The *refdbms* distributed bibliographic database system

Richard A. Golding†
Vrije Universiteit, Amsterdam, The Netherlands

Darrell D. E. Long‡
University of California, Santa Cruz

John Wilkes
Hewlett-Packard Laboratories, Palo Alto, California

Abstract

Refdbms is a database system for sharing bibliographic references among many users at sites on a wide-area network such as the Internet. This paper describes our experiences in building and using *refdbms* for the last two years. It summarizes the collection of facilities that *refdbms* provides, and gives detailed information on how well *refdbms* functions as a collaborative, wide-area, distributed information system.

1 Introduction

A bibliography in a paper serves two purposes: it acknowledges prior work, and it provides pointers to related efforts for readers. Useful bibliographies are accurate and complete, and hard work for a single person to generate. *Refdbms* is a system that assists this process. Bibliographies (like the one in this paper) can be generated easily and quickly, and their contents assured of high degrees of correctness.

Refdbms provides tools to add new entries, edit existing ones, search for interesting entries by keyword, and format the resulting bibliographies for LaTeX and Frame Maker documents. Each bibliographic entry is part of a reference database; any number of databases can be accessed together. Each database can be maintained by a disjoint group of people, but the combined databases are available for searching and retrieval. Several people, who may be physically dispersed, can access or update a single database.

Refdbms is not a single, monolithic system; rather, it consists of a set of tools that can be used to integrate it with other systems, such as text formatters and text retrieval tools. In this way the bibliographic databases can be used throughout the process of conducting scholarly research and writing.

In addition, *refdbms* databases can be replicated at many sites, providing reliable access to their contents even in the face of unreliable network connections, such as the world-wide Internet, or portable computers with only intermittent connectivity. Users at any of the sites can retrieve entries, submit new ones, or augment existing ones. All this is accomplished by exploiting a new class of weak-consistency protocols.

By comparison to other bibliography-maintenance tools, *refdbms* provides better search facilities, and roughly comparable output formatting facilities. Unlike them, however, its distributed, scalable nature means that several people can collaborate on the arduous task of generating – and keeping up to date – a bibliography pool of relevant and interesting items.

Refdbms is an example of a *collaborative* information system, by comparison to other wide-area information systems that use a *publishing* metaphor [Schwartz92]. It is also better integrated with the process of research and writing than most wide-area systems, and provides better availability and performance. At present, *refdbms* provides for either open, public collaboration, or private local use, though many other options are possible.

---

† Supported in part while at the University of California by a graduate fellowship from the Santa Cruz Operation.

‡ Supported in part by the National Science Foundation under Grant NSF CCR-9111220 and by the Office of Naval Research under Grant N00014–92–J–1807.

The first versions of the *refdbms* system were written at Hewlett-Packard Laboratories [Wilkes91]. Over several years, this became the *refdbms version 1* system, which provided a bibliographic database shared within a single research group. In 1990, an effort began at U.C. Santa Cruz to build a portable version, as a first step toward using it as a prototype of a wide-area information system. A first version of this wide-area distributed system (known as *refdbms version 3*) was released in November 1992. Development since then has continued at U.C. Santa Cruz and the Vrije Universiteit Amsterdam to improve and extend the prototype.

Refdbms is publicly available software. It is currently in use at several sites, supporting at least 40 world-wide databases that contain nearly 14,000 computer-science-related bibliography entries.

The remainder of the paper is organized as follows. We begin with an overview of the *refdbms* functions, and compare these against related bibliography systems. In Section 3 we discuss the internal architecture of the system, how the pieces are connected, and how additional tools can be used with the system. We follow this in Section 4 by an analysis of the performance of the system in practice. In Section 5 we report some of the lessons we have learned and how we are using them to develop the next version of *refdbms*.

2 Services

Refdbms can generate bibliographies for documents from embedded citation information – for example, the bibliography for this paper was built using the system. Like Harrison and Munson [Harrison89], we believe that a good bibliographic database system should be integrated with the writing process, and should support other aspects of academic research. To this end, *refdbms* can also be used as a browsing tool to find interesting papers to read, it can be used for maintaining reading lists, and it has been used to manage technical report libraries.

Refdbms supports four general types of operations: storing, retrieving, sharing, and using references. Users can both retrieve information and update it, so that the effort of maintaining a database can be shared among them, and so they can evolve the database to meet their needs. Since we are concerned with geographically dispersed users, each database can be replicated at multiple sites, and updates made to one replica are propagated to other replicas.

In this section we briefly discuss these facilities. We start by defining references and databases, and then discuss how references are retrieved, modified, shared, and used.

2.1 References and databases

Refdbms maintains a collection of databases, each containing a set of references relevant to some topic. Each database has a name that is unique at the site. Table 1 lists some of the databases available. A site will usually maintain replicas of several databases, some of which will be publicly readable by every user on the site, and some of those databases will be shared with other sites as well. Users can also create their own databases, which are protected from other users to the degree of the permissions on the files storing the database. Each database must have a name that is unique within the site.

Refdbms stores references in a format reminiscent of that used by refer [Lesk78], with influence from BibTeX [Lamport85], as shown in Figure 1. A reference consists of a sequence of lines, each beginning with %⟨letter⟩⟨space⟩. The letter indicates the *field* of which the line is a part. Some fields, such as the title (%T), consist of only a single line. Other fields, such as the list of authors (%A), consist of repeated lines. Finally, some fields, such as the extract (%x), are logically a single value spread over multiple lines.

Every reference has a *type,* such as TechReport or Article. This is indicated in the %z field, which must be the first line of the reference. Each reference also has a *tag,* such as Lamport78a, which is a unique name for the reference within its database. The tag is stored in the %K field, which must be the second line in the reference. Other fields, such as the title, authors, location, or date, may or may not be present depending on the type of the reference.

Refdbms allows *abbreviations* in a reference. An abbreviation is a sequence of letters and digits ending in a period, such as CACM. in the example in Figure 1. Most abbreviations are specific to a particular set of fields – for example, CACM. expands to Communications of the ACM only in a journal name (%J) field. Control files allow abbreviations to be expanded to a greater or lesser degree. Two control files are provided with *refdbms*, for full and minimal expansion, and it is easy to add a control file to conform to the particular conventions of a target journal or publisher.

