer also allows users to include *manually generated* information (such as hand-chosen keywords or taxonometric classification references) along with the

Essence-extracted information. This allows users to improve index quality for data that warrants the added manual labor. For example, Harvest can incorporate site markup records specified using IETF-specified IAFA templates. The Gatherer stores manually and automatically generated information separately, so that periodic automated summarizing will not overwrite the manually created information.

The other type of flexibility supported by the Gatherer is in the distribution of the Gatherer process. Ideally, a Gatherer will be placed at each Provider site, to support efficient data gathering (discussed below). Because this may not always be feasible, we also allow Gatherers to be run remotely, accessing objects using FTP/Gopher/HTTP/NetNews.

Measurements

In this subsection we present measurements of the Gatherer's efficiency with respect to provider site load and network traffic.

Provider Site Load Measurements

To quantify Provider-site server savings from running local Gatherers, we measured the time taken to gather information three different ways: via individual FTP retrievals, via individual local file system retrievals, and from the Gatherer's cache. To focus these measurements on server load, we collected only empty files and used the network loopback interface in the case where data would normally go across the network. We collected the measurements over 1,000 iterations on an otherwise unloaded DEC Alpha workstation. Our results indicate that gathering via the local file system causes 7.0 times less load than gathering via FTP. [4] More importantly, retrieving *an entire stream* of SOIF records from the Gatherer's cache incurs 2.7 times less server load than retrieving a *single object* via FTP (and 2.8 times less load for retrieving all objects at a site, because of an optimization we implemented that keeps the data for this commonly-requested case cached). Thus, assuming an average archive site size of 2,300 files [5], serving indexing information via a Gatherer will reduce server load per data collection attempt by a factor of 6,429. Collecting all objects at the site increases this reduction to 6,631, because of the aforementioned common-case cache.

As noted earlier, Harvest allows information to be collected remotely so that it can interoperate with sites that do not run the Harvest software. Because this poses so much more load on Provider-site servers, we implemented a *connection caching* mechanism, which attempts to keep connections open across individual object retrievals.[6] When the available UNIX file descriptors are exhausted (or timed out with inactivity), they are closed according to a replacement policy. This mechanism reduces server load by a factor of 7.0 for FTP transfers.

Network Traffic Measurements

As discussed previously, Harvest reduces network traffic four ways: extracting indexing data before transmitting across the network, supporting incremental updates, transmitting the data in com-

[4] Harvest provides a means by which site administrators can specify mappings from URL roots to local file system names, allowing URLs listed in the Gatherer configuration file to be accessed via the local file system.

[5] We computed this average from measurements of the total number of sites and files indexed by Archie [47].

[6] At present this is only possible with the FTP control connection and NetNews. Gopher and HTTP close connections after each retrieval.

pressed form from Gatherers to Brokers, and by using a local cache of recently retrieved objects. Here we discuss the first three ways, since the cache is primarily needed for insulating the network from re-retrievals in case of a system crash.

The savings from pre-extraction are derived from measurements we performed for an earlier paper about Essence [28]. We found that Essence content summaries require from 10-50 times less space than the data they summarize, depending on data type. For example, the reduction for PostScript files is quite high (since a good deal of PostScript content is positioning information that can be discarded when extracting indexing terms), while the reduction for short text files is fairly low (since Essence extracts full keyword content for short text files).

To measure the savings from incremental gathering, we sampled "last update" times for a set of approximately 4,600 HTTP objects distributed among approximately 2,000 sites, between September 1 and November 1, 1994. We then computed the cost of incremental gathering for a range of update periods, based on the average update period for each object and the object's size. Figure 6 plots the proportion of bytes saved by incremental vs. full gathering for the sampled objects. For example, this figure shows that full gathering sends approximately 4.8 times as many bytes as incremental gathering does, for daily updates. The savings drop off rapidly as the gathering period increases because many objects are updated frequently.

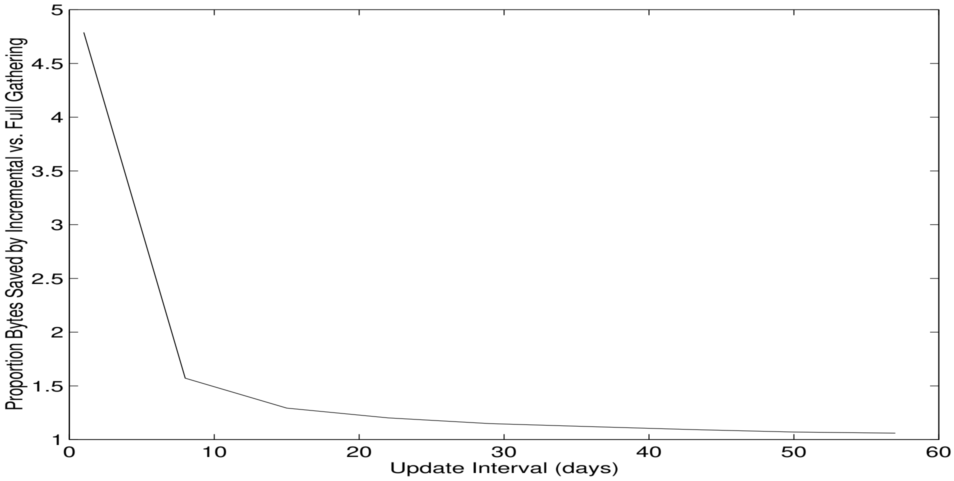

Figure 6: Measured Savings from Incremental Gathering for Sampled HTTP Documents

While a factor of 4.8 is clearly worthwhile, we were initially surprised the savings were not higher. The explanation is that the savings depends heavily on how quickly the data change. The average lifetime of the sampled objects was approximately 44 days, with over 28% of the objects being updated at least every 10 days, and 1% being updated at least daily. [7] We believe that WWW data volatility is particularly high at present because the WWW is a new phenomenon. WWW may reach a somewhat less volatile steady state over time. Older systems like FTP probably experience much slower change rates, and hence incremental gathering would be of more value for such systems.

[7] In some cases HTTP objects appear to be updated very frequently because they include constructs that recompute a value upon each access, such as the current user readership count.

One other consideration is that frequent incremental gathering allows more current indexing information to be maintained. For example, most Web "robots" perform infrequent gathering (e.g., only every several months), which means the majority of their content is out-of-date for the average lifetime of 44 days.

The last way that Harvest reduces network traffic while gathering indexing data is through the use of compression. The savings from compression is again data type-dependent, providing an additional factor of 2-10 in savings.

Examples of Overall Traffic and Provider Site Savings from Harvest Gathering

The combination of pre-filtering, incremental updates, and compression can reduce network traffic significantly. As an example of these savings, our content index of Computer Science technical reports (see Section 10) starts with 2.44 GB of data distributed around the Internet, and extracts 111 MB worth of content summaries. It then transmits these data as a 41 MB compressed stream to requesting Brokers, for a 59.4 fold network transmission savings over collecting the documents individually and building a full-text index. Note that these computations do not include the additional savings from doing incremental updates.

