April 3,2003

Montage on the Grid

Gurmeet Singh, Ewa Deelman
gurmeet@isi.edu, deelman@isi.edu

1. Introduction

Montage is an astronomical image mosaic service for the National Virtual Observatory[1][2]. One of the goals of Montage is to run the computations on the Grid. In the remaining part of this document we describe the procedures used to run Montage on a Grid like environment (a pool of linux machines), the results obtained and the issues that we faced. The computation executed was to generate a custom mosaic from a set of 91 2MASS image data sets. The image data set used for computation is available at http://gaul.isi.edu/montage/raw_data and the final generated mosaic files are available at http://gaul.isi.edu/montage/final_mosaic.fits and http://gaul.isi.edu/montage/final_mosaic_area.fits. This work was performed on an evaluation release of Montage, made available with the permission of the Montage Project.

2. Porting Montage to Chimera/Pegasus

Chimera is a virtual data system for representing, querying and automating data derivations [3] . Chimera provides a framework for representing a set of application programs and their instantiations as transformations and derivations respectively. For example a particular Montage operation such as

```
mImgtbl     data_dir     images.tbl
```

for picking up all the image files in the data_dir and creating a metadata table called images.tbl can be expressed as the following chimera transformation

```
TR    mImgtbl( in imagedir,        out imagestbl) {
      Argument = ${imagedir};
      Argument = ${imagestbl};
}
```

A particular derivation of the same can be expressed as

```
DV    d1->mImgtbl(
      Imagesdir="/home/data/raw_data",
      Imagestbl=@{out:"raw_images.tbl"}
);
```

The complete set of transformations and derivations used for the computation is shown in appendix I.

3. Montage Computations

The Montage version used for the computation was 1.4. In the following discussion we refer to individual montage methods like mImgtbl, mSubset etc as montage methods or operations while the whole process of getting the final image mosaic by composing these operations is called a montage computation. For running montage over chimera we had to generate the equivalent transformations and derivations for all the montage methods we used. Output and input arguments like image tables, corrections table etc were given logical file names which is used to chimera to identify dependencies between the operations. These logical file names are the parameters in the derivations shown in the appendix. An abstract dag is created for computing the final image mosaic fits file. This abstract graph of derivations produced by chimera for a set of operations is transformed into an executable DAG by the Pegasus planner. This executable DAG is then submitted to the Condor-G [4] to be executed over the Grid.

The Montage computation was run on a pool of linux machines at ISI. The input to the computation was a cache of 2 Micron All Sky Server (2MASS) image files in fits format and the output is the final image mosaic fits file. There is a series of 12 operations run to achieve the final output. (mImgtbl -> mMakehdr -> mSubset -> mProjExec -> mImgtbl -> mOverlaps -> mDiffExec -> mFitExec -> mBgModel -> mBgExec -> mImgtbl -> mAdd). There were 91 image files in the input set. The number was chosen keeping memory requirements in mind. The computation ran for about 5 hours. The header template used for the computation was derived from the raw data by using the mMakehdr operation. The following chart shows all the operations invoked and the average run time for each operation.

