The BIMA data archive: the architecture and implementation of a real-time telescope archiving system

Raymond L. Plante[a] and Richard M. Crutcher[a,b]

[a]National Center for Supercomputing Applications
[b]Department of Astronomy
University of Illinois
Urbana, Illinois 61801 USA

ABSTRACT

We describe the design model and implementation of the data archiving system used by the Berkeley–Illinois–Maryland Association (BIMA) Millimeter Array Telescope. This system transmits data in real-time from the BIMA telescope at the Hat Creek Observatory in northern California via the Internet to an archive server at the National Center for Supercomputing Applications (NCSA) in Urbana, Illinois. Once at NCSA, the data undergo minor processing to make it available to astronomers via an HTML interface: *(1)* the data are checked for successful transmission, *(2)* metadata are extracted from the data and inserted into a searchable database, and *(3)* the data are sent to the NCSA Mass Storage System. When necessary, the system can carry out *rollback* operations which allow it to easily recover from errors, particularly those associated with the often unstable Internet. We also comment on some ways in which the system can be improved and expanded to adapt to changing observing strategies.

Keywords: remote real-time observing, data archive, networked systems, radio astronomy, synthesis imaging

1. INTRODUCTION

The efforts at many modern telescopes to operate at the forefront of science and technology have resulted in a rapid growth in the rate of data production, and the recent evolution of the Berkeley–Illinois–Maryland Association (BIMA) Millimeter Array (located at Hat Creek, CA) is no exception.[1] Improvements in the telescope's speed and flexibility through additional antennas and a spectrometer with more channels contribute directly to BIMA's growing data rate. Efforts to extend telescope operation to higher resolutions and frequency and the need for shorter data sampling times are also having major effects.

Archiving and processing data that are being produced at a high rate can present real technical challenges, especially if these operations cannot happen at the telescope site. Traditionally, data from the BIMA telescope have been off-loaded to an astronomer's desktop workstation for processing and analysis; however if a project is made up of particularly large amounts of data, the astronomer may need to call on a supercomputer to process the data, especially when fast turn-around is needed.[2] To prevent a data bottleneck, high-performance resources are necessary to archive and process the data at a rate faster than the rate at which they are being produced. In particular, the data need to be transferred in *real-time* to an archive where it can be made available for immediate processing, either by individual workstations or by a supercomputer.

The Radio Astronomy Imaging (RAI) Group within the National Center for Supercomputing Applications (NCSA) has collaborated with BIMA to implement a real-time archiving system. The goal of the system is to transfer raw telescope data to a central archive where it can be made available to astronomers soon after the observations are made. Like most archives, the accumulation of data over time provides an opportunity for so-called *archival research* in which old data can be reused to answer new questions beyond the data's original intent. The addition of a "real-time" capability to the system opens the door for a new kind of research in aperture-synthesis radio astronomy: *real-time observing.* In this mode, the observer can send the data to a supercomputer for fast pipeline processing and then use the results of that processing to alter the observing strategies.

Other author information: R.L.P.: Email: rplante@ncsa.uiuc.edu; Telephone: 217-244-4209; Fax: 217-244-7638.
R.M.C.: Email: crutcher@uiuc.edu; Telephone: 217-333-9581; Fax: 217-244-7638

The power of near-real-time observing was recently demonstrated in the early spring of 1997 during a study of the molecular composition of Comet Hale-Bopp. Astronomers only had a limited amount of time to observe the comet as it passed by Earth's view in its 2000-year long orbit around the Sun, so BIMA observed the comet almost daily for the two weeks the comet was closest to Earth. The data were transferred in real-time to the archive at NCSA where it was then sent to a Silicon Graphics Origin 2000 supercomputer for processing. At the end of each observing day, astronomers used the 32-processor machine to process fourteen hours worth of data, containing observations of several molecular species, in about 100 minutes. Based on the quality of the data and the results they revealed, the astronomers were then able to plan the next day's observations. The study resulted in an unprecedented view of molecular line emission from a comet, including the first detection of the HCO^+ ion from a source within our own Solar System.

