ASTRONOMICAL IMAGE MOSAICKING ON A GRID: INITIAL EXPERIENCES*

DANIEL S. KATZ† NATHANIEL ANAGNOSTOU‡ G. BRUCE BERRIMAN‡
EWA DEELMAN§ JOHN GOOD‡ JOSEPH C. JACOB†
CARL KESSELMAN§ ANASTASIA LAITY‡ THOMAS A. PRINCE¶
GURMEET SINGH§ MEI-HUI SU§ ROY WILLIAMS‖

Abstract. This chapter discusses some grid experiences in solving the problem of generating large astronomical image mosaics by composing multiple small images, from the team that has developed Montage (http://montage.ipac.caltech.edu/). The problem of generating these mosaics is complex in that individual images must be projected into a common coordinate space, overlaps between images calculated, the images processed so that the backgrounds match, and images composed while using a variety of techniques to handle the presence of multiple pixels in the same output space. To accomplish these tasks, a suite of software tools called Montage has been developed. The modules in this suite can be run on a single processor computer using a simple shell script, and can additionally be run using a combination of parallel approaches. These include running MPI versions of some modules, and using standard grid tools. In the latter case, processing workflows are automatically generated, and appropriate data sources are located and transferred to a variety of parallel processing environments for execution. As a result, it is now possible to generate large-scale mosaics on-demand in timescales that support iterative, scientific exploration. In this chapter, we describe Montage, how it was modified to execute in the grid environment, the tools that were used to support its execution, as well as performance results.

Key words. grid applications, astronomy, mosaics

1. Introduction. Astronomy has a rich heritage of discovery from image data collections that cover essentially the full range of the electromagnetic spectrum. Image collections in one frequency range have often been studied in isolation from those in other frequency ranges. This is a consequence of the diverse properties of the data collections themselves; images are delivered in different coordinate systems, map projections, spatial samplings and image sizes, and the pixels themselves are rarely co-registered on the sky. Moreover, the spatial extent of many astronomically important structures, such as clusters of galaxies and star formation regions, is substantially greater than those of individual images.

Astronomy thus has a need for image mosaic software that delivers science-grade mosaics from multiple image data sets as if they were single images with a common coordinate system, map projection etc. That is, the software must preserve the astrometric and photometric integrity of the original data, and rectify background emission

*Montage is supported by the NASA Earth Sciences Technology Office Computing Technologies program, under Cooperative Agreement Notice NCC 5-6261. Pegasus is supported by NSF under grants ITR-0086044 (GriPhyN) and ITR AST0122449 (NVO). Part of this research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

†Parallel Applications Technologies Group, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109

‡Infrared Processing and Analysis Center, California Institute of Technology, Pasadena, CA 91125

§USC Information Sciences Institute, Marina del Rey, CA 90292

¶Astronomy Department, California Institute of Technology, Pasadena, CA 91125

‖Center for Advanced Computing Research, California Institute of Technology, Pasadena, CA 91125

from the sky or from the instrument using physically based models. The Montage project [1] provides the astronomer with the tools needed to build mosaics in Flexible Image Transport System (FITS) [2] format, including support for all common astronomical coordinate systems, all World Coordinate System (WCS) map projections [3], arbitrary image sizes (including full-sky images) and rotations, and user-specified spatial sampling.

Montage has been designed as a scalable, portable tool kit that can be used by astronomers on their desktops for science analysis, integrated into project and mission pipelines, or run on computing grids to support large-scale product generation, mission planning and quality assurance. It will be deployed operationally on the Distributed Terascale Facility (TeraGrid) [4] and be accessible to astronomers through existing astronomy portals. In its initial deployment, Montage will serve images from the 2 Micron All Sky Survey (2MASS) [5], Digital Palomar Observatory Sky Survey (DPOSS) [6] and Sloan Digital Sky Survey (SDSS) [7], but will process all WCS-compliant images. It can therefore be considered an enabling technology, in that the mosaics it generates will widen avenues of astronomical research, and be a valuable tool in mission planning and quality assurance, including:

- Deep source detection by combining data over multiple wavelengths
- Predictions of source counts and wavelength extrapolations of fluxes
- Spectrophotometry of each pixel in an image
- Position optimization with wavelength
- The wavelength dependent structure of extended sources
- Image differencing to detect faint features
- Discovering new classes of objects

Two previously published papers provide much background for Montage. The first described Montage as part of the architecture of the National Virtual Observatory [8], and the second described some of the initial grid results of Montage [9]. This chapter covers much of the material in those papers, and describes some potential solutions to the problems in the initial grid implementations.

2. Montage architecture.

2.1. Processing steps. Montage employs the following four steps to compute a mosaic:

- Re-projection of input images to a common spatial scale, coordinate system, and WCS projection
- Modeling of background radiation in images to achieve common flux scales and background levels by minimizing the inter-image differences
- Rectification of images to a common flux scale and background level
- Co-addition of re-projected, background-corrected images into a final mosaic

Montage accomplishes these computing tasks in independent modules, written in ANSI C for portability. This toolkit approach controls testing and maintenance costs, and provides considerable flexibility to users. They can, for example, use Montage simply to re-project sets of images and co-register them on the sky, or implement a custom background removal algorithm without impact on the other steps, or define a specific processing flow through custom scripts.

TABLE 1
The components of Montage version 1.7.

Component	Description
`mImgtbl`	Extracts the FITS header geometry information from a set of files and creates from it an ASCII image metadata table used by several of the other programs.
`mProject`	Reprojects a single image to the scale defined in a pseudo-FITS header template file (an ASCII file with the output image header lines, but not padded to 80 characters and with new lines at the end of each line). Actually produces a pair of images: the reprojected image and an "area" image consisting of the fraction of input pixel sky area that went into each output pixel.
`mProjExec`	A simple executive that runs `mProject` for each image in an image metadata table.
`mAdd`	Co-adds the reprojected images using the same FITS header template and working from the same `mImgtbl` list.
`mOverlaps`	Analyzes an image metadata table to determine a list of overlapping images.
`mDiff`	Performs a simple image difference between a single pair of overlapping images. This is meant for use on reprojected images where the pixels are already lined up exactly.
`mDiffExec`	Runs `mDiff` on all the pairs identified by `mOverlaps`.
`mFitplane`	Fits a plane (excluding outlier pixels) to an image. Meant for use on the difference images generated above.
`mFitExec`	Runs `mFitplane` on all the `mOverlaps` pairs. Creates a table of image-to-image difference parameters.
`mConcatFit`	Concatenates the output of multiple `mFitplane` runs into a single file for use by `mBgModel`. It is only used when `mFitplane` is called directly by the user, rather than through `mFitExec`. (Note: `mConcatFit` was not in Montage version 1.7, but was added for grid processing very soon thereafter.)
`mBgModel`	Modeling/fitting program which uses the image-to-image difference parameter table to interactively determine a set of corrections to apply to each image to achieve a "best" global fit.
`mBackground`	Removes a background from a single image (planar has proven to be adequate for the images we have dealt with).
`mBgExec`	Runs `mBackground` on all the images in the metadata table.

2.2. Sequential processing. The first public release of Montage [10], version 1.7 (October 2003), supports serial processing of images, with processing of the four steps described above controlled through a set of simple executives. It is written in ANSI C for portability, and does not use shared memory. It has only been rigorously tested on machines running Red Hat Linux, but has been successfully run under Solaris and Mac OS X, among others. Figure 1 shows the high level design for the original release of Montage, which is described in more detail in Table 1.

Montage version 1.7 emphasized accuracy over speed. In order to support the broadest range of applications, the basic Montage reprojection and image flux redistribution algorithm works on the surface of the celestial sphere. All pixel vertices from both input and output images are projected onto this sphere; if necessary, a coordinate system transform is applied to the input pixel vertices to put their sky coordinates in the same frame as the output. Then, for overlapping pixels, the area of overlap (in steradians) is determined. This overlap, as a fraction of the input pixel area, is used to redistribute the input pixel "energy" to the output pixels.

