***Title*:** ALMA pipeline bench-marking and profiling report #1	***Author*:** Felipe Madsen, James Robnett, K. Scott Rowe	***Date*:** 03/05/2019
NRAO Doc. #:		***Version*:**

ALMA pipeline benchmarking and profiling report #1

PREPARED BY	ORGANIZATION	DATE
Felipe Madsen	NRAO	03/05/2019
James Robnett	NRAO	03/05/2019
K. Scott Rowe	NRAO	05/05/2019

Change Record

VERSION	DATE	REASON

TABLE OF CONTENTS

1 INTRODUCTION

A framework for benchmarking and profiling the ALMA pipeline(s) has been developed by the Scientific Computing Group at NRAO. The main goal is to characterize the execution of the pipelines with respect to computing resources. Our expectation is that this work will help ALMA ARCs have a deeper understanding of the computational cost of data processing jobs, while providing developers an additional tool to help track specific areas where CASA can be made more resource efficient.

This is the first of a series of reports associated with the profiling work. A complete documentation of the profiling framework will be released with the initial profiling report. Depending on findings, we may break out specific profiling topics, e.g. parallelization, memory footprint, I/O patterns into their own report. In this report, we present an overview of the framework's capabilities: the data it collects and automatic plots that are generated. We also describe the methodology applied in this work, the criteria for data selection, parameter space coverage and the hardware environment on which the tests were performed. We present our first results, discuss how we can do a deeper analysis of the data for this first round of tests, indicate further testing that we plan to execute on different hardware and software environments and areas we request input as to priority.

2 OVERVIEW OF THE PROFILING FRAMEWORK

The pipeline profiling framework is a set of python and bash tools that interact with the CASA pipeline to measure metrics information. At present, the following measurements are available on a per task basis (for both CASA and pipeline tasks) and can be obtained from any pipeline profiling run.

- Timing
- Memory footprint (aggregate for process and child processes)
- Memory load of a node (used, cached, swap and the largest slab block)
- Number of file descriptors (aggregate for process and child processes)
- IO statistics on lustre file system (number of files per IO size range - 0k-4k, 4k-8k...)
- Number of system calls (open, close, read, write, fcntl, fsync)

3 DATASETS SELECTED FOR PROFILING

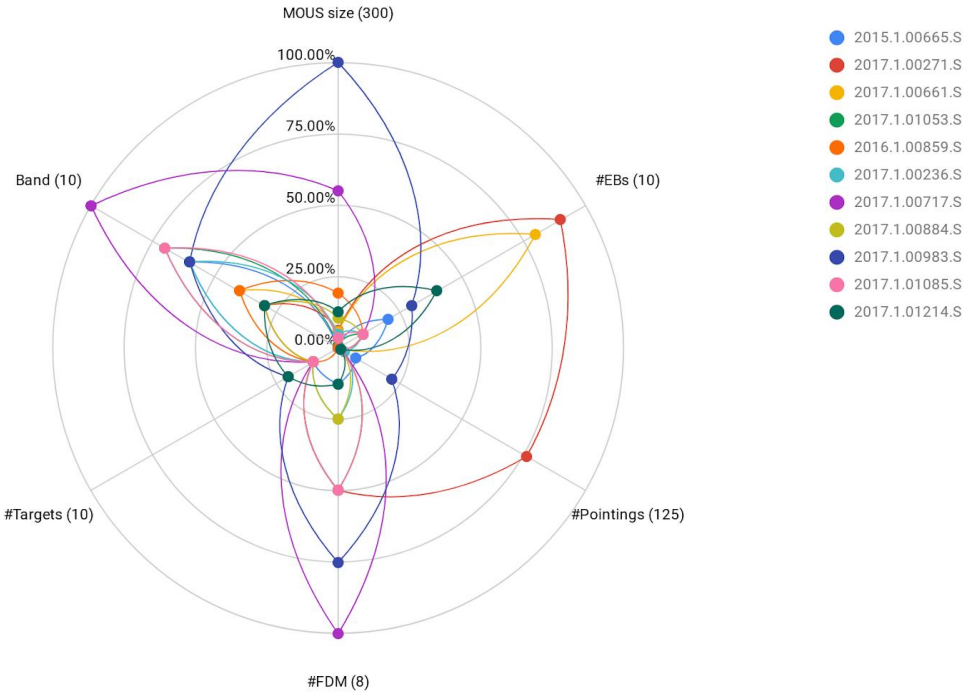
We have selected a subset of ALMA datasets that have already been used for pipeline regressions and other testing. The selection was done in order to maximize the coverage of the parameter space defined by the following:

- Data size (number of visibilities)
- Frequency band
- Number of execution blocks (EBs)
- Number of pointings
- Number of targets
- Number of FDM spectral windows

We are currently running the profiling tests with data from the following projects and corresponding MOUS:

- 2015.1.00665.S - uid://A001/X2d8/X2c5
- 2016.1.00859.S - uid://A001/X87c/X61c
- 2017.1.00236.S - uid://A001/X1296/X8ef
- 2017.1.00271.S - uid://A001/X1273/X2e3
- 2017.1.00661.S - uid://A001/X128e/X222
- 2017.1.00717.S - uid://A001/X1273/Xc66
- 2017.1.00884.S - uid://A001/X1296/X7b1
- 2017.1.00983.S - uid://A001/X12a3/X3be
- 2017.1.01053.S - uid://A001/X1288/Xf93
- 2017.1.01085.S - uid://A001/X1284/X95f
- 2017.1.01214.S - uid://A001/X1284/Xb3e

The parameter space coverage is represented in the plot below. The numbers in parenthesis close to the axes labels correspond to the maximum value considered for that axis. A table showing the parameter space coverage for our selection is also shown in Appendix 1. To simplify notation throughout the document, we will use only the project name.



4 METHODS AND MEASUREMENTS

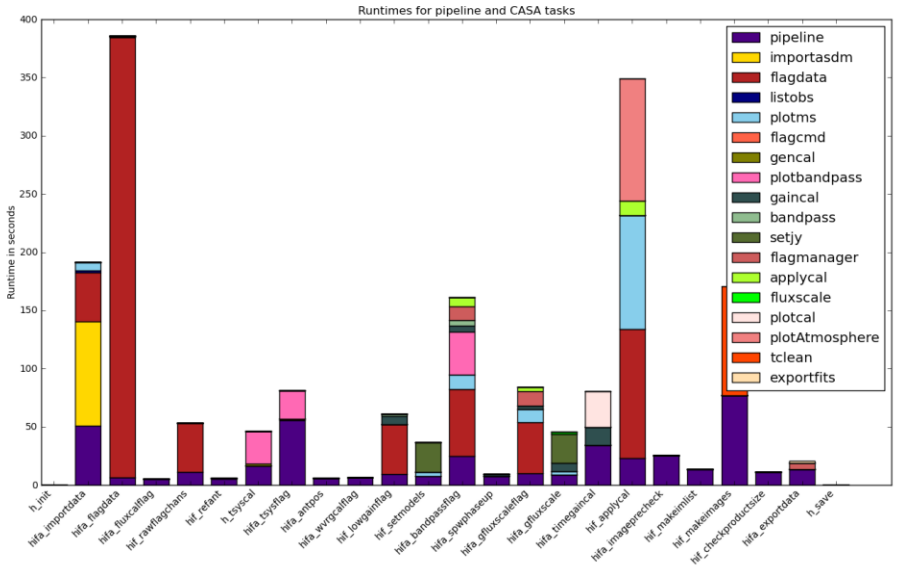
All the tests reported here were run as serial batch jobs in the NM post processing cluster, and we have taken special care to make sure they all run on nodes with the same hardware. The jobs were run on nodes nmpost001 through nmpost050, with 192 GB of RAM, ES-2670 CPUs and running Red Hat Enterprise Linux (RHEL) 6.9. We have also chosen the current casa release (5.4.0-68 by the time we started the testing) as the baseline version, on which we performed all

the testing reported here. To ensure consistency of results, we requested all the memory on the nodes, so that a profiling run was the only processing job running on a node during its entire execution. We also cleared the filesystem cache at the beginning of each run to ensure clean I/O statistics.

To record data we rely on two separate mechanisms. The first mechanism relies on slightly modifying the pipeline script to add a method to record timing and I/O shape (counts by size) data before and after each discrete pipeline task. Execution times are recorded for each CASA and pipeline task, as well as the entire pipeline run. The timings are recorded in such a way that we retain the ability to associate a specific instance of a CASA task with a specific pipeline task, for instance different 'plotms' calls through the pipeline. A plot of the absolute timings for each pipeline and the enclosed CASA tasks is produced automatically by the framework at pipeline completion. We then calculate the percentage that each task (CASA or pipeline) contributes towards the total execution times.

The relative (or percentage) time data can be used to compare the profiles for datasets that differ in any of the axes of the parameter space, looking for variations that can be correlated with the data structure. We do that by computing the average percentage execution times, the standard deviation, minimum and maximum for each task. The tasks that show larger standard deviations are identified and selected for detailed analysis.

The following is an example of the plot of absolute timings for pipeline and CASA tasks that is produced automatically by the framework.



The second mechanism involves an external monitoring task which continuously measures memory usage by the pipeline and child processes at 20 second intervals as well as every time a task starts or ends. We also record at the same rate: current running task, total memory used, cache and swap used, number of file descriptors and size of the largest slab block (usually 512 MB). For this report, we have primarily considered the memory footprint of entire jobs, but the data allows a deeper analysis - down to the level of pipeline and CASA tasks. We also look in some detail at the job memory time series to investigate the effect of setting the parameter "system.resources.memory" in the ~/.casarc file.

5 RESULTS

Initial profiling results are reported in subsections below and are broken out by timing, memory, file descriptor count, I/O shape and system calls. Results for both the calibration and imaging pipelines are included together per profiling characteristic.