2. THE ARCHIVING MODEL

The design of the archiving system is based on a basic model for data production, transmission, and use that includes assumptions concerning the following:

1. the telescope model already in use
2. what defines a dataset
3. the criteria by which users of the archive will search for data
4. measures of robustness.

2.1. The Existing Telescope Model

- *The archiving system must fit into an existing telescope model for acquiring and recording data onto local disk with minimal changes to existing procedures and software.*

Observing with the BIMA telescope has traditionally been highly automated. An observer usually creates an observing program file days or weeks prior to the actual observations. An operator (almost always someone other than the observer) runs the program file through a scheduling script. Although real-time interaction with the observing procedures is rare, essentially all information about the telescope performance is available over the Internet, so the array can be completely controlled from a remote site for real–time observing. Calibration and conversion of the raw data into images are almost always done "off-line", days or weeks after the data are taken.

The core of the telescope control and data acquisition software is the Miriad software package.[3] (It is also used by the astronomers for processing the data.) Miriad stores data in its own format, designed especially for multichannel *uv** and image data. The Miriad software does not provide any support for data archiving. Written in FORTRAN and C, it is considered impractical to adapt this software to support archiving. However, the compiled programs are usually combined together into C-shell scripts which drive the day-to-day observing; support for archiving is most easily inserted at this level.

2.2. Definition of a Dataset

- *BIMA data are stored in Miriad format and therefore a dataset is a directory under which the components of the dataset are stored as files.*

Examples of dataset components include `header`, `vartable`, `visdata`, `flags`, and `wflags`. The `visdata` component (or Miriad *item*) is usually the largest component, containing the actual *uv*-data itself.

- *Most items within a Miriad dataset are assumed to be written sequentially (i.e. new data are appended at the current end of the file); those files that are not (such as* `header` *items, which are updated via random access) are fairly small.*

Real-time transfer is most efficient for sequentially written data, since with each update to the file, only the newly appended data need to be transferred to the archive. For a file updated via random access, the simplest way to update the version in the archive (and thus the way least susceptible to error) is to retransmit the entire file. Thus, such files should be small to keep the cost of re-copying low.

---

*the raw data taken by the telescope which represent an irregular sampling of the two-dimension Fourier transform of the sky brightness in each frequency spectral channel.

- *Data are collected as part of a project, which may contain several datasets. A dataset is therefore uniquely identified by:*
  1. *the date*
  2. *the project name*
  3. *the dataset name*

  Once a project ends, it is not restarted within the same 24-hour period.

This identification of the datasets reflects how the data are written to disk. All Miriad datasets written by the telescope observing software are written under a directory we will refer to as `$newdata`. All the datasets for a project that begin on the same date go into a single directory of the form `$newdata/`*ddmmmyy* (e.g. `$newdata/05apr97`). The data are further segregated beneath this directory by project name; each directory under this then represents a separate Miriad dataset. Thus, the path to a dataset might be `$newdata/05apr97/n341c110.oria/3c273.n`. Usually, each dataset represents a different set of observing parameters in use. This usually means that data for different sources (i.e. objects in the sky) go into different datasets; however, differences may be in the way the telescope backend was configured.

During the normal observations for a particular project, the telescope software will alternate between the various datasets, reopening files and appending new data according to the project's observing script. For example, during the bulk of a typical observing run, the telescope will observe a calibration source (say, a quasar of known brightness) for a few minutes between periods of observing the source of interest; the data from the two sources would go into different datasets.

2.3. Locating Data in the Archive

Archive users will need a way of searching the contents of the archive through useful search criteria. Unfortunately, given our very first assumption (that the archive must fit within the existing system), the criteria are restricted to the information about the data given by the telescope system.