In this way, total energy is conserved for those input pixels which do not extend beyond the bounds of the output image area. Even when a pixel has "undefined" vertices, such as at the boundaries of an Aitoff all-sky projection, the same process can be applied by determining an edge pixel's outline on the sky, described in the general case as a spherical polygon. The co-addition engine then creates the final

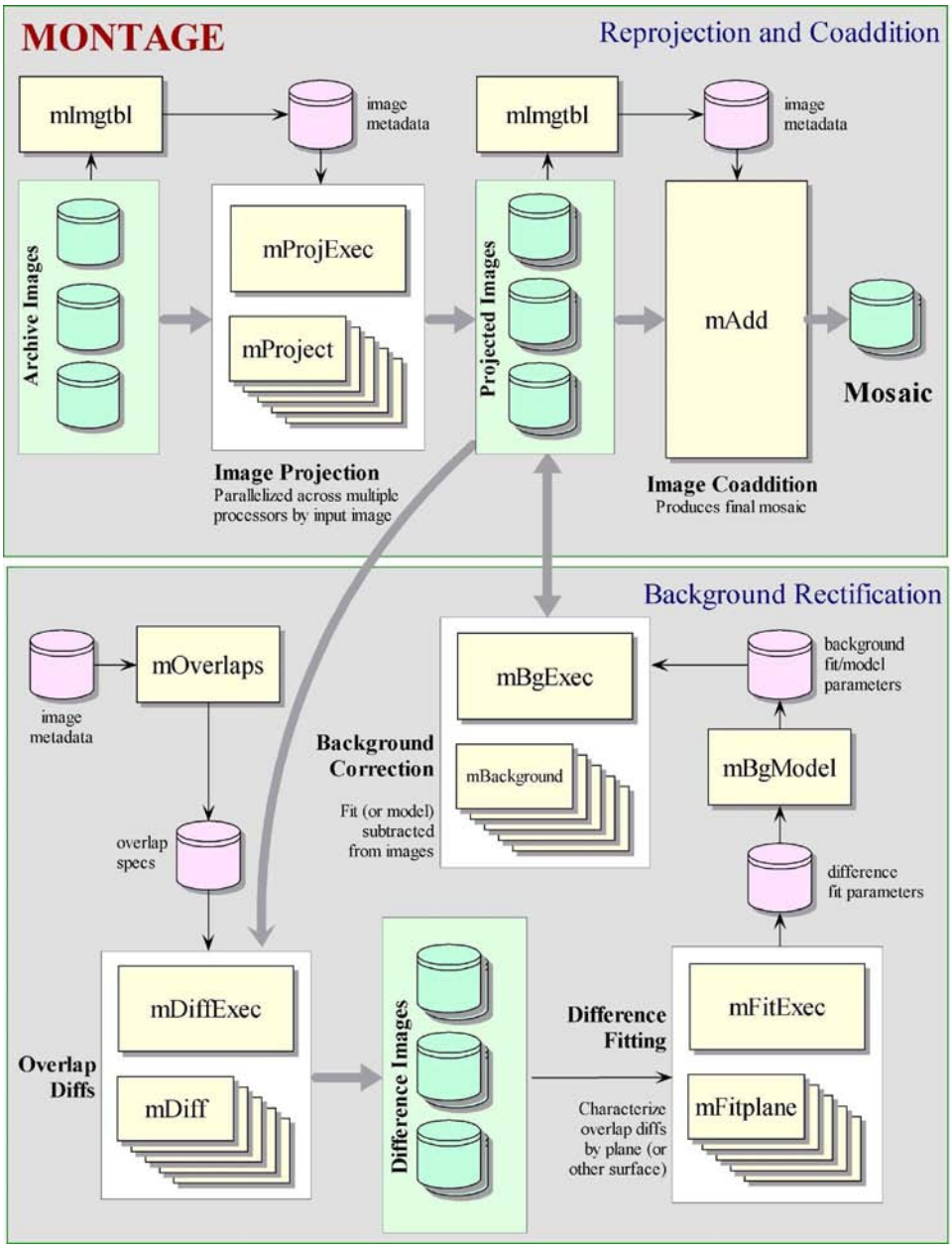

FIG. 1. *The high-level design of Montage. The figure shows the background rectification engine, and the reprojection and co-addition engines. The components of each engine are shown.*

mosaic by reading the reprojected images from memory and weighting each pixel's flux by total input area.

This approach is completely general and preserves the fidelity of the input images. A comparison of sources extracted from the mosaics with the SExtractor source extraction [11] program shows that, in general, Montage preserves photometric accuracy to better than 0.1% and astrometric accuracy better than 0.1 of a pixel, for the 10

WCS projections subjected to rigorous testing (using 2MASS images) [12]. Generality in reprojection is achieved at the expense of processing speed. For example, reprojection of a 512 × 1024 pixel 2MASS image takes 100 seconds on a machine equipped with a 2.26-GHz Intel processor and 1 GB memory, running Red Hat Linux 8.0.

As an example, consider a 1-degree square mosaic of the Galactic Center as measured by 2MASS. Figure 2a shows an unrectified mosaic. The striped appearance arises because different scan paths were observed at different times and through different atmospheric path lengths. Figure 2b shows an example mosaic generated by the Montage prototype code: the images in Figure 2a have been background rectified to generate a seamless mosaic in Figure 2b. The image is not, however, of science grade and should be considered as a proof-of-concept.

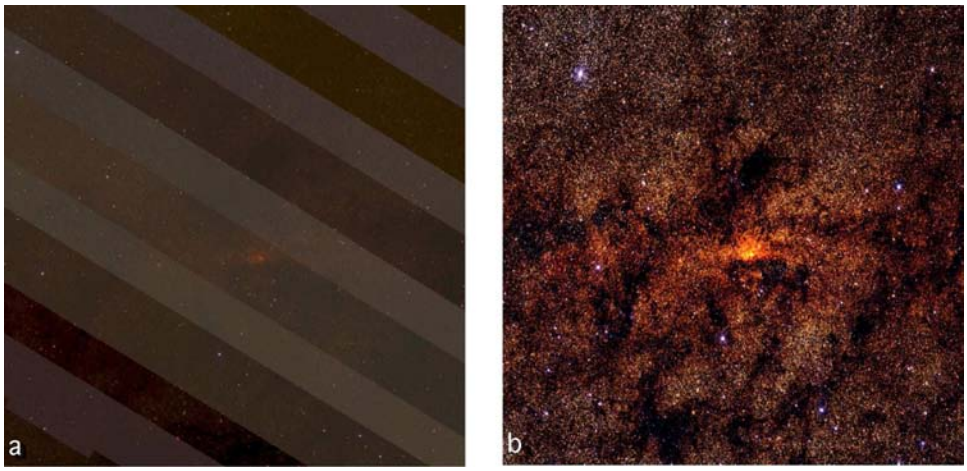

FIG. 2. *A 1 degree square mosaic of the Galactic Center, constructed by Montage from images released by the 2MASS project. In frame (a), the images are not rectified for background emission, and the variation of background radiation at different times and air masses is apparent in the stripes. Frame (b) shows the same region after applying a background rectification algorithm as described in the text. The mosaic is a demonstration product generated with a prototype code; it is not a science grade image and it is not endorsed by 2MASS.*

Two additional drawbacks inherent in this distribution of the software are that the maximum image mosaic size is limited by the available computer memory, and co-addition of flux in the reprojected pixels only supports weighting by area coverage.

The Montage team has taken advantage of the software's modular design to address these limitations in a new distribution, version 2.2, that was released in September 2004. Co-addition was redesigned to overcome the limitations of memory and weighting just described, and the use of a dedicated module allowed the redesign to proceed without impact on the design of performance improvements for reprojection. These performance improvements have taken two forms:

- development of custom, fast reprojection algorithms applicable to commonly used astronomical projections; these algorithms bypass projection of pixels onto a sphere, and transform input pixel flux directly into output pixel space [9], and
- exploitation of the parallelization inherent in the design; many of the steps needed to compute a mosaic can be performed in parallel.

2.3. Improved co-addition. The limitations of the available memory on the processing machine were overcome by simply reading the reprojected images one line at a time. Assuming that a single row of the output file does not fill the memory, the only remaining limitation on file size is that imposed by the file system. Images of up to 6 GB have been built with the new software. The algorithm has also been developed further to support quite general co-addition methods. For each output line, `mAdd` determines which input files will be contributing pixel values, and opens only those files. Each contributing pixel value is read from the flux and area coverage files, and the value of each of these pixels is stored in an array until all contributing pixels have been read for the corresponding output row. This array constitutes a "stack" of input pixel values; a corresponding stack of area coverage values is also preserved. The contents of the output row are then calculated one output pixel (i.e., one input stack) at a time, by averaging the flux values from the stack. Different algorithms to perform this average can be trivially inserted at this point in the program. The greater flexibility of the new software comes at the modest expense of 30% in speed.

Montage currently supports mean and median co-addition, with or without weighting by area. The default is the mean algorithm, which accumulates flux values contributing to each output pixel, and then scales them by the total area coverage for that pixel. The median algorithm ignores any pixels whose area coverage falls below a specific threshold, and then calculates the median flux value from the remainder of the stack. This median input pixel is scaled by its corresponding area coverage, and written as the output pixel. If there are no area files, then the algorithm gives equal weight to all pixels. This is valuable for science data sets where the images are already projected into the same pixel space (e.g., MSX). An obvious extension of the algorithm is to support outlier rejection, and this is planned for a future release.

2.4. Fast reprojection. In its general form, the Montage reprojection algorithm transforms pixel coordinates in the input image to coordinates on the sky and then transforms that location to output image pixel space. Under certain circumstances, this can be replaced by a much faster algorithm which uses a set of linear equations (though not a linear transform) to transform directly from input pixel coordinates to output pixel coordinates. This alternate approach is limited to cases where both the input and output projections are of "tangent plane" type (gnomonic, orthographic, etc.), but since these projections are by far the most common, it is appropriate to treat them as a special case.

This "plane-to-plane" approach is based on a library developed at the Spitzer Science Center [13]. When both images are tangent plane, the geometry of the system can be viewed as in Figure 3, where a pair of gnomonic projection planes intersect the coordinate sphere. A single line connects the center of the sphere, the projected point on the first plane and the projected point on the second plane. This geometric relationship results in transformation equations between the two planar coordinate systems that require no trigonometry or extended polynomial terms. As a consequence, the transform is a factor of thirty or more faster than using the normal spherical projection formulae.

