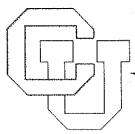
# Harvest: A Scalable, Customizable Discovery and Access System

C. Mic Bowman
Peter B. Danzig
Darren R. Hardy
Udi Manber
Michael F. Schwartz

CU-CS-732-94



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

# Harvest: A Scalable, Customizable Discovery and Access System

C. Mic Bowman
Peter B. Danzig
Darren R. Hardy
Udi Manber
Michael F. Schwartz

CU-CS-732-94 July 1994

University of Colorado at Boulder

Technical Report CU-CS-732-94
Department of Computer Science
Campus Box 430
University of Colorado
Boulder, Colorado 80309

# Harvest: A Scalable, Customizable Discovery and Access System

C. Mic Bowman
Transarc Corp.
mic@transarc.com

Peter B. Danzig
University of Southern California
danzig@usc.edu

Darren R. Hardy
University of Colorado - Boulder
hardy@cs.colorado.edu

Udi Manber
University of Arizona
udi@cs.arizona.edu

Michael F. Schwartz
University of Colorado - Boulder
schwartz@cs.colorado.edu

July 1994

**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 a 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 Mosaic and with HTTP, FTP, and Gopher information resources. We discuss the design and implementation of each subsystem, and provide measurements indicating that Harvest can reduce server load, network traffic, and index space requirements significantly compared with previous indexing systems. We also discuss a half dozen indexes we have built using Harvest, underscoring both the customizability and scalability of the system.

# 1 Introduction

Over the past few years a progression of Internet publishing tools have appeared. Until 1992, FTP [43] and NetNews [39] were the principal publishing tools. Around 1992, Gopher [38] and WAIS [31] gained popularity because they simplified network interactions and provided better ways to navigate through information. With the introduction of Mosaic [2] in 1993, publishing information on the World Wide Web [3] 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 grows, it is 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 structured, complex data. For a more detailed discussion of these problems, the reader is referred to [10].

In this paper we introduce a system that addresses these problems using a variety of techniques. We call the system *Harvest*, to connote its focus on reaping the growing crop of Internet information. Harvest supports resource discovery through topic-specific content indexing made possible by a very efficient distributed information gathering architecture. It resolves 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 the Harvest system, including a variety of performance measurements. In Section 4 we offer several demonstrations of Harvest, including WWW pointers where readers can try these demonstrations. In Section 5 we discuss work in progress, and in Section 6 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/menu name indexes of widely distributed information (such as Archie [18], Veronica [19], or WWWW [37]); and full content indexes of individual sites (such as Gifford's Semantic File System [21], WAIS [31], and local Gopher [38] 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 are less useful as the information space grows (causing queries to match too much information). In contrast to full content indexes that support

powerful queries but require so much space that they cannot hold more than a few sites' worth of data, Harvest achieves space efficient, content indexes of widely distributed data.

Recently, a number of efforts have been initiated to create content indexes of widely distributed sites (e.g., Gifford's Content Router [45] and Pinkerton's WebCrawler [40]). 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 3.1.

The WHOIS and Network Information Look Up Service Working Group in the Internet Engineering Task Force is defining an Internet standard called "WHOIS++" which gathers concise descriptions (called "centroids") of each indexed database [46]. 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++.

Many of the ideas and some of the code for Harvest were derived from our previous work on the agrep string search tool [47], the Essence customized information extraction system [26], the Indie distributed indexing system [15], and the Univers attribute-based name service [12].

### 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 [28] and ISIS [5]). More recently, a number of efforts have begun to build object caches into HTTP servers (e.g., Lagoon [41]), and to support replication [22] in Internet information systems such as Archie.

In contrast to the flat organization of existing Internet caches, Harvest supports a hierarchical architecture 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 [14]. 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 [39] 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 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 [23] and OLE [29]), 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 The Harvest System

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, and HTTP). Bold boxes indicate excessive load being placed on servers, primarily because the indexing systems gather information from Providers that fork a separate process for each retrieved object. 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.

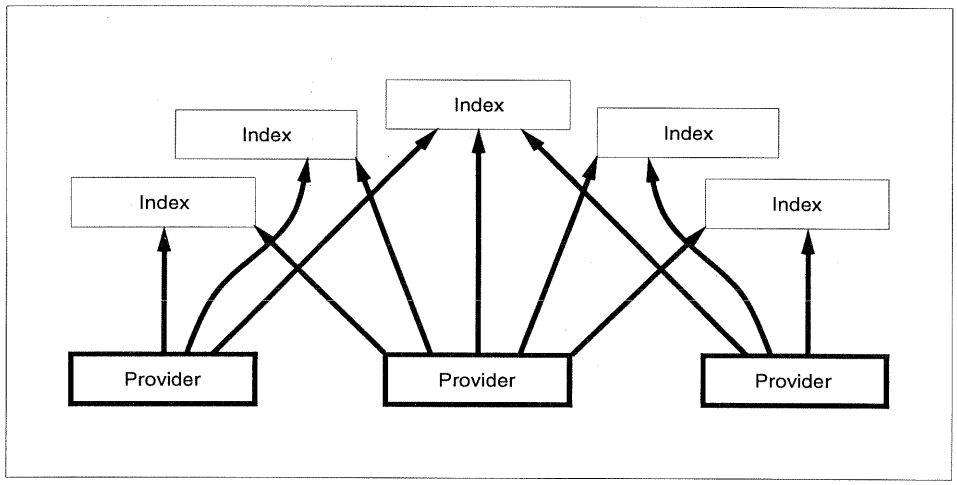



Figure 1: Inefficiencies Caused by Uncoordinated Gathering of Information from Object Retrieval Systems