The system enforces a few syntactic conventions on references. These include having the reference's type and tag as the first and second lines, and each type of reference is required to have some minimal set of fields (such as title, publisher, or author.) Some additional rules about formatting dates and names are enforced so every reference can be uniformly translated by simple programs.

TABLE 1: Some of the databases available in October, 1993. Several organizations have made available existing bibliographies used in their research.

Database	References	Contents
hpdb	3412	General computer systems bibliography from HP Labs.
usenix	1337	Many Usenix Association publications through 1989.
parallel.miya	1274	Eugene Miya's parallel systems bibliography.
theory	1007	General references on the theory of computing.
inria.ooos	965	Object-oriented operating systems bibliography from INRIA.
ucsc.grad.os	959	A canonical reading list for graduate OS courses; also contains references collected by students for term projects.
comp.doc.techreports	853	Technical reports announced on comp.doc.techreports.
logicprog	851	General bibliography on logic programming.
docbib	785	Database on document analysis and understanding.
dartmouth.io	234	References on parallel I/O systems, from Dave Kotz at Dartmouth.
inria.stochastic	188	Stochastic processes, from INRIA.
jacm	166	Selected articles from the Journal of the ACM.
x11bib	160	A collection of papers on the X Window System.
comp.os.research	148	Collected bibliographies from comp.os.research.
ieee.tcos	142	Contents of the IEEE TCOS (Technical Committee on Operating Systems) newsletter.
parallel	131	General bibliography on parallel computation.
compression	125	General bibliography on data compression.
ai	117	General bibliography on AI.
reinas	112	The REINAS project (UC Santa Cruz and MBARI).
inria.cache	104	Caching bibliography from INRIA.
inria.risc	99	RISC processor bibliography from INRIA.
crypt	90	General bibliography on cryptography.
comp.parle	77	Papers from PARLE '92 (Parallel Architectures and Languages Europe).
inria.bin-packing	77	General bibliography on bin packing, from INRIA.
compilers	75	General bibliography on compilers.
mitl.migration	53	References on process migration, from Fred Douglis at MITL.
ucsc.tr	53	Technical reports from UC Santa Cruz.
concurrent	48	General bibliography on programming current systems.
chorus	31	References on the Chorus operating system.
continmedia	27	General bibliography on continuous (multi-) media.
ucsb.cs.tr	15	Computer Science technical reports from the University of California at Santa Barbara.
hashing	14	General bibliography on hashing algorithms.
graphics	8	General references on computer graphics.
rfc	6	Selected Internet Requests for Comments.
Total	13743	

A reference tag is a unique name for the reference within a database. However, it is also supposed to be mnemonic and user-specified. Since two users could simultaneously try to add two different references with the same tag, the system internally uses a machine-generated unique identifier consisting of a host address and timestamp for each reference, and modifies the tag of one of the references to ensure its uniqueness. The internal unique identifier is invisible to the user. (The way the system copes with two simultaneous additions of the same reference is discussed in Section 2.3.)

When a reference must be unambiguously named among all databases, a *qualified* tag can be used, which has the form ⟨dbname⟩:⟨tag⟩.

The *reference path* organizes a user's view of the databases available as a list of databases and the search order within them. The default is to construct a list by concatenating a file .refpath in the user's home directory with a list of the publicly available databases. This can be overridden by specifying the reference path explicitly on the command line, or in the REFPATH environment variable. Reference paths can nest: one element of a path can refer to a file containing additional path information. Further, user environment variables can be used within a path, so that complex context-dependent paths can be created when needed.

```
%z Article (the type)
%K Lamport78a (the tag)
%A Leslie Lamport
%T Time, clocks, and the ordering of events in a distributed system
%J CACM.
%V 21
%N 7
%D 1978
%P 558 565
%x The concept of one event happening before another in a distributed
%x system is examined, and is shown to define a partial ordering of
%x the events. A distributed algorithm is given for synchronizing a
%x system of logical clocks which can be used to totally order the
%x events. The use of the total ordering is illustrated with a method
%x for solving synchronization problems. The algorithm is then
%x specialized for synchronizing physical clocks, and a bound is
%x derived on how far out of synchrony the clocks can become.
%k causal consistency, asynchrony, happens before
%k clock synchronization
```

FIGURE 1: An example reference.

2.2 Retrieving references

Refdbms provides two ways of retrieving references: name-oriented and content-oriented. Name-oriented retrieval uses the reference tag as a unique name for a single reference. Content-oriented retrieval searches a database for references matching a query, returning the tags of those references that match. Retrieval by name is useful when a specific reference must be unambiguously named, while searching is more useful when the results are not so definitely known.

Searching is done by keyword. The system maintains an inverted index of words from the author, editor, title, and keyword fields. These words are *wordstemmed,* so that `timing`, `times`, and `time` all are indexed and searched as `time`.

For example, one can search the databases by keywords:

```
oak> refsearch Lamport time
comp.doc.techreports:abadi92
hpdb:lamport78a
hpdb:lamport84
hpdb:lamport86a
hpdb:lamport87
hpdb:lamport89b
ucsc.grad.os:lamport78
inria.ooos:lamport78
inria.ooos:lamport87
```

The text of one of these references can be retrieved using the `refget` command:

```
oak> refget hpdb:lamport78a
%z Article
%K hpdb:Lamport78a
%A Leslie Lamport
%T Time, clocks, and the ordering of events in a distributed system
%J CACM.
%V 21
%N 7
%D 1978
%P 558 565
(etc.)
```

The `refget` command can also expand the abbreviations in the reference.

In most circumstances one wants to see the text of the results of a search. The reflook command is a shorthand for piping the results of refsearch to refget.

Refdbms also provides *active* searches: a user can specify a query once, then periodically run the refinterest program to find what references have been added to local databases that match that query.

2.3 Modifying references

References can be added, changed, and deleted from a *refdbms* database. The programs that process add and change updates enforce the syntactic correctness constraints discussed earlier. The effects of an update are not immediately visible; instead, the update is written to a log that is processed periodically. An update must be received by every replica of the database before it can be processed. The time required depends on whether all the sites holding replicas are available; how often each site propagates updates; and how many databases there are.