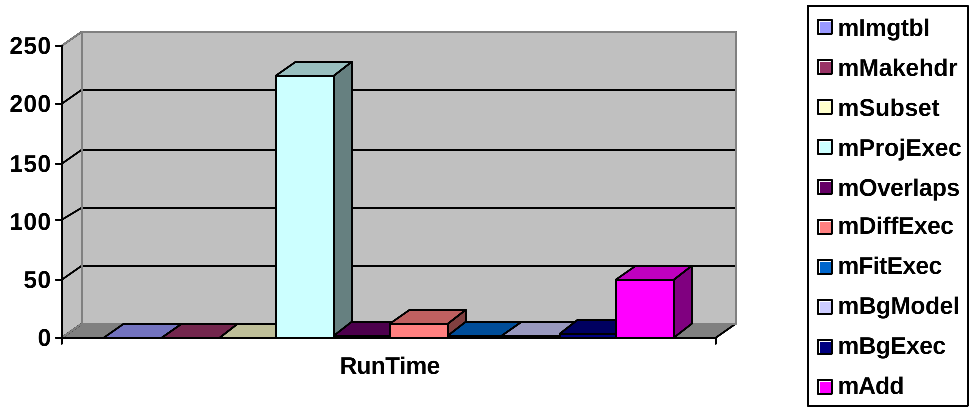

Fig 1. The run time of the operations. The Y axis shows the time in minutes. Most of the operations finish within a couple of minutes. However mProjExec takes 224 minutes and mAdd takes 50 minutes.

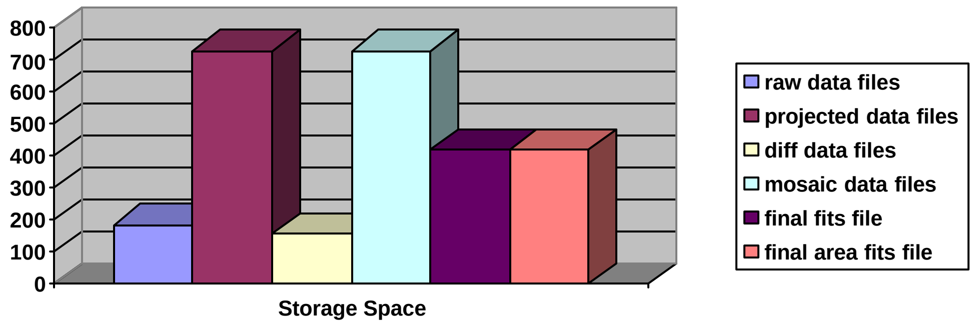

Fig 2. The storage requirements for the total computation. The Y axis show the space requirement in Mbytes. The size of input image data is 183 Mb. The computation creates the projected data files, difference data files, mosaic data files. The final operation mAdd creates two fits file each of 420 Mb. The total space requirement is around 2.7 Gb.

4. Issues involved

While porting montage to run on the grid wasn't particularly difficult, several issues were observed which are worth mentioning. Some of these issues may need to be addressed.

- ***User Interface to Montage***: What should be the user interface to Montage and what should be the granularity of the interface ? We can provide a web service kind of interface to the montage computation over the Grid. Since the computation can take a long time, the web service operations has to be asynchronous. The user can be returned a url where he can track the progress of the computation. Another issue is the granularity of the interface. We can provide a service which takes as input a set of image files and returns the final image mosaic or we can expose each of montage methods as web service operations and let the user compose them to achieve the final outcome. The second option is more difficult than the first one as it would increase the complexity and would require changes to how the montage operations are invoked.
- ***Specifying the input***: Presently the computation takes as the input the fits image files present in a directory. Obviously this will work if the image files are present on the same machine or file system accessible for a machine on which the computation is run. But this may not be true in most of the cases. First of all the user may not know on which machines the computation is going to run and he may not have the authorization to directly transfer image files to that pool. We need a standard interface for specifying the data input to the montage service. We can use a VOTable for specifying the input image files. This provides a standardized way to indicate the list of input images. Also we can specify logical file names and physical urls for each of these image files in the VOTable. The logical file name will help Pegasus to determine if the image file has already been fetched before as part of

some previous computation and if not then the url can be used to fetch the image file over the internet. There are a few operations in montage that take as input the data present in a particular directory. All these methods can use VOTable for specifying the data set. Using VOTable would require the montage code to be changed or we can write wrappers around those methods to work with VOTables.

