Astronomical Data Analysis Software and Systems XII
ASP Conference Series, Vol. 295, 2003
H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

Status of the BIMA Image Pipeline

David M. Mehringer & Raymond L. Plante

National Center for Supercomputing Applications

Abstract. The BIMA Image Pipeline is nearing production mode. In this mode it will automatically process data that has recently been transferred from the telescope. Its products will be calibrated *uv* datasets, calibration tables, FITS images, etc. that will be ingested in the BIMA Data Archive and be retrievable by astronomers.

1. Introduction

The BIMA Image Pipeline[1] processes data from the BIMA Array[2] using the AIPS++ astronomical data processing package[3]. Currently, processing is initiated manually, but soon this process will start automatically after data from observing tracks have been ingested into the archive. Only single tracks can be processed at present, but in the future multiple tracks (e.g., from different telescope configurations) from the same project will be combined and processed. Processing jobs normally consist of multiple stages (e.g., serial processing for filling and calibration and parallel processing for image deconvolution). After the processing is complete, the data products are re-ingested into the BIMA Data Archive[4] where they are available for users to download.

2. Architecture of the Pipeline

Figure 1 is a block diagram of the BIMA Image Pipeline. The major components are discussed below.

2.1. The Event Server

Raw BIMA data are automatically transferred from the telescope at Hat Creek, California to NCSA over the internet where they are ingested into the BIMA Data Archive by the Ingest Engine. The Ingest Engine notifies the Event Server that new data have been archived and are ready for processing. The Event Server determines if the new data should be processed, and if so, creates a so-called

[1] `http://monet.astro.uiuc.edu/BIP/index.html`

[2] `http://bima.astro.umd.edu`

[3] `http://aips2.nrao.edu`

[4] `http://bimaarch.ncsa.uiuc.edu`

© Copyright 2003 Astronomical Society of the Pacific. All rights reserved.

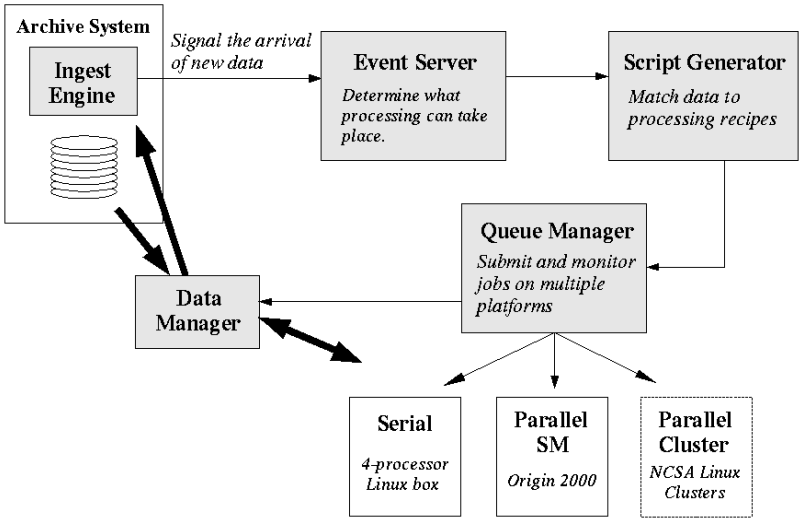

Figure 1. Architecture of the BIMA Image Pipeline

Shot document. This document is an XML document containing the metadata for the data collection (e.g., observing track) to be processed, wrapped in a root SHOT node. The Event Server places the SHOT document in an area where the Script Generator can locate it.

2.2. The Script Generator

The Script Generator is responsible for creating various scripts (Glish/AIPS++, shell/MIRIAD, and other shell scripts used for moving data) that are ultimately used for processing the data collection. It does this by transforming the Shot document created by the Event Server using XSLT into the various scripts to be used as well as XML metadata describing the products of the processing. Generally, a processing job is composed of several sub-stages. For example, a processing job to fill, calibrate, and image data is composed of a sub-stage that uses a shell script to copy the data to be processed to the work area, a sub-stage that uses a Glish script for filling and calibration in AIPS++ on a serial machine, a sub-stage that uses a Glish script for imaging and deconvolution on a parallel machine such as a Linux cluster, and a sub-stage that uses a shell script to archive the data products. The type of processing to be done (e.g., filling only, filling and calibration only, filling, calibration, and imaging) is encoded in the XML metadata of the collection. Each type of processing has an associated XSL stylesheet that is used to transform a Shot document to create the scripts appropriate for that specific type of processing. Upon successful generation of these files, the Script Generator notifies the Queue Manager that a collection is ready to be processed. This notification includes informing the Queue Manager in what order the scripts that were just generated should be run as well as in which queue each of these sub-stages should be placed.