An added benefit to the plane-to-plane approach is that the computation of pixel overlap is much easier, involving only clipping constraints of the projected input pixel polygon in the output pixel space.

This approach excludes some commonly-used projections such as "Cartesian" and "zenithal equidistant," and is essentially limited to small areas of few square degrees. Processing of all-sky images, as is almost always the case with projections such as

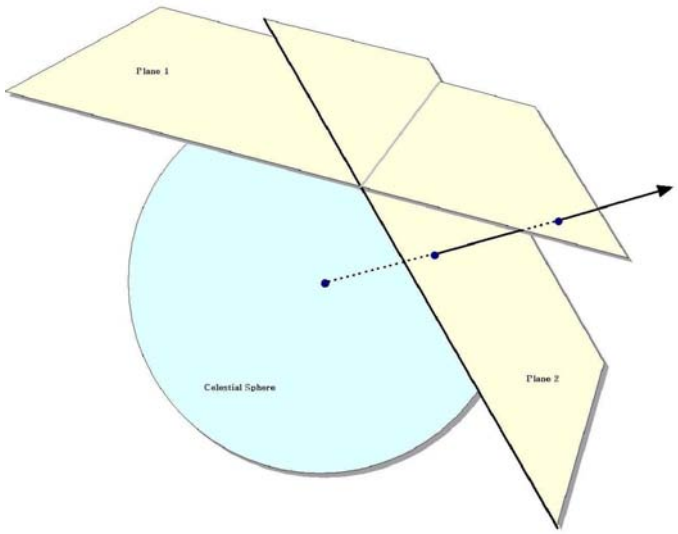

FIG. 3. *The principle of plane-to-plane reprojection.*

Aitoff, generally requires the slower plane-to-sky-to-plane approach.

There is, however, a technique that can be used for images of high resolution and relatively small extent (up to a few degrees on the sky). Rather than use the given image projection, we can often approximate it with a very high degree of accuracy with a "distorted" Gnomonic projection. In this case, the pixel locations are "distorted" by small distances relative to the plane used in the image projection formulae. A distorted space is one in which the pixel locations are slightly offset from the locations on the plane used by the projection formulae, as happens when detectors are slightly misshapen, for instance. This distortion is modeled by pixel-space polynomial correction terms which are stored as parameters in the image's FITS header.

While this approach was developed to deal with physical distortions caused by telescope and instrumental effects, it is applicable to Montage in augmenting the plane-to-plane reprojection. Over a small, well-behaved region, most projections can be approximated by a Gnomonic (TAN) projection with small distortions. For instance, in terms of how pixel coordinates map to sky coordinates, a two-degree "Cartesian" (CAR) projection is identical to a TAN projection with a fourth-order distortion term to within about a percent of a pixel width. Figure 4 shows this in exaggerated form for clarity, with the arrows showing the sense of the distortion.

3. Grid-enabled Montage.

3.1. Single-processor performance on benchmark problem. In order to examine grid-enabling Montage, we choose a sample problem that could be computed on a single processor in a reasonable time to use as a benchmark. The results in this section involve this benchmark, unless otherwise stated. The benchmark problem is generating a mosaic of 2MASS data from a 6 degree $\times$ 6 degree region around M16. Construction of this mosaic requires 1254 2MASS images as input, each having about 0.5 Megapixels, for a total of about 657 Megapixels input (or about 5 GB with 64 bits per pixel double precision floating point data). The output is a 3.7 GB FITS file

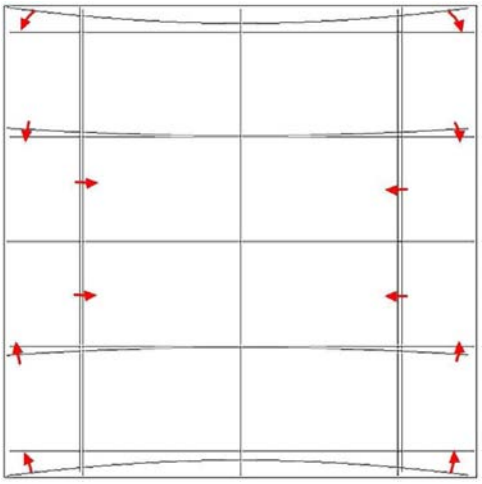

FIG. 4. *Representation of a WCS projection as a distorted Gnomonic (TAN) projection, exaggerated for clarity. The arrows indicate the sense of the distortions.*

with a 21600 pixel × 21600 pixel data segment, and 64 bits per pixel double precision floating point data. Note that the output data size is a little smaller than the input data size because there is some overlap between neighboring input images. For the timing reported in this section, we assume that the input data has been pre-staged to a local disk on the compute cluster.

Table 2 shows timings from running this problem on a single processor of the "Phase 2" TeraGrid cluster at the National Center for Supercomputing Applications (NCSA), University of Illinois at Urbana-Champaign. This cluster consists of 887 nodes, each with dual Itanium-2 processors and each having at least 4 GB of memory. 256 of the nodes have 1.3 GHz processors, and the other 631 nodes have 1.5 GHz processors. The timing tests reported in this chapter used the faster 1.5 GHz processors. The network interconnect between nodes is Myricom's Myrinet and the operating system is SuSE Linux. Disk I/O is to a 24 TB General Parallel File System (GPFS). Jobs are scheduled on the system using Portable Batch System (PBS) and the queue wait time is not included in the execution times since that is heavily dependent on machine load from other users.

Table 2 also shows the processing steps for the benchmark problem through the ordering of the rows of the table. Each step must be completed before the next step (with the slight exception that the `mImgtbl` steps can be excluded from the processing stream). However, each of the executive modules (`mProjExec`, `mDiffExec`, `mFitExec` and `mBgExec`) calls additional sub-modules (as described in Table 1), and the execution of each sub-module is independent of the others, as long as all the sub-modules complete before the next module starts. This is shown in Figure 5. These sub-modules are the initial cases where parallelism is found in Montage. This parallelization is the most basic, as the sub-processes are started by the executive, but do not communicate with either the executive or each other as they run. Additionally, because `mAdd` builds the output mosaic by rows, it can also be parallelized, though it is slightly more complicated than the executives and sub-modules, because the parallel components of `mAdd` need to work together to write a single file. In the following sections of this paper, two different methods of taking advantage of this

TABLE 2
Execution times (minutes) for a benchmark 2MASS mosaic covering 6 degrees × 6 degrees centered at M16, computed with Montage (version 2.1) on a single 1.5 GHz node of the NCSA TeraGrid cluster.

Component	Run Time (minutes)
`mImgtbl`	0.7
`mProjExec` (orig.)	2408.1
`mProjExec` (fast)	142.3
`mImgtbl`	1.1
`mOverlaps`	0.05
`mDiffExec`	30.0
`mFitExec`	20.2
`mBgModel`	1.1
`mBgExec`	11.3
`mImgtbl`	1.05
`mAdd`	73.0
Total (w/ orig. `mProject`)	2546.6
Total (w/ fast `mProject`)	280.8

parallelization on a grid are discussed.

3.2. Data access. A large variety and amount of astronomical data are now available in archives around the world. In order to standardize and encourage use of these data, the International Virtual Observatory Alliance (IVOA) [14] has been formed by a number of smaller organizations with a set of common interests. The IVOA has proposed a number of draft standards, ranging from a table format called VOTable [15] to a method for retrieving data called the Simple Image Access Protocol (SIAP) [16]. These standards become important in the context of distributed computing, where the data usually needs to be brought to the processing in an automated manner.

For example, if one wants to obtain data needed to build a mosaic of SDSS [7] data for band z, in a circular region around (RA, DEC) (39.9995925, 0.0000885), with radius 0.99554875, one can use the following URL:

```
http://skyserver.sdss.org/vo/dr2siap/SIAP.asmx/getSiapInfo?
bandpass=z&POS=39.9995925,8.85e-05&SIZE=0.99554875&FORMAT=image/fits
```

This returns a VOTable containing a list of files (URLs) that can be accessed. One can then parse this table, and actually retrieve the files. Where this is done depends on the context of the mosaicking job to be run. Using the single processor version of Montage, one would likely do this on the machine on which Montage was to be run. Using the grid-enabled versions of Montage that are described in the next sections, this data retrieval would be done in some manner to make the files local to the processing. This will be discussed in the next two sections: §3.3 and §3.4.

Alternatively, a scientist may wish to work with data that is on her desktop. In this case, if using a grid-enabled version of Montage, the user needs a mechanism for getting the data to the machine on which the processing (or at least the first parts of it) will be done. This will also be discussed in §3.3 and §3.4.