- ***Identifying data provenance***: Each run of the montage computation should be tagged so that it can reuse data products computed by previous runs and the data products produced by this run can be used by future computations. This means that we should be able to assign logical names of the computed data products which are unique. The montage computation can run for days and produces gigabytes of intermediate data products. It would be an inefficient utilization of resources if the computations that have been done already need to be done again. These intermediate data products need to be given unique logical names. The data products derived for the same input data but with different parameters should be named differently to reflect their derivation. Either the logical names should reflect those parameters or attributes can be assigned to the files. The second figure shows that the computation has a storage space requirement of 2.7 Gb. Thus it would be very efficient to reuse the projected data files, diff data files, mosaic data files if they have been created in the past instead of creating them again.
- ***Identifying parallelism***: There is lot of scope to parallelize the current montage computation to decrease the effective run time. As can be seen from figure 1, the mProjExec operation takes 224 minutes while the total computation takes about 300 minutes. Thus there can be lots of saving if we can decompose the mProjExec operation. mProjExec is a wrapper which calls mProject sequentially 91 times (since there were 91 input 2mass image files). Thus if we can start the execution of these mProject operations in parallel we can reduce the total run time by utilizing more machines. Also it may be that some of these operations may need not be called as the projected image files may already have been computed in past and we can reuse them. However it will increase the communication cost of the model and it will require the implementation of mProjExec to be changed.
- ***Security issues***: Once we have decided on the interface to the service, we will need to decide on the security model. Should we allow anonymous users to access and utilize the service or should we allow only authenticated (plain password, GSI authentication etc) users to access the service. Another model can be to allow anonymous users to run limited size computations while requiring authentication for larger computations.
- ***Recovery issues***: Since the montage computation is supposed to run on the grid, we have to look into how to handle failures. When the computation is running standalone on a machine by a user, he can

observe if the computation has crashed and try to debug it by using the core dump. However in the grid environment, the computation can be scheduled on any available machine by the meta scheduling system (condor-g etc). In such environment it is very important for the application to do a graceful exit leaving enough information for the user to later go through the logs and infer what went wrong. For example the machine on which the mAdd computation is scheduled may not have enough memory to run the computation (for this particular run, it required about 880 Mb, for a input image set of about 242 files it requires more than 3 Gb). Thus instead of assuming that the memory will always be there, it would be more prudent to check the memory allocation before using it and logging and doing a graceful exit in case the memory is not there.

- ***Control dependencies***: The chimera virtual data system recognizes data dependencies between the operations and generates the abstract DAG accordingly. Some of the dependencies between the methods in montage is not explicit i.e. mDiffExec writes files in a difference directory and mFitExec reads file from that directory. Given that both these operations take a string as an argument which is the name of the difference directory, it is not possible for chimera to identify the control dependency between them. Thus we may like to modify the signature of these operations to make the dependency between them more explicit. One option is that mDiffExec creates an output VOTable which lists the created files and mFitExec reads this VOTable and uses the files. This will also allow these operations to be used as standalone services. The transformations and derivations for mDiffExec and mFitExec in the appendix show an extra dummy file being created by mDiffExec and consumed by mFitExec. This dummy argument has been added only for chimera to understand the control dependency between them. Similar arguments have been added to other methods where the dependency is not explicit.

5. References

[1] http://montage.ipac.caltech.edu
[2] http://www.us-vo.org
[3] http://www.griphyn.org/chimera
[4] http://www.cs.wisc.edu/condor/condorg