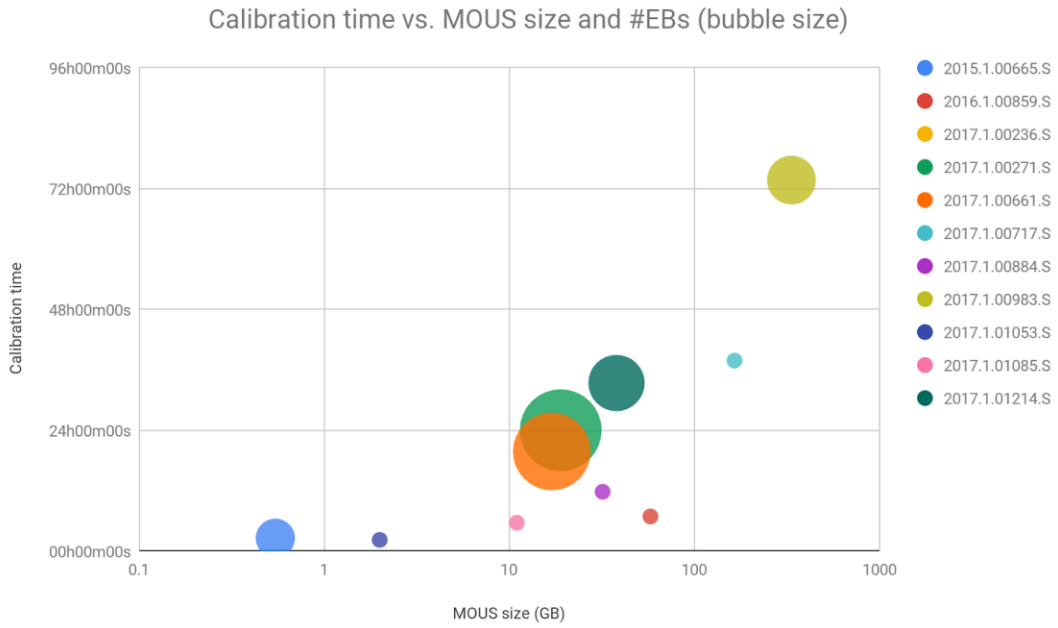
Results for absolute timing and memory footprint are summarized in the table below. The table also shows the size of both the MOUS and the calibrated MS files. Execution times range from 00h28m00s (for an imaging job that diverges quickly) to 829h38m52s for imaging the largest dataset (335 GB MOUS size; 1100 GB calibrated MS size). Memory footprints range from 1.57 to 104.29 GB in the imaging case, while it spans much less for calibration, ranging from 2.25 to 14.70 GB.

			Execution time		memory footprint (GB)	
Project	MOUS size (GB)	Cal MS size (GB)	Calibration	Imaging	Calibration	Imaging
7m array						
2015.1.00665.S	0.546	3.4	02h36m50s	00h28m00s	2.53	2.31
2017.1.00271.S	19	102	24h03m48s	37h37m09s	10.78	13.46
2017.1.00661.S	17	91	19h49m31s	08h34m59s	8.95	4.84
2017.1.01053.S	2	11	02h16m42s	02h11m38s	2.25	1.57
12m array						
2016.1.00859.S	58	33	06h55m34s	04h47m53s	6.65	9.94
2017.1.00236.S	15	49	09h42m40s	62h50m56s	7.12	71.34
2017.1.00717.S	165	510	37h53m05s	69h44m55s	10.96	13.02
2017.1.00884.S	32	103	11h49m46s	32h05m30s	14.70	104.29
2017.1.00983.S	335	1100	73h44m01s	829h38m52s	10.71	70.05
2017.1.01085.S	11	36	05h40m49s	02h36m42s	6.55	8.94
2017.1.01214.S	38	130	33h25m51s	18h41m15s	10.20	26.05

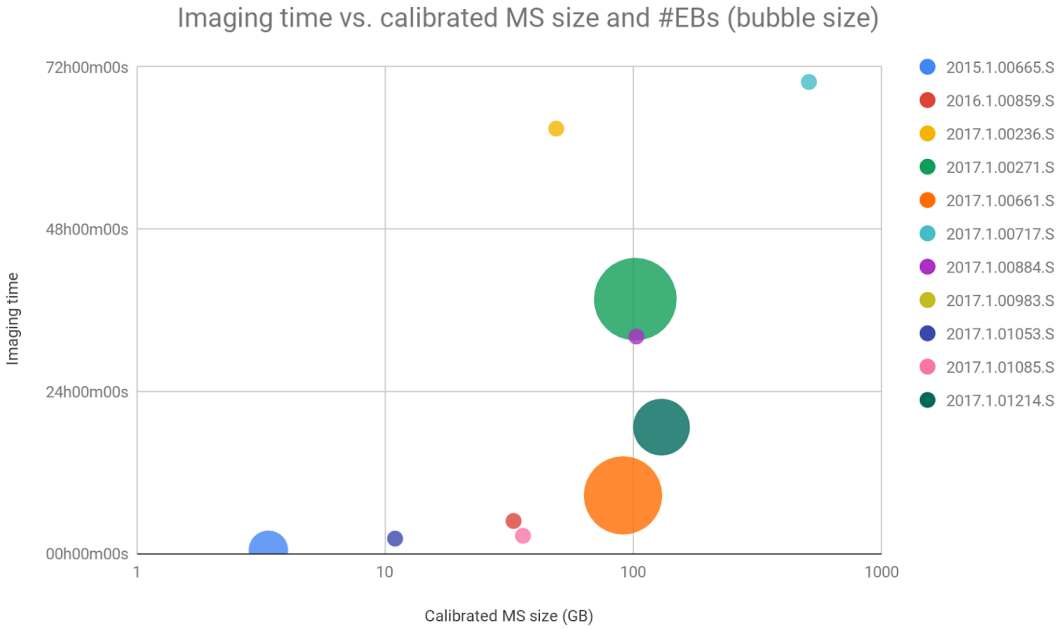
5.1 Timing

In this section, we analyze timings to understand their variations as a function of number of visibilities (MOUS or calibrated MS size) and number of Execution Blocks (EBs).

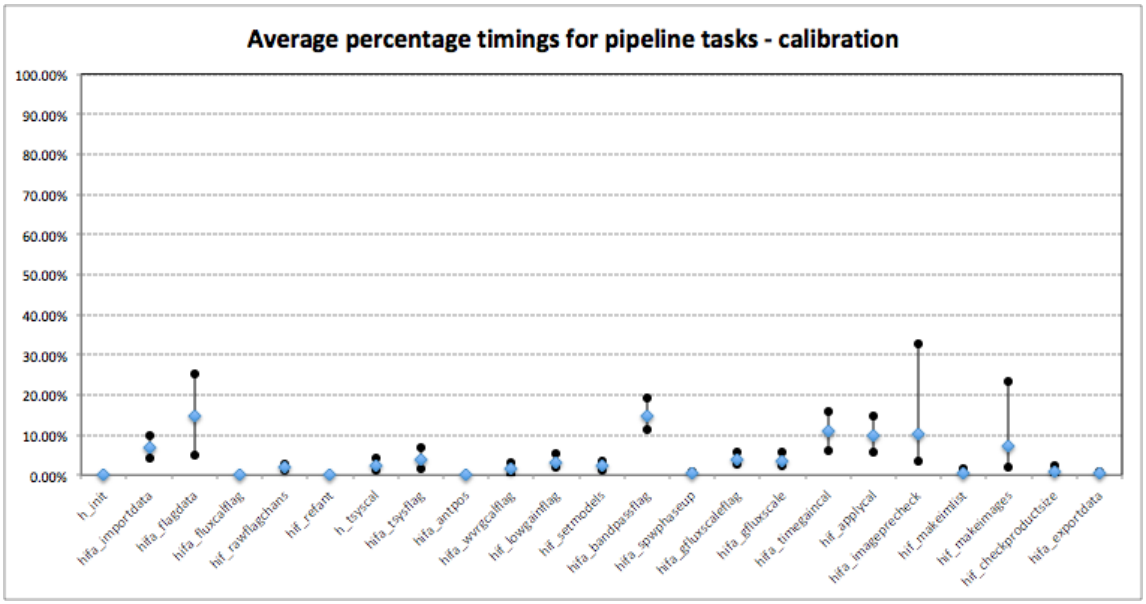
The following plot shows run times for calibration as a function of MOUS size and number of EBs. It shows that calibration timing strongly depends on the MOUS size, with a weaker relation to number of EBs.

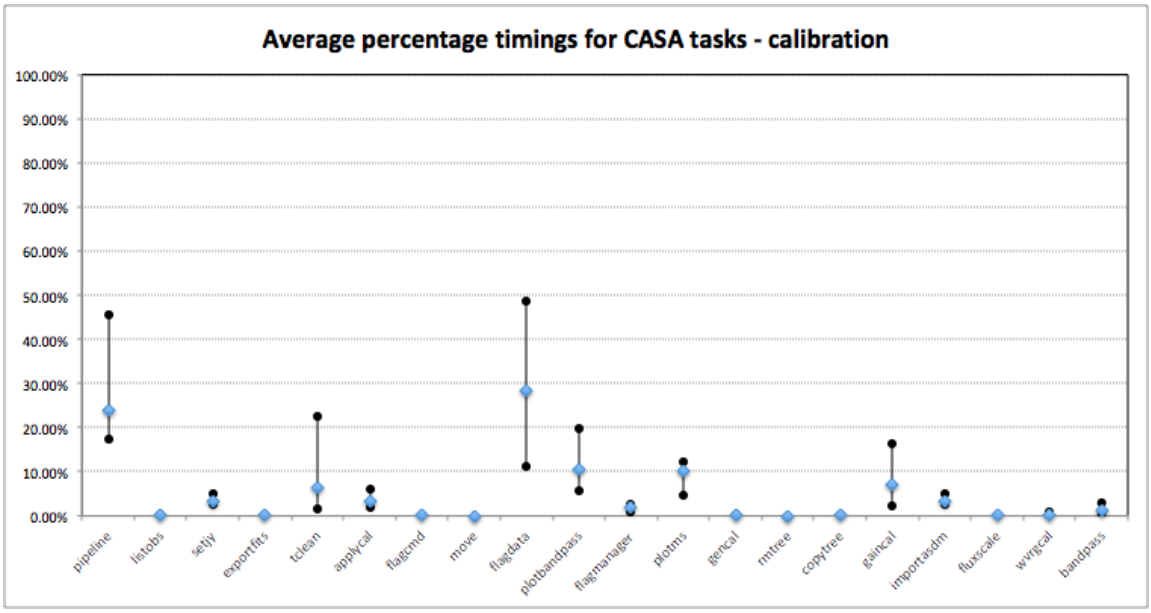


The next plot shows run times for imaging as a function of MOUS size and number of EBs. In general, the imaging time increases as the calibrated MS size increases. However, 2017.1.00236.S warrants more consideration as to what drives its imaging time, as it is not apparently MS size or number of EBs. Possible causes are chanchunks from memory limit or number of pointings.