3.3. Grid-enabling Montage through use of MPI parallelization. The first method of running Montage on a grid is to use grid-accessible clusters, such as the TeraGrid. This is really very similar to traditional, non-grid parallelization. By use

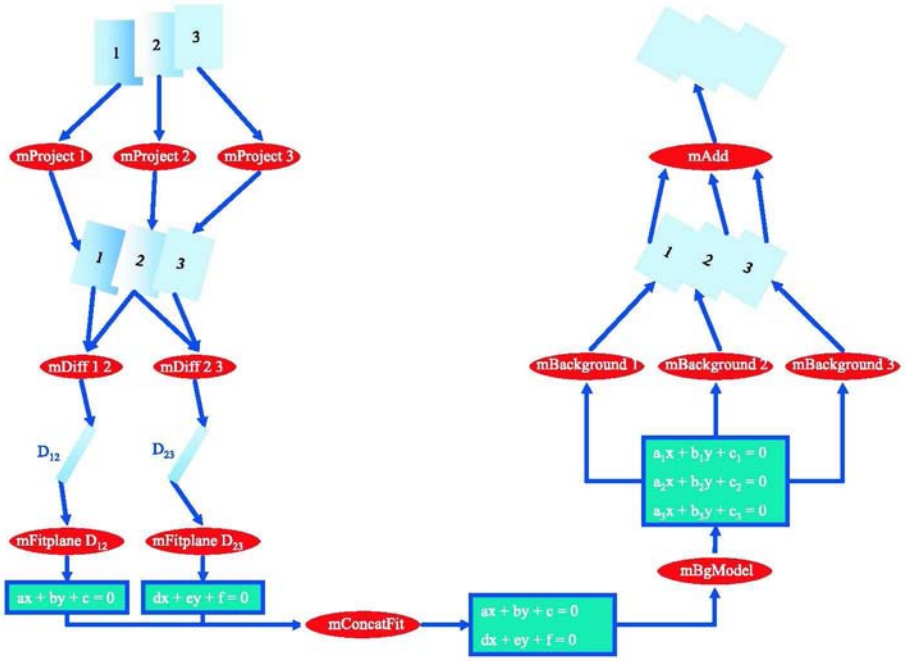

FIG. 5. *Example workflow for Montage, for a case where three images are to be made into a mosaic. The reprojection, calculation of the inter-image differences and application of the background rectification can all be performed in parallel.*

of MPI (the Message Passing Interface [17]), the executives (`mProjExec`, `mDiffExec`, `mFitExec` and `mBgExec`) and `mAdd` are run on multiple processors. The Atlasmaker [18] project previously developed an MPI version of `mProject`, but was not closely coupled to the released Montage code, and therefore was hard to maintain. In this case, the MPI versions and single-processor versions of modules are generated from a single set of source code, by use of preprocessing directives.

3.3.1. Methodology. The structure of the executives are similar to each other, in that each has some initialization that involves determining a list of files on which a sub-module will be run, a loop in which the sub-module is actually called for each file, and some finalization work that includes reporting on the results of the sub-module runs. Additionally, two of the executives write to a log file after running each sub-module. The executives, therefore, are parallelized very simply, with all processes of a given executive being identical to all the other processes of that executive. All the initialization is duplicated by all processes. A line is added at the start of the main loop, so that each process only calls the sub-module if the remainder of the loop count divided by the number of processes equals the MPI rank. All processes then participate in global sums to find the total statistics of how many sub-modules succeeded, failed, etc., as each process keeps track of its own statistics. After the global sums, only the process with rank 0 prints the global statistics. For the two executives that write to a log file, the order of the entries in the log file is unimportant.

In these cases, each process writes to a unique temporary log file. Then, after the global sums, the process with rank 0 runs an additional loop, where it reads lines from each process's log file, then writes them to a global log file, deleting each process's temporary log file when done.

As was discussed in §2.3, `mAdd` writes to the output mosaic one line at a time, reading from its input files as needed. The sequential `mAdd` writes the FITS header information into the output file before starting the loop on output lines. In the parallel `mAdd`, only the process with rank 0 writes the FITS header information, then it closes the file. Now, each process can carefully seek and write to the correct part of the output file, without danger of overwriting another process's work. While the executives were written to divide the main loop operations in a round-robin fashion, it makes more sense to parallelize the main `mAdd` loop by blocks, since it is likely that a given row of the output file will depend on the same input files as the previous row, and this can reduce the amount of input I/O for a given process.

A set of system tests are available from the Montage web site. These tests, which consist of shell scripts that call the various Montage modules in series, were designed for the single-processor version of Montage. The MPI version of Montage is run similarly, by changing the appropriate lines of the shell script, for example, from:

```
mAdd arg1 arg2 ...
```

to:

```
mpirun -np N mAddMPI arg1 arg2 ...
```

These are the only changes that are needed. Notice that when this is run on a queuing system, some number of nodes will be reserved for the job. Some parts of the job, such as `mImgtbl`, will only use one processor, and other parts, such as `mAddMPI`, will use all the processors. Overall, most of the processors are in use most of the time. There is a small bit of overhead here in launching multiple MPI jobs on the same set of nodes. One might change the shell script into a parallel program, perhaps written in C or in python, to avoid this overhead, but this has not yet been done for Montage.

A reasonable question to ask is how this approach differs from what might be done on a cluster that is not part of a grid. The short answer to this somewhat rhetorical question is that the processing part of this approach is not different. In fact, one might choose to run the MPI version of Montage on a local cluster by logging in to the local cluster, transferring the input data to that machine, submitting a job that runs the shell script to the queuing mechanism, and finally, after the job has run, retrieving the output mosaic.

In a grid environment, the Globus Toolkit (GTK) [19] could be used for both secure data communication and remote job execution. GTK uses the Grid Security Infrastructure (GSI) for secure communication between grid elements and to allow "single sign-on" access to a collection of grid resources. With GSI, a user of a grid may have an X.509 certificate, issued by a trusted certificate authority, that grants access to the various resources. Authentication can then be done with Globus by requesting a proxy certificate that permits access to all the resources for a finite period.

To run the mosaicking code on a cluster that is part of the grid, the staging of the input data to the remote computer performing the computations could either be done

TABLE 3

Execution times (wallclock time in minutes) for a benchmark 2MASS mosaic, covering 6 degrees × 6 degrees centered at M16, computed with Montage (version 2.1) on various numbers of 1.5 GHz nodes of the NCSA TeraGrid cluster. Parallelization is done with MPI.

	Number of Nodes (1 processor per node)							
Component	1	2	4	8	16	32	64	128
`mImgtbl`	0.7	1.05	1.05	1.2	1.4	1.3	1.2	0.7
`mProjExec` (orig., MPI)	2408.1	1228.1	620.0	302.4	153.6	75.8	39.6	21.9
`mProjExec` (fast, MPI)	142.3	68.0	34.1	17.2	8.8	4.8	3.05	2.3
`mImgtbl`	1.1	1.0	0.9	1.3	1.1	1.2	1.1	1.1
`mOverlaps`	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
`mDiffExec (MPI)`	30.0	16.9	9.5	9.5	9.2	9.4	9.5	8.8
`mFitExec (MPI)`	20.2	10.6	5.3	2.9	1.7	1.1	1.1	1.2
`mBgModel`	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
`mBgExec (MPI)`	11.3	6.0	3.8	2.6	2.3	2.4	2.6	3.0
`mImgtbl`	1.05	0.9	0.9	1.1	1.1	1.2	1.1	1.4
`mAdd (MPI)`	73.0	44.6	36.4	28.7	17.2	14.8	15.8	12.8
Total (w/ orig. `mProject`)	2546.6	1310.3	679.0	350.9	188.8	108.3	73.4	52.0
Total (w/ fast `mProject`)	280.8	150.2	93.1	65.7	44.0	37.3	36.9	32.4

by running the appropriate SIAP service and fetching the data files from the returned list of URLs on the remote cluster or by using the GridFTP [20] implementation included in GTK, if the data to be mosaicked was on a local desktop. GridFTP is a data transfer protocol that is optimized for high-performance, secure communication over high-bandwidth networks. The mosaic computations can be launched using the remote job submission provided by `globusrun`. Finally, transfer of the output mosaic back to a local disk could be done using GridFTP.

The grid paradigm just described requires an active proxy certificate during the data communication and remote job execution. This authentication is straightforward when a user is typing at a command-line prompt. However, if a user is interacting with the grid through a web portal, the web server then has to authenticate the user on the grid. In this case, a repository of grid credentials called MyProxy [21] can be used to store a user's grid certifications for easy access by the web portal. The portal then acts on the user's behalf to submit commands for use of grid resources. The portal can notify the user of job completion via email notification, or provide a more sophisticated status reporting mechanism if more feedback is required.

3.3.2. Timing results. The timing results of the MPI version of Montage are compiled in Table 3, which shows wall clock times in minutes for each Montage module run on the specified number of nodes (with one processor per node) on the NCSA TeraGrid cluster. The end-to-end runs of Montage involved running the modules in the order shown in the table. The modules that are parallelized are labeled as MPI; all other modules are serial implementations. Note that timings are shown for two versions of the `mProject` module: one that calls the slower original implementation and one that calls the fast implementation. Total times are shown for both implementations, with only the one specified implementation (original or fast) called in each run. For clarity, the execution times for the parallel modules on the different number of cluster nodes are plotted in Figure 6 for the original `mProject` implementation and in Figure 7 for the fast `mProject` implementation. Figure 8 shows a plot of the speedups that were achieved for each size of cluster partition.

