BIBDB: A Bibliographic Database for Collaboration

David J. Musliner **James W. Dolter** **Kang G. Shin**

Computer Science and Engineering Division
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122
(313) 936–2495 {djm, jdolter, kgshin}@eecs.umich.edu

ABSTRACT

While researchers strive to develop new systems to enhance the cooperative document editing process, many authors already collaborate, using *existing* text processing systems to produce papers and reports. Using these tools, one of the most time-consuming and error-prone collaboration tasks is maintaining a consistent shared bibliography. We have designed and implemented the BIBDB system to simplify collaborative authoring by providing a shared, cooperatively maintained bibliographic database. BIBDB uses existing networking technology and merges seamlessly into the LaTeX/BIBTeX text processing system [5]. The contributions of BIBDB include a set of user interface policies and software implementation techniques that support cooperative database maintenance.

KEYWORDS

Bibliographic databases, collaborative writing, distributed & replicated databases, partial locking, relaxed consistency, incremental indexing.

INTRODUCTION

Despite their overwhelming popularity in the scientific community, computerized text formatting systems have remained quite primitive, and do not yet take advantage of the potential for sharing and cooperation which is embodied in local-area networks (LANs) and the nationwide Internet. For instance, every formatting system has some technique for constructing a bibliography of references automatically, but each user must type the bibliographic reference material into his/her own database file. In addition to a tremendous amount of replicated effort, this isolation of personal databases leads to pervasive inconsistency in reference formats (especially in abbreviations of journal and proceedings titles) and incompatible citation keys. As a result, collaborating authors find it nearly impossible to smoothly merge their individual databases and reference styles. A few user groups have cooperated to form larger, shared databases of references, but the performance of programs which access these databases does not scale well, because the databases are simply large text lists of bibliographic entries.

To address the deficiencies of current bibliographic systems within the context of existing text processing and networking technology, we have designed and implemented the BIBDB bibliographic database system. The BIBDB system is designed to maintain a single vast bibliographic database, shared by all users and accessible to any UNIX machine on the Internet. The database is not stored or accessed linearly, so search and access performance does not depend linearly on the size of the database. Local copies of the database are maintained at each installation (perhaps one installation per university or LAN), so that most accesses to the database are done over LANs rather than the nationwide network. Additions to the database are incremental and global, and data is never removed from the system: once an entry is added at one installation, it is available (and will not be added again) at all installations.

Our work on BIBDB has been carefully focused on developing practical mechanisms to allow cooperative database maintenance and to enhance collaborative authoring capabilities. We have developed a set of user interface policies that encourage harmonious cooperative efforts. We have also developed several unique software mechanisms useful for implementing this shared distributed database. These implementation details include a uniquely powerful client/server organization, a relaxed database consistency criterion, a partial database locking method, and a dynamic overload scheme leading to graceful degradation.

The remainder of this paper is organized into six sections. The next section presents an overview of the BIBDB system goals and organization. The following three sections describe the server and access programs which make up the system. The last two sections compare BIBDB to competing bibliographic systems, and describe the current status and future extensions of the system.

THE BIBDB SYSTEM

Design goals

We designed the BIBDB system to address the variety of problems perceived in existing bibliographic systems. The

Appears in Proc. 1992 ACM Conf. on Computer-Supported Cooperative Work pp. 386–393

primary goal was to create a system which takes advantage of cooperative efforts at database maintenance and eases collaborative authoring, within the context of currently popular text processing packages. Based on personal experience and discussions with other users, we designed BibDb with the following specific aims:

- Data sharing. BibDb maintains a very large database of bibliographic information entered, maintained, and shared by all users and accessible over LANs and the Internet. Sharing the database avoids redundant data entry.
- A one-to-one mapping of keys to references. BibDb goes to great lengths to prevent duplicate entries, and will never issue the same citation key for two different references. Thus there is never any confusion about what information a citation points to, and collaborating authors can exchange simple citation keys, rather than extensive bibliographic data.
- Citation permanence. Once a citation key has been issued for a new entry, it will always point to that entry. Documents produced using the BibDb system will never be invalidated by changes to the database.
- Consistent referencing. BibDb encourages users to employ standard string aliases, so that references to similar works are formatted similarly.
- Powerful searching. BibDb includes an associative cross-indexing mechanism, so users can perform keyword searches for articles. These searching facilities make the database a major resource both for finding specific references and for blind literature searches.
- Speed and scalability. BibDb stores references in a hashed database format which makes retrieval extremely fast. The database is never processed entirely or searched linearly.
- Incremental growth. As users add new entries, BibDb does not re-process the entire database in any way.