- *Archive users must be able to locate data in the archive by at least date, project name, dataset name, position in the sky, or observing frequency.*

2.4. System Robustness

- *The archiving process must be highly robust in the following ways:*
  - *All data stored in the archive are exact copies of the data recorded by the telescope.*
  - *No data intended for archiving are lost.*
  - *All archived data can be located via in a general archive search.*
  - *The system must operate smoothly in an unstable environment, recovering from errors easily with minimal human intervention.*

The telescope can produce data at a large range of rates all day and night under a variety of conditions. The archiving system must be prepared to recover from problems associated with one or more of its various unstable components, the most unstable being the Internet. Data transfer may be slowed significantly or cut off completely due to high network traffic. The connection might be interrupted at any stage of the transfer. Furthermore, human error and system failures are expected to occur at both the observatory and the archive with the usual frequency. Because the archiving system is expected to be somewhat complex, flaws in the archiving system itself are expected to interrupt or hinder the data flow, sometimes only under certain pathological conditions.

In order to achieve this level of robustness, the system must be able to carefully track the state of the system and data being transferred. When an error occurs, the system must be able to determine the last successful step in the archiving and resume with the next necessary step. The system must also be able to check that previous steps were performed correctly before proceeding with non-reversible operations. (For instance, the system should check to make sure a dataset has been archived before it is removed from disk.) This basically describes our *rollback* recovery model.