MPI parallelization reducing the one processor time of 2546.6 minutes down to 52.0 minutes on 128 nodes, for a parallelization speedup of 49.0. Note that with the

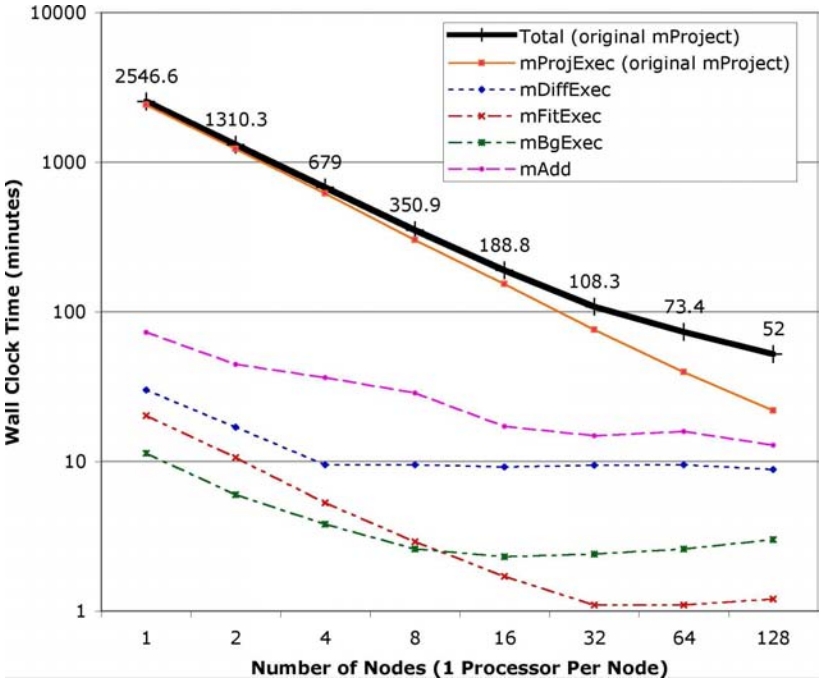

FIG. 6. *Plot of wall clock time versus number of nodes on the NCSA TeraGrid cluster for the parallel components of Montage (version 2.1), using the baseline mProject algorithm, building the benchmark 2MASS mosaic covering 6 degrees × 6 degrees centered at M16. The total time for the end-to-end run is shown as the thick black line.*

exception of some small initialization and finalization code, all of the parallel code is non-sequential. The main reason the parallel modules fail to scale linearly as the number of processors is increased is I/O. On a computer system with better parallel I/O performance, one would expect to obtain better speedups; i.e., the situation where the amount of work is too small for the number of processors has not been reached, nor has the Amdahl's law limit (where speedup is limited by the serial fraction of the program) been reached. With the algorithmic improvements of the fast `mProject`, the 128-node time has been further reduced down to 32.4 minutes, for an overall speedup (including parallelization and algorithmic improvements) of 78.6.

Note that there is certainly some variability inherent in these timings. For example, the time to run `mImgtbl` should be the same in all cases, since it is always run on a single processor. However, the measured results vary from 0.7 to 1.4 minutes. Additionally, since some of the modules' timings are increasing as the number of processors is increased, one would actually choose the fastest timing and run the module on only the number of processors that were used for that timing. For example, `mBgExec` on this machine should only be run on 16 processors, no matter how many are used for the other modules.

These timings are probably close to the best that can be achieved on a single cluster, and can be thought of as a lower bound on any parallel implementation, including any grid implementation. However, there are numerous limitations to this implementation, including that a single pool of nodes with shared access to a common file system is required, and that any single failure of a module or sub-module will cause the entire job to fail, at least from that point forward. The next section discusses a

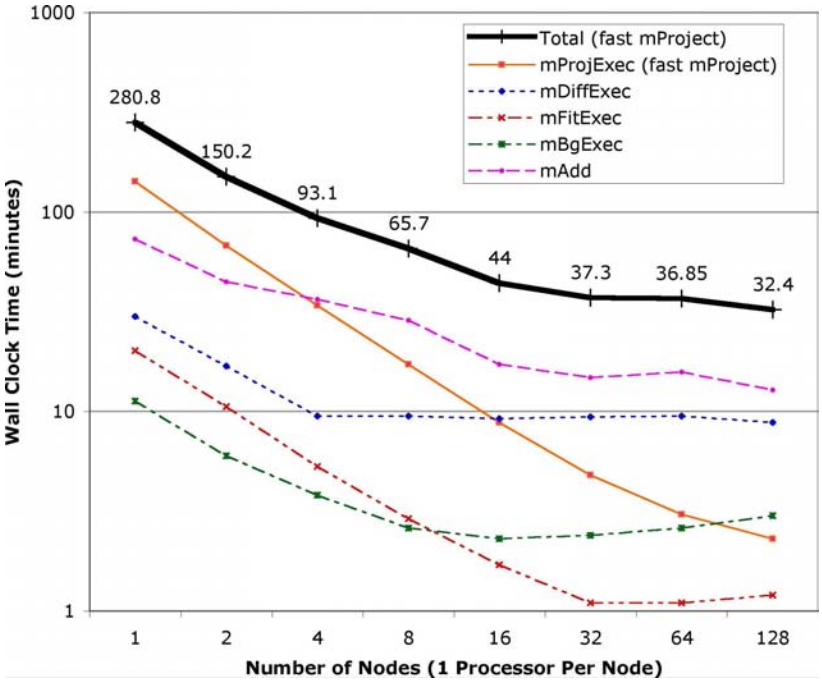

FIG. 7. *Plot of wall clock time versus number of nodes on the NCSA TeraGrid cluster for the parallel components of Montage (version 2.1), with the fast mProject algorithm, building the benchmark 2MASS mosaic covering 6 degrees × 6 degrees centered at M16. The total time for the end-to-end run is shown as the thick black line.*

more general approach that can overcome these limitations.

3.4. Grid-enabling Montage with Pegasus. Pegasus [22, 23, 24, 25], which stands for Planning for Execution in Grids, is a framework that enables the mapping of complex workflows onto distributed resources such as the grid. In particular, Pegasus maps an abstract workflow to a form that can be executed on the grid, on a variety of computational platforms from single hosts, to condor pools, to compute clusters, to the TeraGrid. The Pegasus framework allows users to customize the type of resource and data selections performed as well as to select the information sources. Pegasus was developed as part of the GriPhyN Virtual Data System [26].

Abstract workflows describe the analysis in terms of logical transformations and data without identifying the resources needed to execute the workflow. The abstract workflow for Montage consists of the various application components shown in Figure 5. The nodes of the abstract workflow represent the logical transformations such as `mProject`, `mDiff` and others. The edges of the workflow represent the data dependencies between the transformations. For example, `mAdd` requires all the files generated by all the previous `mBackground` steps.

3.4.1. Mapping application workflows. Mapping the abstract workflow description to an executable form involves finding the resources that are available and can perform the computations, the data that is used in the workflow, and the necessary software. Pegasus consults various grid information services to find the above information (Figure 9). In the following description, we illustrate the Pegasus framework configured to use a specific set of information sources. Pegasus uses the logical

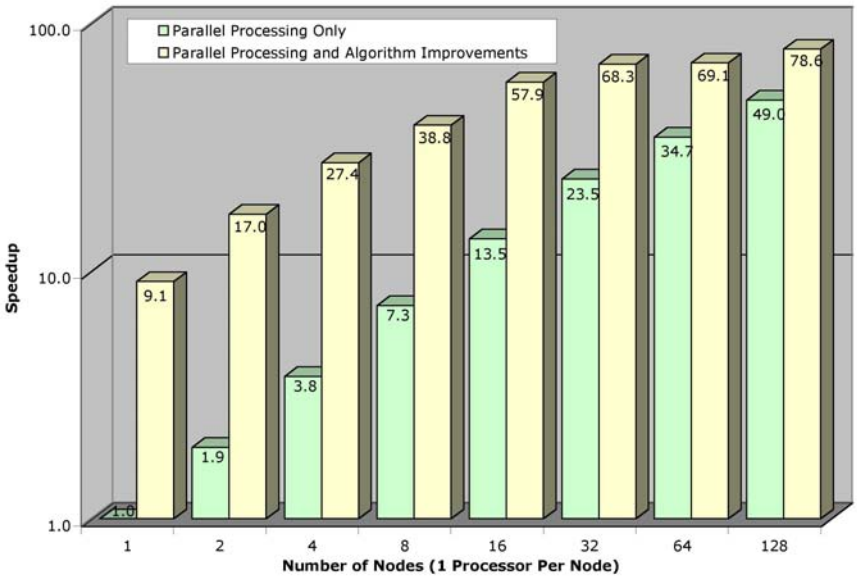

FIG. 8. *Speedup of Montage (version 2.1) building the benchmark 2MASS mosaic covering 6 degrees × 6 degrees centered at M16, on the specified number of nodes of the NCSA TeraGrid cluster. For each number of nodes, two speedup numbers are shown, one from simply parallelizing parts of the Montage code, and one that includes the parallelization as well as algorithmic improvements. The primary algorithmic improvement involved replacing the mProject code with a new, fast plane-to-plane implementation.*

filenames referenced in the workflow to query the Globus Replica Location Service (RLS) [27] to locate the replicas of the required data. We assume that data may be replicated across the grid and that the users publish their data products into RLS. Given a set of logical filenames, RLS returns a corresponding set of physical file locations. After Pegasus produces new data products, it registers them into the RLS as well (unless the user does not want that to happen). Intermediate data products can be registered as well.