The BibDb system is intended to merge seamlessly into the LaTeX/BibTeX formatting system [5]. In the LaTeX system, text formatting commands are included in a document to control its eventual printed form. The companion BibTeX system provides bibliographic reference indexing and formatting. BibTeX bibliographic entries are typed in a fixed format, with each entry assigned a unique key by which it is cited. Figure 1 shows an example BibTeX entry for a (mythical) journal article, as well as the use of *strings* (or aliases) to abbreviate commonly used journal names and other text items. The "@article" portion specifies that the entry's *type* is article, which the `bibtex` program knows how to format. The entry type can also be "@inproceedings", "@book", etc. After the type specification, the entry's unique key is specified (here, "musliner:92"). Then a series of "*field* = *value*" pairs specifies the bibliographic data.

```
@string{CCJ = "The Cool Computing Journal"}
@article{musliner:92,
author = "D. J. Musliner and J. W. Dolter",
title = "A Cool Bibliographic System",
journal = CCJ,
year = 1992 }
```

Figure 1: An example BibTeX entry.

System overview

Figure 2 shows the organization of the entire BibDb system, which is composed of database files, server programs, and client programs. A copy of the entire database exists at each installation, where installations are allocated to LANs with fairly large groups of users (e.g., universities, research companies). Each installation runs the `bibdb-server` program, which provides database maintenance and access services to users. Users connect to the `bibdb-server` by running client programs on their own computer. The client programs usually connect to the closest installation's server, but may connect to a server at any installation. The database copies at each installation are kept consistent by their connection to the central `keymaster`. The `keymaster` coordinates all updates to the database, systemwide, and ensures that different users do not simultaneously add the same entry.

All of the BibDb programs are written in *perl* [9], a commonly available and freely-licensed interpreted language which contains nearly all of the features of awk, sed, c-shell and C. Like Lisp, perl has an 'eval' function which runs the interpreter on a perl expression. A perl program can write its own code by assigning valid perl code to a string variable and then eval-ing the variable.

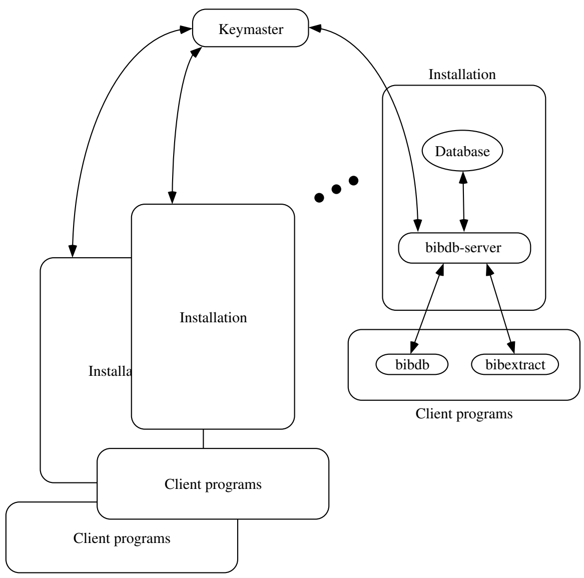

Figure 2: Overview of the BibDb system.

The server programs

The `bibdb-server` provides all database services to the user. This program embodies the various administrative policies of the BIBDB system, which are described in the next section. Each installation's server program opens a "well-known" socket to which the client programs connect. The server forks copies of itself for each connecting client, so that multiple clients may access the database simultaneously.