2.3. The Queue Manager

The Queue Manager is responsible for scheduling, initializing, and monitoring jobs and sub-stages. The Queue Manager manages several queues. In general, each processing queue is configured to run a maximum number of jobs at any given time. A job starts out its life in the *pending queue*. When room becomes available in the queue in which the job's first sub-stage will run, the Queue Manager moves the job to the *running queue*. The first sub-stage is placed into the type of queue that is responsible for processing that type of job (e.g., the *unix queue* for processing shell scripts, the *serial queue* for processing AIPS++ jobs on serial machines, or the *parallel queue* for processing AIPS++ jobs on parallel platforms). If that sub-stage ends successfully (generally determined by the exit status of the script that it runs), the next sub-stage is started and placed in the appropriate queue. If all sub-stages complete successfully, the job finishes and is moved to the *success queue*. The data products (filled and calibrated *uv* data in AIPS++ Measurement Set 2 format, AIPS++ calibration tables, images in FITS format, etc.) of successful processing runs are ingested into the archive where they can be retrieved by users. If any sub-stage fails, the main job is terminated and moved to the *error queue* and none of the data products are ingested into the archive.

The primary means of monitoring the states of the various queues by both users and administrators is via web interfaces. These interfaces are powered by Apache Tomcat, which transforms XML documents describing the current queue state to HTML using Java Servlets. Information provided includes the start and end times of all the sub-stages associated with a job, the script used to run that stage, etc. The administrator view has the added features of allowing jobs to be purged from queues, jobs to be moved to other queues, etc.

3. Recipes

The scripts that are used for processing come from a *recipe library*. Each recipe is written to do a single phase of the processing. For example, there are standard recipes for filling, calibration, and imaging. Jobs (and their sub-stages) generally call numerous recipes. The recipes needed by a sub-stage are called by a top-level Glish script. For example, below is an example of a top-level Glish script that fills, calibrates, and images data from a single track. This script is invoked by a single sub-stage of a processing job (in this case, the sub-stage is run on a serial platform, though normally the deconvolution portion of imaging will be run on a parallel platform to take advantage of AIPS++'s ability to utilize such systems).

```
# set some project dependent parameters
sources := ['1733-130','3c84','sgb2n','venus'];
targets := ['sgb2n'];
phcals := ['1733-130'];
# include individual recipes
include 'pipelineinit.g';  # initialization
include 'filling.g';       # recipe for filling
include 'flagging.g';      # recipe for flagging
```

```
include 'calibration.g';   # recipe for calibration
bip.numprocs := 1          # number of processors
    include 'imaging.g';   # recipe for imaging
if(bip.imaging())          # main imaging function that returns
                           # true if successful
    note('Imaging completed successfully',priority='NORMAL');
else
    note('Imaging failed',priority='SEVERE');
include 'pipelinefinalize.g';      # finalization
note('Pipeline processing successful',priority=bip.note.NORMAL);

exit 0;
```

4. Summary

The BIMA Image Pipeline processes BIMA data and makes the products of this processing available to users by sending them to the BIMA Data Archive to be ingested. There are several components to the pipeline. The Event Server receives processing events and notifies the Script Generator that a data collection is awaiting processing. The Script Generator generates the various scripts that will be used to process the data as well as XML files that contain some of the metadata for the data products. The Script Generator informs the Queue Manager of the collection that is waiting to be processed and also tells how many sub-stages there are to the job and what type of processing each sub-stage requires. The Queue Manager manages the various sub-stages, and if all are successful, alerts the BIMA Data Archive of the new data products to ingest.