Image Processing with HTCondor

Montage: K-band 2MASS data

Table of Contents

CHAPTER 1: INTRODUCTION

1.1 Background

In this modern world of data and high-performance computing, distributed platforms have become a crucial medium for researchers and organizations to do large-scale and complex scientific works. Scientific research not only need to cater big data for data processing, but it also requires seamless integration between hosts and executor, scalable and have collaborative computing environments.

Cloud platforms such as Amazon Web Services (AWS) has been one of the most common platforms for the solution due to its scalability and cost-effective solution. It offers a robust infrastructure by providing extensive computing, storing, and networking capabilities for users to tailor their data distribution platforms according to requirements and needs. One notable study that has utilized the AWS platform is the International Centre for Radio Astronomy Research (ICRAR)'s the SkyNet project (Bektesevic et al., 2020). ICRAR have utilized AWS to process and analyse up to 20TB raw data and sky images from the Legacy Survey of Space and Time (LSST) to enable efficient processing, storage and sharing of astronomical datasets.

While distributed platforms serves abundance of benefits to compute large scale data and execute scientific research, configuring the platform has its own challenges and complexity. This project aims to study on distributed platform solutions by leveraging a combination of tools and technologies to address the challenges in resource management, data sharing, and workflow orchestration. After implementing the proposed solution, the goal of this study is to efficiently execute scientific workflows as a practical use to demonstrate the platform's capabilities.

1.2 Selected Tools for Distributed Platform

To achieve the objectives of this project, the following tools and technologies were selected to demonstrate the solution in a distributed cloud-based environment of AWS:

Task	Tool/Technology	Selection Justification
Resource Management/ Task Scheduler	HTCondor	Provides high throughput computing system in cloud environment, scalable and fault-tolerance. Gives efficient allocation of computational resources across distributed network, and able to handle job submission, execute and monitor.
Distributed Data Management	Network File System (NFS)	Efficiently share data across distributed nodes by providing centralized file system and ensuring all computational nodes have consistent access. Allows multiple machines to access large datasets simultaneously and able to reduce bottlenecks.
Workflow Management Engine	HTCondor Directed Acyclic Graph Manager (DAGMan)	Able to extend HTCondor functionality by enabling DAG management to ensure jobs are executed in correct sequence based on dependencies. This can allow a more complex workflow be executed by breaking it down into smaller tasks that are managed systematically.

Table 1: *Selected Tools and Technologies in AWS Distributed Platform*

By combining the tools together in the distributed platform, this can address the challenges to manage computational workflows and achieve the objective of handling job schedules, workflow orchestration and seamless data sharing while maintaining the scalability, flexibility and performance aspect of the platform.

1.3 Montage Use Case

Montage is a toolkit for processing and assembling Flexible Image Transport System (FITS) images into custom mosaics, which are widely used in astronomy for storing image and metadata information (Montage, 2022). Montage will be used as the primary use case for this project due to its real-world example of scientific computational workflow that requires efficient resource management, data sharing and task orchestration that serves the objective of this study.

In this project, the focus is to process K-band data from Montage 2MASS dataset to highlight the benefits of the proposed distributed platform. K-band data is a subset of near-infrared observations from the Two Micron All-Sky Survey (2MASS) that is highly valuable to study

celestial objects occurred by interstellar dust and give insights about the universe. The data volume and complexity is a great use case example to examine on how the integration of HTCondor, NFS, and DAGMan can effectively address the challenge of large-scale scientific workflow.

Therefore, this project aims to process the data, orchestrate and execute image mosaicking by utilizing the shell script suggested by Montage (Montage, 2022) and using the suggested distributed platform.

1.4 Architectural Design

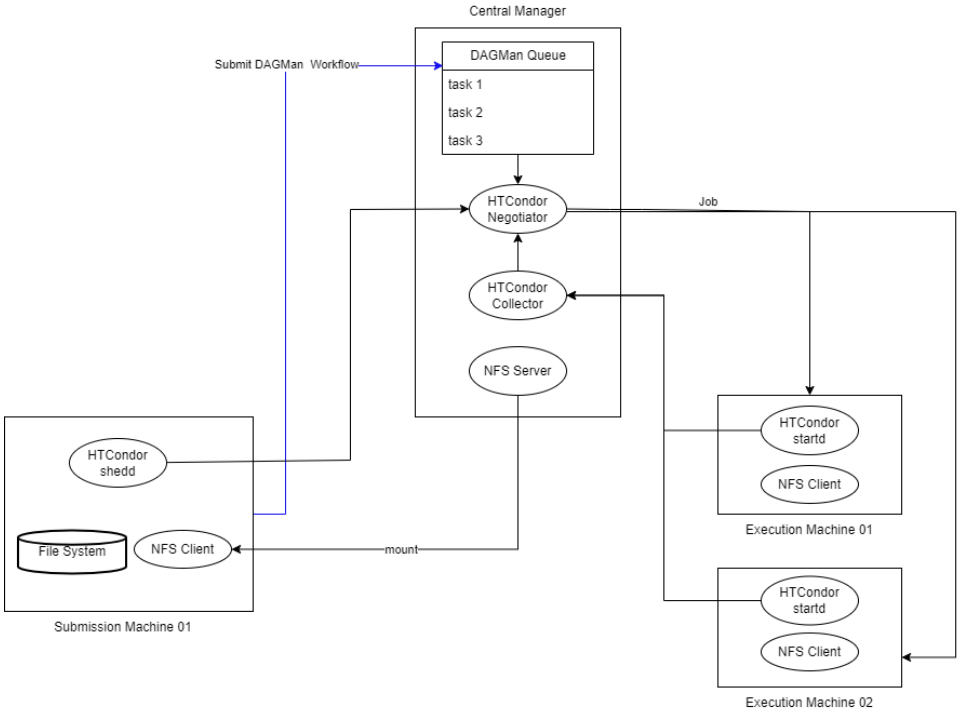

Figure 1: *Architecture of Distributed Data Processing Platform in AWS*

In this project, the entire system resides within the AWS cloud environment and Figure 1 showed the architecture design of the platform. At the compute layer, the HTCondor Central Manager act as the core component, managing job queues, resources, and the overall cluster. Users submit their jobs from HTCondor Submission Nodes to the Central Manager, which assigns the tasks to HTCondor Execution Nodes including instance type of t2.medium and

t2.large. These nodes carry out the computations defined in the Montage workflow, such as metadata generation (imgtbl.sh) and single-band mosaic creation (singleband.sh).