The `bibdb-server` connects to the `keymaster` via a well-known socket, and tells the `keymaster` about all changes to the database. The `keymaster` forks copies of itself for each connecting `bibdb-server`, and tells each `bibdb-server` about changes that others have made to the database. These programs and their interactions will be described in more detail in the following sections.

The client programs

The client programs `bibdb` and `bibextract` are actually very small perl scripts that simply connect to the appropriate server and act as a "programmable conduit" for information. Aside from a single routine to connect to the appropriate server program, the entire body of the client programs is shown in Figure 3. The client programs connect to a server and then read lines from the connected socket, either printing them out or eval-ing them. Thus the server program can send executable code to the client program, causing the client to, for instance, open a file and send its contents back to the server. All file operations, prompting, and data manipulation are actually controlled by the server program, and the client programs just provide a channel over which information can flow to and from the user's computer. This organization has several beneficial aspects:

- Security. Since the data operations are all completely controlled from the server program, there is virtually no way a user can interfere with or alter the database in an unacceptable manner. The `bibdb-server` gives users a limited ability to modify or delete information from the database, and malicious users cannot expand upon that ability because the information exchange protocol is dynamically downloaded from the server. The server even has to tell the client program when to prompt the user for input.
- Speed. Since the client programs are very small perl programs, they start very quickly. The high cost of parsing the complex database management routines is paid only when the server program is started, not each time a user wishes to access the database.
- Optional distribution. The server program can download subroutine definitions to the client programs and then send invocations of those subroutines, thus offloading computations from the server computer to the user's computer. This feature is most useful for I/O routines which can be offloaded without compromising security.
- Protocol locality. Because the entire interaction protocol is contained within the server program, it is much easier to recognize errors in the protocol when writing these programs. While debugging, we do not need to constantly switch between the client and server programs to understand how information is being exchanged, and what the response to various inputs will be. The `keymaster/bibdb-server` interface, which does not share this feature, has been much more difficult to develop and debug.
- Easy code updates. Making changes to the BIBDB implementation is extremely easy, because there are relatively few copies of the server program (one per installation). There may be many thousands of copies of the trivial client programs, but these need never be changed, because the actual interface is contained solely in the server. In fact, since the client programs were first developed in this form, we have made hundreds of changes to the server program, but not one to the client programs.

The fact that the client programs are such minimalist shells is not visible to the user. The user simply starts one of the client programs, which then appears to prompt and interact with the user in a normal fashion. Thus we describe the behavior of the client programs as though they actually define those interaction patterns, when in fact the server program does.

BIBDB

The `bibdb` program is the database maintenance program, allowing the user to add and modify entries and strings, and communicating with the keymaster to keep the local database copy up-to-date with the other installations. The program also provides a powerful user interface to the database's searching mechanisms, including an associative cross-index which allows keyword search without linear scanning.

Policy-directed functions

The `bibdb` program implements many of the policies with which the BIBDB system addresses the shortcomings of other bibliographic databases. For instance, each user of a referencing program tends to develop his/her own style of citations. This means that combining or exchanging databases or even single citations can be difficult, because citation keys may conflict, or duplicate entries may not be detected. To avoid these problems, `bibdb` automatically generates the citation keys for all entries when they are added to the database. All authors then use the same key to cite the same entry in the shared BIBDB database. Keys are formed from the last name of the first author, a colon, the last 2 digits of the year, and a suffixed 'a', 'b', 'c', etc, to distinguish otherwise identical entries (e.g., musliner:92a). No two entries may have the same key. When entries are added, if entries with the same *key-root* (all but suffix character) exist, the system queries the user to make sure a duplicate entry is not being introduced, as illustrated in Figure 4. Barring typographic errors in the first author's last name or the year, this technique detects all possible duplicate entries. A title comparison heuristic then

```
while (<SERVER>)              # While (get a line of input from the server)
  {
      chop;                   # Strip newline from input.
      if (s/^\001//)          # If input starts with CTRL-A,
          { eval; }           # it is executable perl code, so execute it.
      else                    # Else,
          { print "$_\n"; }   # just print the input to the user.
  }
```

Figure 3: Main body of client programs (in perl).

makes continue/abort recommendations to the user. Unfortunately, differing interpretations and typographic errors are much more common in the titles of papers, and thus human intervention is still required to make the final decision to abort an addition. Fortunately, bibliographic systems are not so critical that we need to worry too much about errors. The inevitable typographic errors [1] are easily fixed, since bibdb allows users to modify any entry in the database.