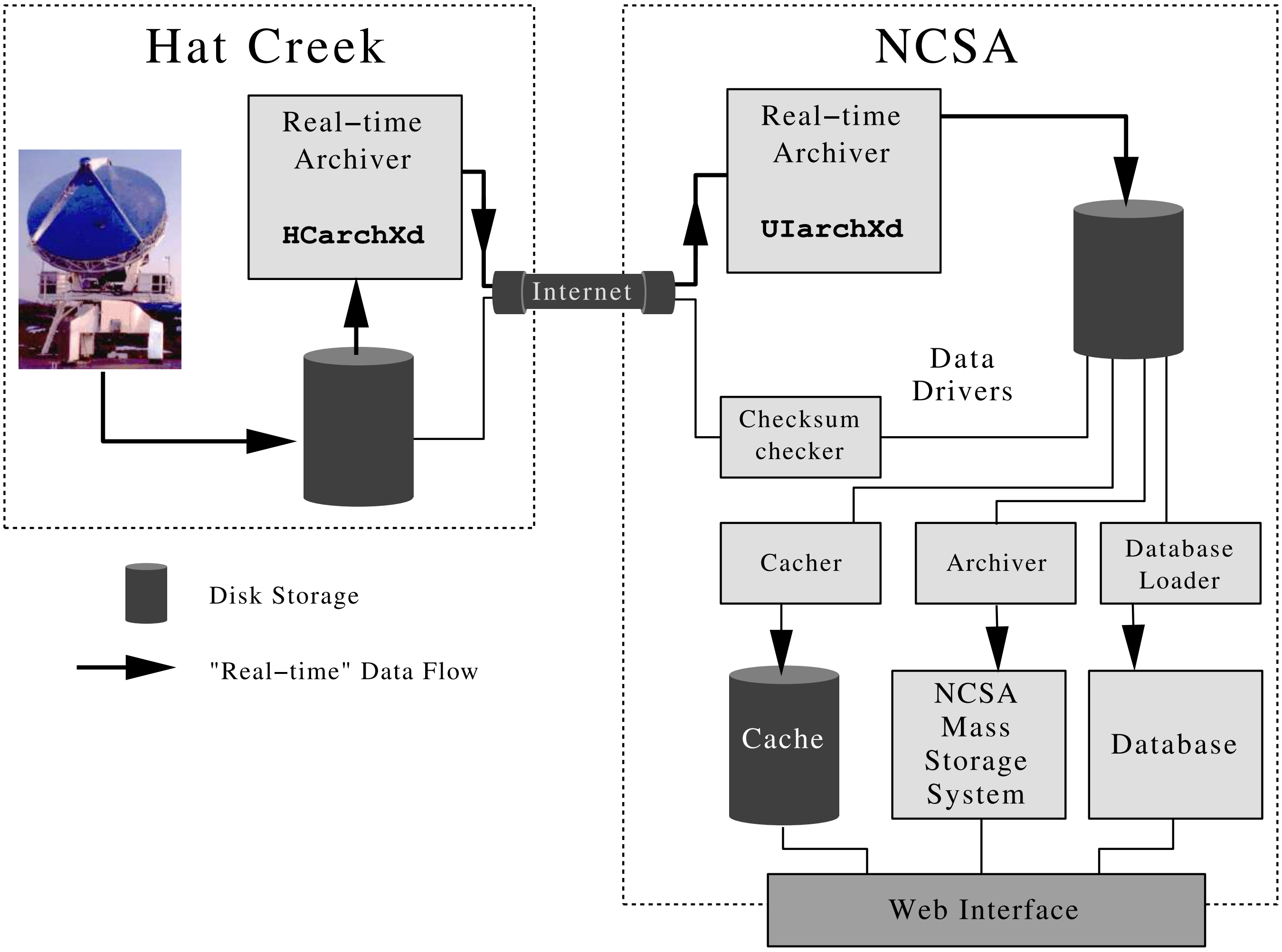

Figure 1. The BIMA real-time archiving system

3. SYSTEM COMPONENTS

Figure 1 illustrates the basic design of the archiving system. The telescope server at Hat Creek that is used to drive the telescope and send new data to NCSA is a Sun Sparcstation which uses a 4-Gigabyte drive to store new data. The archive server at NCSA that receives those data and makes them available to observers is also a Sun Sparcstation.† It uses approximately 15 GB of locally mounted diskspace for a data cache and for staging data currently being transferred. The Internet provides the data transfer link between the telescope and archive servers.

The *real-time archiver* refers to the portion of the archiving system that transfers data from the Hat Creek Observatory to the NCSA archive server machine within approximately one minute of its recording by the telescope operating system (assuming the network is available and a backlog of data to be transferred does not exist). This is represented in fig. 1 by the two daemon processes, `HCarchXd` and `UIarchXd`, which run on the Hat Creek and NCSA machines, respectively. Post-transfer data drivers prepare the data for access by users by copying the data to long-term storage and loading metadata into a searchable database.

The BIMA archive uses a commercial SQL database management system, called Interbase, to store metadata that allow archive users to search for BIMA datasets. Currently, this system is located a separate machine. Eventually, the database will be migrated to PostreSQL installed locally on the archive server.

†This machine is also used as the server for the NCSA Astronomy Digital Image Library (http://imagelib.ncsa.uiuc.edu/imagelib).

The NCSA Mass Storage System (MSS) serves as the long-term storage device. This system is based on a bank of fast IBM Magstar tape drives (loaded by a robotic juke box) and more than 285 Gigabytes of its own disk cache. The drives feature a data rate of 9 Megabytes/second, and they can seek to any position in their 10-Gigabyte tapes in less than 60 seconds. The MSS is connected to the archive server with an FDDI network connection providing 100 Megabits/second transfer rates. Given the performance of the MSS, the bottleneck in a typical session in which a user downloads tens of Megabytes worth of data to a remote workstation is almost always the Internet itself. Of course, loading data onto one of the local NCSA supercomputers (which are also connected to the MSS via FDDI) for processing is not subject to this bottleneck.

The MSS uses the UniTree Archival System which provides access to its data by wrapping a UNIX-like filesystem around them. UniTree automatically migrates files between tape and its disk cache as needed. Read-write access to the files is provided via an FTP interface. The archive server also gets read-only access to the files via an NFS mount of the UniTree filesystem. Among the special functions provided by UniTree is the ability to stage files from tape to disk prior to actual access.

The MSS is the main archive system for NCSA and has proven to be highly reliable. Users can request multiple copies to be stored on separate tapes (a feature employed by the BIMA archive). The data are periodically migrated to new tapes, and even the hardware is upgraded every few years for improved performance and reliability.