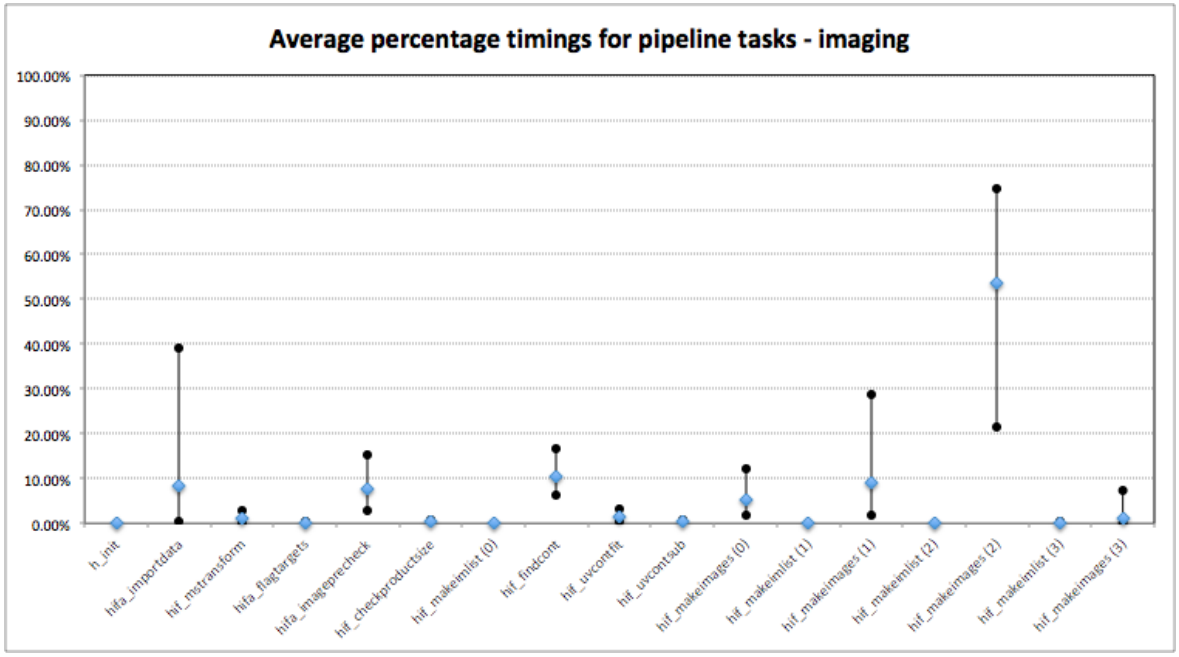
To identify which pipeline or casa task contributes to runtime variations between different datasets, we compute percentage timings for pipeline and CASA tasks to a normalized scale at which point we can compare the task contributions between those datasets. The plots below show the average, minimum and maximum percentage timings per pipeline task and per CASA task, for both the calibration and imaging pipelines (due to skewing from small sample size and under representation of larger data sets, standard deviations were not included). The plots of percentage timings per CASA task also show the aggregate pipeline percentage timing for comparison.

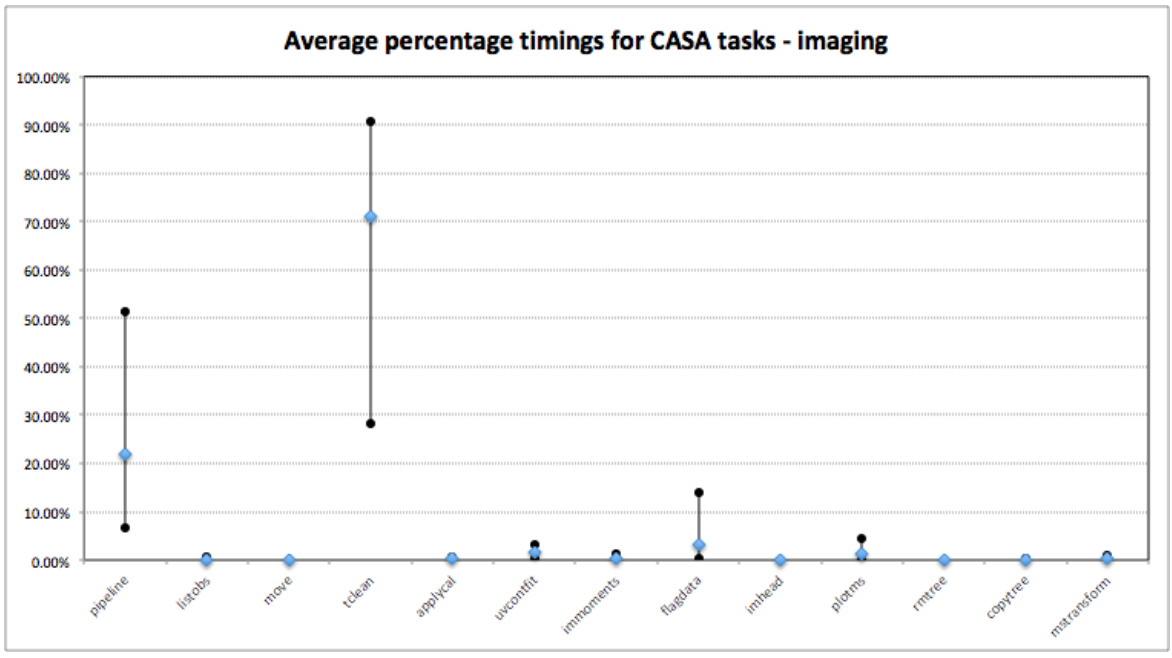




The above plots show that only a small number of tasks are significant contributors to runtime. Note that the pipeline layer itself accounts for, on average, 24% of the total runtime, with a peak of nearly 50%. The calibration pipeline tasks with the longest runtime are ‘hifa_flagdata’, ‘hifa_bandpassflag’, ‘hifa_timegaincal’, ‘hif_applycal’ and ‘hifa_imageprecheck’, while ‘flagdata’,

'plotbandpass' and 'plotms' are the longest CASA tasks. The pipeline tasks with largest variances are 'hifa_flagdata', 'hifa_imageprecheck' and 'hif_makeimages', while 'tclean', 'flagdata', 'plotbandpass' and 'gaincal' show the largest variances among CASA tasks.





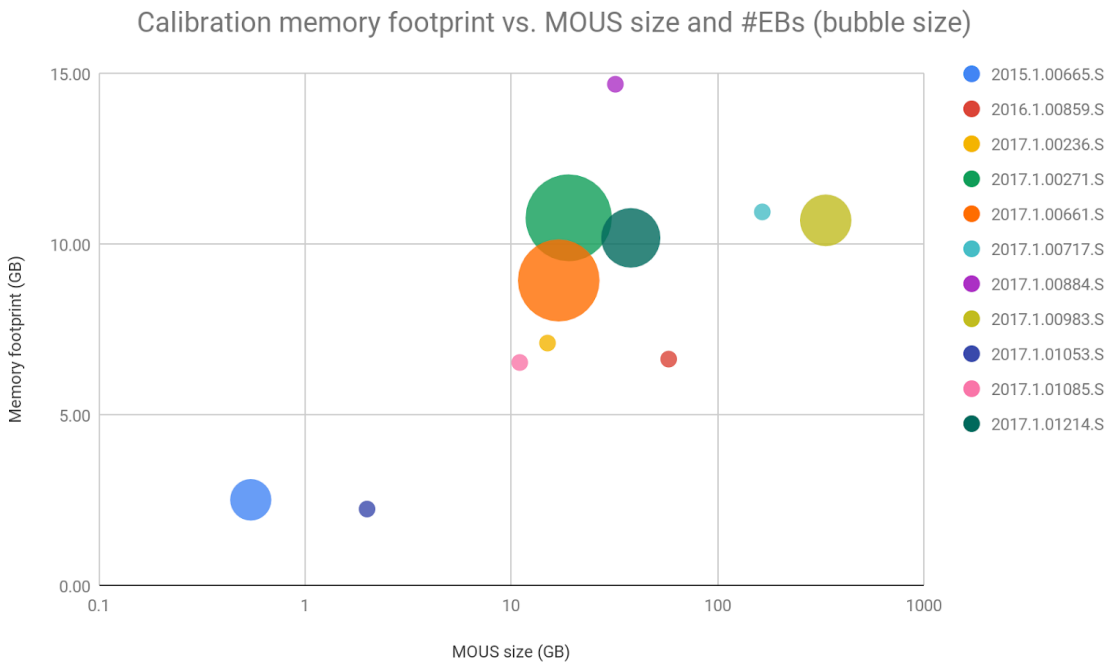
The above plot shows that the imaging case is dominated by the pipeline task 'hif_makeimages' (the third call, identified here as 'hif_makeimages (2)') and the CASA task 'tclean', which is an expected result as most of the processing in the imaging pipeline is known to come from these tasks. Note that the pipeline layer itself accounts for, on average, 22% of the total runtime, with a peak of roughly 50%. As in the calibration case, the variations in percentage timings are also not large for most of the pipeline and CASA tasks. The pipeline tasks with the largest variances are

'hif_makeimages (2)', 'hifa_importdata' and 'hif_makeimages (1)', while 'tclean' is the only CASA task that shows large variance.