Since BIBDB depends on users to add its entries in the first place, we must explicitly trust the user when necessary. Despite the best heuristic efforts, some duplicate entries will undoubtedly be introduced. In that case, two different citation keys will point to (potentially) different data for the same real reference. This violates one of our fundamental goals, since there is no longer a one-to-one mapping of unique keys to unique references. However, we cannot allow a user to fix this problem by simple deleting a citation key and its corresponding reference, because that key would no longer be valid. Some other user who cited that key in a document would then have an invalid document, violating the citation permanence design goal. Once a document is successfully generated, changes to the bibliographic database should not alter the citation structure such that the document's citations are no longer valid. Therefore, we allow a user to fix duplicate entries by "redirecting" one of the keys to the other key for the same reference. The redirected key is still a valid citation key, but now it points to the data associated with the other key. Changes made to the data will be visible to citations using either key.

BIBTEX provides the string alias mechanism both to save typing effort and to lend consistency to references. For example, good bibliographic style dictates a consistent format for the titles of conferences and workshops. A bibliography which cites "The Proceedings of the Third International Conference on Widgets" and also "Proc. 3rd Int'l Conf. Widgets" is inconsistent and undesirable. Thus bibdb provides a powerful automatic string substitution mechanism which encourages users to employ string aliases, leading to more compact references and consistent formatting. The program examines the fields of each entry, heuristically checking to see if they resemble strings which are defined. If so, the program offers those substitutions to the user, as illustrated near the bottom of Figure 4.

Finally, since one of the major goals of BIBDB is to conveniently share the bibliographic data, multiple users may run bibdb in parallel, concurrently accessing and modifying an installation's database. This is a particularly tricky feature, since UNIX does not strongly support multiple-writer files. In fact, all changes to the database files must be serialized, but bibdb restricts the serialized sections of code (monitors) to very short operations, thus allowing most bibdbs to carry on interactions with their users while one bibdb has the database briefly locked. Locks on the database are only acquired once a final decision has been made to modify the database. When any bibdb user is modifying the database, all bibdb accesses to the same installation are preceded by cache flushes and reloads, so that the data which the bibdb user sees is as up-to-date as possible.

The bibliographic database

The BIBDB database is optimized for access by the unique key which must be associated with each BIBTEX entry. The bibliographic data is stored in "gdbm" format, which is a freely-licensed database format based on the "ndbm" system supplied with most UNIX installations. Perl can access these database files as though they were associative arrays (that is, arrays indexed by arbitrary strings). So, bibliographic entries are stored in a huge associative array, indexed by their unique citation key.

```
key -->  [database] --> entry
```

The cross-index

To allow the user to find an entry whose key is unknown, all words in each entry are used as indices into an associative cross-index, which maps arbitrary words to the keys of entries which contain those words.

```
word --> [cross-index] --> list of keys
```