For the storage layer, a Network File System (NFS) server is used to provide shared storage accessible to all compute nodes. This centralized storage hosts input data, intermediate files, and output results. Compute nodes are configured as NFS clients, mounting the shared directory from the NFS server, enabling access to required data during job execution.

The workflow is managed by HTCondor DAGMan, a tool that manages the execution of the Montage workflow defined as a Directed Acyclic Graph (DAG). The DAG file specifies tasks and their order, such as defining the image processing steps in this project,ensuring that each step is executed in the correct order.

The flow of execution is as below:

1. The user submits the Montage DAG file from submission node to a Central Manager.
2. DAGMan, running on the Central Manager, reads the DAG and starts scheduling the individual tasks (jobs).
3. The Central Manager is responsible for matching submitted jobs with available resources (Execute Nodes). This involves the Negotiator and Collector components within the Central Manager, which work together to match jobs to Execute Nodes that meet the job's requirements.
4. Execution Nodes access the necessary input data from the NFS server, perform the computations, and write the results back to the NFS server.
5. DAGMan monitors the progress of the workflow and ensures that all tasks are completed in the correct order.

CHAPTER 2: DISTRIBUTED PLATFORM SETUP

2.1 Resource management and task scheduler (HTCondor)

In this project, 4 instances are setup in the AWS as follows Chapter 1.4 to set up a Condor pool that consists of a central manager, a submission host, and 2 execution machines.

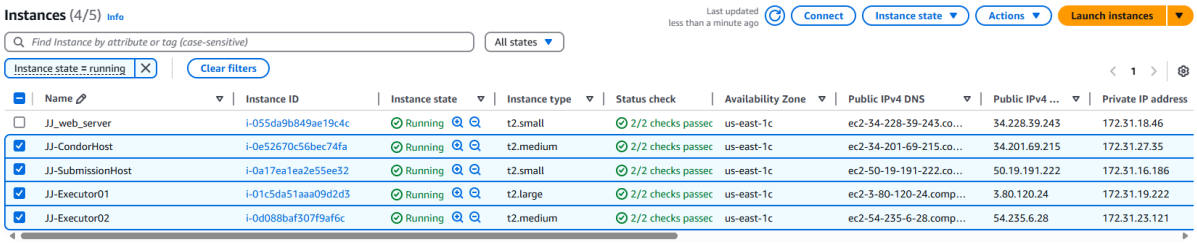

Figure 2: *Instances created at AWS*

To set up HTCondor across four machines, begin by adding the IP addresses and hostnames of all machines to the */etc/hosts* file for proper hostname resolution. Next, update the package repositories using *sudo apt update*. For each machine, download and run the HTCondor startup script from the official website using the command: *curl -fsSL https://get.htcondor.org | sudo GET_HTCONDOR_PASSWORD="abc123" /bin/bash -s -- --no-dry-run*, specifying the appropriate role. Use *--central-manager CondorHost* for the central manager, *--submit* for the submission host, and *--execute* for the execution hosts. Once the script completes, restart the HTCondor service using *sudo systemctl restart condor*. Repeat these steps for all machines, ensuring their roles are correctly specified. Finally, verify the pool's configuration and status by running the *condor_status* command (HTCondor, 2024).

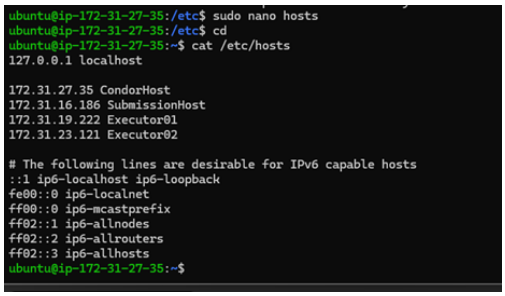

Figure 3: *Added IP address and hostname of all machines for hostname resolution*

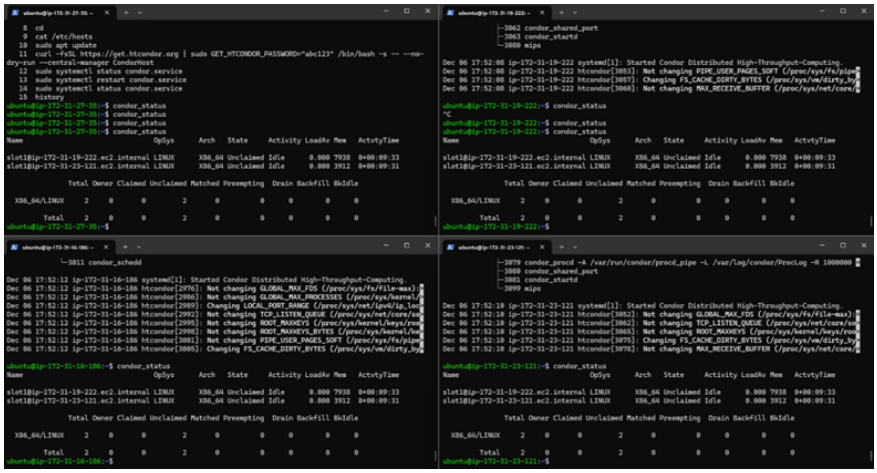

Figure 4: *Condor Pool status after setup*

2.2 Distributed data management (NFS)

2.2.1 Mount the NFS Share

To configure NFS for an HTCondor setup, the process begins with installing the NFS server on the CondorHost machine using the command *sudo apt install nfs-kernel-server* and creating a directory named "server-data" in the home directory. This directory is then exported through the NFS configuration file, enabling it to be shared with other machines in the network.