The tasks identified above as the largest contributors to runtime, and those that show largest variances in percentage timings warrant further investigation.

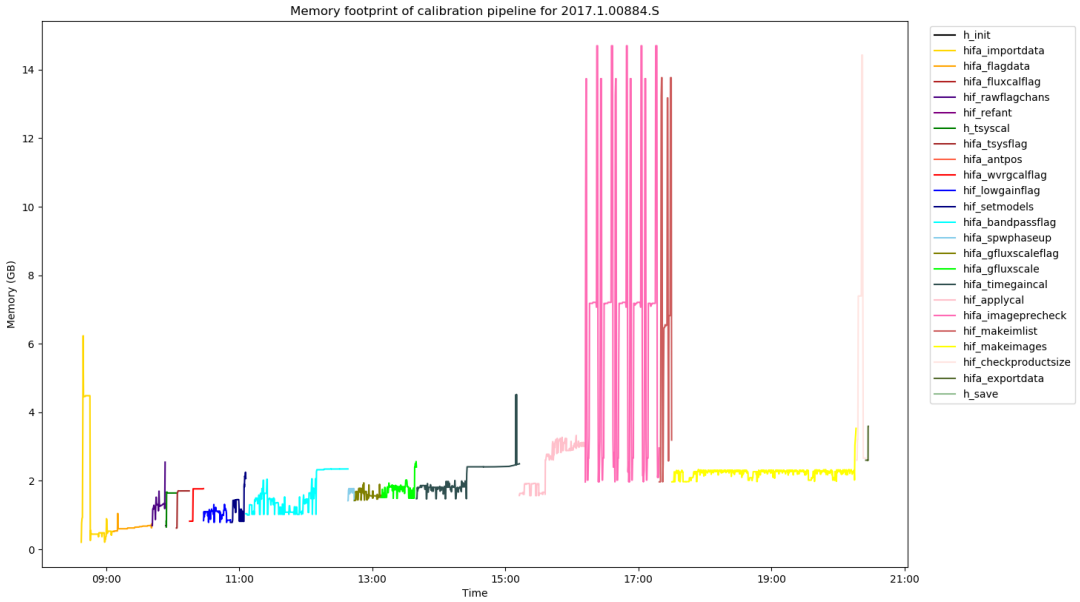
5.2 Memory footprint

The memory footprint of the calibration pipeline is shown in the plot below as a function of the ASDM size in GB and the number of EBs (bubble sizes).



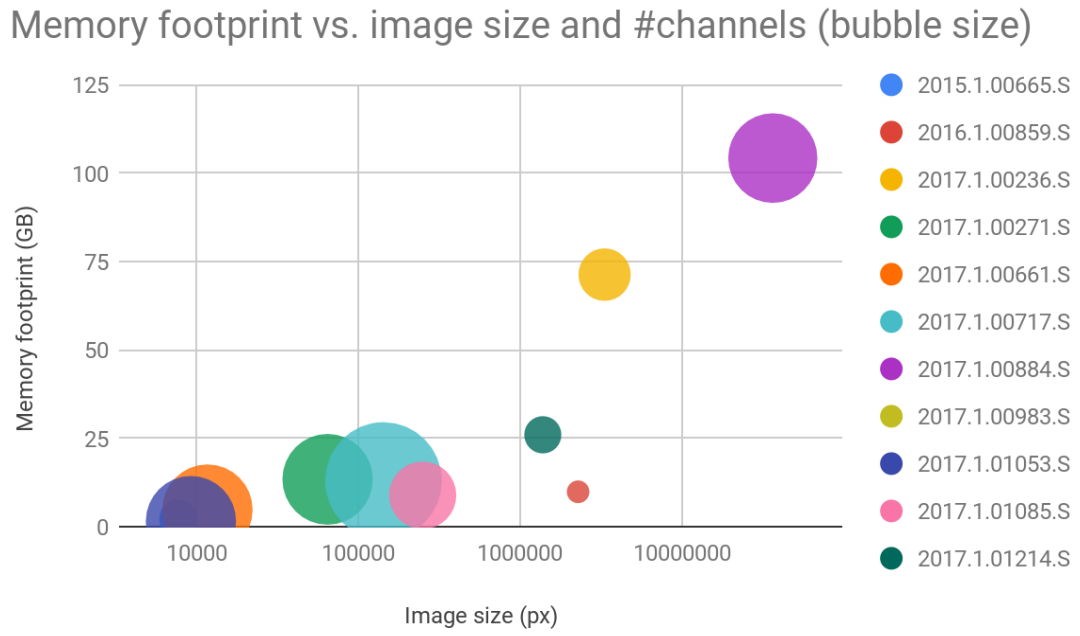
The above plot indicates that the memory footprint is increasing as the MOUS size increases, while we don't see a dependency of the memory footprint with the number of EBs. We also see what appears to be a clustering around 10 GB, with two outliers at larger MOUS sizes - 2017.1.00884.S with a memory footprint close to 15 GB, and 2016.1.00859.S around 6 GB.

To have a deeper look at the largest outlier, we plotted the memory usage as a function of time for project 2017.1.00884.S as shown in the figure below.



The plots show that the peaks in memory usage are associated with pipeline tasks ‘hifa_imageprecheck’, ‘hif_makeimlist’ and ‘hif_checkproductsize’, which do not call CASA tasks.

The memory footprint of the imaging pipeline is expected to be strongly dependent on image (cube) size, as this is what determines the amount of memory that has to be allocated to do all the processing for making images. The plot below shows the memory footprint of the imaging pipeline as a function of image size (total number of pixels), and the number of channels is represented as the bubble size.



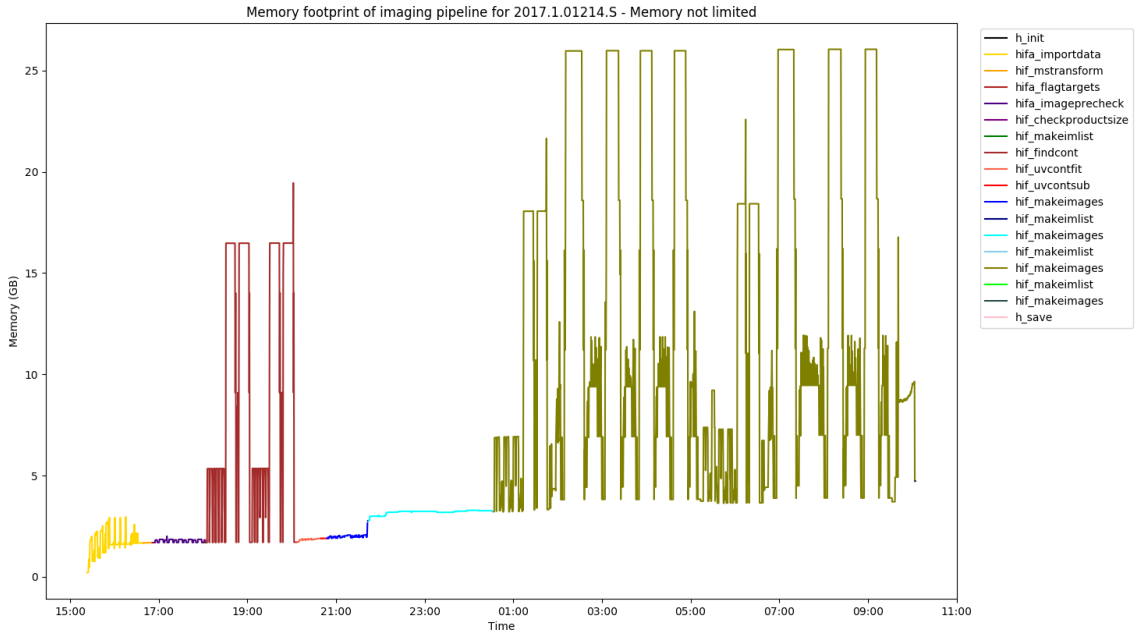
The plot shows that the per plane image size dominates over the number of channels, as we see large number of channels associated with small memory footprints, while larger memory footprints are only associated with larger image size. Three data sets, 2017.1.00236.S, 2017.1.0884.S, and 2017.1.00983.S, automatically reduced their memory footprint due to system limitations by selecting smaller channel ranges to store at any one time via the 'chanchunk' mechanism.