While one might suspect that Essence's use of content summaries causes the resulting indexes to lose useful keywords, our measurements indicate that Essence achieves nearly the same precision [8] and 70% the recall of WAIS, but requires only 3-11% as much index space and 70% as much summarizing and indexing time as WAIS [28]. For users who need the added recall and precision, however, Essence also supports a full-text extraction option.

As an overall illustration of the savings on Provider-site server load, it took 10 hours to gather and extract the information for our AT&T 1-800 Broker (see Section 10), pulling the data across the Internet for each of 4,100 HTML files. (This Gatherer ran remotely from the Provider site.) Once we had computed the compressed SOIF summary stream for these data, we could retrieve the entire stream across the Internet in approximately 3 minutes—a 200 fold reduction! This reduction is explained in part by the reduced amount of data that needed to be sent (2.9 MB for the compressed SOIF summary stream vs. 20 MB for the original data); partly by the time required to perform content extraction only when the data are first gathered; and mostly by the fact that retrieving from the Gatherer can be done without any forking, whereas retrieving via HTTP required a fork for each of the 4,100 HTML objects. Once we had gathered the data from the AT&T machine, we could serve it to other Brokers much more efficiently than could be accomplished by gathering the information independently.

5 The Broker Subsystem

As illustrated in Figure 4, the Broker consists of four software modules: the Collector, the Registry, the Storage Manager and Index/Search Engine, and the Query Manager.

The Registry and Storage Manager maintain the authoritative list of summary objects that exist in the Broker. For each object, the Registry records a time-to-live value and unique object identifier (OID) consisting of the object's URL plus information about the Gatherer that generated

[8] Intuitively, precision is the probability that all of the retrieved documents will be relevant, while recall is the probability that all of the relevant documents will be retrieved [6].

the summary object. The Storage Manager archives a collection of summary objects on disk, storing each as a file in the underlying file system. The Broker also eliminates duplicate objects (e.g., those reached via multiple HTML pointers) based on a combination of MD5 signatures and Gatherer IDs.

The Collector periodically requests updates from each Gatherer or Broker specified in its configuration file. The Gatherer responds with a list of object summaries to be created, removed, or updated, based on a timestamp passed from the requesting Broker. The Collector parses the response and forwards it to the Registry for processing.

When an object is created, an OID is added to the Registry, the summary object is archived by the Storage Manager, and a handle is given to the Index/Search Engine. When the collection is finished, the Registry is written to disk to complete the transaction. When an object is destroyed, the OID is removed and the summary object is deleted by the Storage Manager. When an object is refreshed, its time-to-live is recomputed. If the time-to-live expires for an object, the object is removed from the Broker. Since the state of the Storage Manager may become inconsistent with the Registry, periodic garbage collection removes any unreferenced objects from the Storage Manager.

The Query Manager exports objects to the network. It accepts a query, translates it into an intermediate representation, and then passes it to the search engine. The search engine responds with a list of OIDs and some search engine-specific data for each. For example, one of our search engines (Glimpse; see Section 6) returns the matching SOIF attribute for each summary object in the result list. The Query Manager constructs a short object description from the OID, the search engine-specific data, and the summary object. It then returns the list of object descriptions to the client.

A Broker can gather objects directly from another Broker using a bulk transfer protocol within the Query Manager. When a query is sent to the source Broker, instead of returning the usual short description, the Query Manager returns the entire summary object. The query filters summary objects so that one Broker can gather a subset of the objects in another Broker. In this way a Broker can create an index on a specialized set of summary objects by filtering them from other Brokers.

The Query Manager supports remote administration of the Broker's configuration and operations. Several Broker parameters can be manipulated, including the list of Gatherers to contact, the frequency of contact with a Gatherer, the default time-to-live for summary objects, and the frequency of garbage collection.

6 The Index and Search Subsystem

To accommodate diverse indexing and searching needs, Harvest defines a general Broker-Index/Search Engine interface that can accommodate a variety of "backend" search engines. The principal requirements are that the backend support Boolean combinations of attribute-based queries, and that it support incremental updates. One can therefore use a variety of different backends inside a Broker, such as Ingres or WAIS (although some versions of WAIS do not support all the needed features).

We have developed two particular search and index subsystems for Harvest, each optimized for different uses. Glimpse [36] supports space-efficient indexes and flexible interactive queries, while

Nebula [10] supports fast searches and views based on complex standing queries that scan the data on a regular basis and extract relevant information.

Glimpse Index/Search Subsystem

The main novelty of *Glimpse* is that it requires a very small index (as low as 2-4% of the size of the data), yet allows very flexible content queries, including the use of Boolean expressions, regular expressions, and approximate matching (i.e., allowing misspelling). Another advantage of Glimpse is that the index can be built quickly and modified incrementally.

The index used by Glimpse is similar in principle to inverted indexes, with one additional feature. The main part of the index consists of a list of all words that appear in all files. For each word there is a pointer to a list of occurrences of the word. But instead of pointing to the exact occurrence, Glimpse points to an adjustable-sized area that contains that occurrence. This area can be simply the file that contains the word, a group of files, or perhaps a paragraph. [9] By combining pointers for several occurrences of each word and by minimizing the size of the pointers (since it is not necessary to store the exact location), the index becomes rather small. This allows Glimpse to use agrep [50] to search the index, and then search the areas found by the pointers in the index. In this way Glimpse can easily support approximate matching and other agrep features.

Since the index is quite small, it is possible to modify it quickly. Glimpse has an incremental indexing option that scans all file modification times, compares each to the last index creation time, and reindexes only the new and modified files. In a file system with 6,000 files of about 70MB, incremental indexing takes about 2-4 minutes on a Sparcstation IPC. A complete indexing takes about 20 minutes.

The combination of Glimpse features allows users to perform much more selective queries than simple keyword searches when the Gatherer was able to extract attributes from the gathered objects (such as "author's name," "institution," and "abstract"). For example, one can search for all articles about "incremental indexing" by "unknown author," allowing one spelling error in the author's name.

Harvest's use of Essence content summarizing and Glimpse indexing provides a great deal of space savings compared with WAIS. In Section 4 we cited our earlier Essence measurements showing that Essence requires only 3-11% as much index space as WAIS. Because we now use Glimpse to index the space-efficient summaries generated by Essence, the space savings are drastically improved. Combining these two changes, we compute that Harvest now requires a factor of up to 42.6 less index space than WAIS. This savings makes it possible for Harvest to index a great deal of information using a moderate amount of disk space – a concrete example being a Broker we made of Computer Science technical reports from 280 sites around the world (see Section 10). This Broker would have taken 9 GB with a standard WAIS index, but only required 270 MB using Harvest.

Nebula Index/Search Subsystem

In contrast to Glimpse, Nebula focuses on providing fast queries at the expense of index size. It also provides particular support for constructing *views*, which are defined by standing queries against the database of indexed objects. Because views exist over time, it is easy to refine and extend

[9] Paragraph-level indexing is not implemented yet.

them, and to observe the effect of query changes interactively. A view can scope a search in the sense that it constrains the search to some subset of the database. For example, within the scope of a view that contains computer science technical reports, a user may search for networks without matching information about social networks.