Appendix – I

The virtual data language transformations and derivations used to populate the virtual data catalog.

```
TR mImgtbl ( input dummy = @{input:"dummy":"temp"}, in imagedir , out imagestbl) {
            argument = ${imagedir};
            argument = ${imagestbl};
            argument = " -d 1";
}

TR mMakeHdr(in imagestbl, out templatehdr) {
    argument = ${imagestbl};
    argument = ${templatehdr};
            argument = " -d 1";
}

TR mSubset( in imagestbl, in templatehdr, out imagessubsettbl) {
            argument = ${imagestbl};
            argument = ${templatehdr};
            argument = ${imagessubsettbl};
            argument = " -d 1";
}

TR mProjExec( in imagessubsettbl, in templatehdr, in projdir, out statstbl,
                                                  output dummy = @{output:"dummy":"temp"} ) {
            argument = ${imagessubsettbl};
            argument = ${templatehdr};
            argument = ${projdir};
            argument = ${statstbl};
            argument = " -d 1";
}

TR mOverlaps( in imagestbl, out diffstbl) {
            argument = ${imagestbl};
            argument = ${diffstbl};
            argument = " -d 1";
}

TR mDiffExec( output dummy = @{output:"dummy":"temp"}, in diffstbl, in templatehdr, in diffdir ) {
            argument = ${diffstbl};
            argument = ${templatehdr};
            argument = ${diffdir};
            argument = " -d 1";
}

TR mFitExec( input dummy = @{input:"dummy":"temp"}, in diffstbl, out fitstbl, in diffdir) {
            argument = ${diffstbl};
            argument = ${fitstbl};
            argument = ${diffdir};
            argument = " -d 1";
}
```

```
TR mBgModel( in imagestbl, in fitstbl, out correctionstbl) {
            argument = ${imagestbl};
            argument = ${fitstbl};
            argument = ${correctionstbl};
            argument = " -d 1";
}

TR mBgExec( in imagestbl, in correctionstbl, in mosaicdir, output dummy =
                                                @{output:"dummy":"temp"} ) {
            argument = ${imagestbl};
            argument = ${correctionstbl};
            argument = ${mosaicdir};
            argument = " -d 1";
}

TR mAdd( in imagestbl, out finalmosaicfits, in templatehdr) {
            argument = ${imagestbl};
            argument = ${finalmosaicfits};

            argument = ${templatehdr};
            argument = " -d 1";
}

DV d1->mImgtbl(
                        imagedir="raw_data",
                        imagestbl=@{out:"raw_images.tbl"}
);

DV d2->mMakeHdr(
        imagestbl=@{in:"raw_images.tbl"},
        templatehdr=@{out:"template.hdr"}
);

DV d3->mSubset(
                        imagestbl=@{in:"raw_images.tbl"},
                        templatehdr=@{in:"template.hdr"},
                        imagessubsettbl=@{out:"raw_images_subset.tbl"}
);

DV d4->mProjExec(
                        imagessubsettbl=@{in:"raw_images_subset.tbl"},
                        templatehdr=@{in:"template.hdr"},
                        projdir="proj_data",
                        statstbl=@{out:"stats.tbl"},
                        dummy=@{output:"dummy1":"temp"}
);

DV d5->mImgtbl(
                        dummy=@{input:"dummy1":"temp"},
                        imagedir="proj_data",
                        imagestbl=@{out:"proj_images.tbl"}
);

DV d6->mOverlaps(
                        imagestbl=@{in:"proj_images.tbl"},
                        diffstbl=@{out:"diffs.tbl"}
);

DV d7->mDiffExec(
                    dummy=@{output:"dummy2":"temp"},
                    diffstbl=@{in:"diffs.tbl"},
                    templatehdr=@{in:"template.hdr"},
                    diffdir="diff_data"
);

DV d8->mFitExec(
                    dummy=@{input:"dummy2":"temp"},
                    diffstbl=@{in:"diffs.tbl"},
                    fitstbl=@{out:"fits.tbl"},
                    diffdir="diff_data"
);

DV d9->mBgModel(
                    imagestbl=@{in:"proj_images.tbl"},
                    fitstbl=@{in:"fits.tbl"},
                    correctionstbl=@{out:"corrections.tbl"}
);

DV d10->mBgExec(
                    imagestbl=@{in:"proj_images.tbl"},
                    correctionstbl=@{in:"corrections.tbl"},
                    mosaicdir="mosaic_data",
                    dummy=@{output:"dummy3":"temp"}
);

DV d11->mImgtbl(
                    dummy=@{input:"dummy3":"temp"},
                    imagedir="mosaic_data",
                    imagestbl=@{out:"mosaic_images.tbl"}
);


DV d12->mAdd(
                    imagestbl=@{in:"mosaic_images.tbl"},
                    finalmosaicfits=@{out:"final_mosaic.fits"},
                    templatehdr=@{in:"template.hdr"}
);
```