On the submission and execution hosts, the NFS client is installed using command *sudo apt install nfs-common*, and a corresponding mount point directory, "/mnt/server-data," is created. This directory is mounted to the "server-data" directory on CondorHost, providing access to shared resources.

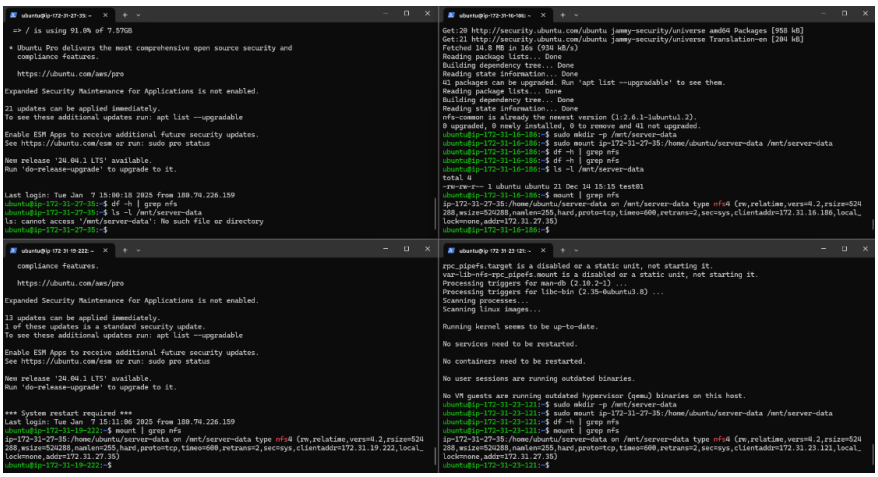

Figure 5: *Mount the NFS Share*

To ensure the configuration persists across reboots, the necessary settings can be added to the /etc/fstab file on the client machines, establishing a reliable, centralized data-sharing mechanism for the HTCondor environment.

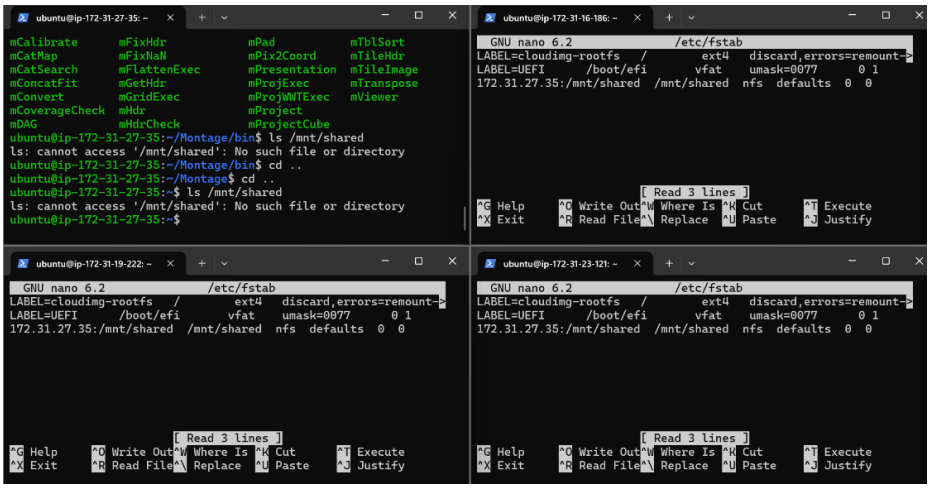

Figure 6: *Configure Permanent Mounting*

2.2.2 Input Montage data into NFS shared directory

To set up Montage, begin by downloading the source code from the official Montage website or directly from the provided tar file link of Montage_v6.0.tar.gz : http://montage.ipac.caltech.edu/download/Montage_v6.0.tar.gz . Extract the tar file and follow the installation instructions included in the package to complete the setup (Montage, 2018). Once Montage is installed, obtain input data using the mArchiveList command to query the 2MASS K-band for the target "NGC 3372" with a 1.0 x 1.0-degree field of view, saving the results to a file named archive_small.tbl. Next, use the mArchiveExec command with the -p raw option to download the data files listed in the table. Move the downloaded data files to the input directory located at /mnt/shared/input_data, ensuring they are accessible for further processing.