Figure 2 illustrates the Harvest approach to information gathering. A *Gatherer* collects indexing information from a Provider, while a *Broker* provides an indexed 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 the network and servers than the approach illustrated in Figure 1. The leftmost part of Figure 2 shows a Gatherer accessing a Provider from across the network (using the native FTP, Gopher, or HTTP protocols). This arrangement mimics the configuration shown in Figure 1, and is primarily useful for interoperating with systems 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.

In the second part of Figure 2, the Gatherer is run at the Provider site. As suggested by the lack of bold boxes and lines in this part of the figure, doing this 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 3.1.

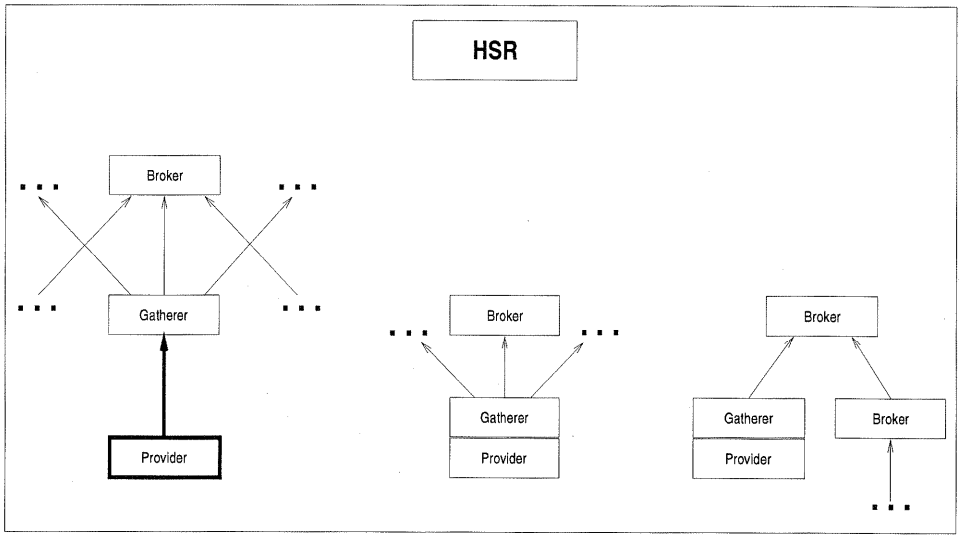



Figure 2: Harvest Information Gathering Approach

The ellipses in Figure 2 indicate that a Broker can collect information from many Gatherers (to build an index of widely distributed information) and that 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/refine the information from one Broker to the next.

The final element illustrated in Figure 2 is the *Harvest Server Registry (HSR)*. This is a distinguished Broker instance that registers information about each Harvest Gatherer, Broker, Cache, and Replicator in the Internet. The HSR is useful when constructing new Gatherers and Brokers, to avoid duplication of effort. It can also be used when looking for an appropriate Broker at search time.

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 [20] and scaling problems of unfocused global indexes (such as Archie and WWWW).

Gatherers and Brokers communicate using an attribute-value stream protocol called the Summary Object Interchange Format (SOIF) [9], 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 [17] and BibTeX [33]. Each template contains a type, a Uniform Resource Locator (URL) [4], and a list of byte-count delimited field name/field 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].

```
@DOCUMENT{ ftp://ftp.cs.psu.edu/pub/techreps/TR123.ps
Title{28}: Type Structured File Systems
Author-Name{33}: C. Dharap, R. Balay and M. Bowman
Author-Organization-Type{11}: educational
Author-Department{32}: Computer Science and Engineering
Publication-Status{19}: Accepted, To appear.
Keywords{42}: types object-oriented semantic filesystems
Access-Protocol-v*{3}: FTP
Description{457}: The paper describes the design of Nebula.  Nebula is a dynamic,
        object based typed file system.  Nebula uses types to create well
        structured files as well as to export existing files in the internet
        environment.  This allows for backward scalability.  Files in Nebula are
        abstractions and are defined by objects.  Nebula assumes a flat global
        namespace and operations are provided to manipulate logical namespaces.
}
```

Figure 3: Example SOIF Template

Figure 4 illustrates the Harvest architecture in more detail, showing the Gatherer/Broker interconnection arrangements and SOIF protocol, the internal components of the Broker (discussed in Section 3.2), and the caching, replication, and client interface subsystems.

The *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 *Object Cache* reduces network load, server load, and response latency when accessing located information objects. Additionally, it will be used for caching *access methods* in the future, when we incorporate an object-oriented access mechanism into Harvest. At present Harvest simply retrieves and displays objects using the standard "hard coded" Mosaic procedures, but we are extending the system to allow publishers to associate access methods with objects (see Section 5). Given this support, it will be possible to perform much more flexible display and access operations.