4. THE REAL-TIME ARCHIVING SYSTEM

In overview, data transfer is "driven" by `HCarchXd`, which runs continuously within a processing loop. With each cycle, `HCarchXd` goes through all the data in the `$newdata` directory, determines which datasets have been updated since the last cycle, and transmits the new data to the `UIarchXd` process. To accomplish this efficiently, the archiver must be able to record and keep track of the state for the data transfer; i.e. a record of how much has been transferred from each dataset at any given time must be kept.

4.1. Registering Data to be Transferred

A pair of routines, `HCarchTd` and `HCarchT`, are used for registering projects whose data should be archived. As part of the standard observing procedure, data are recorded for each project according to a master observing script. At the beginning of each project, the master script creates the appropriate date and project directories under the `$newdata` directory. At the same time, the master script runs `HCarchT -a` to signal to the real-time archiver that a new project should be prepared for archiving.

`HCarchT` does its job by sending the request to the daemon process `HCarchTd`, which does the real preparation. It first creates a directory structure (under a directory we will refer to as `$hc.outgoing`) that *shadows* the directory structure for the project under `$newdata`. It then creates a flag file under the shadow project directory called `startfile`. The content of this one-line file is the full path to project directory (under `$newdata`) that contains the real data. The existence of this `startfile` indicates that data contained within the project can and should be archived; therefore, versions of the datasets located on the NCSA machine are susceptible to updates.

When the observations for a project are complete, the master script runs `HCarchT -f` to tell the real-time archiver that no more data will be recorded for that project. This command causes the `HCarchTd` daemon to create a file called `stopfile` in the project's shadow directory under `$hc.outgoing`.

4.2. HCarchXd to UIarchXd: Sending Data to the Archive Server

The `HCarchXd` process transfers new data to the archive server and manages the state of the transfer in a few basic steps as part of a continuous loop.

1. *Create new shadow files.*

   For every project in progress, `HCarchXd` looks for new dataset items created since the last loop. For each new dataset item it finds, it creates a *shadow file* under `$hc.outgoing` which holds the following state information for that dataset item:

   - the pathname to the file containing the real data,
   - the number of bytes from the real data file that has been transferred so far, and

- whether the file has been completely transferred.

This state information is consulted and updated by the subsequent steps below.

2. *Determine which data files have been updated since the last transfer.*

   Prior to the actual data transfer, a tentative file transfer file list (i.e. a list of updated files) is generated. This is done by comparing the number of transferred bytes recorded in each shadow file with the size of the corresponding data file under `$newdata`; if the data file has grown, it is added to the list of files to transfer. Next, `HCarchXd` steps through this list, contacting the `UIarchXd` daemon at NCSA and requesting the size of its copy of each of these files. New data will only be transferred if the copy at the telescope is larger than the one on the archive server.

   The two-step process for determining which files have new data in need of transferring minimizes the communication overhead between the two archiving daemons. In addition to isolating the files that have been updated since the last cycle, it reduces the number of files whose sizes on the archive server must be checked. (Note that this scheme only works for the data files that are written sequentially.)

3. *Transfer new data*

   When `UIarchXd` is contacted by `HCarchXd` for a file size request, it opens that file in append mode, assuming that the next read from its socket will be new bytes to be added to the file. Meanwhile, `HCarchXd` receives the number of bytes already written to the archive copy from `UIarchXd`, opens its own copy for reading, and "fast-forwards" through the file past that number of bytes. `HCarchXd` then starts reading the new bytes and sends them to `UIarchXd` which appends them onto the archive copy. If `HCarchXd` decides that there are no new data to send (based on the file size returned by `UIarchXd`), it closes the socket, signalling `UIarchXd` to close its copy of the file.

4. *Update the transfer state for partially transferred data*

   Once the transfer of the new bytes from a data file is complete, the number of transferred bytes stored in the corresponding shadow file is updated. If a `stopfile` exists for the project that contains the data file, then the state stored in the shadow file is updated to indicate that no further transfers for the data file will be necessary.

   Note that if `HCarchXd` were to die or otherwise encounter an error while sending the data to `UIarchXd`, the recorded bytes transferred number would not get updated; thus the bytes would be resent in `HCarchXd`'s next cycle. This is why `UIarchXd` positions its file pointer just past the number of correctly sent bytes: it guarantees that the new bytes get written to the correct place in the data file, regardless of any previous attempts.