Users at different sites can submit conflicting updates. These conflicts are resolved by holding an update until it has been received by every database replica, and by not processing an update until it can be globally ordered with respect to other potentially-concurrent updates. There are three sources of conflict:

1. Two references added with the same tag. This is resolved by ordering the add operations, and adding a suffix letter to the tag of the second. For example, if the tags of both references were Lamport78, the tag of the second reference would be modified to Lamport78a.

2. Two concurrent changes to the same reference. We use two techniques to resolve conflicts: first, we *avoid* many conflicts by transmitting the *changes* each update operation makes to the reference, rather than a complete image of the reference. The difference is done by field, and different fields have different rules: the title or extract is overwritten, while lines can be added to or removed from the list of locations where the paper can be obtained. Second, the differences are processed in a definite order by every database replica. This yields a consistent view of the reference when all updates have been processed – "single-copy" consistency.

3. Deletion and other operations. Changes to a deleted reference are discarded, as are attempts to delete it more than once.

Users may need to retrieve a new or modified reference while the database is waiting to propagate the update. For example, a user might add a reference to an article they have just read, then want to embed a citation to it in a document they are writing. Unfortunately, the tag of a new reference might have to be modified to make it unique when the update is finally processed, but the user must embed the tag into their document. We felt that immediate access was important enough that we had to allow it, but we were not willing to compromise the idea that the tag is a stable name for a particular reference. Our solution was to maintain two versions of a reference: a stable version and a "pending" version. A pending version is created whenever a reference is added or changed, and can be retrieved using a tag of the form ⟨dbname⟩:⟨tag⟩.pending. When all outstanding updates for a reference have been processed, the pending version is deleted, so references embedded in user documents become invalid. This mechanism allows a user to explicitly state that they are willing to use information that may change, while allowing them to detect that the change has occurred. We considered another approach, where a user could qualify a tag by version or time information, but rejected it as too complex and unnecessary.

2.4 Sharing references

Many information systems have used the *publishing* metaphor: an individual or organization creates a database and makes it available for others to read. We are interested in a more flexible approach, where users can *collaborate* to manage a database. At the same time, *refdbms* has the flexibility to support publishing where it is appropriate.

Databases are the unit of sharing in *refdbms*. A user can create a new database and designate it either private or public. A private database is placed on that user's personal list of databases. Other users can add it to their personal database list if the files in the database allow them read access. A public database is usually placed in a subdirectory of a public directory, and is readable and writable by all users at the site. Local operations on a database are coordinated by locking database files.

A public database can be made available for sharing with users at other sites by *exporting* it. Users at other sites can then *import* a replica of the database. An update made to any replica will be propagated to every other replica, as discussed in the last section. Updates can be restricted by exporting the database read-only to other sites, so that updates can only originate from the site that first exported the database – supporting the publishing model.

TABLE 2: Other bibliography-database systems.

System	Formatter	Typed	Distribution	Indexing	Search	Citation
refer	troff	no	no	nonincremental	absolute keyword	content
Tib	(La)TeX	no	no	nonincremental	absolute keyword	content
BibIX	troff	no	remote query	nonincremental	absolute keyword	content
Scribe	Scribe	yes	no	no	no	name
BibTeX	LaTeX	yes	no	no	no	name
Refdbms	LaTeX Frame Maker	yes	replication	incremental	word stem	name

2.5 Using references

We believe that it must be possible to integrate an information system with the work processes it supports – for *refdbms*, the processes of research and writing.

One can use *refdbms* to learn of interesting papers as their references are added to databases using an experimental notification facility, which allows a user to specify a set of keywords that they find interesting. As new references are added to databases, a filter is invoked and those references that match a user's keywords will be mailed to them. For example, a user can create a file .refinterest in their home directory:

```
query : distributed
query : operating system
```

then run the refinterest program to find all the references that match the keywords "distributed" or "operating system" that have been added in the last few days.

The support for LaTeX and Frame Maker documents uses BibTeX. In the LaTeX source, one embeds the tag in a \cite command:

```
...that uses synchronized clocks \cite{hpdb:Lamport78a}.
```

One then builds a BibTeX database from the references cited in the LaTeX source using the refbibtex utility, and processing proceeds as always. The refmaker command is the equivalent for Frame Maker documents.

2.6 Related systems

2.6.1 Other bibliographic database systems

Several other bibliographic database systems are in use, including refer, Scribe, BibTeX, Tib, and BibIX. Table 2 summarizes these systems. In this section we will consider only those systems that integrally support the writing and research processes. None of the other systems listed support collaborative maintenance of a database.

The refer system supports the troff family of formatters [Kernighan78, Kernighan81], and was the original inspiration for Tib and BibIX. Tib [Alexander87], which supports TeX, was derived from an earlier similar formatter called bib, which in turn was derived from refer. The reference file format is almost identical to refer, except that many field types have been added to support documents that have been translated from other languages, and the system comes with control files that describe many different styles of bibliography and citations. The BibIX system [Rodgers90] supports the troff formatter, and unlike other systems allows queries to remote databases.

The main functional differences between these systems and *refdbms* are that (1) the type of each reference is made explicit in *refdbms*, but left implicit in refer, BibIX, and Tib; and (2) refer reference entries don't have a unique tag by which they can be identified: instead, "sufficiently many" keywords have to be given to identify a single reference. We believe that an imprecise citation is inappropriate when large, changing databases are used because a citation that uniquely defines a particular reference one day might match many references the next. We also believe that type information helps to improve the accuracy of information in the database and improves the quality of translations into other forms.

Scribe was developed as a research vehicle to demonstrate the practicality of separating document content from form. A part of that demonstration was a nicely designed bibliography package. The Scribe program is now commercially available, at a not inconsiderable cost. BibTeX [Lamport85] is an add-on package and program for the LaTeX formatting system that uses (essentially) the Scribe format for its references. Neither system provides support

TABLE 3: Other wide-area systems.