The bibdb program gives the user a powerful set of searching primitives which locate references based on boolean combinations of keywords, as listed in the cross-index. For example, the command "find author musliner and title database" specifies a search for entries in which the author field contains "musliner" and the title field contains "database". These searches are always case-insensitive. The cross-index database maps each field/value pair in the search

```
unix% cat new.bib                                          Print out the file of new data.
@article{ignored-key,
        author = "D. J. Musliner and J. W. Dolter",
        title = "A Cool Bibliographic System",
        journal = "J. of Cool Computing",
        year = 1992 }

unix% bibdb                                                Start the database maintenance program.
Connected to bibdb-server at [huron.eecs.umich.edu]
bibdb> add new.bib                                         Add the new data file.
WARNING: a key collision has occurred.                     Bibdb detects a similar entry.
The key for the existing entry :
    @inproceedings{musliner:92,
               author = "D. J. Musliner and J. W. Dolter",
               title = "Another Neat Bibliographic System",
               booktitle = "Proc.  Conf.  on Important Things",
               year = 1992 }
Conflicts with the proposed key for the new entry :
    @article{musliner:92,
               author = "D. J. Musliner and J. W. Dolter",
               title = "A Cool Bibliographic System",
               journal = "J. of Cool Computing",
               year = 1992 }
Enter one of the following options:                        Bibdb lists options.
    (c)autiously continue:  try the next lexical key
    (a)bort the addition
    (r)eplace existing entry with new entry
Title comparison suggests the entries are NOT the same     And provides advice based
and you should choose the (c)autious option                on heuristics.
key-collision-action> cautious                             User agrees entries are different,
Continuing addition attempt:  generating new key           so bibdb tries new key
Adding new entry with key [musliner:92a]                   and finds no conflict.

The following strings are similar to the                   Bibdb detects a common string.
original journal ["J. of Cool Computing"]:
         1 :  CCJ = "The Cool Computing Journal"
Please choose a number, or <Return> for the original value
string choice> 1                                           User chooses the suggested alias.
bibdb>
```

Figure 4: Showing how the `bibdb` interface tries to prevent duplicate entries and encourages common string usage when a new data file is added. Note the new entry's citation key in the data file is ignored, and `bibdb` creates the key itself.

command to a list of citation keys. The "and" indicates that the result of the whole search command should be the intersection of the two intermediate lists. Replacing "and" with "or" yields the union of the intermediate lists.

Finding the intersection or union of lists involves a linear scan of the lists, and thus can have relatively high cost. To limit the maximum time which a search can take, and also to restrict the size of the cross-index entries, words which map to too many[1] entry keys are declared "overloaded," and are not allowed as cross-index indices. However, they may still be used in searches: as long as a non-overloaded keyword is specified, the search routines return a list of keys[2] which is then scanned by a post-processing step for the overloaded search terms.

THE KEYMASTER

The `keymaster` is responsible for ensuring that each of the installation databases is "consistent." Essentially, the `keymaster` must make sure that every valid citation key is assigned to exactly one entry (hence the name). No installation must be able to assign a key that has already been used, and all

[1] N = "too many" is a constant on the order of 500-1000.

[2] of length $\leq N$.

installations must agree on which entry a particular key points to. However, since our bibliographic database does not have the temporal consistency requirements of a banking database, the installations need not be identical at all times. In fact, the only time it is important for an installation to be completely up-to-date is when a user is trying to modify the database: `bibdb` must have access to all the existing entries, so it can issue an unused key and make sure the user is not creating a duplicate entry. Therefore, we have implemented a type of "relaxed consistency" in which installations can become outdated if they have not been changed recently.

Relaxed consistency

The `keymaster` assigns a unique, monotonically increasing *update-id* to each database change sent from a `bibdb-server`. When a `bibdb-server` initiates a modification to the database by contacting the `keymaster`, it first requests all the updates since its last contact (as identified by its most recent update-id). The `keymaster` sends all the more recent updates back to the `bibdb-server`, thus ensuring that the installation's database is up-to-date before the change.

This approach has the advantage that installations only need to be connected to the `keymaster` when the user tries to change the database. Periods of Internet downtime do not completely incapacitate the BIBDB system: installations which cannot connect to the `keymaster` essentially become read-only, so users can still extract entries which they have used before. Documents which were successfully produced in the past are not suddenly crippled by network problems.

Despite the relaxed consistency technique, the `keymaster` is the bottleneck of the BIBDB system. All changes to the database must eventually be serialized at the `keymaster` (because UNIX does not support multiple-writer files). The `keymaster` does not actually have a gdbm copy of the database: it only maintains a file containing all of the modifications to the database (the *updates-file*). The updates-file is the single-writer bottleneck, since database changes must be recorded serially in the file. Although a `keymaster` is forked for each `bibdb-server` which is trying to change the database, these `keymasters` cannot all write to the updates-file at the same time. We have minimized the cost of this restriction through a technique we call "key-root locking."