Nebula's views facilitate the construction of hierarchical classification schemes for indexed objects. For example, a simple classification scheme for our PC software Broker (see Section 10) implements, at the top level, a view that selects all summary objects that represent PC software. A more specific class, like software for Microsoft Windows, is constructed from objects in the PC software view; i.e., the PC software view scopes the Windows view.

Nebula can also be extended to include domain-specific query resolution functions. For example, a server that contains bibliographic citations can be extended with a resolution function that prefers keywords found in "abstract" and "title" attributes to keywords found in the text of the document. The resolution function is constructed with domain-specific information—namely, that the title and abstract contain the most important and descriptive information about a document.

For performance reasons, each attribute tag has a corresponding index that maps values to objects. The indexing mechanism can differ from tag to tag, depending on the expected queries. Attributes like "description" and "summary" use a mechanism that indexes individual keywords in the value. The keyword mechanism supports queries based on pattern matching. Other tags, like "name" and "size", index values with a dynamic hash table for fast, precise retrieval.

While Nebula prefers fast queries over small indexes, for structured summary objects some of the cost of indexing is reclaimed by storing a single copy of an attribute that appears in many summary objects. As a result, for a database of 20,000 bibliographic citations, the index adds about 9% overhead. A query on this database with two attribute expressions and one Boolean operator is resolved in less than 100 milliseconds on a Sparc IPC.

7 The Query Interface

There are two conflicting goals in designing a query interface for use in as diverse an environment as the Internet. On the one hand, we want to provide a great degree of flexibility and the ability to customize for different communities and for different users. On the other hand, we want a reasonably simple, uniform interface so that users can move from one domain to another without being overwhelmed.

Uniform Interface

Harvest supports a uniform client interface through the Query Manager in the Broker. The Query Manager in each Broker implements the same query language, which supports Boolean combinations of keyword- and attribute-based expressions. Query results are presented in a uniform format that includes some indexer-specific information plus the object identifier for each object.

The Query Manager uses an HTML form to collect the query from the user, and then passes the query to the Harvest gateway. The gateway retrieves the query from the form and sends it to the query manager. The set of objects returned by the Query Manager is converted into an HTML page with hypertext links to complete summary objects in the Broker's database and to the original resource described by the summary object. An example query is shown in Figure 7.

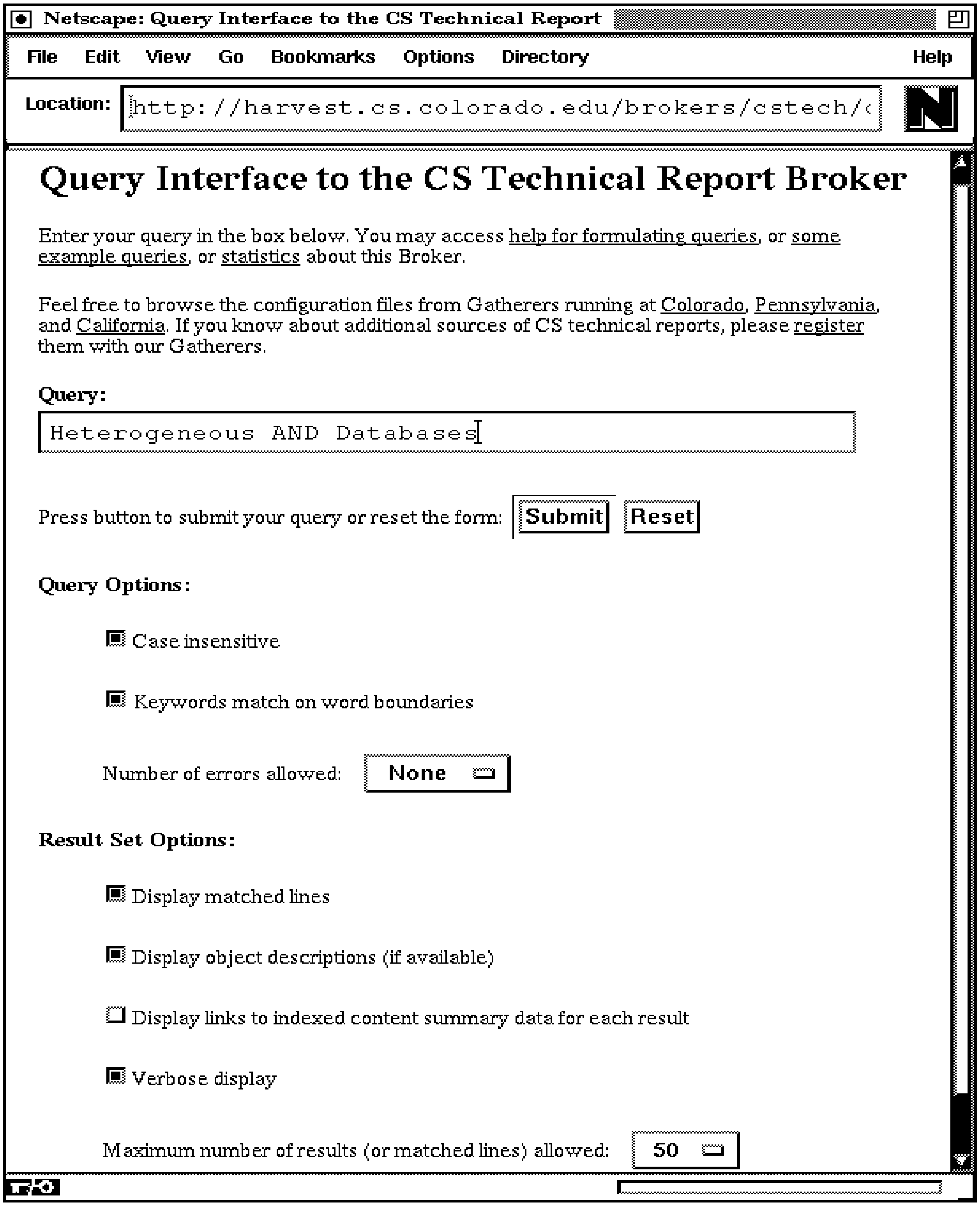

Figure 7: Example of Query Interface

Currently, the Query Manager maintains no state across queries. Thus, the Broker's uniform interface does not support query refinement. However, simple query refinement is implemented by most WWW clients. For example, Mosaic caches recently accessed documents in virtual memory. Previous queries can be retrieved and refined by moving through the cache. We also support a customizable means of displaying query results, which allows users to define result layout, displayed fields, string manipulations, warning messages, and many other aspects of the display output.

Interface Customization

Harvest supports query interface customization through direct access to each Index/Search Engine. For example, the Glimpse-specific interface supports additional switches for approximate matching and record specific document retrieval, while the Nebula-specific interface supports extensive query refinement and improved integration of organization and search.

Direct access lets the user take advantage of the specific strengths of each Index/Search Engine. While this means users must learn multiple interfaces, it also increases flexibility. A domain-specific Index/Search Engine can use the Broker to collect information from several sites. This information can be accessed by neophytes through the uniform interface. Expert users benefit from the additional functionality provided by the Index/Search Engine through the direct interface. In this way, Harvest facilitates the construction of domain-specific databases (such as image or genome databases) that require a unique query language or result presentation.