In order to be able to find the location of the logical transformations defined in the abstract workflow, Pegasus queries the Transformation Catalog (TC) [28] using the logical transformation names. The catalog returns the physical locations of the transformations (on possibly several systems) and the environment variables necessary for the proper execution of the software. Pegasus queries the Globus Monitoring and Discovery Service (MDS) [29] to find the available resources and their characteristics such as the load, the scheduler queue length, and available disk space. The information from the TC is combined with the MDS information to make scheduling decisions. When making resource assignments, Pegasus prefers to schedule the computation where the data already exist; otherwise, it makes a random choice or uses a simple scheduling technique. Additionally, Pegasus uses MDS to find information about the location of the GridFTP servers [20] that can perform data movement, job managers [30] that can schedule jobs on the remote sites, storage locations, where data can be pre-staged, shared execution directories, the RLS into which new data can be

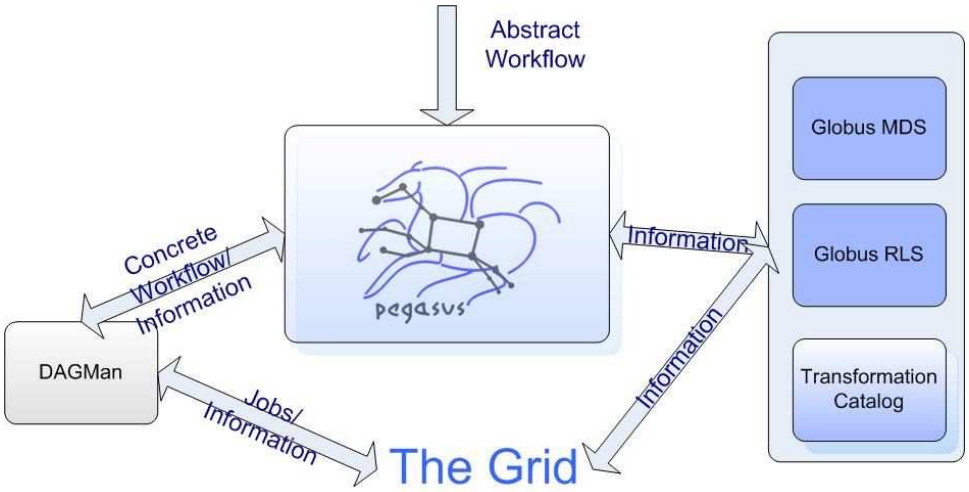

FIG. 9. *Mapping an abstract workflow onto the grid resources. (A particular set of information sources is used.)*

registered, site-wide environment variables, etc. This information is necessary to produce the submit files that describe the necessary data movement, computation and catalog updates.

The information about the available data can be used to optimize the concrete workflow. If data products described within the abstract workflow already exist, Pegasus can reuse them and thus reduce the complexity of the concrete workflow. In general, the reduction component of Pegasus assumes that it is more costly to execute a component (a job) than to access the results of the component if that data is available. For example, some other user may have already materialized (made available on some storage system) part of the entire required dataset. If this information is published into the RLS, Pegasus can utilize this knowledge and obtain the data, thus avoiding possibly costly computation. As a result, some components that appear in the abstract workflow do not appear in the concrete workflow.

Pegasus also checks for the feasibility of the abstract workflow. It determines the root nodes for the abstract workflow and queries the RLS for the existence of the input files for these components. The workflow can only be executed if the input files for these components can be found to exist somewhere on the grid and are accessible via a data transport protocol.

The final result produced by Pegasus is an executable workflow that identifies the resources where the computation will take place. In addition to the computational nodes, the concrete, executable workflow also has nodes for data transfer that stage data in and out of the computations, data registration nodes that can update various catalogs on the grid (for example, RLS), and also nodes that can stage-in statically linked binaries. Figure 10 shows a snapshot of a small Montage workflow that consists of 1200 executable jobs.

3.4.2. Workflow execution. The concrete workflow produced by Pegasus is in the form of submit files that are given to DAGMan and Condor-G [31] for execution. The submit files indicate the operations to be performed on given remote systems and the order in which the operations need to be performed. Given the submit

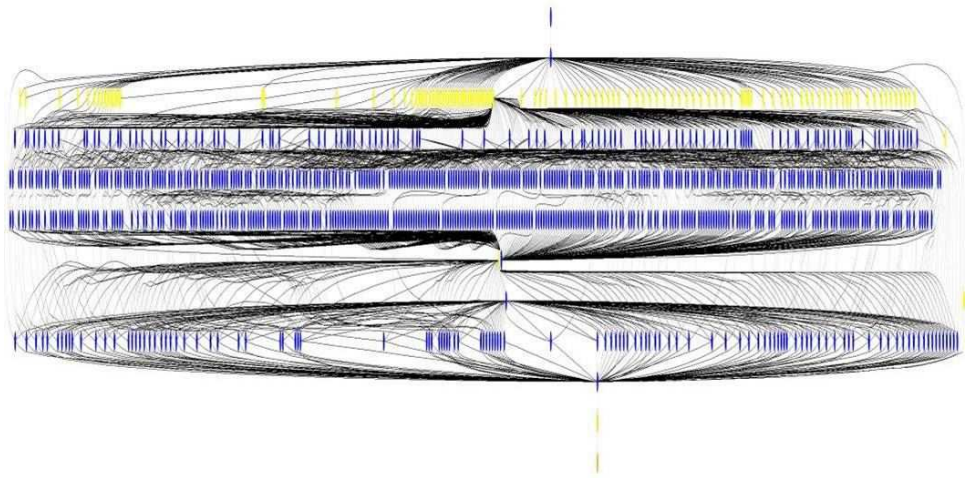

FIG. 10. *Concrete Montage workflow produced by Pegasus. The light colored-nodes represent data stage-in and the dark colored nodes, computation.*

files, DAGMan submits jobs to Condor-G for execution. DAGMan is responsible for enforcing the dependencies between the jobs defined in the concrete workflow.

In case of job failure, DAGMan can retry a job a given number of times. If that fails, DAGMan generates a rescue workflow that can be potentially modified and resubmitted at a later time. Job retry is useful for applications that are sensitive to environment or infrastructure instability. The rescue workflow is useful in cases where the failure was due to lack of disk space that can be reclaimed or in cases where totally new resources need to be assigned for execution. Obviously, it is not always beneficial to map and execute an entire workflow at once, because resource availability may change over time. Therefore, Pegasus also has the capability to map and then execute (using DAGMan) one or more portions of a workflow [25].

3.4.3. Montage portal. In this section, we illustrate how we can combine application-specific services and grid-based services to provide users with a Montage portal.

The Montage TeraGrid portal has a distributed architecture, as illustrated in Figure 11. The portal is comprised of the following five main components, each having a client and server: (i) User Portal, (ii) Abstract Workflow Service, (iii) 2MASS Image List Service, (iv) Grid Scheduling and Execution Service, and (v) User Notification Service. These components are described in more detail below.

User Interface. Users on the internet submit mosaic requests by filling in a simple web form with parameters that describe the mosaic to be constructed, including an object name or location, mosaic size, coordinate system, projection, and spatial sampling. After request submission, the remainder of the data access and mosaic processing is fully automated with no user intervention. The server side of the user portal includes a CGI program that receives the user input via the web server, checks that all values are valid, and stores the validated requests to disk for later processing. A separate daemon program with no direct connection to the web server runs continuously to process incoming mosaic requests. The processing for a request is done in two main steps:

1. Call the abstract workflow service client code.

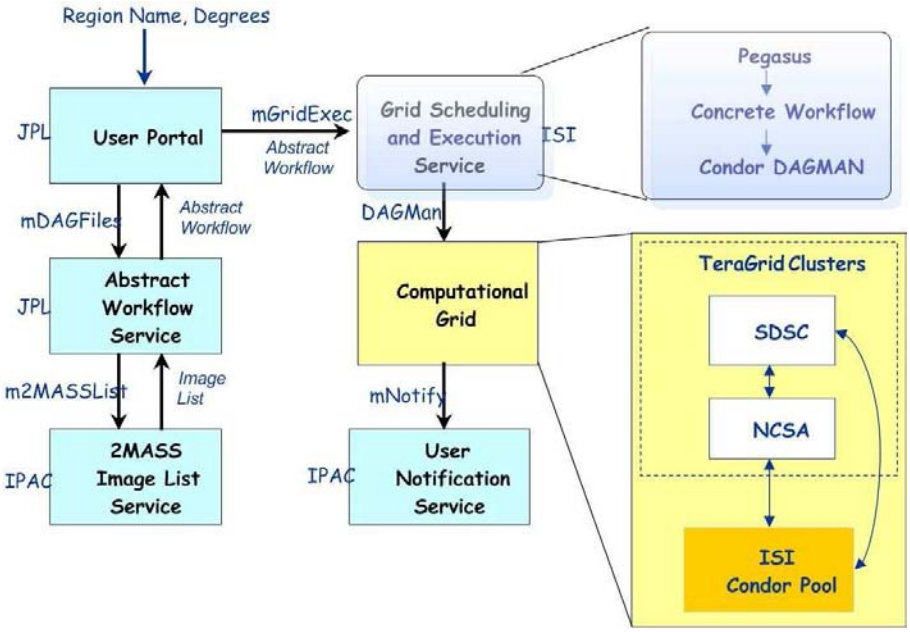

FIG. 11. *The distributed architecture of the Montage TeraGrid portal.*

2. Call the grid scheduling and execution service client code and pass to it the output from the abstract workflow service client code.