Key-root locking

If a user is going to add a new entry, the addition may take as much as a minute or two, because the user may have to resolve key conflicts and string-substitutions. We do not wish to have the updates-file locked for that entire time. So, we allow the `bibdb-server` to request an exclusive key-root lock, disallowing all other changes which involve database entries with the same key-root (author name and year). The key-root lock is held for the duration of the bibdb/user interaction dealing with the new entry. But the updates-file is locked only briefly, once the user has confirmed the addition. The `bibdb-server` sends the confirmation to the `keymaster`, which locks the updates-file, writes out the update information, and immediately releases the lock on the updates-file. Thus, the time during which the forked `keymasters` must be serialized (essentially, the monitor section of the `keymaster`) is extremely short. Multiple users can simultaneously interact with `bibdb` to arrange changes to the database, as long as the key-root locks do not conflict.

In fact, the forked `keymasters` are also serialized when they issue a new update-id, since that number must be unique and monotonically increasing. The key-root locks and update-id locks are implemented through *flock*, the UNIX file locking facility.

BIBEXTRACT

The `bibextract` program interfaces the BIBDB database system into the normal LaTeX/BibTeX system. `Bibextract` finds the citations in a LaTeX document and retrieves the corresponding entries (and all necessary string definitions) from the BIBDB database, building a reference file tailored to the exact needs of the document.

Normally, the "\cite{}" commands in a LaTeX document will specify the exact BIBDB citation key, so that `bibextract` can use a simple lookup in the BIBDB database. The time required for this extraction process does not grow linearly with the size of the database, because the database is indexed associatively.

`Bibextract` also allows users to pass the "\cite{}" command an "imprecise citation," consisting of a set of semicolon-separated words which appear in the desired bibliographic entry (i.e., an imprecise citation for this paper might have the form "\cite{musliner;bibdb;1992}"). `Bibextract` will attempt to resolve imprecise citations to unique entries using the BIBDB associative cross-index.

COMPETING DATABASES

This section compares BIBDB to a variety of competing bibliographic database systems, both commercial and public domain. We demonstrate that BIBDB provides a unique set of features, combining the best aspects of many other systems while largely avoiding their disadvantages.

Refer

The *refer* system mentioned in the introduction is an older reference-maintenance system which was developed to work with *troff*. Since *refer* has been around so long, many people have very large *refer* databases, and there are various modifications available to make the system usable with other text processing systems. *Refer* is the most popular competitor to BibTeX.

Aside from minor formatting differences, the primary difference between the BibTeX and *refer* systems is that *refer* allows imprecise citations. *Refer*'s `indxbib` program examines a *refer* text database and creates a cross-index similar to the BIBDB cross-index. The `refer` program uses this cross-index to attempt to resolve imprecise citations to unique articles. If the citation is not sufficiently precise, so that it matches more than one database entry, `refer` prints an error message. The citation must then be enhanced to specify a unique reference.

During the design of BibDb, we carefully considered the strengths and weaknesses of *refer*-style imprecise citations. The main advantage of the technique is that a user need not remember some (possibly cryptic) unique key to cite a paper. There are several significant disadvantages. Primarily, imprecise citations do not permit the "citation permanence" guarantees we desire. Consider the case when a user has cited a reference and generated a paper, but the next day the paper is no longer properly generated because another user has added an entry which also matches the imprecise citation, and thus the citation is no longer sufficient. We consider this an unacceptable failure, especially since we intend BibDb to facilitate collaboration and incremental expansion. Confusion and annoyance could only result if a co-author found that another co-author's imprecise citation suddenly did not indicate a unique entry.