We discuss each of the Harvest subsystems below.

### 3.1 The Gatherer Subsystem

As indexing the Web becomes increasingly popular, two types of problems arise: data collection inefficiencies and duplication of coding effort. Harvest addresses these problems by providing an efficient and flexible system upon which to build many different indexes. Rather than building

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

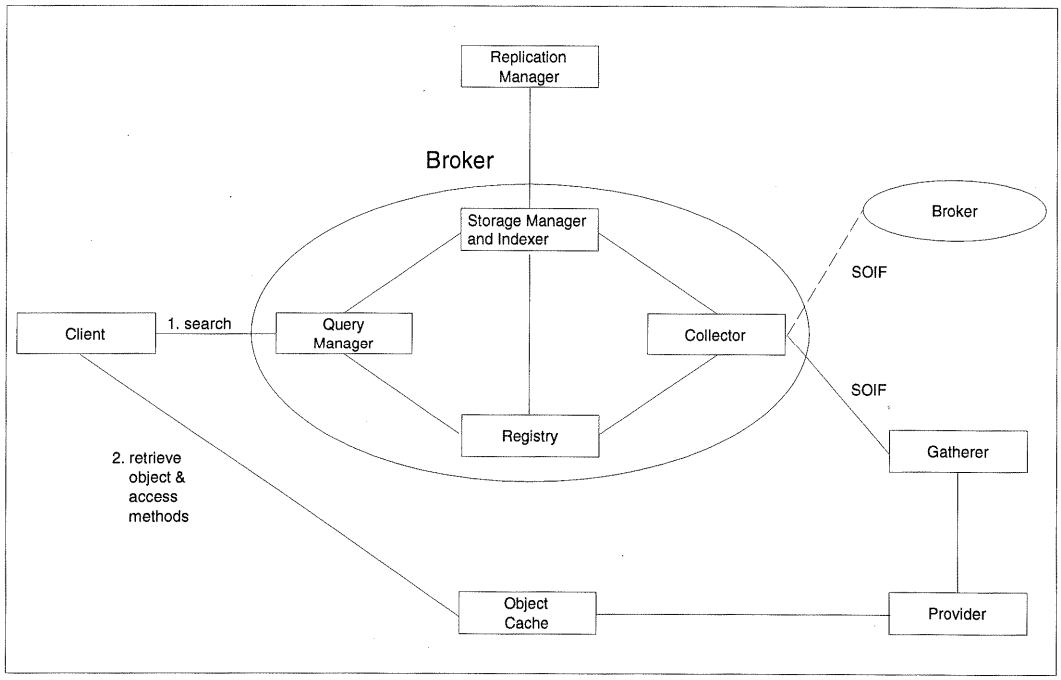



Figure 4: Harvest System Architecture

one indexer from scratch to collect WAIS headlines, 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 server load, network traffic, and index space.

### Efficient Indexing Support

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. Rather than working around the existing shortcomings, we believe a better approach is to provide a much more efficient index collection system.

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 HTML links. This causes a great deal of load, because each object retrieval results in 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.

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 cache of indexing information, so that separate traversals are not required for each request. [2] More

[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.

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 by several orders of magnitude.

As a simple quantification of these savings, 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. For a more precise quantification we could have measured CPU, memory, and disk resource usage, but measuring elapsed time provides a first order approximation of overall load for each of the gathering methods. 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 6.99 times less load than gathering via FTP. More importantly, retrieving *an entire stream* of records from the Gatherer's cache incurs 2.72 times less server load than retrieving a *single object* via FTP (and 2.81 times less load for retrieving all objects at a site, because of a common-case optimization we implemented that keeps a compressed cache of this information around). Thus, assuming an average archive site size of 2,300 files [3], 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.

The Gatherer also reduces network traffic substantially, based on three techniques. First, it uses the Essence system [26] to extract *content summaries* before passing the data to a remote Indexer. 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 current robots might retrieve a set of HTML documents and extract anchors and URLs from each, Essence extracts content summaries at the Provider site, often reducing the amount of data to be transmitted by a factor of 20-50 or more. [4] Beyond generating content summaries at the Provider site, the Gatherer also allows Brokers to request only those summaries that have changed since a specified time, and transmits the response stream in a compressed form. [5] The combination of pre-filtering, incremental updates, and compression can reduce network traffic by several orders of magnitude.

As an example of these savings, our content index of Computer Science technical reports (see Section 4) starts with 2.44 GB of 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 and building a full-text index. The network transmission savings are actually higher still if one considers the number of network links that must be traversed by each byte of data. For the above example, we used traceroute [30] to count hops to each Provider site. In the measured example, the average hop count was 18, magnifying the importance of network transmission savings. This savings corresponds to running local Gatherers at each Provider site, rather than having to pull full objects across the

[3] We computed this average from measurements of the total number of sites and files indexed by Archie [44].

[4] The reduction factor depends on data type. For example, the reduction for PostScript files is quite high, while the reduction for HTML files is fairly low.