Abstract Workflow Service. The abstract workflow service takes as input a celestial object name or location on the sky and a mosaic size and returns a zip archive file containing the abstract workflow as a directed acyclic graph (DAG) in XML and a number of input files needed at various stages of the Montage mosaic processing. The abstract workflow specifies the jobs and files to be encountered during the mosaic processing, and the dependencies between the jobs. These dependencies are used to determine which jobs can be run in parallel on multiprocessor systems.

2MASS Image List Service. The 2MASS Image List Service takes as input a celestial object name or location on the sky (which must be specified as a single argument string), and a mosaic size. The 2MASS images that intersect the specified location on the sky are returned in a table, with columns that include the filenames and other attributes associated with the images.

Grid Scheduling and Execution Service. The Grid Scheduling and Execution Service takes as input the zip archive generated by the Abstract Workflow Service, which contains the abstract workflow, and all of the input files needed to construct the mosaic. The service authenticates users, schedules the job on the grid using Pegasus, and then executes the job using Condor's DAGMan.

Users are authenticated on the TeraGrid using their grid security credentials. The user first needs to save their proxy credential in the MyProxy server [21]. MyProxy is a credential repository for the grid that allows a trusted server (like our Grid Scheduling and Execution Service) to access grid credentials on the users behalf. This allows the appropriate credentials to be retrieved by the portal using the user's username and password.

Once authentication is completed, Pegasus schedules the Montage workflow onto the TeraGrid or other clusters managed by PBS and Condor. Upon completion, the final mosaic is delivered to a user-specified location and the User Notification Service, described below, is contacted.

User Notification Service. The last step in the grid processing is to notify the user with the URL where the mosaic may be downloaded. This notification is done by a remote user notification service so that a new notification mechanism can be used later without having to modify the Grid Scheduling and Execution Service. Currently the user notification is done with a simple email, but a later version will use the Request Object Management Environment (ROME), being developed separately for the National Virtual Observatory. ROME will extend our portal with more sophisticated job monitoring, query, and notification capabilities.

Our design exploits to the maximum the parallelization inherent in the Montage architecture. The Montage grid portal is flexible enough to run a mosaic job on a number of different cluster and grid computing environments, including Condor pools and TeraGrid clusters. We have demonstrated processing on both a single cluster configuration and on multiple clusters at different sites having no shared disk storage.

Figure 12 shows the Sword of Orion (M42, Trapezium, Great Nebula). This mosaic was obtained by running a Montage workflow through Pegasus and executing the concrete workflow the TeraGrid resources.

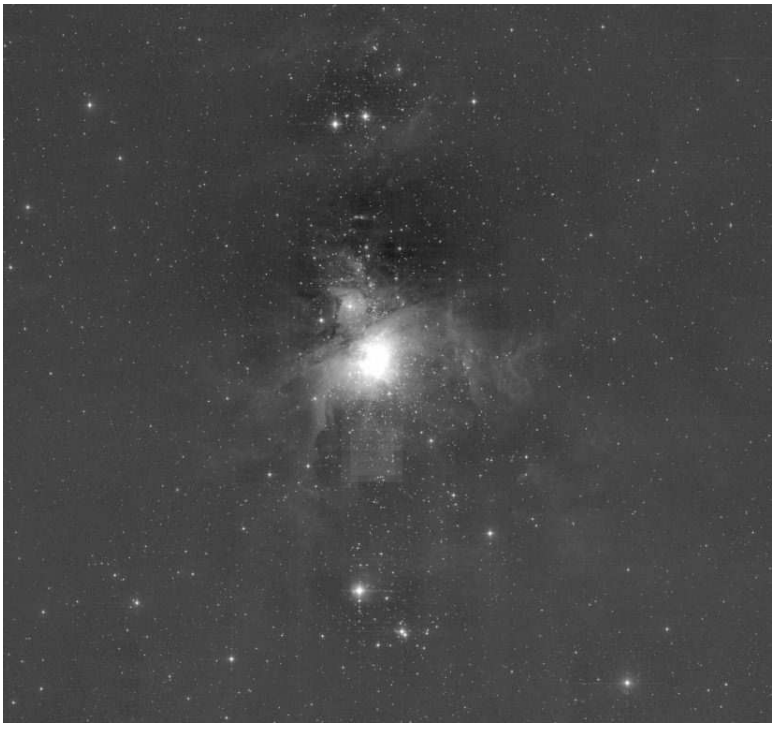

FIG. 12. *Mosaic of M42 obtained using Pegasus and the TeraGrid resources.*

3.4.4. Timing results. When using remote grid resources for the execution of the concrete workflow, there is a non-negligible overhead involved in acquiring resources and scheduling the computation over them. In order to reduce this overhead,

Pegasus can aggregate the nodes in the concrete workflow into clusters so that the remote resources can be utilized more efficiently. The benefit of clustering is that the scheduling overhead (from Condor-G, DAGMan and remote schedulers) is incurred only once for each cluster. In the following results we cluster the nodes in the workflow within a workflow level (or workflow depth). In case of Montage, the `mProject` jobs are within a single level, `mDiff` jobs in another level, and so on. Clustering can be done dynamically based on the estimated run time of the jobs in the workflow and the processor availability.

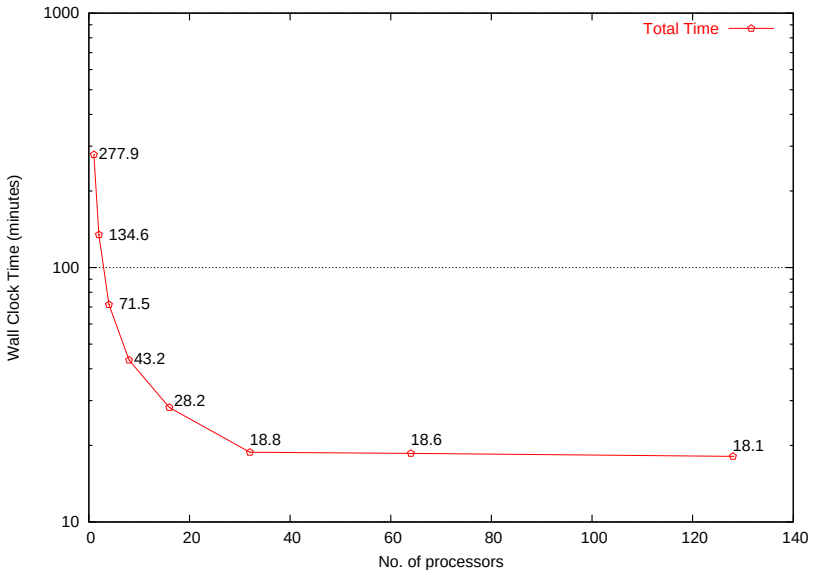

FIG. 13. *Total time (in minutes) for executing the concrete workflow for creating a mosaic covering 6 degrees × 6 degrees region centered at M16. The numbers on the curve indicate the runtime in minutes.*

Figure 13 shows the end-to-end time taken to execute the concrete workflow for creating a 2MASS mosaic covering 6 degree × 6 degree region centered at M16 using resources from the NCSA TeraGrid Cluster. Condor Glidein [32] was used to acquire the resources. Once the resources are acquired, they are available for executing the workflow and there is no queuing delay at the remote resource. The workflow was executed using DAGMan running on a host at ISI. The time taken to transfer the input data and the output mosaic is not included in this figure. These measurements were made using Montage version 3.0β5. The figure shows the time taken in minutes for DAGMan to execute the workflow as the number of processors are increased. The numbers of processors used were 1, 2, 4, 8, 16, 32, 64 and 128. The nodes in the workflow were clustered so that the number of clusters at each level of the workflow was equal to the number of processors. As the number of processors is increased and thus the number of clusters increases, the Condor overhead becomes the dominating factor. (DAGMan takes approximately 1 second to submit each cluster into the condor queue. Condor's scheduling overheads add additional delay.) As a result we do not see a corresponding decrease in the workflow execution time as we increase the number of processors. There are also sequential sections in the workflow that limit the overall parallel efficiency.