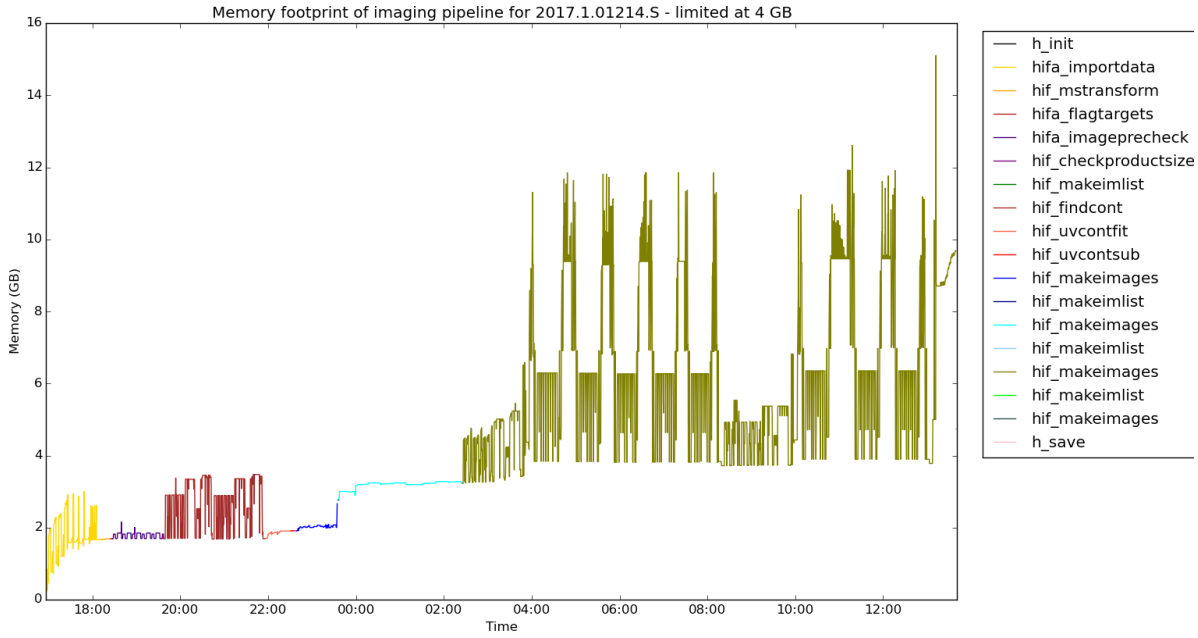
The imaging pipeline memory requirement can be constrained via the ~/.casarc parameter 'system.resources.memory', which tells CASA how much memory is available for imaging. Tests were run to measure the actual memory usage and pipeline runtime vs. various parameter settings. The results for five test cases constrained to different memory limits and the unconstrained case are shown in the table below.

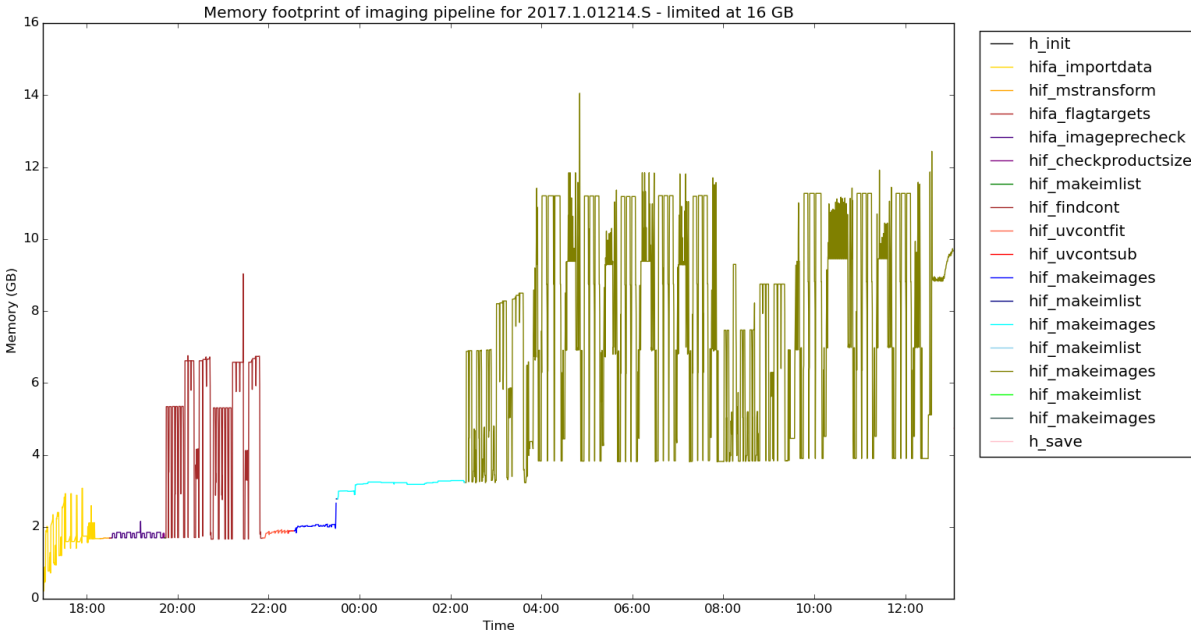
Memory limit (GB)	Run time	Memory footprint (GB)
1	24:37:08	14.46
2	21:38:05	15.57
4	20:44:16	15.1
8	20:02:06	14.29
16	20:02:10	14.05
not limited	18:41:15	26.05

The table above shows the rc defined memory limit does not wholly constrain the memory footprint, although memory usage clearly differs from the unconstrained case. In addition the more constrained memory cases suffer slightly longer run times.

The external monitoring task that records time series data also receives a message from CASA informing the current running task and stores that information. To identify the cause of excess memory usage by the defined limit we make use of the complete time series to show the memory footprint as a function of task. The following plots show the unconstrained, 4 GB and 16 GB limited cases, respectively:


Memory footprint of imaging pipeline for 2017.1.01214.S - Memory not limited
Memory (GB)
Time
h_init
hifa_importdata
hif_mstransform
hifa_flagtargets
hifa_imageprecheck
hif_checkproductsize
hif_makeimlist
hif_findcont
hif_uvcontfit
hif_uvcontsub
hif_makeimages
hif_makeimlist
hif_makeimages
hif_makeimlist
hif_makeimages
hif_makeimlist
hif_makeimages
h_save


Memory footprint of imaging pipeline for 2017.1.01214.S - limited at 4 GB
Memory (GB)
Time
h_init
hifa_importdata
hif_mstransform
hifa_flagtargets
hifa_imageprecheck
hif_checkproductsize
hif_makeimlist
hif_findcont
hif_uvcontfit
hif_uvcontsub
hif_makeimages
hif_makeimlist
hif_makeimages
hif_makeimlist
hif_makeimages
hif_makeimlist
hif_makeimages
h_save

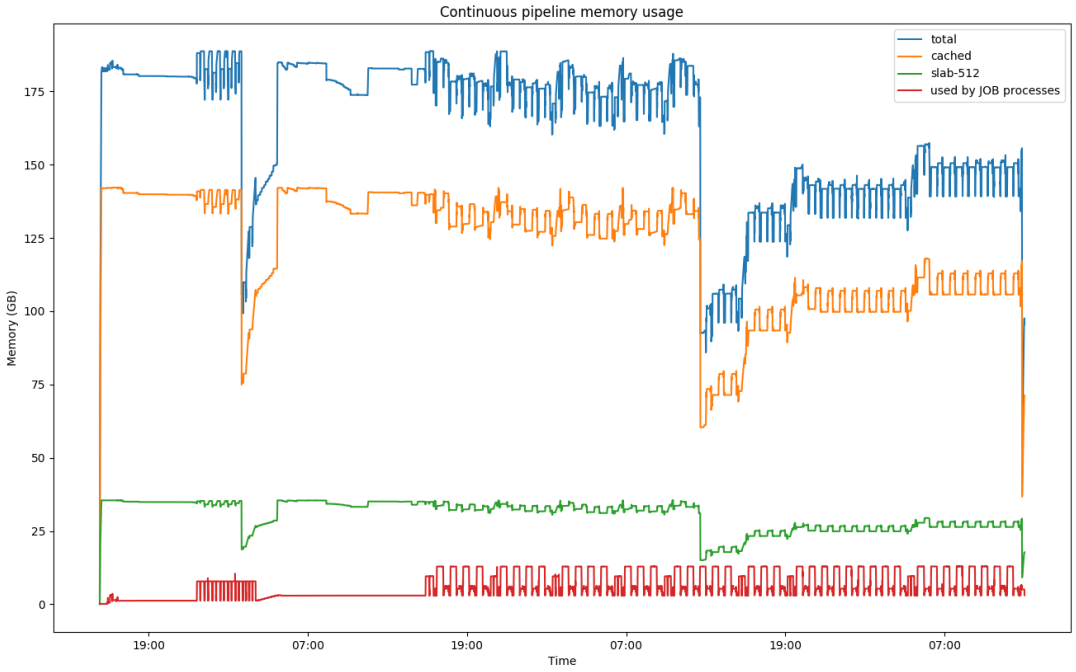


Analysis of the plots shows that they only differ significantly when 'hif_findcont' and the third instance of 'hif_makeimages' are running. For these two tasks, we see that the shapes of the memory footprint time series change with the amount of memory available to CASA. We also notice that sections of the time series show a well-defined limit in the constrained cases, that this limit is larger for larger values of 'system.resources.memory', and that the corresponding intervals in the unconstrained case are where the peak memory usage is for that case. All this indicates that only a particular code, or call, is affected by the memory limit. This is consistent with the knowledge that these tasks repeatedly call 'tclean', and also with the expectation that only 'tclean' should be affected by 'system.resources.memory'.

A more in depth analysis of the per-task memory footprints for both the imaging and calibration pipelines will be reported and analyzed in a future report.