We are currently working on several tools to allow much greater customizations.

8 The Replication Subsystem

Harvest replicates its Brokers with the *mirror-d* replication tool. Mirror-d maintains a weakly consistent, replicated directory tree of files. Mirror-d implements eventual consistency [5]: if all new updates cease, the replicas eventually converge. Each replica belongs to one or more data propagation groups and, within a group, mirror-d's companion process, flood-d [15], estimates the available network bandwidth and observed round trip delay between each pair of replicas. A group master periodically computes a graph that specifies which replicas should synchronize and propagate updates among each other. We call this the group's *logical topology.* Replicas synchronize updates with their neighbors in the logical topology graph using the "ftp-mirror" software [40]. Basically, mirror-d dynamically configures the ftp-mirror software between replicas in a replication group so that updates propagate along the highest bandwidth, lowest delay network paths. Since groups can be organized hierarchically, mirror-d should be able to scale to thousands of autonomously operated replicas. Groups become stitched together when one or more replicas belong to both groups. While mirror-d is written in Perl and flood-d is written in C, both mirror-d (http://catarina.usc.edu:4000) and flood-d (http://catarina.usc.edu:9500) speak HTTP and HTML.

Each replication group is named, and one replica in each group is distinguished as the group leader. Flood-d's do not elect new leaders because the access control list of sites that can join the group resides at the group leader. Access control can be disabled or limited to a specific list of sites. If the leader fails, replication proceeds but the logical topology is not updated and new replicas cannot join the group.

Implementation Notes

Upon startup, a new replica issues a join request. When the group master receives the request, it generates a new topology with the new member included and distributes the topology to the new group members. Frequently during the first hour of operation, and less frequently afterwards, a replica selects another replica and measures the elapsed round trip for a null UDP-based remote procedure call and measures the elapsed time to transfer a 64KB data block via TCP. Periodically, replicas flood these estimates to one another. Flood-d does not implement an explicit group leave operation. Rather, if a replica is silent for a long time, each group member independently reclaims resources associated with it, and the group leader, when it recomputes the group topology, leaves out the silent replica. When new members join the group or when a timer expires the group leader computes and floods a new logical-topology to its members.

Periodically or when requested through its HTTP interface, a replica's mirror-d process attempts to synchronize with its logical neighbors. Mirror-d obtains the list of its logical neighbors from its flood-d process and then requests that each neighbor return the version number of its directory tree. If no neighbor has a more recent version, then mirror-d does nothing. If one or more neighboring replicas has a more recent version, mirror-d spawns an ftp-mirror process that synchronizes the two replicas. For ease of synchronization, mirror-d will spawn only one concurrent ftp-mirror and enqueues other relevant update notifications. If more than one neighbor have the highest version number, mirror-d chooses the neighbor with highest bandwidth to delay ratio.

The group leader constructs a two- or three-connected logical topology that attempts to minimize the sum of the logical link costs. These costs are computed as the estimated bandwidth to delay ratio. While we originally employed a search algorithm to compute this graph, recently we switched this to a heuristic: take a minimum spanning tree and add low cost links until the graph's connectivity requirement is met. Since neighbors attempt to propagate updates along the minimum spanning tree, data transfers usually do not take place to a replica's less desirable neighbors.

Performance and Experience

We have tested mirror-d configurations of thirty replicas, organized into two or three overlapping groups. Two of the project members replicate the 60MB Harvest www-home-pages Broker at our homes, over 28KB/s PPP links. At the time of writing, four sites replicate Harvest's www-home-pages Broker, and during testing we frequently run another ten replicas. Had these sites been replicated with ftp-mirror, rather than flood-d, all of them would have pulled their updates from the primary copy of the www-home-pages Broker. With mirror-d, the home machine mirrors off of a set of machines in a computing laboratory at the University of Southern California, the ten laboratory machines mirror off of each other, and one USC lab machine mirrors off of a machine at the University of Colorado.

9 The Object Caching Subsystem

To meet ever-increasing demand on network links and information servers, Harvest includes a hierarchically organized Object Caching subsystem [13], as illustrated in Figure 8. At each level there can be several *neighbor* caches, allowing some load-sharing among cache servers.

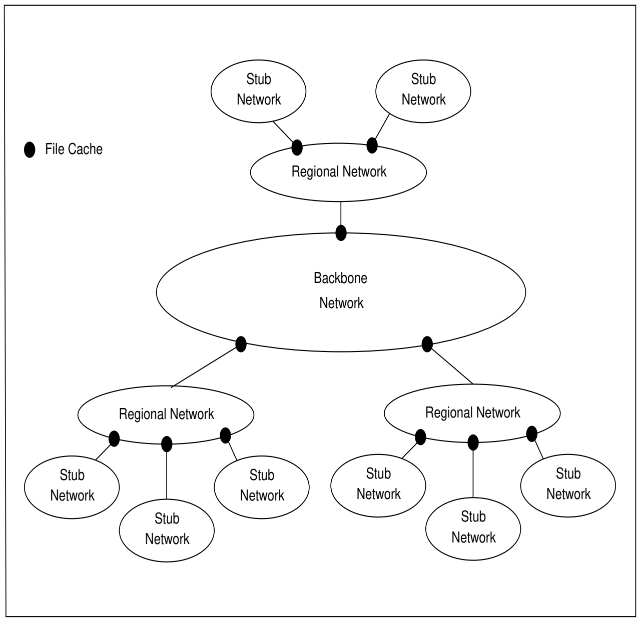

Figure 8: Topology-Based Cache Organization

Below, we describe the resolution policy, implementation philosophy, and performance of the Object Caching subsystem.

Hierarchical Caching

Clients request objects through the Cache's proxy-HTTP interface, modeled after the CERN httpd cache [35]. The proxy interface allows Mosaic, Netscape and Lynx clients to use the cache simply by setting three environment variables.

A cache resolves a miss by sending a "query" datagram to each of its *neighbor* and *parent* caches and to the echo port of "inetd" process [10] at the requested object's home site. Each neighbor and parent responds with a *hit* or *miss* message, depending on the state of the object in their caches. If the object's home is running a UDP [44] echo daemon, the object's home site echos a *hit* message. The cache fetches the object from the fastest site to return a *hit* message, whether it be another cache or the object's home site. If all caches miss and the home site is slower than all the parent caches, the cache retrieves the object through the fastest parent cache to miss. Otherwise, the cache retrieves the object from the home site if its response time is close to the fastest cache. In addition to caching Gopher, HTTP, FTP, and NetNews objects, we maintain a cache of recent DNS name-to-address mappings to optimize common-case cache behavior. We also cache *negative* responses, so that temporarily unavailable servers are not repeatedly contacted.