[5] For access methods that do not support time stamps (like the current version of HTTP), the Gatherer extracts and compares "MD5" cryptographic checksums of each object before transmitting updates.

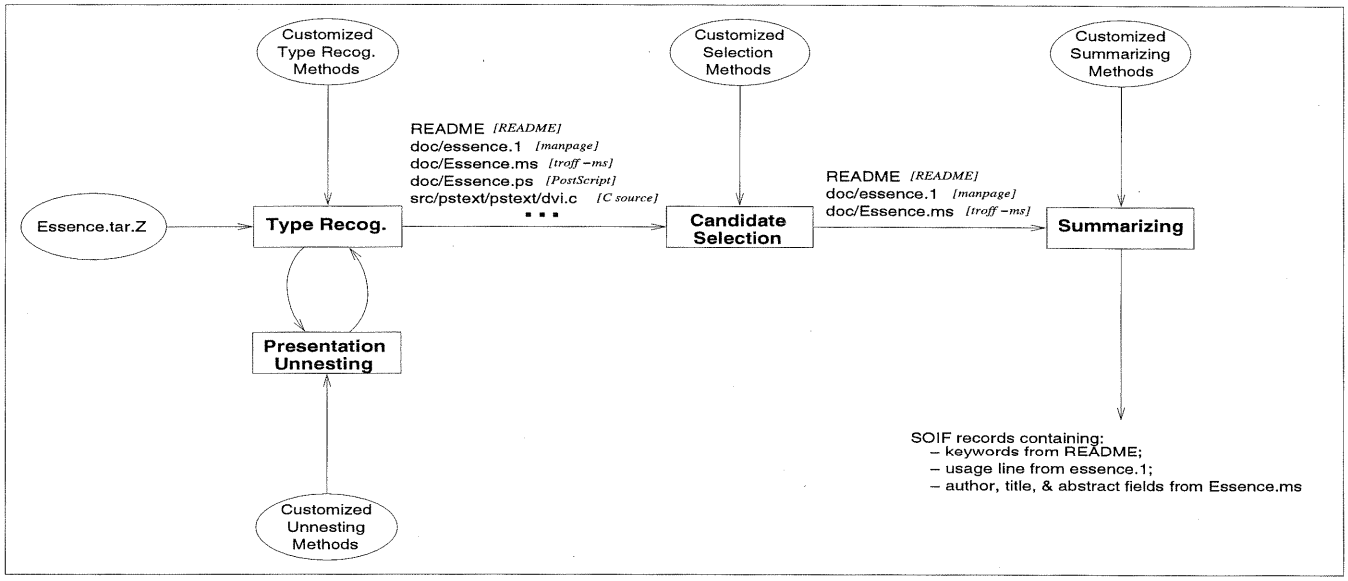



**Figure 5**: Example of Essence Customized Information Extraction

Internet before summarizing at a central site. 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 [6] 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 [27]. For users who need the added recall and precision, however, Essence also supports a full-text extraction option.

### Flexibility of Index Collection

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.

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, it enumerates FTP objects using a recursive directory listing, and enumerates HTTP objects by

[6] 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 [7].

recursively extracting the URL links that point to other HTML objects within the same server as the original RootNode URL. [7]

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 edited* 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 [17]. The Gatherer stores the manual and automatically generated information separately, so that periodic automated summarizing will not overwrite manually created information.

The other type of flexibility supported by the Gatherer is in the placement of Gatherer processes. Ideally, a Gatherer will be placed at each Provider site, to support efficient data gathering. Because this requires cooperation from remote sites, we also allow Gatherers to be run remotely, accessing objects using FTP/Gopher/HTTP. As an optimization, we are currently building a translator that will translate FTP/Gopher/HTTP URLs to local file system references if the Gatherer is run locally. Doing so will provide a factor of 6.99 reduction in server load while gathering locally, as indicated by the server load measurements discussed above.

Because remote gathering imposes so much load, we implemented a *connection caching* mechanism, which attempts to keep connections open across individual object retrievals.[8] When the available UNIX file descriptors are exhausted (or timed out with inactivity), they are closed according to a replacement policy. This change resulted in a 700% performance improvement for FTP transfers.

### 3.2 The Broker Subsystem

As illustrated in Figure 4, the Broker consists of four software modules: the Collector, the Registry, the Storage Manager, 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 unique identifier and a time-to-live. The unique object identifier (OID) consists of the resource's URL plus information about the Gatherer that generated 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 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.

[7] This restriction prevents a misconfigured Gatherer from traversing huge parts of the Web. As a second limitation, Gatherers refuse to enumerate RootNode URLs beyond a (configurable) maximum number of resulting LeafNode URLs.

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

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 Indexer. Finally, 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 and Indexer. 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 Indexer and Storage Manager may become inconsistent with the Registry, periodic garbage collection removes any unreferenced objects from the Storage Manager and Indexer.

The Query Manager exports objects to the network. It accepts a query, translates it into an intermediate representation, and 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 3.3) returns the matching 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 Manager supports remote administration of the Broker's configuration. 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.