System	Model	Updates	Replicated	Active
Prospero	files	no	no	no
Alex	files	no	no	no
AFS	files	yes	yes	no
Jade	files	yes	no	no
Clearinghouse	name service	yes	yes	no
DNS	name service	no (publish)	yes	no
UNS	name service	no (publish)	yes	no
Archie	database	no	yes (not transparent)	no
Gopher	document graph	no (publish)	no	no
WAIS	full-text search	no (publish)	no	no
WWW	hypertext	no (publish)	no	no
Indie	general distributed index	no (publish)	yes	yes
Usenet	messages	yes (post)	yes (not transparent)	yes
Refdbms	bibliography	yes	yes (not transparent)	yes

for searching bibliography files: their purpose is to supply data for formatting, not to act as a database, though recently some tools have been made available to index and search BibTeX files. Further, although the format is elegant, it is not easy to use text-processing tools like grep and awk with it. *Refdbms* uses the type and tag from Scribe and BibTeX, but uses a basic reference format similar to refer for ease of manipulation by simple programs.

2.6.2 Other wide-area systems

Several information systems are available on the Internet. Some of these systems extend the file system model to span wide-area networks, some provide naming services, while others use an information-retrieval model. Table 3 summarizes several of these systems.

We have several criteria for judging a wide-area system. Availability and reliability are first among these: if the system cannot provide service, it is not useful. We believe that replication is an absolute requirement for a reliable wide-area system because it reduces the probability that the service is unavailable, decreases the mean network distance between client and server, and works to share processing and communication load across many machines. Some systems, including *refdbms*, provide non-transparent replication, where users must explicitly interact with one local replica, even though the overall service is replicated. Unreplicated systems fare even worse when one uses a disconnected computer. Unless the system can place a copy of interesting information on the local system, the service is completely unavailable as long as the computer is not connected. Cost is also important, including the cost in both time and money to set up a site, to begin sharing information with others, and to maintain the system. Finally, an active system can inform users of new information that is likely to be of interest to them.

The Prospero [Neuman92], Alex [Cate92], Jade [Rao93] and AFS [Howard88] file systems, among others, allow users at different sites to share different forms of information and to integrate this information with other tools. Unfortunately, they *only* provide the semantics of a file system: there is no possibility of supplying application-specific semantics to the information shared among different sites. These systems therefore cannot properly reconcile conflicting modifications made by different sites in all cases.

Name services, including the Xerox Clearinghouse [Oppen81], DNS [Mockapetris87], and Cambridge UNS [Ma92], implement specialized databases for translating names to addresses. All of these systems have very high reliability requirements, and so all provide replication. All restrict updates to part of the database to the organization that has authority for that part. The Clearinghouse and UNS systems use weak consistency replication techniques similar to those used in *refdbms*.

Other systems provide a read-only mechanism for *publishing* information. These systems include Archie [Emtage92], which allows users to search a database of files available for anonymous FTP; Gopher [Anklesaria93], a menu-oriented "document delivery system"; the WAIS [Kahle89] full-text search system; and the World Wide Web (WWW) [Berners-Lee92] global hypertext system. Except for Archie, these tools do not integrate with other tools.

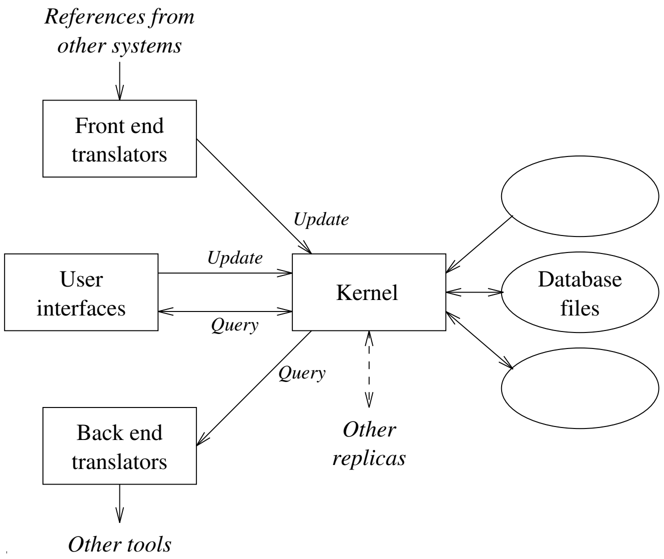

FIGURE 2: General architecture of the *refdbms* system.

Archie achieves better integration by using the Prospero file system protocol to connect clients to servers. Most of these systems have severe performance problems because of their lack of replication.

The Indie distributed indexing system [Danzig92] uses an interesting architecture that is an alternative to that used in *refdbms*. The Indie system consists of a number of distributed indexes, each of which collect "interesting" information from other information sources, and which can provide this information to other indexes. This is fundamentally a system for storing and discovering information published by other sources, and any changes must be made at the original source of the information.

Usenet is an often-overlooked information system, even though it is currently clearly the largest such system. Users can post messages to and read messages from a newsgroup.

3 Architecture

The *refdbms* system is divided into four parts: the kernel, which provides the database functions; front ends, which translate references from other formats; user interfaces, which allow the user to interact with the kernel; and back ends, which integrate *refdbms* with other tools. Figure 2 shows these components.

In this section we will discuss the implementation of each of these parts.

3.1 Kernel

The kernel implements the basic database functions, including updates, indexing, search and retrieval, and distributed operation. It is structured as a number of separate programs and libraries, and other parts of the system, such as user interfaces, use the kernel either by invoking the programs or calling library routines. Figure 3 shows the parts of the kernel.

There is one *submission* program for each of the kinds of updates that can be made to a database. When adding and changing a reference, the programs read a copy of how the reference should appear. The add program writes the new reference to the log in its entirety, while the change program computes a difference between the current value and the new, and stores that difference in the log. The deletion program merely takes the unique identifier or tag of the reference and writes a deletion record to the log.

The *log* stores the updates that have not yet propagated to every replica of the database.

The *replication* programs maintain the list of replicas for a database, propagate references between replicas, and control their creation, importation, exportation, and removal. Section 3.2 discusses this in detail.