However, since we already have a fully developed cross-index to allow nonlinear searches of the database, there is no reason we can not also allow imprecise citations in the *refer* manner. Thus, `bibextract` does attempt to resolve imprecise citations, as described earlier. In fact, BibDb implements imprecise citations with lower cost and greater flexibility than *refer*. If a *refer* user does not run `indxbib` on the entire database, `refer` must use a linear scan to resolve citations. Each change to a *refer* database requires that the entire database be re-indexed by an `indxbib` linear scan. BibDb, on the other hand, performs its cross-indexing incrementally, as each entry is added, so once an entry has been cross-indexed it need never be done again. The `bibextract` operation never involves processing the entire database.

The `indxbib` program itself has a number of limitations which the BibDb cross-index does not share. `Indxbib` truncates all words to six characters and discards words shorter than 3 characters, numbers less than 1900 or greater than 2000, and the 100 most common English words. The BibDb overloaded-word mechanism is a far more flexible, dynamic implementation of the same attempt to reduce the size of the cross-index. And, since overloaded words can still be used in imprecise citations as long as at least one non-overloaded word is used, BibDb provides more powerful and less costly imprecise citations than *refer*.

Mail-servers

A number of bibliographic database projects have recently made their databases available through mail-servers [2, 4]. In these systems, users email specially formatted queries to a public address, where an automated mail server processes the requests, usually overnight. The main disadvantage of these systems is that they are not on-line. Users must wait for unpredictable electronic mail to transmit their data, and they cannot find out if their queries were even properly formatted until at least the next day. Our queries to the LIDO mail-server [4], for instance, took over 2 days to return.

Mail-servers have the advantage that, since their processing can be batched to run when other demands on the host system are low, they can provide computationally expensive services. In particular, all of the mail-servers we have encountered use a simple linear scan of a text database. Since they are already doing linear scanning, the mail-servers can afford to allow regular expressions in the queries. Thus, mail servers can easily be implemented as front-ends to UNIX filters of the "grep" family. While regular expressions provide a more powerful mechanism for blind searching when the desired reference is not known, we consider the delays associated with mail-servers to be unacceptably long. Essentially, these services can be useful for literature searches, but are not integrated with text formatting systems and provide no way for users to add or modify entries.

However, mail-servers do not require Internet access: any user who can send and receive email can access the mail-servers. This vastly increases the number of people who can use these services. Fortunately, nothing in the nature of BibDb prevents us from implementing a simple mail-server interface to accommodate those users. While the interactions required to verify changes to the database could become complex and slow, the retrieval functions could certainly be on par with any existing mail-server.

Other BibTeX systems

Bibliographic database managers are a recurrent theme on the electronic newsgroups related to text formatting. A number of users have made their systems available to the Internet community [3, 7]. Most of these systems essentially provide front-end interfaces to textual BibTeX databases, usually including regular-expression matching. While these systems are useful for maintaining small, personal databases, they have no provisions for sharing data or avoiding linear scanning. BibDb is a far more powerful and wider-scope database management system.

Commercial systems

Most modern research libraries have electronic card catalogs which maintain records of bound publications. There are also a few services which list not just bound publications, but the separate articles within those publications. For example, the University of Michigan library provides network access to the Wilson Indexes to Journal Articles, a commercial database listing references from the Applied Science and Technology Index, Art Index, Social Sciences Index, and others. This resource provides exactly the sort of on-line keyword searching which BibDb provides, over a much larger database than BibDb currently controls. While the system is not interfaced to any bibliographic system, it probably could be. However, this may never occur, because the commercial systems all copyright their data. Using their data to directly create a document might violate that copyright.

The main advantage of these systems is that they can achieve complete coverage of the contents of periodicals, because the licensing fees are used to hire people whose sole responsibility is to input data. By relying on users for input, BibDb grows slowly and provides only spotty coverage of publications. On the other hand, it also necessarily includes exactly the articles which users find useful, and thus it may be considered a pre-filtered source for literature searches, less complete than commercial systems, but more convenient when it comes time to write a paper or report.