Figure 14 shows the time taken to execute the concrete workflow end-to-end (not including the initial data stage in and the final data stage out) as the size of the desired mosaic increases. The size of the mosaic increases from 1 degree × 1 degree to 10 degree × 10 degree centered at M16. The numbers near the data points in the graph show the number of nodes (transformations) in the abstract workflow. DAGMan was used to execute the workflow using 64 processors at the NCSA TeraGrid Cluster. Consequently, there were 64 clusters at each level of the workflow. The figure shows the scalability of Pegasus and the Montage application as the size of the abstract workflow increases. There is an increase of approximately 5 minutes in the workflow execution time per degree increase in the desired mosaic size.

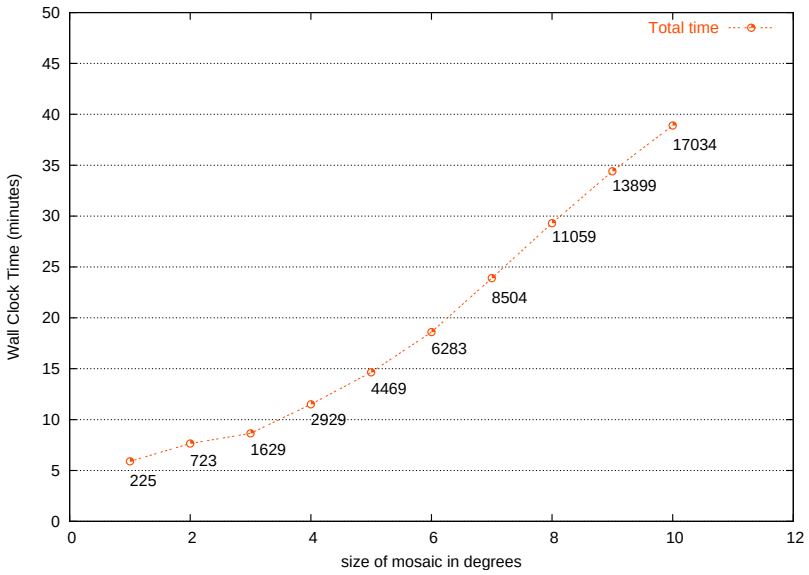

FIG. 14. *Total time taken (in minutes) for executing the concrete workflow as the size of the desired mosaic increases from 1 degree × 1 degree to 10 degree × 10 degree centered at M16. The numbers on the curve indicate the number of nodes in the abstract workflow.*

Note that these timings are for computing with Montage version 3.0β5, which is not the same as that done with the MPI version of Montage version 2.1. Specific changes currently in version 3.0β5 include running `mOverlap` and `mImgtlb` on the portal rather than on the compute nodes, combining `mDiff` and `mFit` into a new module called `mDiffFit`, and not generating a single output mosiac but rather multiple small mosaics. These changes make it difficult to make a direct comparison of the run times of the Pegasus and MPI versions on Montage, but it certainly appears that the Pegasus version will come close to the lower bound established by the MPI version.

4. Conclusions. Montage was written as a very general set of modules to permit a user to generate astronomical image mosaics. Mosaics are large images that are built from multiple small images. Montage includes modules that are used to reproject images onto a common space, calculate overlaps between images, calculate coefficients to permit backgrounds of overlap regions to be matched, modify images based on those coefficients, and co-add images using a variety of methods of handling multiple pixels in the same output space. The Montage modules can be run on a single processor computer using a simple shell script. Because this approach can take a long time

for a large mosaic, alternatives to make use of the grid have been developed. The first alternative, using MPI versions of the computational intensive modules, has very good performance but is somewhat limited. A second alternative, using Pegasus and other grid tools, is more general and allows for the execution on a variety of platforms without having to change the underlying code base. Various scheduling techniques can be used to optimized the concrete workflow for a particular execution platform. Other benefits of Pegasus include the ability to opportunistically take advantage of available resources (through dynamic workflow mapping) and the ability to take advantage of pre-existing intermediate data products, thus potentially improving the performance of the application.

5. Acknowledgments. The Pegasus software has been developed at the University of Southern California by Gaurang Mehta and Karan Vahi. Use of TeraGrid resources for the work in this chapter was supported by the National Science Foundation under the following NSF programs: Partnerships for Advanced Computational Infrastructure, Distributed Terascale Facility (DTF), and Terascale Extensions: Enhancements to the Extensible Terascale Facility.

REFERENCES

[1] The Montage project web page, http://montage.ipac.caltech.edu/
[2] The Flexible Image Transport System (FITS), http://fits.gsfc.nasa.gov, http://www.cv.nrao.edu/fits.
[3] E. W. Greisen and M. Calabretta, *Representation of Celestial Coordinates In FITS*, http://www.atnf.csiro.au/people/mcalabre/WCS.htm
[4] The Distributed Terascale facility, http://www.teragrid.org/
[5] The 2MASS Project, http://www.ipac.caltech.edu/2mass
[6] The Digitized Palomar Observatory Sky Survey (DPOSS), http://www.astro.caltech.edu/~george/dposs
[7] The Sloan Digital Sky Survey, http://www.sdss.org/
[8] G. B. Berriman, D. Curkendall, J. C. Good, J. C. Jacob, D. S. Katz, M. Kong, S. Monkewitz, R. Moore, T. A. Prince, and R. E. Williams, *An architecture for access to a compute intensive image mosaic service in the NVO* in Virtual Observatories, A S. Szalay, ed., Proceedings of SPIE, v. 4846, 2002, pp. 91-102.
[9] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, and M.-H. Su, *Montage: A grid enabled engine for delivering custom science-grade mosaics on demand*, in Optimizing Scientific Return for Astronomy through Information Technologies, P. J. Quinn, A. Bridger, eds., Proceedings of SPIE, v. 5493, 2004, pp. 221-232.
[10] Montage Version 1.7.x documentation and download http://montage.ipac.caltech.edu/docs/
[11] E. Bertin and S. Arnouts, *SExtractor: software for source extraction*, Astronomy and Astrophysics Supplement, v. 117, 1996, pp. 393-404.
[12] Montage Version 1.7.x. Photometric and Calibration Accuracy, http://montage.ipac.caltech.edu/docs/accuracy.html
[13] Mopex, the Spitzer Science Center Mosaic Engine, http://ssc.spitzer.caltech.edu/postbcd/doc/mosaicer.pdf
[14] International Virtual Observatory Alliance, http://www.ivoa.net/
[15] F. Ochsenbein, R. Williams, C. Davenhall, D. Durand, P. Fernique, D. Giaretta, R. Hanisch, T. McGlynn, A. Szalay, M. B. Taylor, and A. Wicenec, *VOTable format definition version 1.1*, http://www.ivoa.net/Documents/latest/VOT.html
[16] D. Tody, R. Plante, *Simple image access specification version 1.0*, http://www.ivoa.net/Documents/latest/SIA.html
[17] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra, *MPI: The Complete Reference*, The MIT Press, Cambridge, MA, 1996.
[18] R. D. Williams, S. G. Djorgovski, M. T. Feldmann, and J. C. Jacob, *Atlasmaker: A grid-based implementation of the hyperatlas*, Astronomical Data Analysis Software & Systems (ADASS) XIII, 2003.
[19] The Globus Toolkit, http://www.globus.org/toolkit/

[20] B. Allcock, S. Tuecke, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, and D. Quesnel, *Data management and transfer in high performance computational grid environments*, Parallel Computing, v. 28(5), 2002, pp. 749-771.

[21] J. Novotny, S. Tuecke, and V. Welch, *An online credential repository for the grid: MyProxy*, in Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing, 2001.

[22] E. Deelman, S. Koranda, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, and R. Williams, *GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists*, in Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing, 2002.

[23] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, and K. Vahi, *Mapping abstract complex workflows onto grid environments*, Journal of Grid Computing, v. 1(1), 2003.

[24] E. Deelman, R. Plante, C. Kesselman, G. Singh, M.-H. Su, G. Greene, R. Hanisch, N. Gaffney, A. Volpicelli, J. Annis, V. Sekhri, T. Budavari, M. Nieto-Santisteban, W. OMullane, D. Bohlender, T. McGlynn, A. Rots, and O. Pevunova, *Grid-based galaxy morphology analysis for the national virtual observatory*, in Proceedings of SC2003, 2003.

[25] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, *Pegasus: mapping scientific workflows onto the grid*, in Across Grids Conference 2004.

[26] GriPhyN, http://www.griphyn.org/

[27] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney, *Giggle: a framework for constructing scalable replica location services*, in Proceedings of SC2002, 2002.

[28] E. Deelman, C. Kesselman, and G. Mehta, *Transformation catalog design for GriPhyN*, GriPhyN Technical Report 2001-17, 2001.

[29] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, *Grid information services for distributed resource sharing*, in Proceedings of the 10th IEEE Symposium on High-Performance Distributed Computing, 2001.

[30] K. Czajkowski, A. K. Demir, C. Kesselman, and M. Thiebaux, *Practical resource management for grid-based visual exploration*, in Proceedings of the 10th IEEE Symposium on High-Performance Distributed Computing, 2001.

[31] J. Frey, T. Tannenbaum, M. Livny, and S. Tuecke, *Condor-G: a computation managament agent for multi-institutional grids*, in Proceedings of the 10th IEEE Symposium on High-Performance Distributed Computing, 2001.

[32] Condor-Glidein, http://www.cs.wisc.edu/condor/glidein