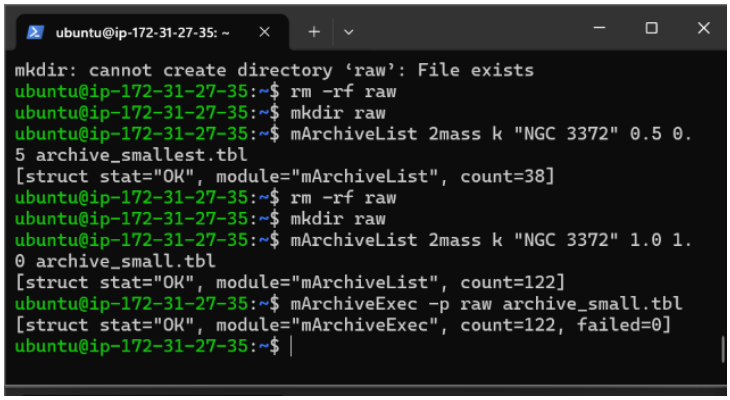

Figure 7: *Successfully download the data from mArchiveList command*

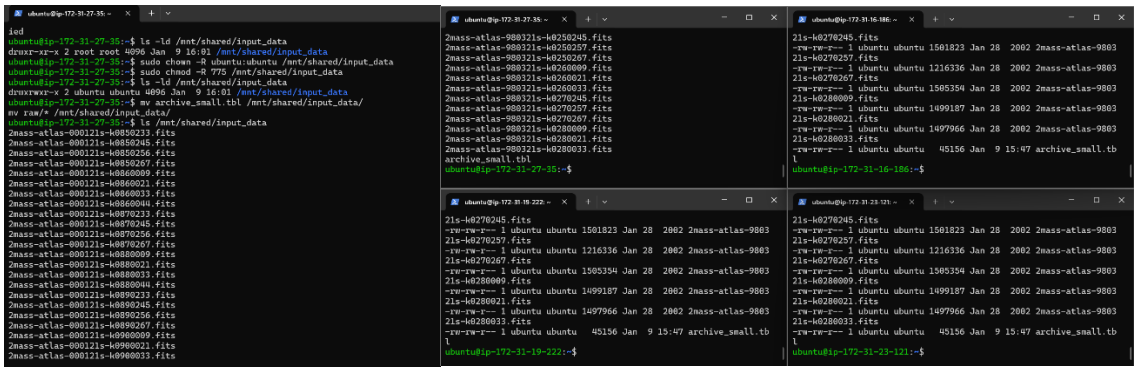

Figure 8: *Move the input data to NFS shared directory*

2.3 Workflow management engine (HTCondor & DAGMan)

To set up and execute a Montage workflow in HTCondor, job submission scripts are created for each task in the workflow. These scripts, such as imgtbl.sub and singleband.sub, define the execution parameters, input/output paths, and log file locations for each respective tasks.

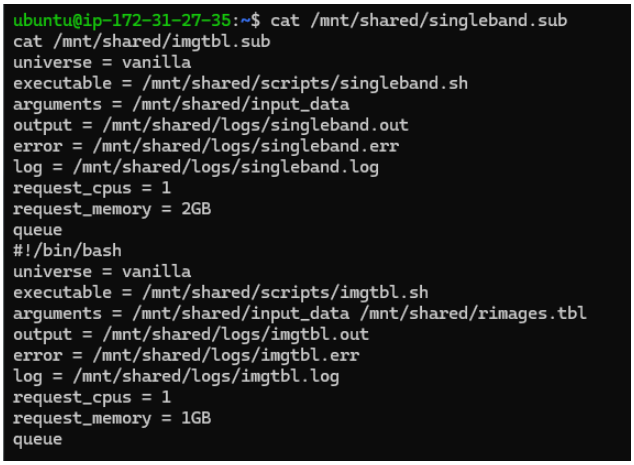

Figure 9: *Contents of singleband.sub & imgtbl.sub file*

The imgtbl.sub file configures the execution of imgtbl.sh, which generates metadata tables for raw astronomical images. These metadata tables are critical for subsequent tasks such as image reprojection and mosaic creation.

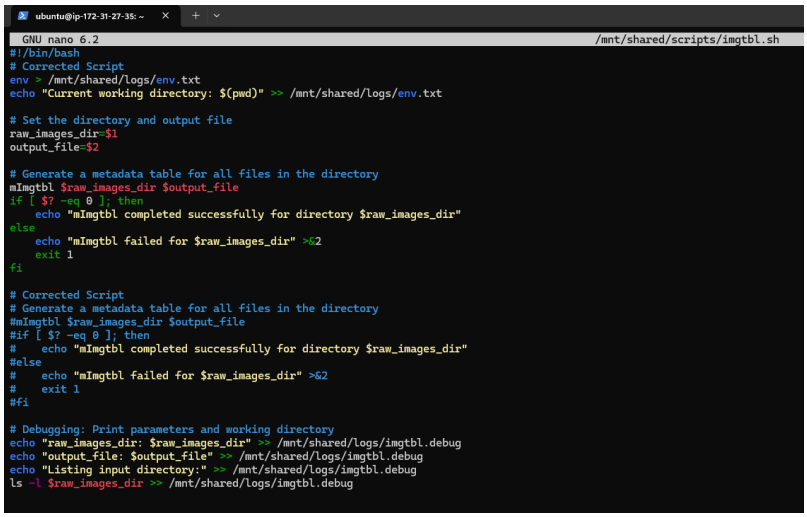

Figure 10: *Contents of imgtbl.sh file*

On the other hand, singleband.sub configures the execution of singleband.sh, a script that processes the K-band data. This script performs tasks such as metadata creation, image corrections, and the generation of a single-band mosaic, resulting in the final K-band mosaic image.