Currently, Web cache consistency is not altogether satisfactory, since no standard MIME header [7] exists that defines an object time-to-live [18]. Mosaic and Netscape cache images and Netscape caches objects, but verifying that an object is consistent requires nearly the same effort as retriving the object again. Until this standard emerges, all Web caches (including the Har-

[10]Inetd is the process in UNIX systems that spawns a new process to handle each incoming service request.

vest cache) are forced to assign default time-to-live values to objects. [11] In Mosaic and Netscape, re-loading an object causes the cache to re-fetch the object from its source.

The cache runs as a single, event-driven process. I/O to disk and to cache clients is non-blocking. I/O between cache clients begins as soon as the first few bytes of an object fault their way into the cache. For ease of implementation, the cache spawns a separate process to retrieve FTP files, but retrieves HTTP and Gopher objects itself. The cache separately manages replacement of objects on disk and objects loaded in its virtual address space. It also keeps all meta-data [12] for cached objects in virtual memory, to eliminate access latency to the meta-data. Finally, we use non-blocking name lookups. Since the cache is single threaded but otherwise non-blocking, page faults are the only source of blocking.

Cache Performance

We contrast non-hierarchical cache performance using a workload of 1,000 objects collected from logs of Mosaic activity. We collected the response time to retrieve each of these objects from our cache and from CERN's cache under the load of a single client and again from ten clients, referencing all 1,000 URLs in a random order.

With one client, all 1,000 references miss. As illustrated in the top half of Figure 9, the Harvest cache returns 600 of these objects in less than 1 second each, while it takes the CERN cache 2 seconds to return each of 600 objects. Since a handful of these objects come a long way through the network, some response times are quite large. In summary, the Harvest cache tends to be twice as fast on the objects that can be retrieved quickly and is never slower than the CERN cache for slow to retrieve objects.

Under our multiple client workload, one or more clients see a miss when they reference their object; the rest see a hit. As illustrated in the bottom half of Figure 9, under these conditions the Harvest cache is again roughly twice as fast as CERN's cache for most of these references, and has a much shorter tail of longer response times. Of these 10,000 references, only 172 references to our cache while 938 references to CERN's cache experience more than 4 second response time.

10 Demonstration Brokers

To provide some concrete examples of its flexibility and scalability, in this section we discuss a number of Brokers we have built using Harvest. Table 1 summarizes the features demonstrated by these Brokers.

Below we discuss some of these Brokers in more detail. The reader can try these and other Brokers via the WWW at http://harvest.cs.colorado.edu/harvest/demobrokers.html.

NIDR Brokers

To demonstrate the idea of topic-specific indexing, we constructed several Brokers focused on Networked Information Discovery and Retrieval (NIDR)—a topic about which we have the background

[11] Harvest allows the system administrator to set the default TTL values.

[12] File name and access statistics for each URL.

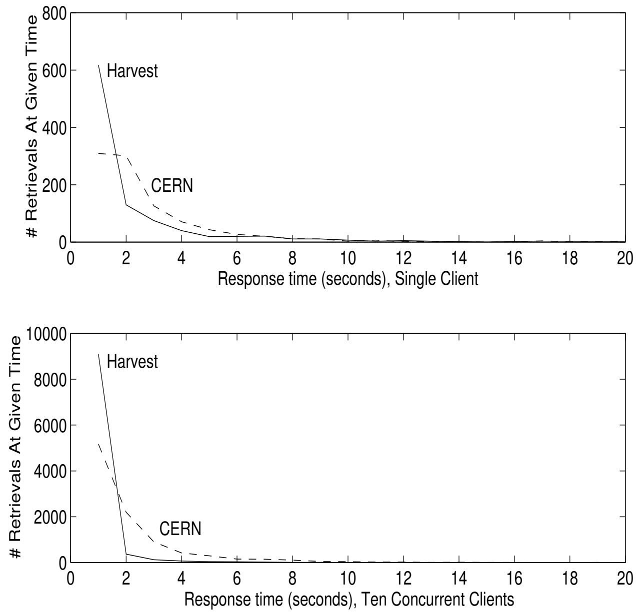

Figure 9: Harvest vs. CERN Cache Performance

Feature	Demo Brokers	#Objects	#Sites
Scalable content indexing Multiple formats Gathering distribution	CS Technical Reports	29,340	280
Incorp. manual indexing info Support for other standards Structured Indexing	PC Software Packages Linux Software Maps IAFA	30,910 1,510 -	6 39 -
Focused global content index Structured indexing	WWW Home Pages	27,360	11,630
Customized search scoping	UColo. CS Department	3,280	1
Topic-specific indexing; Broker cascading	NIDR-SW, NIDR-Doc, NIDR	4,600	110
Customized indexing; Structured indexing	SFI Time Series Papers	170	7
Structured Indexing	AT&T 1-800 phone #s	4,560	1
Handling rapidly changing data	Network News	616	1
Locate Harvest servers during search & index config.	Harvest Server Registry	87	34

Table 1: Summary of Demonstrated Features of Constructed Brokers

necessary to construct meaningful lists of objects to index, and about which a good deal of information is available online. We constructed three Brokers: one for software, one for documents, and one that gathers from these first two Brokers (with no additional network or remote server load) to provide a combined Broker of NIDR information.

To demonstrate the scaling advantages of topic-specific Brokers, suppose we want to locate technical reports about approximate string searching. Issuing the query "approximate" at our NIDR-Docs Broker locates Glimpse. On the other hand, issuing this same query at our Computer Science technical reports Broker (see below) locates many irrelevant papers in response to this query, such as a paper about approximate analysis of buffer replacement schemes. Using a topic-specific Broker reduces this unwanted breadth (the "vocabulary" problem).

Our NIDR Broker helps underscore the role topical experts can fill in creating well-focused Brokers, as well as the value of distributed indexing software in support of this role. We believe such experts will play an increasingly important role as the Internet moves towards professionally managed information collections.

Computer Science Technical Report Broker

To demonstrate the scaling benefits of our content summarizing approach, we built a Broker of Computer Science technical reports. Because content summaries require so much less space than full content indexing, we were able to index content summaries for over 24,200 documents with this Broker, gathered from 280 sites around the world. In contrast, the current WAIS Computer Science technical reports Broker only indexes authors and titles for 10,000 documents, and abstracts for 2,300 documents.

To construct the list of indexed objects, we began with several previously existing lists, including a WAIS index of Computer Science technical reports maintained at Monash University in Australia, [13], the University of Indiana index of Computer Science technical reports [14], and a large Computer Science bibliography maintained at the University of Karlsruhe in Germany[15]. We augmented these lists with sites located by doing an Archie substring search for "report". We then visited each site and selected paths appropriate to the technical report Broker. Since then we have incorporated URLs from many other sources.

Our technical report Broker also demonstrates two different types of gathering distribution. First, we divided the list of URLs among Gatherers running in California, Colorado, and Pennsylvania in such a fashion that sites were gathered from whichever Gatherer was closest. [16] Second, this Broker gathers from the NIDR Broker in addition to the above URL lists, representing topical distribution in the gathering process.