## 3.3 The Index and Search Subsystem

To accommodate diverse indexing and searching needs, Harvest defines a general Broker-Indexer 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 could therefore use a variety of different backends inside a Broker, such as WAIS or Ingres (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 [35] supports space-efficient indexes and flexible interactive queries, while Nebula [11] supports fast searches and 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 [47] 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.

Glimpse's support for Boolean queries can be particularly useful when the Gatherer was able to extract attributes from the gathered objects—such as "author's name," "institution," and "abstract". So, for example, it is easy to search for all articles about "incremental indexing" by "unknown author" allowing one spelling error in the author's name. Glimpse also allows filtering by file names. This is important for supporting a combined browse/search paradigm because it allows us to keep only one index and filter the search according to the current directory, transparently to the user.

Harvest's use of Essence content summarizing and Glimpse indexing provides a great deal of space savings compared with WAIS. In Section 3.1 we cited our earlier Essence measurements showing that Essence requires only 3-11% as much index space as WAIS. The situation is more pronounced now, because Essence now generates more space efficient summaries, and because for the original measurements we had indexed the content summaries using WAIS, but we now use Glimpse. Combining these two changes, we compute that Harvest now requires a factor of 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.

### Nebula Index/Search Subsystem

Nebula provides two key features in support of complex, regularly executed, standing queries. First, it represents each indexed object as a set of attribute/value pairs. This approach supports more precise queries than typical keyword-based file indexes can, because users can query the database with Boolean queries based on attributes.

The Nebula search engine can 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 of a document contain the most important and descriptive information about the 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.

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

Unlike Glimpse, Nebula prefers efficient lookup over small indexes. However, 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.

Nebula's second feature in support for complex standing queries is its notion of views. A view is defined by a standing query against the database of indexed objects. Because views exist over time, it is easy to refine and extend them over time, and to observe the affect 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 removing 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 4) 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. The Windows view selects summary objects for Windows programs. The query used to define the Windows view depends on the attributes in the summary objects. If a summary objects contain an explicit classification attribute such as "software-class:pc-software.windows", then the query used to define the view is simply "software-class=pc-software.windows". In practice, the query is more difficult to construct. Currently, we construct the query based on the archive and directory that contains the summarized file.

## 3.4 The User Interface

There are two conflicting goals in designing a user 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 6.

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.