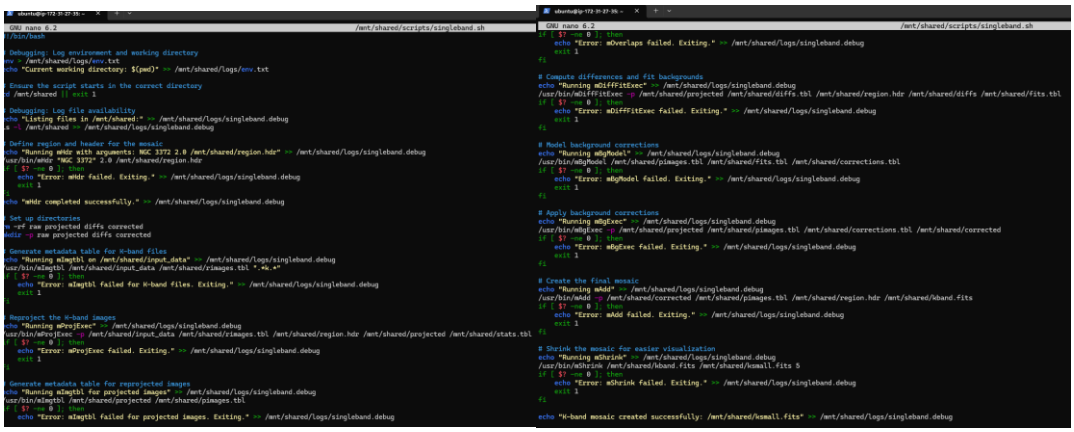

Figure 11: *Contents of singleband.sh file*

After preparing the sub files, the workflow is defined using a Directed Acyclic Graph (DAG) in the montage_workflow.dag file. The DAG specifies the sequence and dependencies of tasks within the workflow. In this setup, the workflow consists of two tasks: Imgtbl, which generates metadata tables using imgtbl.sh, and SingleBand, which processes the K-band data using

singleband.sh to create the mosaic. The DAG ensures that the Imgtbl task completes before the SingleBand task begins.

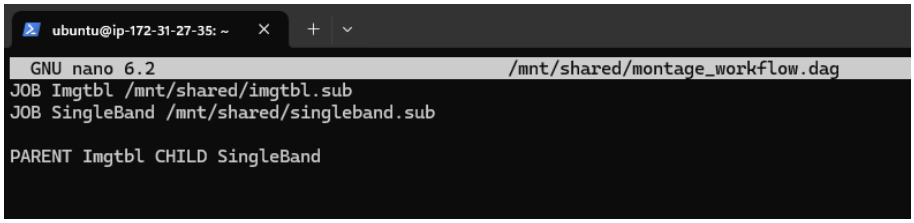

Figure 12: *Contents of .dag file*

Once the sub and DAG files are prepared, the workflow is submitted using the condor_submit_dag command. This triggers HTCondor to execute the jobs in the order specified by the DAG. Execution progress can be monitored using condor_q to check the job queue and reviewing the log files for detailed updates on job progress, completion, or errors. By utilising HTCondor and DAGMan, the Montage workflow can be managed efficiently, ensuring proper task execution order and maintaining dependencies between tasks.

CHAPTER 3: RESULTS & DISCUSSION

3.1 Mosaic Image Generation

After the workflow is completed, mViewer command is run with the FITS file as input and specify the output image as PNG format.

```
ubuntu@ip-172-31-16-186:~$ mViewer -ct 1 -gray /mnt/shared/ksmall.fits min max -out /mnt/shared/ksmall.png
[struct stat="OK", module="mViewer", type="grayscale", width=1209, height=1274, min=533.772, minpercent=0.00, m
insigma=-1.26, max=9683.51, maxpercent=100.00, maxsigma=134.69, datamin=533.772, datamax=9683.51, xflip=0, yfli
p=1, bunit="", colortable=1]
ubuntu@ip-172-31-16-186:~$
```

Figure 13: *Snippet code on executing mViewer command*

After the image file is generated, it can be downloaded it to local machine.

```
PS C:\Users\User> scp -i "C:\Users\User\Downloads\JJKey.pem" ubuntu@ec2-54-164-54-28.compute-1.amazonaws.com:/mnt/shared
/ksmall.png C:\Users\User\Downloads
ksmall.png                                                    100%  159KB 112.2KB/s   00:01
PS C:\Users\User>
```

Figure 14: *Snippet code on downloading the output image to local machine*

The images below show the output of the workflow, producing mosaics from 212 K-band 2MASS image.

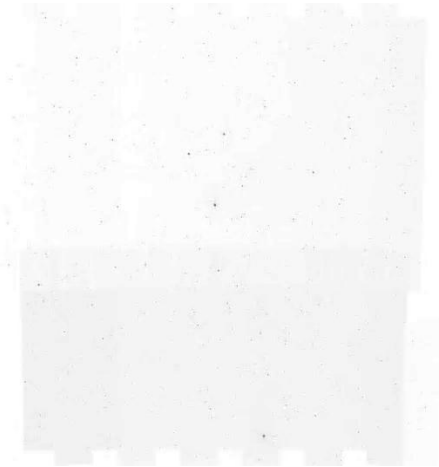

Figure 15: *Image output for ksmall.fits*

3.2 Resource Utilisation

Commands of condor_history is used to find completed jobs and their status. From Figure 16, Job ID of 51.0 is the HTCondor DAGMan process managing the DAG workflow. It handles dependencies and orchestrates the execution of imgtbl and singleband. While Job ID of 52.0 is the imgtbl job, which corresponds to the first job in the DAG (JOB Imgtbl). Lastly, Job ID of 53.0 is the singleband job, which run after imgtbl, corresponds to JOB SingleBand.

```
ubuntu@ip-172-31-16-186:~$ condor_history
 ID     OWNER          SUBMITTED   RUN_TIME     ST COMPLETED   CMD
51.0    ubuntu          1/13 08:47   0+00:02:02 C   1/13 08:49 /usr/bin/condor_dagman -p 0 -f -l . -Lockfile /
53.0    ubuntu          1/13 08:48   0+00:01:18 C   1/13 08:49 /mnt/shared/scripts/singleband.sh /mnt/shared/i
52.0    ubuntu          1/13 08:47   0+00:00:03 C   1/13 08:47 /mnt/shared/scripts/imgtbl.sh /mnt/shared/input
```