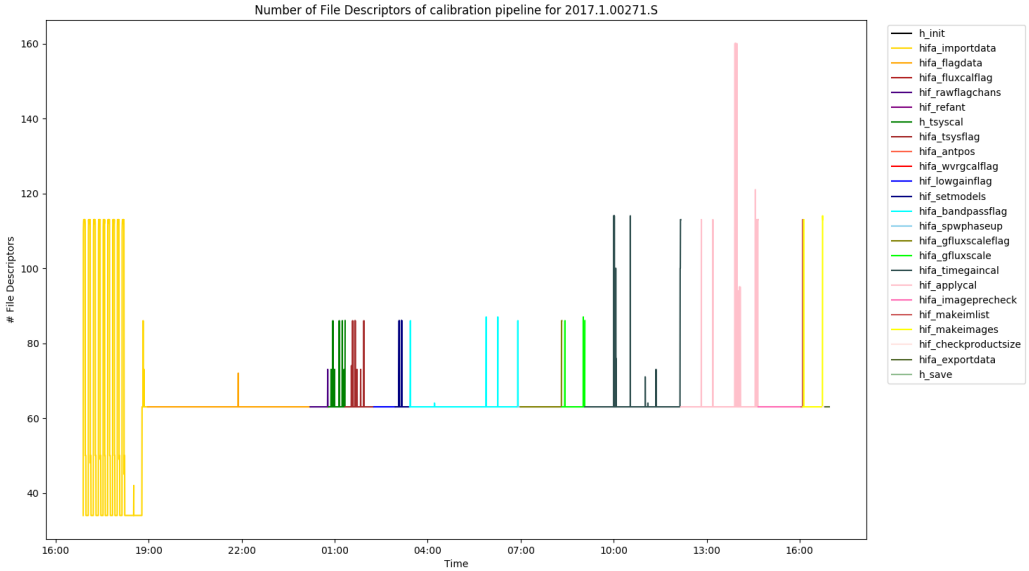
5.3 Memory load on a node

The memory load on cluster nodes is continuously monitored by ganglia that provides plots of that load (and other performance indicators) over time. However, the time resolution of ganglia plots degrades for longer runs, unless the plots are captured during the run. To ensure we have information on memory load with the same time resolution for all data sets irrespective of run time, we collect memory load information at fixed intervals during pipeline runs. The following plot is an example of memory load on a node as a function of time over a pipeline with an approximate execution time of 2.5 days. With such a duration, only the weekly plot in ganglia would show the entire run, but with a much poorer time resolution than we show here.



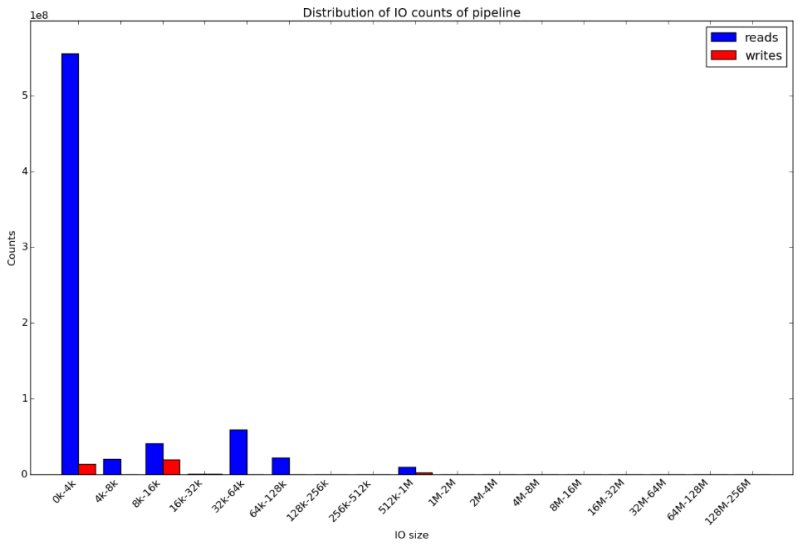
5.4 Number of file descriptors

The plot below is an example showing how the number of file descriptors change over a calibration pipeline run as a function of time and pipeline task. This case is particularly interesting as this data set has 9 EBs, and this clearly shows in the plot as the 9 peaks associated with hif_importdata.

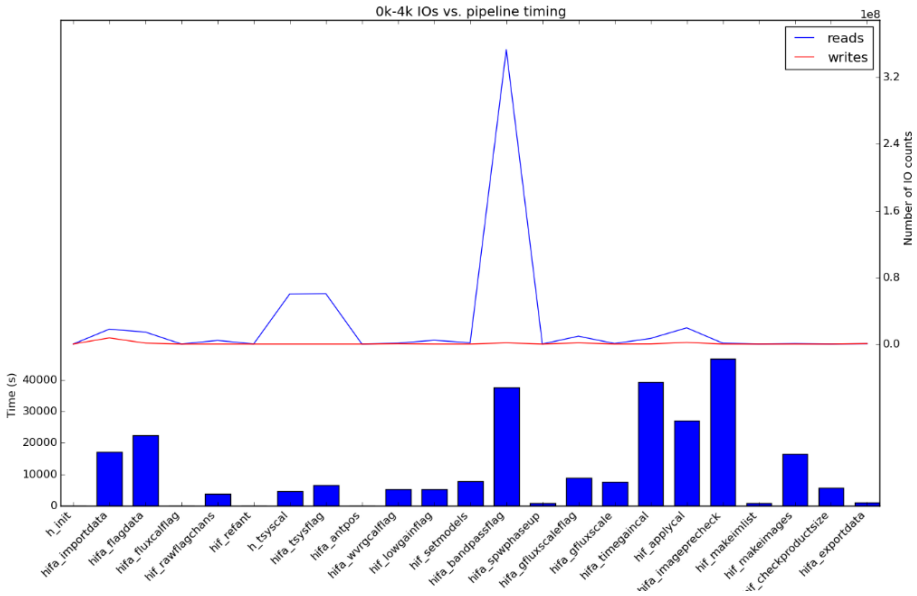


5.5 IO file statistics on lustre

Throughout the pipeline execution counts of read and write requests by size in kilobytes (e.g. 0k-4k, 4k-8k, 8k-16k etc) are recorded from the Lustre filesystem's internal statistics. Note this data recording is unique to Lustre and cannot be easily recorded on executions versus local disk. The following plot shows the distribution of IO requests by size for the pipeline calibration of 2017.1.00983.S - the largest dataset of our sample.



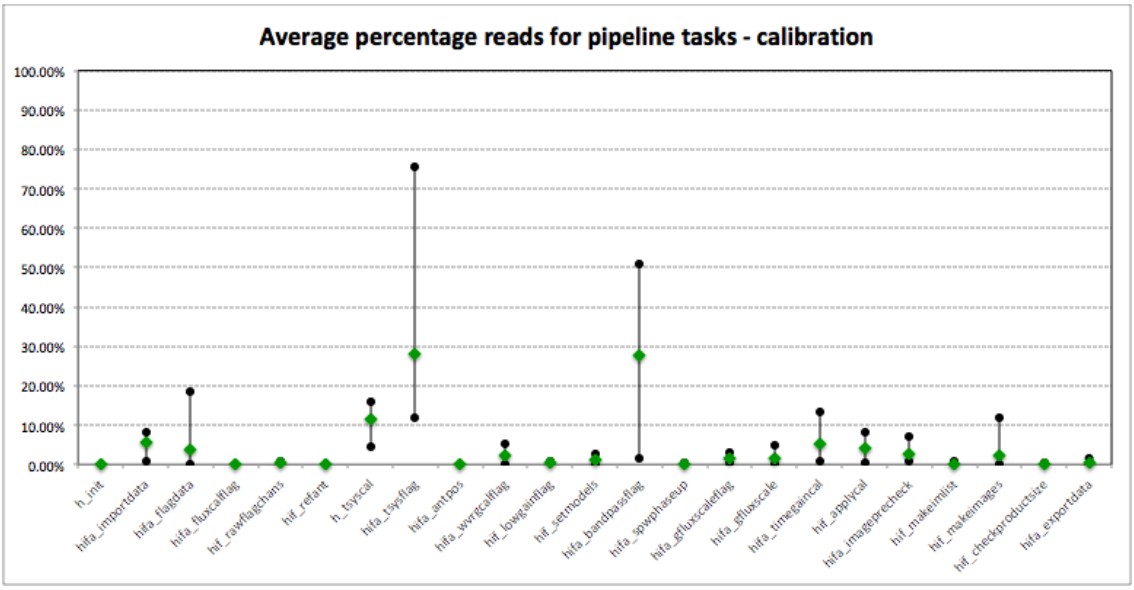
The IO distribution is dominated by small (0k-4k) files, so we look at the distribution of IOs per pipeline task for that IO size range, as shown in the following plot.

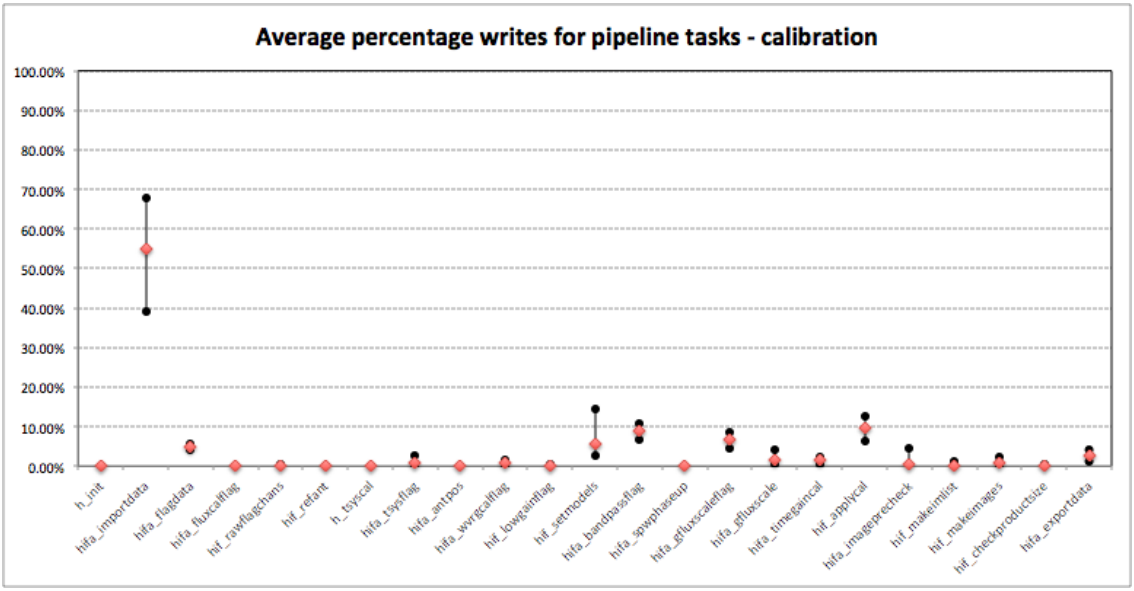


The plot shows that the large number of 0k-4k IOs is mostly associated with 'hifa_bandpassflag' which is also the third largest contributor to calibration run time. Consideration should be made to

examining the apparent unique IO pattern in 'hifa_bandpassflag' for this dataset and possibly coalescing reads to improve IO performance.