5. *Clean up shadow directories for projects that are no longer being observed.*

   When a `stopfile` appears in a project's shadow directory, observations have ceased for the project, and so updates to its component data files will also cease. When all the bytes have been sent for all the datafiles, as indicated by the bytes transferred number in the shadow files, transfer of the project is complete. To finalize the transfer, `HCarchXd` copies the project's `stopfile` to the archive server (via `UIarchXd`). Then all the shadow files and directories for the project are removed from `$hc.outgoing`. (The real data files, of course, are not removed.)

As mentioned above, this design really only works for sequentially-written data files. Miriad `header` item files are updated via random access. To allow for this, `header` files are handled separately, copied in full whenever any other data item from its project is written to. Since these files are small, there is very little overhead doing this.

It's possible that either daemon may die while processing data. To keep the archiver running, attempts to start the daemons are made every ten minutes via the UNIX `cron` facilities on the two servers. If either daemon is already running, the new version silently exits.

5. POST-TRANSFER PROCESSING DRIVERS

`UIarchXd` writes its copies of the datasets into a directory tree, `$hc.incoming`, that matches `$newdata` on the telescope server. The appearance of a `stopfile` indicates that a project is fully transferred and ready for archiving. At this point, some additional processing must take place before the data can be made available to users via the Web interface. The processing is primarily a matter of organizing the data and is carried out mainly by four Perl scripts (referred to as "Data Drivers" in fig 1), performing four basic tasks:

1. *Confirm that the data were transferred completely and accurately.*

   The `checksum` data driver finds datasets from projects in `$hc.incoming` with `stopfile`s and confirms that copies of the dataset files there are exact duplicates of the files on the telescope server. This done by carrying out a *checksum* operation[4] on both versions of each file in the project and checking whether the results are equal. (The checksum test for the telescope's version is executed on the telescope server via a remote-shell command using `rsh`.) Equal values of the checksum result indicate (to an extremely high probability) that the files are identical.

   If any of the files fail the checksum test, those files are recopied from the telescope server via a remote copy (using `rcp`). When the retransmission is complete, the project is ready to be checked again, so the next `checksum` is executed.

   When all the files for a project have passed the checksum test, the `stopfile` is removed, and the entire project is moved to a separate processing directory we will refer to as `$berkarchive`. Thus, the state of the data with respect to the both `checksum` driver and the real-time archiver (`UIarchXd`) is controlled by the location of the data. While the data are in `$hc.incoming`, it is susceptible to updates. By moving the data, we prevent either process from updating the data further.

2. *Extract metadata and bundle datasets into cached tarfiles.*

   The `cacher` script examines new projects under the `$berkarchive` area. For each dataset in the project, `cacher` first extracts metadata from the header and writes them to an ASCII flat file. Next, it bundles the dataset into a tar file, placing it into a local disk cache.

3. *Load metadata into database.*

   The `dbloader` script reads the ASCII flat file and sends the metadata to the Interbase database system. Because this particular database system is also used by other projects, it is located on another machine; thus `dbloader` must first copy the flat file to the remote system, remotely execute an Interbase client to actually load the data, and then remove the file from the remote system after successful loading.

4. *Transfer data to long-term storage.*

   The `archiver` script copies the tarred datasets and the metadata flat files to the NCSA Mass Storage System.

The four scripts are run simultaneously on the quarter-hour; however, when applied to a particular dataset, the tasks need to be performed in the order listed above. As a result, state must be recorded that keeps track of which processing steps have been applied already. This is accomplished through the use of filenames that reflect the state of processing of the dataset. Process lock files are used to prevent more than one copy of the script from running at the same time.

Each driver is designed so that it can be applied repeatedly to a dataset until it is successful. Each time it is applied, it checks to see if data need to be *rolled-back*—i.e, returned to the last successful state. This helps the driver cope with unexpected errors or interruptions. For instance, one can imagine that the archive server could go down while a driver script is in the middle of processing a particular dataset. The next time the script runs, it will see that the dataset has not yet successfully completed that step. If any files were created from the previous attempt, the files would be removed before the re-processing is started.