Figure 16: *Snippet code on executing condor_history for HTCondor DAGMan process*

Next, the workflow is restricted to run a specific executor, executor 2 with instance type of t2.medium, and the performance is compared with the DAGMan process. The condor_history output indicates that the jobs in executor 2 has completed successfully, including the Imgtbl job (Job ID 55.0) and the SingleBand (Job ID 56.0). The overall DAG execution was handled by the condor_dagman process (ID 54.0). All jobs finished successfully with no errors, as indicated by the ST (Status) column showing C (Completed).

```
ubuntu@ip-172-31-16-186:~$ condor_history
 ID     OWNER          SUBMITTED   RUN_TIME     ST COMPLETED   CMD
54.0    ubuntu          1/13 09:49   0+00:02:08 C   1/13 09:51 /usr/bin/condor_dagman -p 0 -f
56.0    ubuntu          1/13 09:49   0+00:01:32 C   1/13 09:51 /mnt/shared/scripts/singleband
55.0    ubuntu          1/13 09:49   0+00:00:05 C   1/13 09:49 /mnt/shared/scripts/imgtbl.sh
```

Figure 17: *Snippet code on executing condor_history for workflow run on the executor 2*

Scenario	Process/ Job	Execution Time	Disk Usage during execution	Physical memory (RAM)	Memory allocation
DAG job execution	**DAGMan process**	122 seconds	750 KB	nil	nil
	Imgtbl job	3 seconds	17 KB	125 KB	1024 MB
	SingleBand job	78 seconds	75 KB	195 MB	2048 MB
Execution in Executor 2	**DAGMan process**	128 seconds	750 KB	-	-
	Imgtbl job	5 seconds.	17 KB	3.25 MB	1024 MB
	SingleBand job	92 seconds	75 KB	250 MB	2048 MB

Table 2: *Performance Comparison*

The table compares the performance of the workflow execution under two scenarios: a general DAG execution and execution restricted to a specific executor (Executor 2, with a t2.medium instance). In the first scenario, the HTCondor scheduler automatically distributed the workflow across available executors. As the distributed execution utilized the capabilities of multiple executors, the resources for these jobs were distributed across available executors without explicit targeting. Hence, this led to faster execution times for both jobs since the resources were shared and there was less contention. The DAGMan process completed in 122 seconds, while the Imgtbl job took 3 seconds with 125 KB of memory usage, and the SingleBand job took 78 seconds with 195 MB of memory usage

In the second scenario, the sub files were modified to constrain the workflow to a specific executor (Executor 2, a t2.medium instance). The DAGMan process execution time increased slightly to 128 seconds, still with 750 KB disk usage. The imgtbl job execution time increased to 5 seconds, using 3.25 MB of physical memory and the same memory allocation of 1024 MB. Similarly, the singleband job took 92 seconds, with physical memory usage increasing to 250 MB while maintaining its memory allocation of 2048 MB. This increase in execution time and

resource utilization is attributed to the restriction to a single executor, leading to serialized job execution and potential resource contention on the single instance.

The comparison showed that the distributed execution performs better as HTCondor scheduler automatically distributed the workflow across available executors, resulting in optimized resource utilization and reduced execution times. In contrast, running jobs on a single executor has several disadvantages. Firstly, the execution time increases due to the serialization of tasks, as jobs must wait for available resources on the single executor, leading to potential restriction. Moreover, there is a higher likelihood of resource contention, particularly if the allocated memory or CPU resources are insufficient to handle the workload efficiently. Lastly, the single executor becomes a single point of failure—any issues with the instance (e.g., downtime or hardware failures) can interrupt the workflow, whereas distributed execution offers redundancy and resilience by spreading the workload across multiple nodes. This demonstrates the benefits of distributed computing in achieving scalability and robustness for complex workflows.

CHAPTER 4: PROJECT LIMITATIONS

One of the limitations in this project is that the disk usage report reveals that the root partition (/dev/root) has a total capacity of 7.6 GB, out of which 6.3 GB is already used, as shown in Figure 18. This indicates that 84% of the disk space is occupied, limiting the capacity for additional data processing or storage. As the constraints of this project, which involves handling astronomical data, such as generating mosaics for various bands (K, H, and J bands), the limited available disk space poses a significant challenge. Hence, the decision to process data only for the K band while excluding the H and J bands to generate a color image. This exclusion conserved the limited disk space and ensuring that the project can be completed within the current storage constraints.

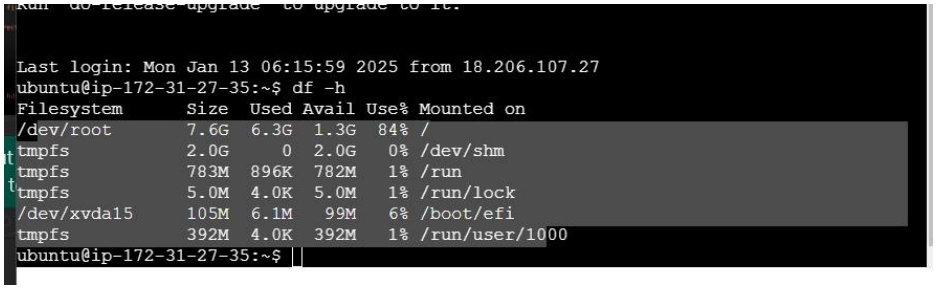

Figure 18: *Snippet code on disk space*

CHAPTER 5: CONCLUSION

This project successfully set up a distributed computing platform using HTCondor, HTCondor DAGMan, and NFS on AWS. It was used to handle complex scientific workflows, as demonstrated through the Montage use case for creating astronomical image mosaics. The results showed that distributed execution is faster and more efficient compared to running it on an executor.