Once a reference has propagated to every replica, it is *posted* to the database proper. In doing so, the reference is written to the references file for the database, which is indexed by the reference's unique identifier. A tag index entry

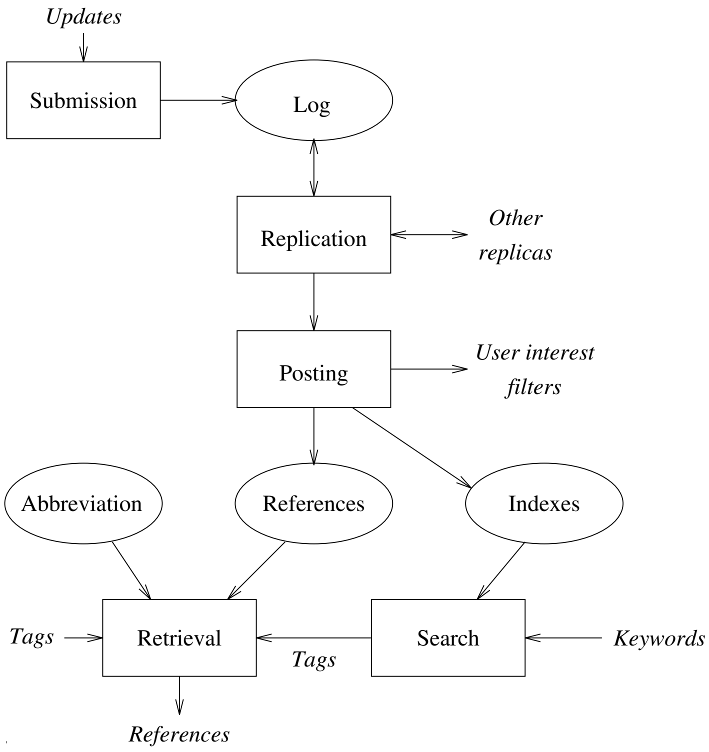

FIGURE 3: Components of the *refdbms* kernel.

is recorded to map the reference's tag to its unique identifier. In addition, the posting program incrementally updates an inverted index of the words in the references' keyword, author, and title fields.

The *search* program uses the inverted index to find the references that include some words. The words, both in the index and in the query, are wordstemmed: a *stem* or *base word* is computed for each word, and matching uses the stems. This ensures that trivial differences in a key, such as plurals, do not affect the search.

Finally, the *retrieval* program can retrieve references by tag. It includes a way to expand abbreviations.

We used several publicly-available libraries to build the *refdbms* kernel. These include Henry Spencer's regexp library for regular expressions; wordstemming routines from the ispell program maintained by Geoff Kuenning; and B-tree and hashed file management from the db library from the CSRG at U.C. Berkeley.

In addition, we have written a number of routines to assist in making the system portable, providing a functional interface behind which most system idiosyncrasies – such as omissions from the standard libraries, system logging facilities, and file locking – can be hidden.

3.2 Replication

Each exported *refdbms* database can be replicated at many sites. We used the *timestamped anti-entropy* (TSAE) weak consistency group communication protocol [Golding92b] to coordinate the replicas. We have described the overall architecture we are prototyping in *refdbms* elsewhere [Golding92a]. The current implementation only provides the essential parts of that architecture: database replication using a weak-consistency group communication protocol.

Weak consistency allows replicas to diverge for short periods of time, but guarantees that all of them will eventually receive every update and that they will compute the same results upon processing the updates. Divergence provides two important benefits: it allows the system to tolerate disconnected and crashed sites, and it makes the delay required for propagation invisible to the client update program. Divergence is acceptable in a system like *refdbms* as long as all the copies reach agreement reasonably quickly.

The TSAE protocol works as follows. When a client program submits an update, that update is written as a message in the log. A few times each hour, a cron job executes the refae program. If this program finds that there are updates that need to be propagated, it randomly selects another replica and opens a TCP connection to the refdbmsd daemon at that site. They perform an anti-entropy session, in which each of the two programs forwards to its partner any update messages it has received and that the other has not. They also exchange information about the status of the group, including updates to the list of replicas and summaries of the messages each replica has received. After the session has been committed, the programs on either end run the refpost program in case some messages have now been acknowledged by every replica, and have become eligible for posting. The proofs that this protocol correctly propagates messages and analyses of its performance can be found elsewhere [Golding92b].

Another set of programs allow users to maintain the replicas at their site. Refcreatedb creates a new database, which can then be exported using refexport. If someone at another site wants to use a replica of the database, they use the refimport program, which contacts the daemon at some other site (the new site's "sponsor") to retrieve a complete copy of the database. When someone no longer wants to maintain a replica, they use the refleave program.

Each replica of a database has an associated *privilege,* which is one of member, rwmember, or romember. The latter two privilege levels allow some databases to be used for publishing information by not allowing updates except from the publisher. Users of a member replica can update the database, and any new replica sponsored by this replica will also be a full member. If a replica has rwmember privilege, its users can update the database, but any sponsored replica will be a romember. A romember copy only allows queries, not updates. In practice, the current mechanism can be subverted with moderate effort, and we are considering whether to provide more resilient authentication mechanisms to prevent this.

The system includes facilities for logging statistics on traffic and for checking other servers.

3.3 Front ends

The front ends translate references from other bibliographic systems into *refdbms* format. At the time of writing there are translators for BibTeX databases and refer databases.

The BibTeX translator uses bibtex itself to perform the translation. The bibtex program reads a list of citations, reads the references from a file, and parses them. It then interprets a special BibTeX style program that performs the actual translation.

Refer uses untyped references, so the translator must infer the type of a reference from its contents. Further, over the years different refer databases have evolved different, sometimes incompatible, conventions for entering references. The translator uses a set of rules to guide type inference and translation, and the standard rules can be augmented or replaced to adapt the translator to the peculiarities of each database. We have translated several thousand references using this tool, and found that with a few augmenting rules it is able to correctly translate a reference in the vast majority of cases.

3.4 Back ends

The back ends integrate *refdbms* with document processing tools. At the time of writing LaTeX and Frame Maker documents are supported.

Integration with both LaTeX and Frame Maker is currently done using BibTeX. For LaTeX, the back end scans a document, finding the \cite commands, retrieves the appropriate references from the databases, and builds a BibTeX file from them. It then uses the bibtex program to format the references. Frame Maker support is similar: a utility converts the document to text form, then scans it for citations. The citations are retrieved and formatted using BibTeX, converted to Maker Markup Language, and imported into the document.