PC Software Broker

Our PC software Broker demonstrates Harvest's ability to incorporate manually generated indexing information in a variety of formats. A number of sites throughout the Internet offer archives of PC "shareware", documenting each package with a single line conceptual description. We built

[13] Accessible from wais://daneel.rdt.monash.edu.au:210/cs-techreport-archives

[14] Accessible at http://cs.indiana.edu/cstr/search

[15] Accessible at ftp://ftp.ira.uka.de/pub/bibliography.

[16] We used traceroute to make simple estimates of network distances.

summarizers to translate these descriptions from the formats used by six different archive sites into uniformly-structured SOIF templates. This Broker demonstrates Harvest's ability to take advantage of high quality manual annotation information when it is available, and the ability to create a Broker without scanning the indexed objects—only the metadata are scanned. This Broker provides better support for searches than more general-purpose software indexes (such as Archie), because it contains conceptual descriptions of a focused collection of information. For example, this Broker allows someone to search for "batch programming language," while Archie could only locate this software if someone supplied the name of batch programming language, like "RAP."

We have also built Gatherer translation scripts for some other manually created indexing information formats, including the "Linux Software Map" (LSM) format and the IAFA format. At present we have a Broker runing for LSM data but none for IAFA data, because there are not yet enough sites using the IAFA format to warrant building a Broker. The LSM Broker is similar to the PC archive Broker, but contains more attributes—including fields for the package description, author, maintainer, copyright policy, and a half-dozen other attributes.

A similar effort would allow us to incorporate other forms of information into Harvest Brokers, such as the U.S. Library of Congress Machine Readable card Catalog (MARC) standard [37].

Building Harvest Brokers Corresponding to Well-Known Existing Indexes

One of the ideas behind Harvest is to provide a customizable system that can be configured in various ways to create many types of Brokers, to reduce the amount of effort that currently goes into building single-purpose indexers. Here we briefly discuss uses of Harvest for creating particular Brokers related to some well-known, existing, single-purpose indexes and indexing systems. In many cases, Harvest Brokers can support more powerful searches, because they index document content summaries, rather than just names or other minimal document information.

McBryan created an index (called the World Wide Web Worm or WWWW) of anchors and HTML links, by gathering documents from around the World Wide Web [38]. Using Harvest we created a related Broker, containing content summaries of Web home pages. We began with a list of Web pointers from various sources, including the WWWW, the "What's New" pages maintained by the National Center for Supercomputing Applications and O'Reilly & Associates, Inc., a list of WWW servers maintained by the European Laboratory for Particle Physics (CERN), McBryan's Mother-of-all-BBS's [38], URLs gleaned from USENET postings, and various other sources. We periodically collect this list, prune it to a set of home pages, gather these home pages, content summarize them, and index them. We chose not to gather beyond home pages, because home pages typically point to other Web pages, and hence we suspect that one need not traverse the entire Web (ala WWWW) to provide a useful global Web index. Moreover, because we index content summaries rather than just anchor and HTML strings, our home page Broker captures much of the content of Web sites without having to collect every last Web page—providing a useful index at lower cost than that incurred by the WWWW. In fact, in some ways our Home page index works better, because HTML documents tend to contain so many cross links among each other that an index of only the Home pages can provide less duplicate-ridden query results than similar queries made to WWWW. Also, the Harvest Broker eliminates a great deal of duplicate information, by comparing MD5s and Gatherer IDs for each object. For our WWW Home Pages Broker this removed 15% of the objects.

We could make similar use of Harvest to build Brokers like those provided by Archie and Veronica. By simply constructing a Gatherer configuration and building a customized Essence extraction script, each of these indexes can be constructed easily.

We also built an index of Computer Science technical report abstracts, from the Karlsruhe bibliography mentioned earlier in this section. In contrast with our Computer Science Broker, this index only captures document abstracts, but covers many documents that are not available online. Because it uses our Glimpse indexing system, this index provides more flexible search power (e.g., allowing misspellings) than the Karlsruhe index server does. In time we plan to build a custom search engine that will first search our Computer Science technical report content index and then search our Computer Science technical report abstract index, without the users' needing to be aware that multiple Brokers are being consulted.

One could use Harvest to provide the functionality of many other indexes and indexing systems. One could build a WebCrawler [43] by defining an appropriate Gatherer configuration, and using the Essence full-text extraction mechanism. One could build a WHOIS++ [49] system by a combination of a Gatherer configuration and Essence extraction script for constructing the centroids, and then providing a front-end query script to search the gathered SOIF records.

As a final example of how Harvest can be used to build other Brokers, we plan to implement WAIS gathering support in Harvest. Once we have done that, we will be able to build a Broker of WAIS headlines, similar to that supported by Gifford's Content Router system [48]. In contrast with his system, though, we will serve the raw indexing information, allowing many other indexes to be constructed from our data at little additional network load (and no additional load on the gathered WAIS servers).

11 Work in Progress

At present we are extending Harvest in a number of directions. First, we are adding support for a more object-oriented style of access to information objects, to manipulate and display complex types of data. For this purpose we will allow organizations to create SOIF descriptions of objects, including types and methods that can be invoked either by fetching executable code to the local machine or by invoking remote TCP services [14]. A customized, Mosaic-invokable graphical object browser will allow users to interact with these objects by selecting among available methods, invoking methods, and saving intermediate objects. It will be possible to run access methods in succession, for example allowing an image to be filtered by one method and then displayed by another. Clearly, there are several security and architecture-dependency issues involved with this approach, which we are still considering.

We are also developing support for selecting among instances of a replicated service. We are doing this initially in the context of Network Time Protocol servers [26], but in time we will extend this system for use with locating nearby Cache and Replica servers as well.

We are developing tools similar to the Nebula information management system [10] to improve searching capabilities. These include recursive query evaluation, iterative query refinement, and integrated support for taxonomies. Recursive query evaluation enables automatic search of several servers located in the HSR. This relieves from the user the burden of manually guiding the search to several candidate Brokers. Iterative query refinement supports user feedback in the query resolution mechanism. Taxonomy support facilitates expressive, precise queries that can be resolved with minimal overhead.

We are working to support a variety of additional index types and index mechanisms. Additional types include a widely replicated, global index of rare words. Like the HSR, the rare word index will point to Brokers that contain related summary objects. Another type of index is the "join" index of individual site indexes. This may be a virtual index that redistributes queries to other sites. To support these and other indexes, we are working on indexer interfaces to WAIS and other popular database systems.

12 Summary

While systems like Gopher and WWW make it easy to publish information on the Internet, making *effective use* of Internet-accessible information grows increasingly difficult. Rapid growth in the volume of information makes it difficult to locate relevant information. In addition, current systems experience acute server and network bottlenecks when many users attempt to access networked information. Finally, current systems provide little support for structured, complex data.

In this paper we introduced the Harvest system, which addresses these problems through a combination of topic-specific content indexing made possible by a very efficient distributed information gathering architecture; topology-adaptive index replication and hierarchical object caching; and structure-preserving indexes, flexible search engines, and data type-specific manipulation and integration mechanisms. We presented measurements indicating that Harvest can reduce server load by a factor of over 6,000, network traffic by a factor of 59, and index space requirements by a factor of 43 when building indexes, compared with previous systems. Harvest also provides a high performance Object Cache that can be arranged hierarchically for scalability reasons.