However, the project faced a challenge with limited disk space, which restricted the amount of data processed. Despite this, the platform proved to be effective for managing large workflows.

This work highlights the benefits of distributed computing for scientific tasks. In the future, improving storage capacity and optimizing resource use could further enhance the platform's performance and handle larger datasets more efficiently.

REFERENCES

Bektesevic, D., Chiang, H.-F., Lim, K.-T., Miller, T. L., Thain, G., Jenness, T., Bosch, J., Salnikov, A., & Connolly, A. (2020, November 11). *A Gateway to Astronomical Image Processing: Vera C. RubinObservatory LSST Science Pipelines on AWS*. ArXiv.org. https://doi.org/10.48550/arXiv.2011.06044

Montage. (2022). *Montage - Image Mosaic Software for Astronomers*. Caltech.edu. http://montage.ipac.caltech.edu

Montage. (2025). *Montage User Documentation*. Caltech.edu. http://montage.ipac.caltech.edu/docs/montagescript.html

APPENDIX (Performance Comparison)

Job id 51:

```
ubuntu@ip-172-31-16-186:~$ condor_history 51.0 -long
Arguments = "-p 0 -f -l . -Lockfile /mnt/shared/montage_workflow.dag.lock -Dag /mnt/shared/montage_workflow.dag -MaxIdle
1000 -CsdVersion $CondorVersion:' '23.10.18' '2024-11-18' 'BuildID:' '769621' 'PackageID:' '23.10.18-1+ubu22' 'GitSHA:' '
0208a1f0' '$ -dagman /usr/bin/condor_dagman -AutoRescue 1 -DoRescueFrom 0 -force"
ClusterId = 51
Cmd = "/usr/bin/condor_dagman"
CommittedSlotTime = 0
CommittedSuspensionTime = 0
CommittedTime = 0
CompletionDate = 1736758172
CondorPlatform = "$CondorPlatform: X86_64-Ubuntu_22.04 $"
CondorVersion = "$CondorVersion: 23.10.18 2024-11-18 BuildID: 769621 PackageID: 23.10.18-1+ubu22 GitSHA: 0208a1f0 $"
CumulativeRemoteSysCpu = 0.0
```

```
DiskUsage = 750
DiskUsage_RAW = 516
EnteredCurrentStatus = 1736758172
Environment = "HOME=/home/ubuntu LANG=C.UTF-8 PATH=/home/ubuntu/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr
/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/mnt/server-data/m101/Montage/bin USER=ubuntu XDG_RUNTIME_DIR=/run
/user/1000 _CONDOR_DAGMAN_LOG=/mnt/shared/montage_workflow.dag.dagman.out _CONDOR_MAX_DAGMAN_LOG=0 _CONDOR_SCHEDD_ADDRES
S_FILE=/var/spool/condor/.schedd_address _CONDOR_SCHEDD_DAEMON_AD_FILE=/var/spool/condor/.schedd_classad"
Err = "/mnt/shared/montage_workflow.dag.lib.err"
ExecutableSize = 750
ExecutableSize_RAW = 516
ExitBySignal = false
```

```
RemoteSysCpu = 0.0
RemoteUserCpu = 0.0
RemoteWallClockTime = 122.0
RemoveKillSig = "SIGUSR1"
RequestCpus = 1
RequestDisk = DiskUsage
RequestMemory = ifthenelse(MemoryUsage =!= undefined,MemoryUsage,(ImageSize + 1023) / 1024)
Requirements = (TARGET.Arch == "X86_64") && (TARGET.OpSys == "LINUX") && (TARGET.Disk >= RequestDisk) && (TA
= RequestMemory)
ShouldTransferFiles = "IF_NEEDED"
```

Job ID 52:

```
ubuntu@ip-172-31-16-186:~$ condor_history 52.0 -long
ActivationDuration = 3
ActivationExecutionDuration = 2
ActivationSetupDuration = 1
ActivationTeardownDuration = 0
Args = "/mnt/shared/input_data /mnt/shared/rimages.tbl"
BlockReadKbytes = 0
BlockReads = 0
BlockWriteKbytes = 0
BlockWrites = 0
BytesRecvd = 1058.0
BytesSent = 158.0
ClusterId = 52
```

```
DAGNodeName = "Imgtbl"
DAGParentNodeNames = ""
DiskProvisioned = 2048
DiskUsage = 17
DiskUsage_RAW = 17
EnteredCurrentStatus = 1736758073
Environment = ""
Err = "/mnt/shared/logs/imgtbl.err"
ExecutableSize = 2
ExecutableSize_RAW = 2
ExecuteDirWasEncrypted = false
ExitBySignal = false
ExitCode = 0
```

```
RemoteSysCpu = 0.0
RemoteUserCpu = 1.0
RemoteWallClockTime = 3.0
RequestCpus = 1
RequestDisk = DiskUsage
RequestMemory = 1024
Requirements = (TARGET.Arch == "X86_64") && (TARGET.OpSys == "LINUX") && (TARGET.Disk >= RequestDisk) && (TARGET.Memory
>= RequestMemory) && ((TARGET.FileSystemDomain == MY.FileSystemDomain) || (TARGET.HasFileTransfer))
ResidentSetSize = 125
ResidentSetSize_RAW = 120
ScratchDirFileCount = 10
```