3.5 User interfaces

The system includes three user interfaces: a command line interface, a GNU Emacs environment, and an experimental X application based on the XView toolkit.

The command line interface includes three programs: refnew, to enter a new reference; refchange, to edit a reference using a text editor; and refdelete, to delete a reference from the database.

The GNU Emacs interface is currently the most sophisticated interface for adding and changing references. It is an Emacs major mode that places each reference in its own buffer. The mode provides field-based manipulation (for example, tab to move to next field); justification; syntax checking; and submission. Three functions create new buffers: ref-new creates a new reference, inserting an empty template; ref-retrieve prompts for a tag and retrieves that reference; and ref-copy clones a buffer.

The experimental X application uses the XView toolkit, which provides an Open Look interface style. The application consists of a control panel from which other operations are invoked. The primary operations – querying and updating databases – can be initiated directly from the panel, while other operations are invoked from menus.

4 Performance

We have considered a number of performance measures in evaluating *refdbms*. These include the time required to perform basic manipulations, such as retrieving and searching for references; the time required to build indices; the storage required; and the network traffic produced.

Where appropriate we compare *refdbms* with refer and BibTeX. We used the bib system [Budd82] and the Tib system (version 2.1) for refer-format references. We used the bibindex program, version 2.2, for indexing BibTeX references.

All our performance measures were made on frans.cs.vu.nl, a diskless Sun SparcStation 1+ with 16Mbytes of memory running SunOS 4.1.2. The system was running X and a number of applications, though none of them were active during the measurements. This setup was selected as a typical workstation environment.

4.1 Retrieval

We characterize retrieval performance by the time required to retrieve some number of references, given their tag. We were unable to construct a fair comparison between *refdbms* and other systems, since refer does not use unique identifiers, and BibTeX does not index by identifier. We analyzed both the case where database files are not likely in memory, and when they have been cached. Under SunOS, this implies that it is likely that the virtual memory system has not yet gotten rid of pages holding file data.

A linear least-squares fit on these data suggest that the total time, in milliseconds, to retrieve r references from d databases, when not cached (t_n) and cached (t_c) is approximately

$$\begin{aligned} t_n &= 8r + 471d - 385 \text{ msec} \\ t_c &= 4r + 103d + 87 \text{ msec} \end{aligned}$$

We caution that the data are insufficient to draw definite conclusions, other than that the time appears to be dominated by the time required to open a database, and that execution is several times faster if the data are already in memory.

4.2 Indexing

All three systems use an inverted index when searching for references by keywords. We compared the time required to build this index from scratch using *refdbms*; using the invert program distributed with bib and using the tibdex program distributed with the Tib system for refer-format references; and using the bibindex program, version 2.2, for BibTeX references. We report both the "real" (wall clock) time required and the estimated CPU time (in seconds):

System	Tool	CPU	Real
refer	invert	54.2	55
	tibdex	55.4	59
BibTeX	bibindex	20.8	24
refdbms	refbuildkeys	50.6	56

The bibindex program is significantly faster than the others. We believe that this is because that program indexes each field separately. We are encouraged that the simple refbuildkeys program – a shell script that uses several sed and awk scripts, and that in addition does wordstemming – is as fast as programs written entirely in C.

However, one rarely builds a *refdbms* index from scratch. Instead, the index is updated incrementally as the database is changed. When a new reference is added, its search keys are simply added to the index. When a reference is changed, all search keys for it are first deleted, then its keys are recomputed and added to the index. We investigated the time required to add, change, and delete various numbers of references from a database:

	Add		Change		Delete	
References	CPU	Real	CPU	Real	CPU	Real
1	3.4	6	8.6	19	5.7	12
2	3.4	6	8.5	15	6.0	11
4	3.6	7	8.8	17	5.9	11
8	4.1	10	9.2	19	5.9	11
16	4.5	10	9.8	20	5.9	12
32	5.2	11	10.4	21	5.9	11
64	6.3	12	11.4	22	5.7	11

4.3 Query

Most a user's direct interaction with a bibliographic database involves searching for keywords. We compared the time required to search for and retrieve references matching various keywords using refer, BibTeX, and *refdbms*:

	refer			BibTeX			*refdbms*		
Keywords	CPU	Real	Refs	CPU	Real	Refs	CPU	Real	Refs
Lamport	0.1	0	0	1.0	1	0	0.7	0	0
files	0.1	0	3	1.4	2	5	1.0	1	12
file	0.1	0	11	1.4	1	15		(same)	
adaptive routing	0.1	0	3	1.3	1	3	1.0	1	3
distributed systems		(failed)		2.1	2	131	1.5	1	92

The lookup program used in refer appears to perform all its operations in memory, causing it to be very fast. However, it is therefore unable to handle queries that return too many results. The biblook program can perform arbitrary boolean expressions, and can limit its searches to particular fields. This complexity makes it slower than the equivalent *refdbms* programs.

4.4 Storage

Refdbms uses a terse data format, similar to refer. We evaluated the storage required for one large sample database containing 1300 references. It appears that the basic syntax is as efficient as that of refer – the difference is the text of the reference type and tag – but a complete *refdbms* database requires nearly twice as much storage as does a simple refer database with its associated index. The same references translated into BibTeX format require significantly more space, but its inverted index is much smaller than that of *refdbms*, so the sum is smaller.

Metric	*Refdbms*		Refer		BibTeX	
	source	with index	source	with index	source	with index
size (bytes)	440553	1724416	413513	825331	686809	1054371
relative size	100%	100%	93.9%	47.9%	155.9%	61.1%

The bulk of the difference is in the hash file that stores the references themselves – it requires 1.2MB to store 440KB of text. Part of the overhead comes from the internal unique identifier: it averages 38 bytes, which is about 9% of the size of the data. This indicates that about 61% the hash file is overhead. Some of this is due to internal fragmentation in the hash file, and we are looking at ways to reduce the problem.

4.5 Traffic

Over the period of June 15 through October 27, 1993, there were 5078 updates propagated between replicas; of these, 4719 (92.9%) were new references; 147 (2.9%) were modifications; and 212 (4.2%) were deletions.