Harvest interoperates with WWW clients and with HTTP, FTP, Gopher, and NetNews information resources, and defines several protocols that can interoperate with a variety of digital library, knowledge robot, and other types of systems.

To demonstrate system scalability and the ease with which users can customize Harvest for building many types of indexes, we built a number of indexes using Harvest, including indexes of Networked Information Discovery and Retrieval software and documents; Computer Science technical reports; documents referencing the Santa Fe Institute time series competition data; PC Software; WWW Home Pages; and other types of data.

The Harvest software is available from a number of distribution sites, pointed to from http://harvest.cs.colorado.edu/harvest/gettingsoftware.html. The system consists of approximately 158,000 lines of code, divided among 45,300 in the Gatherer, 27,000 in the Index/Search system, 13,700 in the Broker, 14,500 in the Cache, 30,800 in the Replicator, and 26,400 in common data structure library routines.

References

[1] Marc Andreessen. NCSA Mosaic technical summary. Technical report, May 1993. Available from ftp://zaphod.ncsa.uiuc.edu/Web/mosaic-papers/mosaic.ps.Z.

[2] T. Berners-Lee, R. Cailliau, J-F. Groff, and B. Pollermann. World-Wide Web: The information universe. *Electronic Networking: Research, Applications and Policy*, 1(2):52–58, Spring 1992. Available from ftp://ftp.cern.ch-/pub/www/doc/ENRAP_9202.ps.

[3] Tim Berners-Lee. *Uniform Resource Locators.* CERN, July 1993. Internet Draft, IETF URL Working Group.

[4] Kenneth P. Birman. Replication and fault-tolerance in the Isis system. *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, pages 79–86, December 1985.

[5] Andrew D. Birrell, Roy Levin, Roger M. Needham, and Michael D. Schroeder. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, April 1982.

[6] David C. Blair and M. E. Maron. An evaluation of retrieval effectiveness for a full-text document-retrieval system. *Communications of the ACM*, 28(3):289–299, March 1985.

[7] Nathaniel Borenstein and Ned Freed. RFC 1521: MIME (Multipurpose Internet Mail Extensions) part one: Mechanisms for specifying and describing the format of internet message bodies. Technical report, September 1993.

[8] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. The Harvest information discovery and access system. *Proceedings of the Second International World Wide Web Conference*, pages 763–771, October 1994. Available from ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz-/Harvest.Conf.ps.Z or ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.Conf.txt.Z.

[9] C. Mic Bowman, Peter B. Danzig, Udi Manber, and Michael F. Schwartz. Scalable Internet resource discovery: Research problems and approaches. *Communications of the ACM*, 37(8):98–107, August 1994.

[10] C. Mic Bowman, Chanda Dharap, Mrinal Baruah, Bill Camargo, and Sunil Potti. A file system for information management. *Proceedings of the Conference on Intelligent Information Management Systems*, June 1994.

[11] Mic Bowman, Larry L. Peterson, and Andrey Yeatts. Univers: An attribute-based name server. *Software Practice & Experience*, 20(4):403–424, April 1990.

[12] Paul M. E. De Bra and Reiner D. J. Post. *Information Retrieval in the World-Wide Web: Making Client-based searching feasible.* Eindhoven University of Technology. Available from http://www.win.tue.nl/win/cs/is-/reinpost/www94/www94.html.

[13] Anawat Chankthod, Peter B. Danzig, Chuck Neerdales, Michael F. Schwartz, and Kurt Worrell. A hierarchical Internet object cache. *In prepartion*, 1995.

[14] Bhavna Chhabra, Darren R. Hardy, Allan Hundhausen, Dave Merkel, John Noble, and Michael F. Schwartz. Integrating complex data access methods into the Mosaic/WWW environment. *Proceedings of the Second International World Wide Web Conference*, pages 909–919, October 1994.

[15] Peter Danzig, Katia Obraczka, Dante DeLucia, and Naveed Alam. Massively replicating services in autonomously managed wide-area internetworks. Technical report, January 1994. Available from ftp://catarina.usc.edu/pub-/kobraczk/ToN.ps.Z.

[16] Peter B. Danzig, Richard S. Hall, and Michael F. Schwartz. A case for caching file objects inside internetworks. *Proceedings of the SIGCOMM '93*, pages 239–248, September 1993.

[17] Peter B. Danzig, Shih-Hao Li, and Katia Obraczka. Distributed indexing of autonomous internet services. *Computing Systems*, 5(4):433–459, Fall 1992.

[18] Peter B. Danzig, Katia Obraczka, and Anant Kumar. An analysis of wide-area name server traffic — a study of the domain name system. *Proceedings of the SIGCOMM Symposium*, pages 281–292, August 1992.

[19] Peter Deutsch and Alan Emtage. *Publishing Information on the Internet with Anonymous FTP.* Bunyip Information Systems Inc., May 1994. Available from ftp://nri.reston.va.us/internet-drafts/draft-ietf-iiir-publishing-01.txt.

[20] Alan Emtage and Peter Deutsch. Archie - an electronic directory service for the Internet. *Proceedings of the USENIX Winter Conference*, pages 93–110, January 1992.

[21] Steve Foster. About the Veronica service, November 1992. Electronic bulletin board posting on the comp.infosystems.gopher newsgroup.

[22] G. W. Furnas, Thomas K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM*, 30(11):964–971, November 1987.

[23] David K. Gifford, P. Jouvelot, Mark A. Sheldon, and Jr. James W. O'Toole. Semantic file systems. *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, pages 16–25, October 1991.

[24] Richard Golding and Darrell D. E. Long. Quorum-oriented multicast protocols for data replication. Technical report, June 1991.

[25] Object Management Group. Common Object Request Broker: Architecture and specification. Technical report, Framingham, Massachusetts, 1991.

[26] James D. Guyton and Michael F. Schwartz. Experiences with a survey tool for discovering Network Time Protocol servers. *Proceedings of the USENIX Summer Conference*, pages 257–265, June 1994.

[27] Darren R. Hardy and Michael F. Schwartz. Harvest user's manual. Technical report, February 1995. Version 1.1.

[28] Darren R. Hardy and Michael F. Schwartz. Customized information extraction as a basis for resource discovery. Technical Report CU-CS-707-94, March 1994. To appear, *ACM Transactions on Computer Systems*.

[29] John Howard, Michael Kazar, Sherri Menees, David Nichols, M. Satyanarayanan, Robert Sidebotham, and Michael West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[30] Microsoft Inc. *OLE 2.01 Design Specification*. Microsoft OLE2 Design Team, September 1993. Describes the Object Linking & Embedding environment.

[31] Brewster Kahle and Art Medlar. An information system for corporate users: Wide Area Information Servers. *ConneXions - The Interoperability Report*, 5(11):2–9, November 1991. Available from ftp://think.com/wais-/wais-corporate-paper.text.

[32] Martijn Koster. Guidelines for robot writers. Technical report, 1994. Available from http://web.nexor.co.uk-/mak/doc/robots/guidelines.html.