Aside from copyright limitations, nothing prevents us from integrating the data from larger databases into the BibDb database. We hope that in the future, major periodical publishers like the IEEE and ACM will send bibliographic information on their new publications directly into the BibDb database, eliminating most user additions and improving coverage.

CURRENT STATUS & FUTURE WORK

The prototype BibDb system operates as described in this paper. The initial release is confined to the University of Michigan, so only a single `bibdb-server` is kept running. The system is currently managing a database of over 15,000 bibliographic entries and over 250 string abbreviations. The associative retrieval of entries from known keys is essentially instantaneous. Searches that require computing the union and/or intersection of lists of entries take no more than a few seconds. This version and a previous, single-user version have been in use for over two years, helping members of the Real-Time Computing Laboratory (RTCL) at the University of Michigan produce dozens of papers. Several users outside of RTCL have also recognized the advantages of the system, and now use BibDb full-time. The database has proven quite useful for blind searches, yielding dozens of useful references in our experience.

We intend to improve several aspects of the implementation, as outlined below.

Locking

Because BibDb shares several kinds of writable data among multiple users, it makes extensive use of locking. We currently use the UNIX *flock* mechanism to implement locking. This method requires file system accesses which make locking fairly slow, thus increasing the severity of the `keymaster` bottleneck. Since all the `keymaster` copies run on the same machine, we could use shared-memory semaphores to implement locking, speeding the `keymaster`'s response time and increasing the number of `bibdb-servers` which could be served without significant performance degradation. Similarly, all the `bibdb-servers` at an installation run on the same machine, and could do their own locking via semaphores.

Data compression

Not surprisingly, the BibDb database occupies large amounts of disk space. While the programs are quite small, the database is huge: the current bibliographic database occupies over 4.6 megabytes of disk space, and the cross-index consumes 31.7 megabytes. Obviously, using data compression techniques to reduce this excessive space requirement would be a desirable feature. We have examined several mechanisms for compressing the bibliographic data.

In order to maintain the incremental nature of BibDb, we cannot compress the entire database at once: each entry must be stored in the gdbm file in its compressed form, independent of the other entries. Thus, so-called "universal" coding algorithms [10] which optimize themselves to their input are inappropriate, because the bibliographic entries do not supply enough data to establish valid statistical features. Tests with the UNIX universal coding `compress` utility confirm that strings of 300 bytes[3] can be compressed as little as 11%.

Coding algorithms which do not require lengthy input streams are more appropriate. Huffman coding provides high compression rates (on the order of 45–65%) when given a fixed set of input probabilities [6, 8]. We have written a utility which computes these probabilities from a BibDb database. The existing database contains a very large sample of the expected BibDb data, and thus provides input probabilities which will be representative of most new entries. All that remains is to integrate Huffman encoding and decoding routines with the gdbm library.

REFERENCES

[1] C. P. Bourne, "Frequency and Impact of Spelling Errors in Bibliographic Databases," *Information Processing Management*, vol. 13, no. 1, pp. 1–12, 1977.

[2] K. Ginther-Webster. *A New Resource for AAAI Members: Project Mercury: An Electronic Library*. in letter from AAAI, 1990.

[3] P. King. *Bibtex Database Management Programs*. USENET news article 4070 on comp.text.tex, December 1990.

[4] A. Kobsa. *The LIDO Mailserver for AI Literature*. USENET news article 7513 on comp.ai, November 1990.

[5] L. Lamport, LaTeX *: A Document Preparation System*, Addison Wesley, 1986.

[6] D. Severance, "A Practitioner's Guide to Database Compression," *Information Systems*, vol. 8, no. 1, , 1983.

[7] A. Shah. *Bibliography Management Tools, Take 2*. USENET news article 4089 on comp.text.tex, December 1990.

[8] J. D. Ullman, *Principles of Database Systems, Second Edition*, Computer Science Press, 1982.

[9] L. Wall and R. L. Schwartz, *Programming perl*, O'Reilly & Associates, Inc., 1991.

[10] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. Information Theory*, vol. IT-23, no. 3, pp. 337–343, May 1977.

[3] The current database averages 280 bytes per entry.