We have used the IO file statistics for each pipeline execution to compute each pipeline task and CASA task's percentage contribution to the total read and write requests across all datasets. The following eight plots show the minimum, mean and maximum percentage reads and writes for both the calibration and imaging pipelines displayed per pipeline task and per casa task.

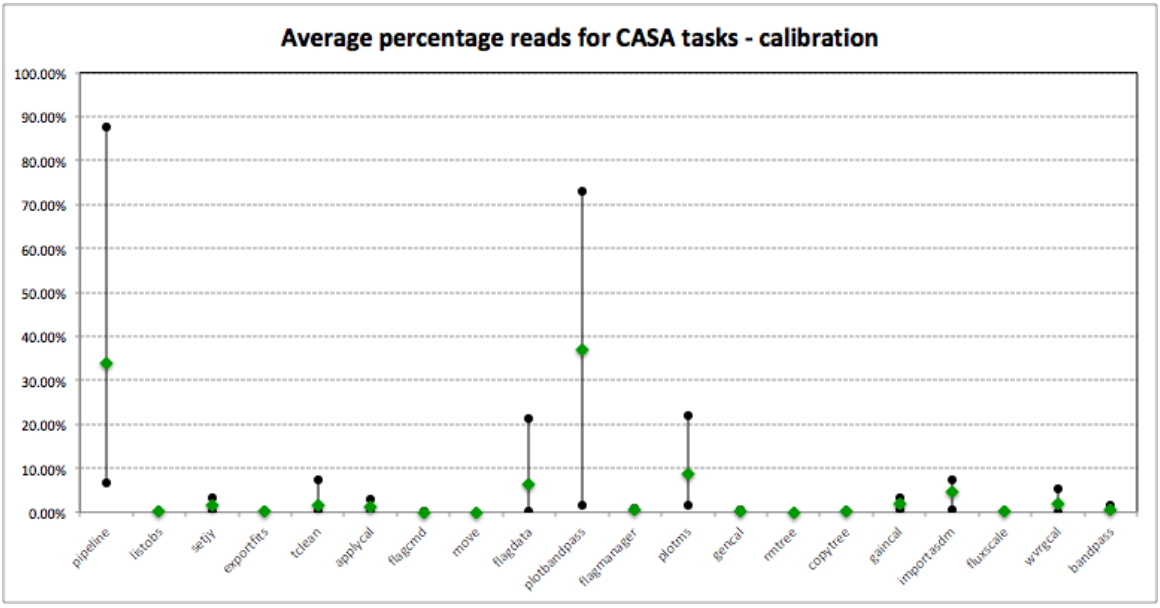


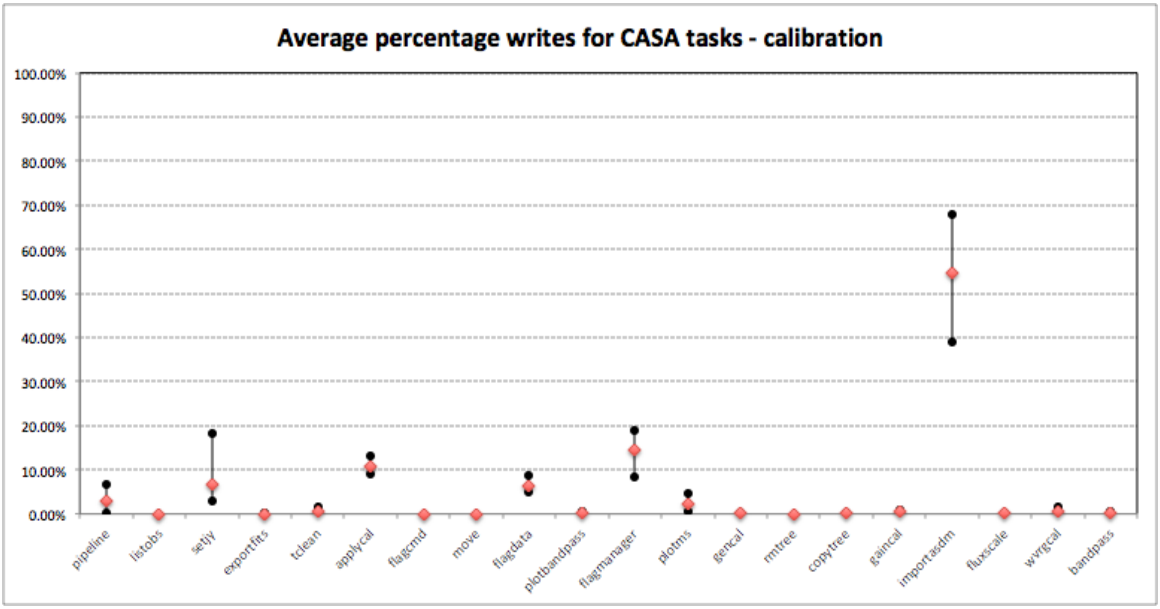


In the above plots, two pipeline tasks, 'hifa_tsysflag' and 'hifa_bandpassflag', contribute most significantly to total reads of the calibration pipeline, and show the largest variances. This is likely indicative of numerous small reads and warrants further examination.

Calibration pipeline writes are dominated by 'hifa_importdata' which is expected as it makes a copy of the data to the working directory. The variance observed in importdata is likely due to differences in data structure and size.

The next two plots show the distribution of reads and writes per CASA task plus the python layer of the calibration pipeline.

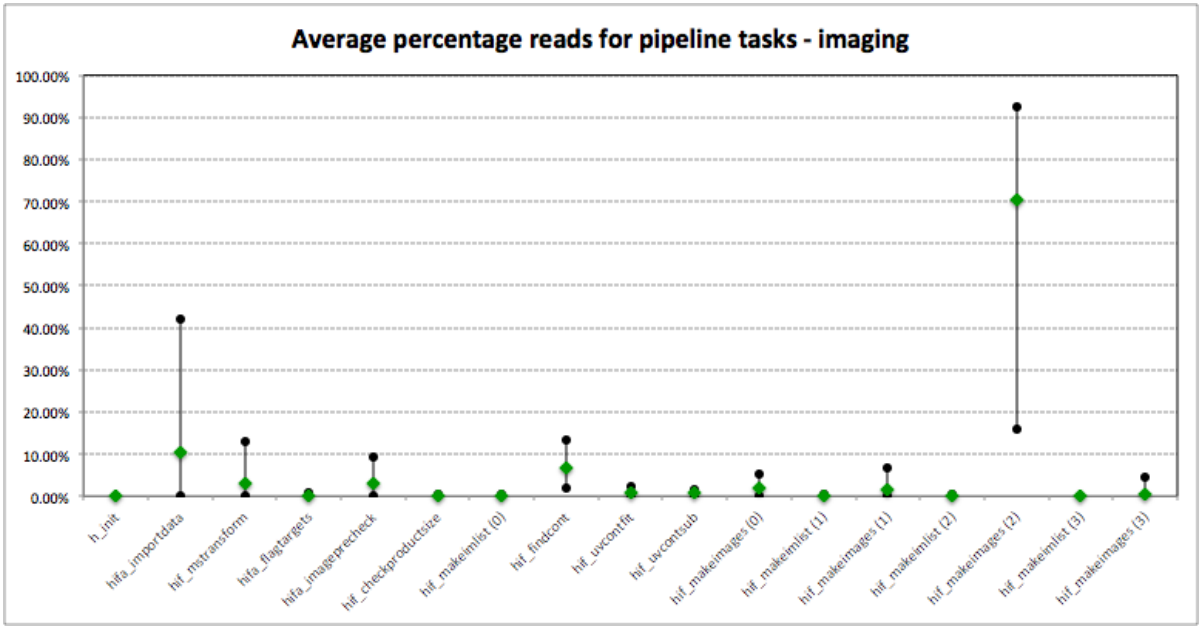


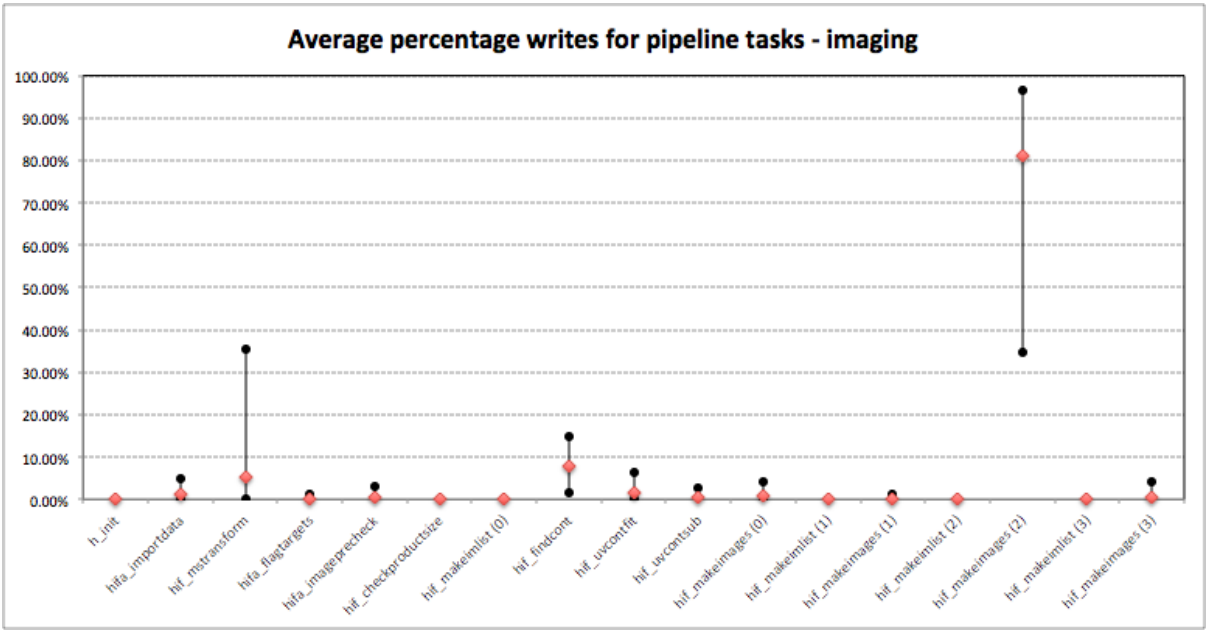


The python layer's contribution to total reads in the calibration pipeline is comparable to that of 'plotbandpass', the largest contributor among CASA tasks. Further examination and coordination with the CASA pipeline group is necessary to understand why this is the case and if it is to be expected.