All of the data drivers keep records of their successes and failures which are compiled and sent to the human archivist daily. These logs are important for catching problems that cannot "fix themselves".[‡] For instance, if the data were corrupted at the telescope prior to the data transfer, there may be no way to correct the problem in an automated way (that is, there may be no way to roll-back to uncorrupted data). In this case, the driver will continually try to process the data and continually fail. To fix this situation, a human must intervene. Such a situation is fairly obvious in the daily logs sent to the human archivist, allowing him to respond.

[‡] An example of a problem that can "fix itself" might be when the server panics and reboots; the "problem" is fixed once the machine comes back up. In this case, the drivers resume their work as if nothing had happened.

6. DATA ACCESS FOR THE END USER

6.1. Interface Frontend

Users access data stored in the BIMA archive through the World Wide Web using a Web browser. In a typical session, the user will first access the archive's Query Page, an HTML form where the user can provide criteria for searching the archive's database. Currently supported fields are date, project name, dataset name, sky position (given as ranges of right ascension and declination), and observing frequency. When the search is submitted, a CGI program submits the query to the database and returns the results as a list of matching datasets identified by date, project name, and dataset name.

The user can browse all of the metadata associated with the dataset (not just the searchable fields) by "clicking" on the dataset name. Included in the metadata display is a link for downloading that individual dataset. Typically, however, a user will want to download all the datasets for a particular project. To make this more convenient, the query results page is actually an HTML form itself with "check-box" (i.e. on/off) buttons next to each dataset. To download multiple datasets, the user toggles the buttons for each of the desired datasets and then clicks the download request button. The requested datasets are then transferred to the client machine and unpacked on the local disk.

Because BIMA datasets are actually directories of files and because users usually want more than one dataset in a session, the Web's stateless, single-file request model is not instrinsically convenient for downloading this type of data. Users prefer to download the data with a single request, so the datasets must somehow be bundled into a single file by the server and then unpacked by the client. This requires that the client browser be configured (via a Mailcap file) to spawn special client applications to do the unpacking. Most BIMA users would rather not concern themselves with these details, so the archive provides users with a preconfigured browser for accessing the archive called `xmbima` which knows how to handle BIMA data.

Distributed as part of the Miriad software package, `xmbima` is actually a script that starts the NCSA Mosaic Web browser with a properly configured Mailcap file. When the user requests a single dataset from the metadata display page, the archive delivers a single UNIX tar file; `xmbima` handles the file by starting the UNIX tar command to unpack the file into a standard Miriad dataset directory. When the user requests multiple datasets, the archive actually returns a small text file listing the URLs for each of the individual datasets. `xmbima` handles this file by starting a specialized program called `xmrecvlist`. This application opens its own HTTP connections to the archive, downloads the requested datasets, and unpacks them on the user's disk. The client applications work transparently without requiring interaction with or even knowledge by the user.

The Web interface also provides a hypertext document that gives access to the data that have been transferred to NCSA but yet fully processed by the archive data drivers, including the dataset currently being observered. Thus, an astronomer has an opportunity to do *real-time observing*. In this mode, the observer can at any time during observations download the accumulated data, do fast processing of the data into images either on a supercomputer or desktop workstation, and use the results to change the course of the observations.

6.2. Interface Backend

The archive contains a master directory tree for holding all the datasets in the archive organized hierarchically by date and project name. (The `cacher` script updates this tree as new datasets arrive from the telescope.) Each project directory contains links to tar files located in the disk cache, each containing a separate dataset. Dataset tar files that have not been recently accessed are purged from the cache to make room for newly archived data or older data recently requested by users. When a user requests a dataset, the archive system first checks to see if it is in the local cache; if it is not, it is automatically restored from the Mass Storage System.[§] Once restored to the disk, it is delivered to the user. Typically, it takes much longer to transfer a project's worth of data over the Internet than it does to restore it from mass storage.