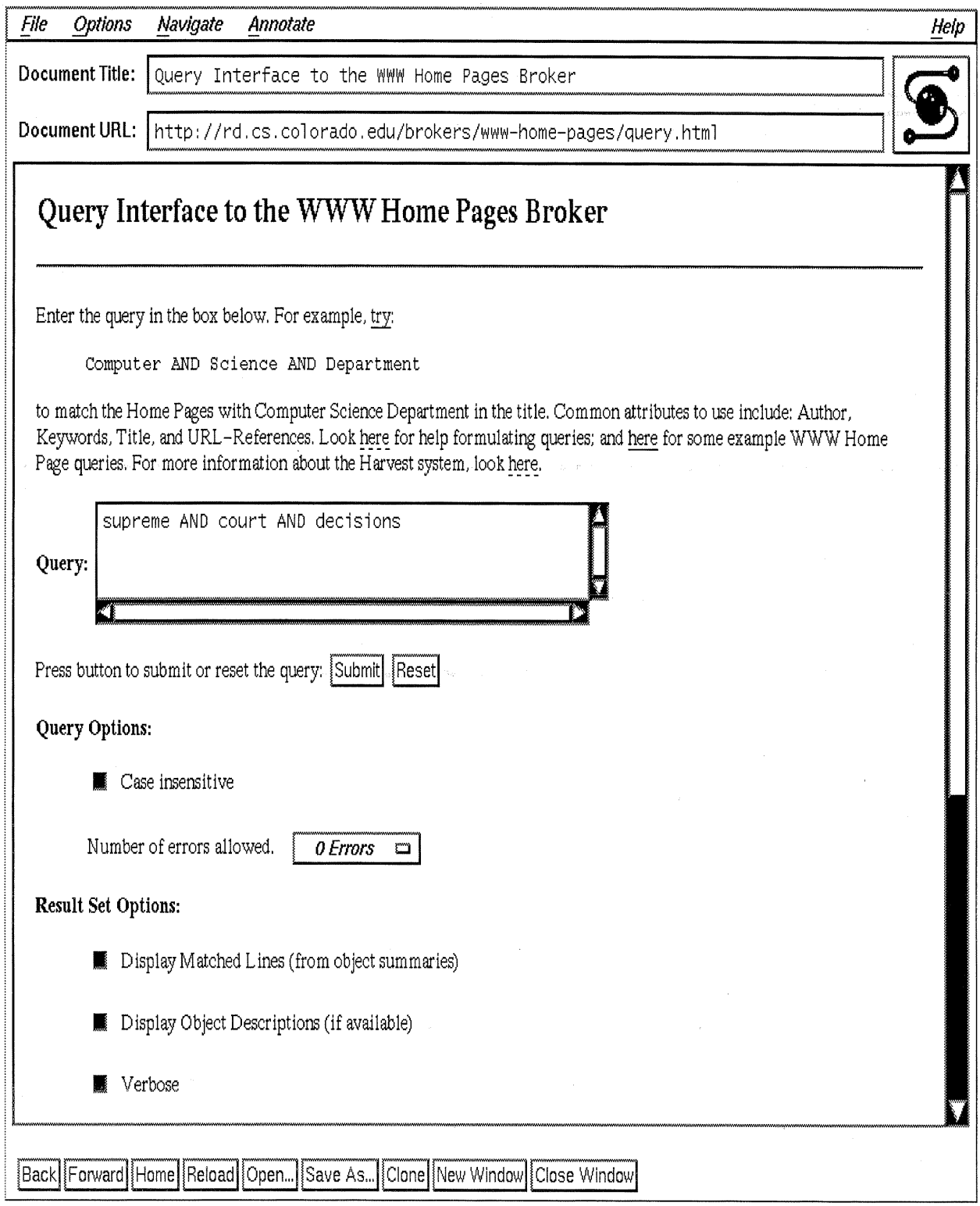



Figure 6: Example Query

Previous queries can be retrieved and refined by moving through the cache. We are currently working on extensions to the Query Manager that will save state between queries.

#### Interface Customization

Harvest supports user interface customization through direct access to each Indexer. 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 Indexer. While this means users must learn multiple interfaces, it also increases flexibility. A domain-specific Indexer 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 Indexer 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.

### 3.5 The Replication Subsystem

Harvest provides a weakly consistent, replicated wide-area file system called *mirror-d*, on top of which Brokers are replicated. Mirror-d itself is layered atop a hierarchical, flooding update-based group communication subsystem called *flood-d* [13].

Each mirror-d instance (or mirror-daemon) in a replication group occasionally floods complete state information to its immediate neighbors, to detect updates that flood-d failed to deliver, possibly due to a long-lasting network partion, site failure, or failure of a flood-d process. Mirror-d implements eventual consistency [6]: if all new updates cease, the replicas eventually converge.

Flood-d logically floods objects from group member to group member along a graph managed by flood-d itself. Each flood-d instance (or flood-daemon) measures the end-to-end network bandwidth achievable between itself and other flood-daemons running at other group member sites. A master site within each group constructs and reliably distributes either a two-connected or a three-connected, low diameter, logical topology of the group members.

A flood-daemon can belong to many groups, making it possible to construct hierarchies of groups and to stitch otherwise independent groups together by sharing two or three common members.

### 3.6 The Object Caching Subsystem

To meet ever increasing demand on network links and information servers, Harvest includes a hierarchically organized Object Caching subsystem, as illustrated in Figure 7.

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, similar to the CERN httpd cache [34]. We also implemented a generic protocol, independent of HTTP. The proxy interface

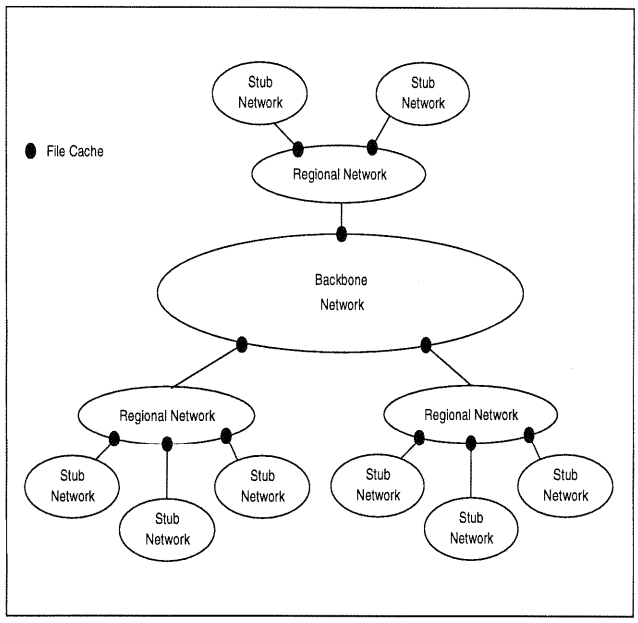



**Figure 7**: Topology-Based Cache Organization

is mostly useful for allowing unmodified Mosaic clients to use the system by simply 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 "inetd" echo port of 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 [42] echo daemon, the object home 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, and FTP objects, we maintain a cache of recent DNS name-to-address mappings to optimize common-case cache behavior.

Some standard (perhaps based on MIME headers [8]) is needed to indicate time-to-live [16] values to objects. Until this standard emerges, our cache will continue to assign default values.

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 [10] for cached objects in virtual memory, to eliminate access latency to the meta-data. Since the cache is single threaded but otherwise non-blocking, page faults are the only source of blocking.