After an update has been submitted, a record of it must propagate to other sites before it can be applied. Figure 4 shows the distribution of the time between submission and posting to the database for updates to the comp.doc.techreports database between June 15 and October 27, 1993. The mean time to posting was 8942 seconds, or about two and a half hours. This database has had six replicas for most of that period, and other databases with as many replicas showed the same delay. Databases with one or two replicas required about ten minutes, on average, before an update could be posted, while one database, which had was subject to an extended network partition, had an average latency of five hours – mostly caused by a few updates being delayed for nearly a week.

5 Lessons learned

While the current version of *refdbms* shows that our architecture for wide-area systems is feasible, there are many improvements that we intend to make in future revisions.

The first is to decouple the identity of a replica from its location – in this case, the IP address of its host and TCP port number on which the daemon listens. This coupling prevents replicas from migrating between addresses, and restricts us to TCP as the transport protocol between replicas. In the next revision of the protocol, location and identification will be treated separately. A replica will be identified by an arbitrary unique value, and a list of ⟨ protocol, address ⟩ pairs will be maintained as the location of each replica. This will allow us to add new transports for communicating

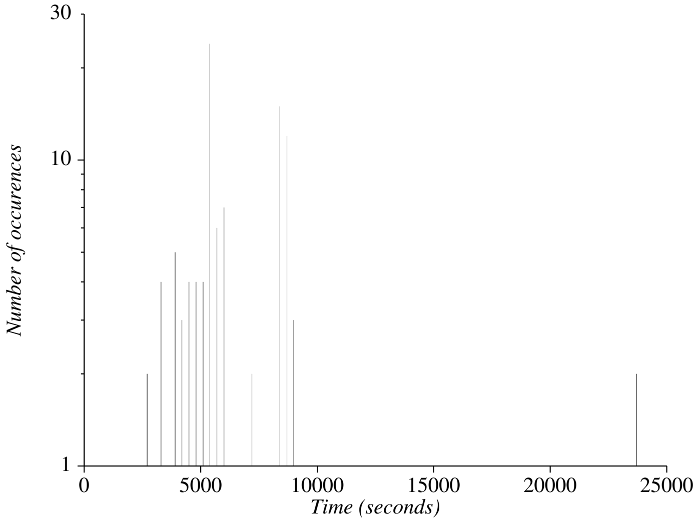

FIGURE 4: Histogram of times between submitting an update and it being applied.

between replicas, including UUCP, and it will allow the addition of new functions, such as a mechanism for remote queries on a replica.

Some databases contain several megabytes of information, and transferring this all in one go when a new database is imported can take quite a while over long-distance or low-bandwidth network connections. To assist this initialization, two new features will be added: first, the state transfer involved in creating a new replica will be split over multiple sessions, with each session transferring a well-defined fraction of the sponsor's state, until both the new replica and the sponsor have equal contents. Second, replicas containing only a subset of a database will be supported. The subset is defined by a set of predicates on references. In this way a site can reduce its storage requirement by only storing those references that are interesting, while forwarding queries it cannot satisfy to other replicas.

Formal analysis of the TSAE protocol identified an optimization that could be applied. As currently implemented, delivery of an update message is currently delayed until every replica has acknowledged receipt of that message. However, updates could correctly be delivered when the *local* replica acknowledges messages from any replica with lesser or equal timestamp, as long as all sites have approximately synchronized clocks. This allows an update message to be processed in $O(\log n)$ time after it is sent, where n is the number of replicas, rather than the $O(n)$ time currently required. The next protocol revision will include this improvement.

The prototype does not support fine-grained control over sharing. We are investigating how existing authentication and access control mechanisms can be used to support per-user protection, and how these mechanisms must be extended to do so.

We are investigating mechanisms for reducing the state information each replica must maintain. Currently, each replica must maintain information on every other replica. We are investigating a hierarchical scheme that would allow each replica to maintain information about $O(\log n)$ replicas. This approach would also make it easier for users to use an *optimistic* view of the database contents.

The system currently does not include any "name service" mechanism for maintaining information about the databases that are available. We are investigating this problem as part of research at the Vrije Universiteit on infrastructure for building wide-area applications.

As other researchers have found, writing a truly portable program of any considerable size is very hard. We have been able, for the most part, to encapsulate system differences in a small set of library routines. However, we have not had the same success in making a simple, portable mechanism for configuring, building, and installing the system. We are investigating several alternatives, including publicly available configuration packages, the X imake tool, and ways to construct a similar tool using the m4 macro processor.

We have also found that the simple database library we are using, which lacks crash recovery, is not adequate for constructing a reliable system: *refdbms* was initially developed on a set of systems that never experienced crashes, so this need was not experienced first hand by its developers. File locking has also been a problem, exercising what appear to be bugs in the SunOS lock daemon. (These difficulties do not matter on more robust host systems, of course.) We are investigating mechanisms by which these problems may be circumvented.

6 Conclusions

A bibliography is an essential part of the process of writing and research. *Refdbms* is a tool for maintaining and using bibliographic information. It differs from other bibliography database systems and wide-are information systems in a number of ways:

- *Refdbms* emphasizes *collaborative* use of information by making it available for all its users to maintain, when appropriate, and allows information *publishing* when needed.
- It is *specific* to the research process, rather than a general-purpose tool.
- It is structured as a set of *tools* rather than as a single integrated system, so that it can become part of other, larger systems.
- It provides better *searching* capabilities than other bibliographic systems, including *active* searches that periodically inform the user of interesting new information.
- It is *distributed* and *replicated,* so that many users can share information while providing acceptable reliability and performance.
- It uses *weak consistency* so that it supports mobile computers and unreliable wide-area networks by strictly controlling communication.

The *refdbms* system is at the same time a prototype that allows us to evaluate our architecture for constructing wide-area applications. The results encourage us to believe that the architecture is suitable as a basis for many other applications, including name and authentication services, as well as other task-specific services. We are investigating some of these options.

The system is publicly available. The current version is available for anonymous FTP on ftp.cse.ucsc.edu, in the directory /pub/refdbms. A list of available databases can be found using finger refdbms@refdbms.cse.ucsc.edu.