[§]The cache managed by the archive server is independent and in addition to the cache managed by the Mass Storage System.

7. FUTURE WORK

As new capabilities of the BIMA telescope come on-line, new observing strategies to take advantage of the capabilities come into use. These changes have suggested some ways in which the archiving system might evolve to efficiently handle data under a variety of observing modes.

The mode of observing that the system has the hardest time dealing with is one that involves high sampling rates. This mode is typically carried out during the winter season when the atmosphere is stable enough for very high angular resolution observing at the expense of requiring that data be recorded to disk as often as every 5 or 10 seconds, which increases the telescope's data rate tremendously. Normally achievable data rates for transferring over the Internet ($\sim$ 700 kilobits/second) are sufficient for keeping up with the production of data; however, occasionally there can be periods of high Internet traffic (which may be restricted to a particular area of the country such as Chicago or San Francisco). In the past, we have seen sustained periods of high traffic that set the archive behind by more than a day. Since Internet traffic is beyond our control, the only remedy to this would be in the form of dedicated bandwidth from an *intra-net* line.

Because the BIMA telescope is such a fast instrument, more projects are getting run for short periods of time. This has caused so-called *filler* projects to become more prevalent. These projects will get run whenever there are a few extra hours to spare between longer projects. Sometimes a filler project can get run more than once in the same day, a condition that is contrary to our basic archive model (see §2.2.). As a result, data can be unnecessarily transferred more than once. Some adjustments could be made to reduce this inefficiency.

Since the BIMA archive was first designed, the Taiwan Institute for Astronomy has joined the BIMA consortium as a supporting partner of the telescope, thus making the primary user group of the archive an international one. Many of the overseas users trying to transfer large datasets from the archive to their local workstations over low-bandwidth lines have found the HTTP channel unstable, since the connection often cuts off before all the data are transferred. To address this, we are planning to allow one to use FTP to do the actual data transfer but still use HTTP to search for data and request its staging to the server's local disk.

Finally, as the number of projects being observed with the BIMA telescope increases, there is growing interest in more careful tracking of what becomes of the data, particularly in terms of published scientific results. This has led to the development of tools for tracking information associated with each project using a database system, beginning with the initial observing proposal. It would useful if these data could be integrated with the archiving system so that users could search for data based on parameters other than those recorded by the telescope, such as author or scientific keywords.

Clearly, there are some disadvantages to developing an archiving system for a pre-existing telescope system. The hooks into the telescope that the archiving system must find are likely to be inelegant if not inconvenient and susceptible to failure. The best approach to designing an archiving system is one in which it is an integral part of the telescope model as a whole and developed in parallel with the rest of the telescope system. Such an opportunity exists the national millimeter array currently under development by the National Radio Astronomy Observatory. Taking advantage of object-oriented design, the archiving sub-system could be one portion of the total data pipeline connecting telescope to the community of users.

ACKNOWLEDGEMENTS

We wish to thank Randall Sharpe and Taisheng Ye for their work on the archive system and the National Center for Supercomputing Applications for support of the BIMA data archive. The work was supported by the HPCC Grand Challenge grant NSF ASC 92–17384.

REFERENCES

1. W. J. Welch, D. D. Thornton, R. L. Plambeck, M. C. H. Wright, *et al.*, "The Berkeley-Illinois-Maryland-Association millimeter array," *PASP* **103**, p. 93, 1996.
2. R. M. Crutcher, "Imaging the universe at radio wavelengths," *CSE* **1**, p. 39, 1994.
3. R. J. Sault, P. J. Teuben, and M. C. H. Wright, "A retrospecitve view of Miriad," in *Astronomical Data Analysis Software and Systems IV*, R. Shaw, H. E. Payne, and J. J. E. Hayes, eds., *ASP Conf. Ser.* **77**, pp. 433–436, 1993.
4. W. R. Stevens, *UNIX Network Programming*, Prentic Hall P T R, Englwood Cliffs, New Jersey, 1990.