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

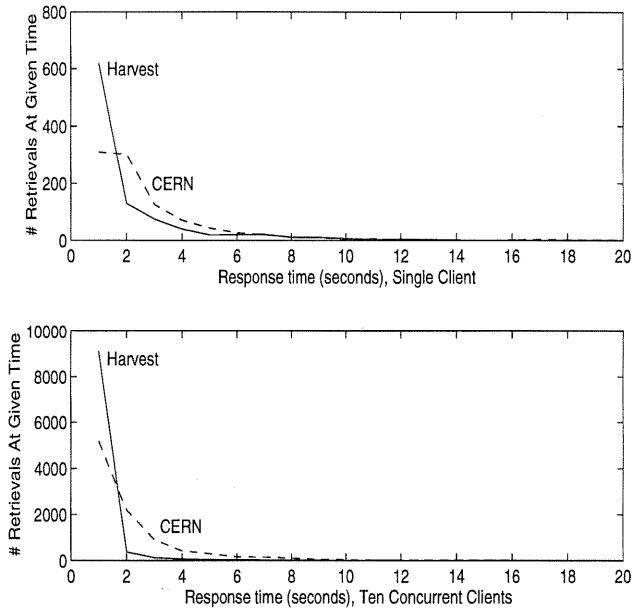



Figure 8: Harvest vs. CERN Cache Performance

### 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 8, the Harvest cache returns 600 of these objects in less than 1 second, while it takes the CERN cache 2 seconds to return 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 8, 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.

## 4 Application Demonstrations

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.

| Feature | Demo Brokers | #Objects | #Sites |
|---|---|---|---|
| Topic-specific indexing; Broker cascading | NIDR-SW, NIDR-Doc, NIDR | 635 | 80 |
| Scalable content indexing; Multiple formats; Gathering distribution | CS Technical Reports | 18,000 | 217 |
| Structured indexing; Indexing customizability | SFI Time Series Papers | 32 | 4 |
| Support for other standards; Incorp. manual indexing info | PC Software Packages | 25,000 | 8 |
| Focused global content index vs. exhaustive name index | WWW Home Pages | 5,000 | 1,761 |
| Locate Harvest servers during search & index config. | Harvest Server Registry | 19 | 3 |

Table 1: Summary of Demonstrated Features of Constructed Indexes

## NIDR Brokers

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

The combined NIDR software + documents Broker gathers 635 objects from 80 sites.

Users can try the NIDR Broker at *http://rd.cs.colorado.edu/brokers/nidr/query.html.*

## Computer Science Technical Report Broker

To demonstrate the scaling benefits of our content summarizing approach, we next 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 18,000 documents with this Broker, gathered from 217 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, [11], the University of Indiana index of Computer Science technical reports [12], and a large Computer Science bibliography maintained at the University of Karlsruhe in Germany[13]. 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.

Our technical report Broker also demonstrates two different types of gathering distribution. First, we divided the list of URLs among Gatherers running at USC, PSU, and Colorado in such a fashion that sites were gathered from whichever Gatherer was closest. [14] Second, this Broker gathers from the NIDR Broker in addition to the above URL lists, representing topical distribution in the gathering process.

Users can try the CS Technical Report Broker at *http://rd.cs.colorado.edu/brokers/cstech/query.html.*

## Time Series Citation Index

To demonstrate Harvest's indexing customizability, we next constructed a citation index of documents that reference the Santa Fe Institute (SFI) time series data [1]. These data consist of several sets of time series, such as measured laser intensity pulsations and records of Swiss/US monetary exchange rate bids. These time series provide a reference set that allows time series researchers to compare the effectiveness of various algorithms in predicting time series.

In addition to supporting Harvest's usual content index of SFI papers, the SFI time series Broker provides a special *SFIRefs* attribute that indicates which time series each paper references. To create these attributes, we built a Perl script that matches each document content summary against approximately 70 regular expression pattern matches, to heuristically determine each document's referenced time series. For example, if the expression "infrared.*laser" matches a document at indexing time, Harvest generates time series "A" attributes in the index, allowing users to locate this document when searching by time series citation. It is also possible to override the indexing-time heuristics with manual updates, to support higher quality attributes when the heuristics do not suffice.

Users can try the SFI time series Broker at *http://rd.cs.colorado.edu/brokers/sfi/query.html.*

## PC Software Broker

Our next Broker demonstrates Harvest's ability to incorporate indexing information in a variety of formats from other sources, including high quality, manually generated information.

There is a fairly large collection of "shareware" and public domain software available at various Internet sites, for which many contributors have manually created structured, conceptual descriptions of each package. Because each site uses a somewhat different format for these files, we used

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

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

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

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

Harvest's customizable extraction features to collect indexing information in site-specific ways, and place this information into uniformly-structured SOIF templates. Note that this Broker is only searchable using the "Description" attribute—no content indexing is supported. This point illustrates the fact that Harvest does not require content summaries to be extracted when building Brokers.

As a result of this effort, we were quickly able to incorporate high-quality indexing information about more than 25,000 software objects into the index. 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 would allow someone to search for "batch programming language", while Archie could only locate this software if someone searched for "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 the Internet Anonymous FTP Archives IETF Working Group (IAFA) [17] 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.

These Brokers are similar to the above PC archive Broker, but contain more attributes. For example, the LSM format contains separate fields for the package description, author, maintainer, copyright policy, and a half-dozen other attributes; the IAFA format contains approximately 100 different fields, including many optional fields.

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 [36].