[33] Martijn Koster. *Introduction to ALIWEB*. NEXOR, Limited, 1994. Available from http://web.nexor.co.uk-/public/aliweb/doc/introduction.html.

[34] Leslie Lamport. *LaTeX: A Document Prepartion System*. Addison Wesley, Reading, Massachusetts, 1986.

[35] Ari Luotonen, Henrik Frystyk, and Tim Berners-Lee. CERN HTTPD public domain full-featured hypertext/proxy server with caching, 1994. Available from http://info.cern.ch/hypertext/WWW/Daemon-/Status.html.

[36] Udi Manber and Sun Wu. Glimpse: A tool to search through entire file systems. *Proceedings of the USENIX Winter Conference*, pages 23–32, January 1994.

[37] MARBI, Network Development, and MARC Standards Office. *The USMARC Formats: Background and Principles*. 1989.

[38] Oliver McBryan. Genvl and WWWW: Tools for taming the Web. *Proceedings of the First International World Wide Web Conference*, May 1994. Available from http://www.cs.colorado.edu/home/mcbryan/mypapers-/www94.ps.

[39] Mark McCahill. The Internet Gopher: A distributed server information system. *ConneXions - The Interoperability Report*, 6(7):10–14, July 1992.

[40] Lee McLoughlin. FTP mirroring software. Available from ftp://src.doc.ic.ac.uk/package/mirror.shar, August 1991.

[41] Jeffrey C. Mogul and Venkata N. Padmanabhan. Improving WWW latency. *Second International World Wide Web Conference*, October 1994.

[42] David A. Nowitz and Michael E. Lesk. *A Dial-Up Network of UNIX Systems*. Bell Laboratories, Murray Hill, New Jersey, August 1978.

[43] Brian Pinkerton. The WebCrawler. Technical report, 1994. Available from http://www.biotech.washington.edu-/WebCrawler/WebCrawler.html.

[44] Jon Postel. RFC 768: User Datagram Protocol. Technical report, August 1980.

[45] Jon Postel and Joyce Reynolds. RFC 959: File Transfer Protocol (FTP). Technical report, October 1985.

[46] Ronald L. Rivest. Rfc 1321: The MD5 message-digest algorithm. Technical report, MIT Laboratory for Computer Science, Cambridge, Massachusetts and RSA Data Security, Inc, April 1992.

[47] Michael F. Schwartz, Alan Emtage, Brewster Kahle, and B. Clifford Neuman. A comparison of Internet resource discovery approaches. *Computing Systems*, 5(4):461–493, Fall 1992.

[48] Mark A. Sheldon, Andrzej Duda, Ron Weiss, Jr. James W. O'Toole, and David K. Gifford. Content routing for distributed information servers. *Proceedings of the FOURTH International Conference on Extending Database Technology*, March 1994. Queryable via http://paris.lcs.mit.edu/Projects/CRS/content-router.html.

[49] Chris Weider, Jim Fullton, and Simon Spero. Architecture of the WHOIS++ index service. Technical report, November 1992. Available from ftp://nri.reston.va.us/internet-drafts/draft-ietf-wnils-whois-00.txt.

[50] Sun Wu and Udi Manber. Fast text searching allowing errors. *Communications of the ACM*, pages 83–91, October 1992.

Acknowledgements

This paper is based on work presented in part in an earlier conference paper [8].

This work was supported in part by the Advanced Research Projects Agency under contract number DABT63-93-C-0052. Bowman was also supported in part by the National Science Foundation under grants CDA-8914587 and CDA-8914587AO2 and an equipment grant from Sun Microsystems, Inc. Danzig was also supported in part by the Air Force Office of Scientific Research under Award Number F49620-93-1-0082, and by a grant from Hughes Aircraft Company under NASA EOSDIS project subcontract number ECS-00009, and by National Science Foundation Institutional Infrastructure Grant Number CDA-921632. Manber was also supported in part by the National Science Foundation under grant numbers CCR-9002351 and CCR-9301129. Schwartz was also supported in part by the National Science Foundation under grant numbers NCR-9105372 and NCR-9204853, an equipment grant from Sun Microsystems' Collaborative Research Program, and from the University of Colorado's Office of the Vice Chancellor for Academic Affairs.

The information contained in this paper does not necessarily reflect the position or the policy of the U.S. Government or other sponsors of this research. No official endorsement should be inferred.

We would like to acknowledge the efforts of the students who have participated in this project: Rajini Balay, William Camargo, Anawat Chankhunthod, Bhavna Chhabra, Gabe Dalbec, Dante De Lucia, Chanda

Dharap, Burra Gopal, James Guyton, Allan Hundhausen, Paul Klark, Shih-Hao Li, Cheng-Che Lue, Dave Merkel, Chuck Neerdaels, John Noble, John Noll, Katia Obraczka, Mark Peterson, Erh-Yuan Tsai, and Kurt Worrell.

We thank all the people who helped beta test Harvest, and who have ported Harvest to other platforms, especially Billy Barron, Shirley Browne, Brad Burdick, Chris Dalton, Paul Everitt, Martin Greving, Yvan Leclerc, Carl Malamud and Stephanie Nile.

Author Information

C. Mic Bowman is a Member of Technical Staff at Transarc Corporation. He received his Ph.D in Computer Science from the University of Arizona in 1990. From 1990 to 1994 he was an assistant professor at the Pennsylvania State University. His research interests include descriptive naming systems for local and wide-area file systems, structured file systems, resource discovery, and protocols for remote computing. Bowman can be reached at mic@transarc.com.

Peter B. Danzig is an Assistant Professor of Computer Science at the University of Southern California. He received his Ph.D in Computer Science from the University of California, Berkeley in 1990. His current research addresses on the measurement and performance debugging of Internet services, distributed system architectures for resource discovery, and flow and admission control for packet communication networks. Danzig can be reached at danzig@usc.edu.

Darren R. Hardy is a Professional Research Assistant in the Computer Science Department at the University of Colorado. He received his M.S. in Computer Science from the University of Colorado, Boulder in 1993. He specializes in network resource discovery, distributed systems, and information retrieval. Hardy can be reached at hardy@cs.colorado.edu.

Udi Manber is a Professor of Computer Science at the University of Arizona. He received his Ph.D in Computer Science from the University of Washington in 1982. His research interests include design of algorithms, pattern matching, computer networks, and software tools. Manber can be reached at udi@cs.arizona.edu.

Michael F. Schwartz is an Associate Professor of Computer Science at the University of Colorado. He received his Ph.D in Computer Science from the University of Washington in 1987. His research focuses on international-scale networks and distributed systems. Schwartz chairs the Internet Research Task Force Research Group on Resource Discovery (IRTF-RD), which built the Harvest system. Schwartz can be reached at schwartz@cs.colorado.edu.

Duane P. Wessels is a Professional Research Assistant in the Computer Science Department at the University of Colorado. He received his M.S. in Telecommunications from the University of Colorado, Boulder in 1995. He specializes in network resource discovery and caching. Wessels can be reached at wessels@cs.colorado.edu.