Acknowledgments

The many *refdbms* users have contributed substantially to the system, both in code, bug fixes, and experience. At Hewlett-Packard Laboratories, David Jacobson wrote the original Emacs mode, and Carl Staelin first integrated the wordstemming code. The Mammoth Project at U.C. Berkeley under National Science Foundation infrastructure grant CDA–8722788 provided resources for testing early versions of the system. George Neville-Neil, Tage Stabell-Kulø, Peter Bosch, and Henri Bal all made helpful comments on this paper.

References

[Alexander87] J. C. Alexander. *Tib: a TeX bibliographic preprocessor* (1987). Department of Mathematics, University of Maryland. Version 2.1.

[Anklesaria93] F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. Torrey, and B. Alberti. The Internet Gopher protocol (a distributed document search and retrieval protocol). Request for Comments 1436 (March 1993). Internet Engineering Task Force.

[Berners-Lee92] Tim Berners-Lee, Robert Cailliau, Jean-François Groff, and Bernd Pollermann. World-Wide Web: the information universe. *Electronic Networking: Research, Applications, and Policy*, **1**(2) (Spring 1992).

[Budd82] Timothy A. Budd and Gary M. Levin. A Unix bibliographic database facility. Technical report 82–1 (1982). University of Arizona.

[Cate92] Vincent Cate. Alex – a global filesystem. *Proceedings of the File Systems Workshop*, pages 1–11 (May 1992). Usenix Association.

[Danzig92] Peter B. Danzig, Shih-Hao Li, and Katia Obraczka. Distributed indexing of autonomous Internet services. *Computing Systems*, **5**(4):433–59 (Fall 1992). Usenix Association.

[Emtage92] Alan Emtage and Peter Deutsch. Archie – an electronic directory service for the Internet. *Proceedings of the Winter Conference* (San Francisco), pages 93–110 (January 1992). Usenix Association.

[Golding92a] Richard Golding. A weak-consistency architecture for distributed information services. *Computing Systems*, **5**(4):379–405 (Fall 1992). Usenix Association.

[Golding92b] Richard A. Golding. *Weak-consistency group communication and membership*. PhD thesis, published as Technical report UCSC–CRL–92–52 (December 1992). Computer and Information Sciences Board, University of California at Santa Cruz.

[Harrison89] Michael A. Harrison and Ethan V. Munson. On integrated bibliography processing. *Electronic Publishing*, **2**(4):193–209 (December 1989).

[Howard88] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, **6**(1):51–81 (February 1988).

[Kahle89] Brewster Kahle. Wide area information server concepts. Technical report TMC–202 (3 November 1989). Thinking Machines Corporation.

[Kernighan78] Brian W. Kernighan and Lorinda L. Cherry. A system for typesetting mathematics. *Communications of the ACM*, **18**(3):151–6 (March 1978).

[Kernighan81] Brian W. Kernighan. A typesetter-independent TROFF. Computing Science technical report 97 (1981). Bell Laboratories, Murray Hill, NJ.

[Lamport85] Leslie Lamport. LaTeX*: a document preparation system* (1985). Addison-Wesley Publishing Company, Reading, MA.

[Lesk78] M. E. Lesk. Some applications of inverted indexes on the UNIX system. Computing Science technical report 69 (June 1978). Bell Laboratories, Murray Hill, NJ.

[Ma92] C. Ma. *Designing a universal name service*. PhD thesis, published as Technical report 270 (1992). University of Cambridge Computer Laboratory.

[Mockapetris87] P. Mockapetris. Domain names – concepts and facilities. Request for comments 1034 (November 1987). ARPA Network Working Group.

[Neuman92] B. Clifford Neuman. The Prospero file system: a global file system based on the virtual system model. *Computing Systems*, **5**(4):407–32 (Fall 1992). USENIX Association.

[Oppen81] D. C. Oppen and Y. K. Dahl. The Clearinghouse: a decentralized agent for locating named objects in a distributed environment. Technical report OPD–T8103 (1981). Xerox Office Products Division, Palo Alto, California.

[Rao93] Herman C. Rao and Larry L. Peterson. Accessing files in an internet: the Jade file system. *IEEE Transactions on Software Engineering*, **19**(6) (June 1993).

[Rodgers90] R. P. C. Rodgers and Conrad Huang. *BibIX 2.1 Manual* (January 1990). University of California Office of Technology Licensing, Berkeley, California.

[Schwartz92] Michael F. Schwartz, Alan Emtage, Brewster Kahle, and B. Clifford Neuman. A comparison of Internet resource discovery approaches. *Computing Systems*, **5**(4):461–93 (Fall 1992).

[Wilkes91] John Wilkes. The refdbms bibliography database user guide and reference manual. Technical report HPL–CSP–91–11 (20 May 1991). Hewlett-Packard Laboratories, Palo Alto, California.

Biographies

Richard A. Golding received his B.S. degree in Computer Science from Western Washington University in 1987. He received his M.S. degree in 1991 and his Ph.D. degree in 1992, both in Computer and Information Sciences from the University of California, Santa Cruz. He is currently a researcher at the Vrije Universiteit Amsterdam. His research interests include distributed operating systems, wide-area systems, and object-based programming environments.

Darrell D. E. Long received his B.S. degree in Computer Science from San Diego State University in 1984. He received his M.S. degree in 1986 and his Ph.D. degree in 1988, both in Computer Science from the University of California, San Diego. He is currently Assistant Professor of Computer & Information Sciences at the University of California, Santa Cruz. He has published articles on protocols for data replication, host reliability, performance guarantees and high-speed I/O systems. His current research interests include distributed computing systems, high-speed I/O systems, performance evaluation and real-time management of large scientific data sets. Dr. Long is a member of the IEEE Computer Society, the Association for Computing Machinery and Sigma Xi. He is the chair of the IEEE Technical Committee on Operating Systems and Application Envinronments. He also serves on the editorial board of the *International Journal in Computer Simulation.*

John Wilkes graduated with degrees in Physics (B.A. 1978, M.A. 1980), and a Diploma (1979) and Ph.D. (1984) in Computer Science from the University of Cambridge. He has worked since 1982 as a project manager, and researcher at Hewlett-Packard Laboratories. His current research interests are in highly available high-performance storage systems, and in resource management in scalable computers. He also enjoys dabbling in photography, interconnects for scalable systems, and performance modelling. He particularly enjoys working with university colleagues on projects such as this one.