Users can try the PC software Broker at *http://rd.cs.colorado.edu/brokers/pcindex/query.html.*

### 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 [37]. 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 the "What's New" page maintained by the National Center for Supercomputing Applications' WWW server, McBryan's Mother-of-all-BBS's [37], URLs gleaned from USENET postings, and various other sources. We periodically gather this list and prune it to a set of home pages, which we then gather, content summarize, and index. 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.

Users can try our WWW home page index at *http://rd.cs.colorado.edu/brokers/www-home-pages/query.html.*

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.

Users can try our Computer Science technical report abstract index at *http://glimpse.cs.arizona.edu:1994/bib.*

One could use Harvest to provide the functionality of many other indexes and indexing systems. One could build a WAIS-like system by using the Essence full-text extraction mechanism, and adding a ranking feature to one of the Harvest search engines. (Or, one could use the WAIS code as an index/search inside of a Broker.) One could build a WebCrawler [40] by defining an appropriate Gatherer configuration, and using the Essence full-text extraction mechanism. One could build a WHOIS++ [46] 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 [45]. 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).

# 5 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 sites to run Brokers that register object types and access methods at their site. In this way, objects exported by HTTP/FTP/Gopher can be given explicit types, and each type can have an associated set of programs for manipulating or displaying object contents. We will modify Mosaic to check for this Broker, allow users to select among methods, retrieve the specified code, and run it on a dedicated method server host. 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 to work out [25].

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 [24], 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 [11] 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.

# 6 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 over 1,000, and index space requirements by a factor of 43, compared with previous indexing systems. Harvest also provides a high performance Object Cache that can be arranged hierarchically for scalability reasons.

Harvest interoperates with Mosaic and with HTTP, FTP, and Gopher 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.

We are interested in getting other groups to use Harvest for indexing their information. Interested readers can send electronic mail to harvest-dvl@cs.colorado.edu.

The Harvest software will be available from ftp://ftp.cs.colorado.edu/pub/cs/distribs/harvest around the end of Summer 1994. The system consists of approximately 100,000 lines of code, divided among 33,000 in the Gatherer, 16,000 in the Index/Search system, 5,000 in the Broker, 9,000 in the Cache, 21,000 in the Replicator, and 15,000 in data structure library routines.

## References


[1] Predicting the future and understanding the past: A comparison of approaches. *Proceedings of the NATO Advanced Research Workshop on Time Series Analysis and Forecasting*, Spring 1993. Information available from ftp://ftp.santafe.edu/pub/Time-Series/competition.

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] C. Mic Bowman, Peter Danzig, Darren Hardy, Udi Manber, and Michael F. Schwartz. Harvest protocol and subsystem specifications. Technical report, 1994. In preparation.

[10] 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. Pre-publication version available from ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/RD.ResearchProblems.Jour.ps.Z.

[11] 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. Pre-publication version available from ftp://ftp.cse.psu.edu/pub/bowman/doc/iims.ps.Z.

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

[13] 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.

[14] 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. Pre-publication version available from ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/FTP.Caching-PS.

[15] Peter B. Danzig, Shih-Hao Li, and Katia Obraczka. Distributed indexing of autonomous internet services. *Computing Systems*, 5(4):433–459, Fall 1992. Pre-publication version available from ftp://catarina.usc.edu/pub/danzig/jcs.ps.

[16] 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. Pre-publication version available from ftp://catarina.usc.edu/pub/danzig/dns.ps.Z.

[17] 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.

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

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

[20] 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.

[21] 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.

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

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

[24] 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. Pre-publication version available from ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/NTP.Discovery.ps.Z.

[25] Darren Hardy, Allan Hundhausen, Dave Merkel, John Noble, and Michael F. Schwartz. *Integrating Complex Data Access Methods into the Mosaic/WWW Environment.* Department of Computer Science, University of Colorado, Boulder, Colorado, 1994. In preparation.

[26] Darren Hardy and Michael F. Schwartz. Essence: A resource discovery system based on semantic file indexing. *Proceedings of the USENIX Winter Conference*, pages 361–374, January 1993. Pre-publication version available from ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Essence.Conf.ps.Z.

[27] Darren R. Hardy and Michael F. Schwartz. Customized information extraction as a basis for resource discovery. Technical report, March 1994. Submitted for publication.

[28] 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.

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

[30] Van Jacobsen. Traceroute software, December 1988. Available from ftp://ftp.ee.lbl.gov/pub/traceroute.tar.Z.

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

[34] 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.

[35] 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. Pre-publication version available from ftp://cs.arizona.edu/reports/1993/TR93-34.ps.Z.

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

[37] 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.

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

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

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

[41] Reinier Post. *"README" File in Lagoon Caching Software Distribution.* Eindhoven University of Technology, 1994. Available from ftp://ftp.win.tue.nl/pub/infosystems/www/README.lagoon.

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

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

[44] 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. Pre-publication version available from ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/RD.Comparison.ps.Z.

[45] 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.

[46] 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.

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

## Acknowledgements

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 the NASA EOSDIS project, 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, and an equipment grant from Sun Microsystems' Collaborative Research Program.

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 thank Brad Burdick and Carl Malamud at the Internet Multicasting Service for their help with porting Harvest to Solaris.

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, and Kurt Worrell.

## 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.