Jod id 53:

```
ubuntu@ip-172-31-16-186:~$ condor_history 53.0 -long
ActivationDuration = 78
ActivationExecutionDuration = 77
ActivationSetupDuration = 1
ActivationTeardownDuration = 0
Args = "/mnt/shared/input_data"
```

```
DiskProvisioned = 2048
DiskUsage = 75
DiskUsage_RAW = 63
EnteredCurrentStatus = 1736758168
Environment = ""
Err = "/mnt/shared/logs/singleband.err"
ExecutableSize = 4
ExecutableSize_RAW = 4
```

```
RecentStatsLifetimeStarter = 69
RemoteSysCpu = 5.0
RemoteUserCpu = 44.0
RemoteWallClockTime = 78.0
RequestCpus = 1
RequestDisk = DiskUsage
RequestMemory = 2048
Requirements = (TARGET.Arch == "X86_64") && (TARGET.OpSys == "LINUX") && (TARGET.Disk >= RequestDisk) && (TARGET.Memor
y >= RequestMemory) && ((TARGET.FileSystemDomain == MY.FileSystemDomain) || (TARGET.HasFileTransfer))
ResidentSetSize = 200000
ResidentSetSize_RAW = 190624
ScratchDirFileCount = 10
ShouldTransferFiles = "IF_NEEDED"
```

Job id 54:

```
ubuntu@ip-172-31-16-186:~$ condor_history 54.0 -long
Arguments = "-p 0 -f -l . -Lockfile /mnt/shared/montage_workflow.dag.lock -Dag /mnt/shared/montage_workflow.dag -MaxIdle 1000 -CsdVersion $Condo
rVersion:' '23.10.18' '2024-11-18' 'BuildID:' '769621' 'PackageID:' '23.10.18-1+ubu22' 'GitSHA:' '0208a1f0' '$ -dagman /usr/bin/condor_dagman -A
utoRescue 1 -DoRescueFrom 0 -force"
ClusterId = 54
Cmd = "/usr/bin/condor_dagman"
CommittedSlotTime = 0
CommittedSuspensionTime = 0
```

```
DAG_Status = 0
DiskUsage = 750
DiskUsage_RAW = 516
EnteredCurrentStatus = 1736761886
Environment = "HOME=/home/ubuntu LANG=C.UTF-8 PATH=/home/ubuntu/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/gam
es:/usr/local/games:/snap/bin:/mnt/server-data/m101/Montage/bin USER=ubuntu XDG_RUNTIME_DIR=/run/user/1000 _CONDOR_DAGMAN_LOG=/mnt/shared/montag
e_workflow.dag.dagman.out _CONDOR_MAX_DAGMAN_LOG=0 _CONDOR_SCHEDD_ADDRESS_FILE=/var/spool/condor/.schedd_address _CONDOR_SCHEDD_DAEMON_AD_FILE=/
```

```
RemoteSysCpu = 0.0
RemoteUserCpu = 0.0
RemoteWallClockTime = 128.0
RemoveKillSig = "SIGUSR1"
RequestCpus = 1
RequestDisk = DiskUsage
RequestMemory = ifthenelse(MemoryUsage =!= undefined,MemoryUsage,(ImageSize + 1023) / 1024)
Requirements = (TARGET.Arch == "X86_64") && (TARGET.OpSys == "LINUX") && (TARGET.Disk >= RequestDisk) && (TARGET.Memory >= RequestMemory)
ShouldTransferFiles = "IF_NEEDED"
StreamErr = false
```

Jod id 55:

```
ubuntu@ip-172-31-16-186:~$ condor_history 55.0 -long
ActivationDuration = 5
ActivationExecutionDuration = 4
ActivationSetupDuration = 1
ActivationTeardownDuration = 0
Args = "/mnt/shared/input_data /mnt/shared/rimages.tbl"
```

```
DiskProvisioned = 2048
DiskUsage = 17
DiskUsage_RAW = 17
EnteredCurrentStatus = 1736761777
Environment = ""
Err = "/mnt/shared/logs/imgtbl.err"
ExecutableSize = 2
```

```
RemoteUserCpu = 1.0
RemoteWallClockTime = 5.0
RequestCpus = 1
RequestDisk = DiskUsage
RequestMemory = 1024
Requirements = (TARGET.Name == "slot1@ip-172-31-23-121.ec2.internal") && (TARGET.Arch == "X86_64") && (TARGET.OpSys == "LINUX") && (TARGET.Di
sk >= RequestDisk) && (TARGET.Memory >= RequestMemory) && ((TARGET.FileSystemDomain == MY.FileSystemDomain) || (TARGET.HasFileTransfer))
ResidentSetSize = 3250
ResidentSetSize_RAW = 3180
ScratchDirFileCount = 10
```

Job id 56:

```
ubuntu@ip-172-31-16-186:~$ condor_history 56.0 -long
ActivationDuration = 92
ActivationExecutionDuration = 91
ActivationSetupDuration = 1
ActivationTeardownDuration = 0
Args = "/mnt/shared/input_data"
BlockReadKbytes = 0
BlockReads = 0
```

```
DiskProvisioned = 2048
DiskUsage = 75
DiskUsage_RAW = 63
EnteredCurrentStatus = 1736761884
Environment = ""
Err = "/mnt/shared/logs/singleband.err"
ExecutableSize = 4
ExecutableSize_RAW = 4
```

```
RemoteSysCpu = 5.0
RemoteUserCpu = 44.0
RemoteWallClockTime = 92.0
RequestCpus = 1
RequestDisk = DiskUsage
RequestMemory = 2048
Requirements = (TARGET.Name == "slot1@ip-172-31-23-121.ec2.internal") && (TARGET.Arch == "X86_64") && (TARGET.OpSys == "LINUX") && (TARGET.Di
sk >= RequestDisk) && (TARGET.Memory >= RequestMemory) && ((TARGET.FileSystemDomain == MY.FileSystemDomain) || (TARGET.HasFileTransfer))
ResidentSetSize = 250000
ResidentSetSize_RAW = 226544
```