The write results are to be expected as ‘importasdm’ should be the dominant write case with some additional writes from ‘setjy’, ‘applycal’ and ‘flagmanager’.

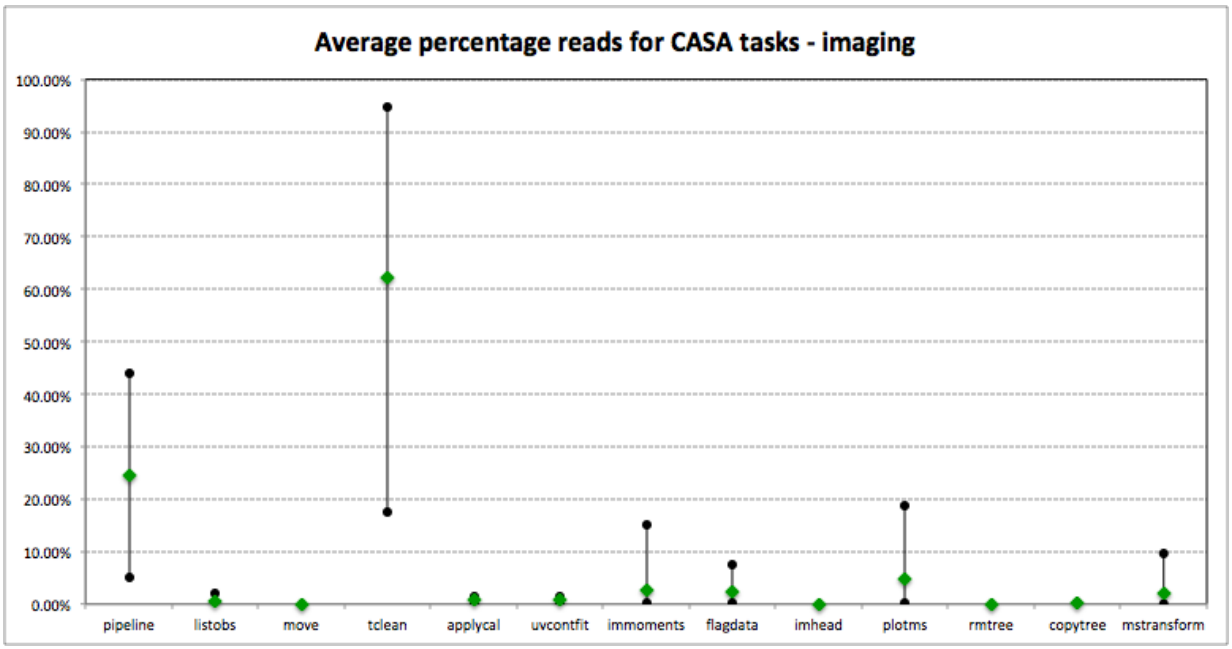
The next two plots show percentage reads and writes for pipeline tasks within the imaging pipeline.

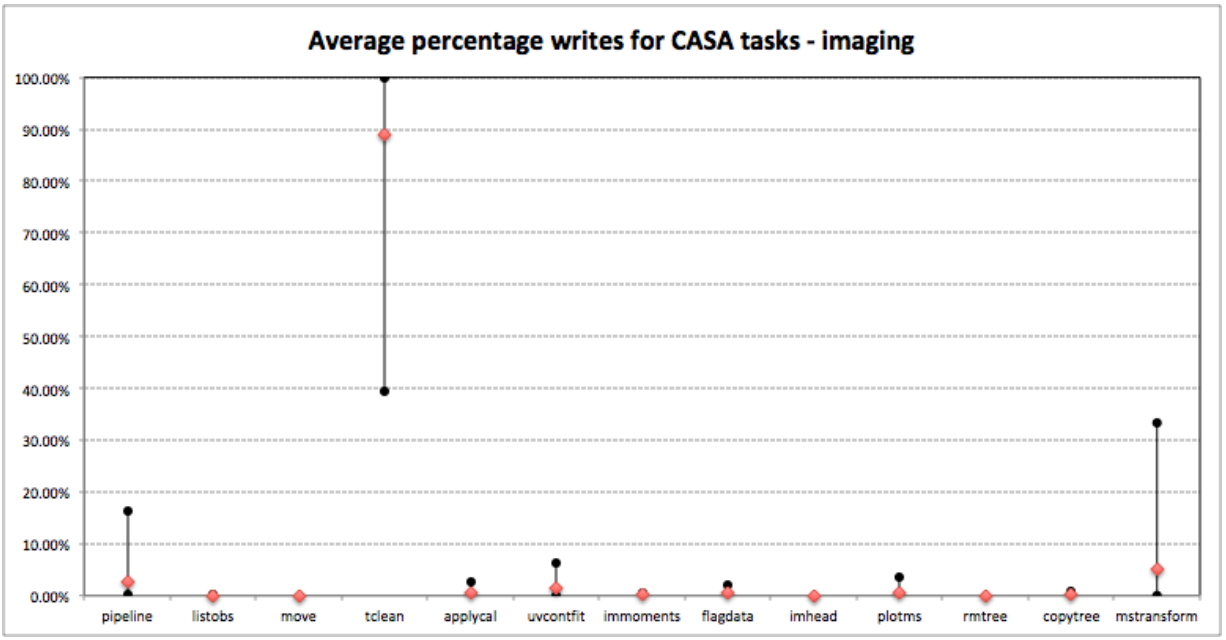




The pipeline task ‘hif_makeimages (2)’ dominates the reads and writes in the imaging pipeline. Further investigation is needed as to whether this is expected behavior or indicative of an issue. The read and write contributions from ‘hifa_importdata’ and ‘hif_mstransform’ respectively is to be expected.

The last two plots in this subsection show the distribution of reads and writes per CASA task plus the python layer of the imaging pipeline.



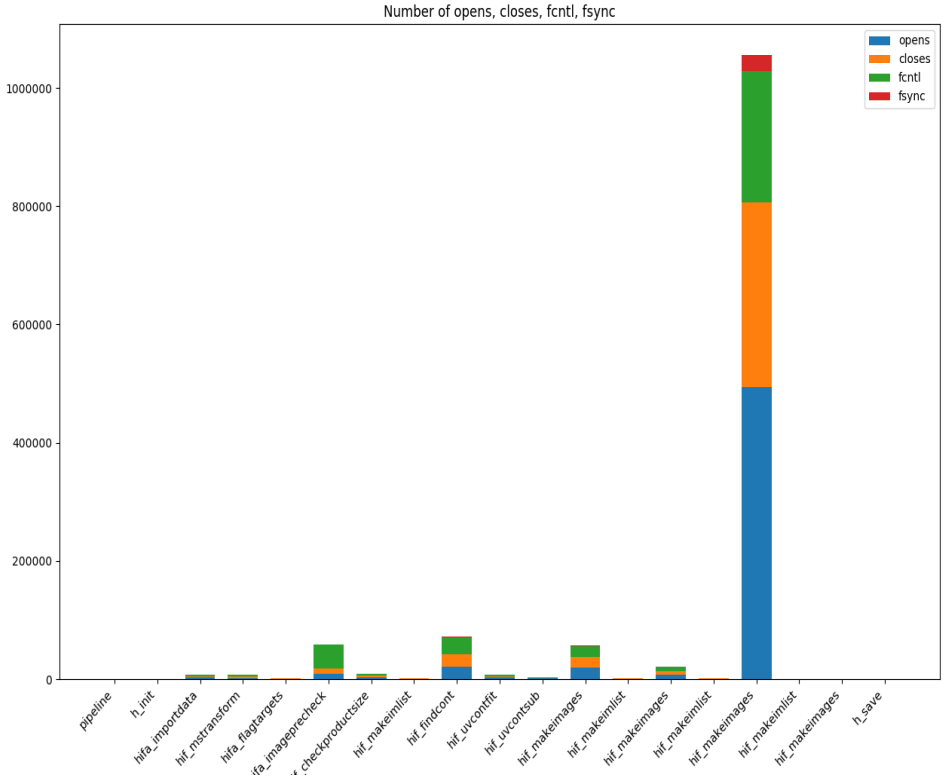


Again the python layer of the pipeline contributes significantly to the total reads throughout the pipeline execution. Further examination and coordination with the CASA pipeline group is necessary to understand this behavior. The contribution from 'tclean' to reads and writes as seen here is from the repeated calls to 'hif_makeimages' and is to be expected.

5.6 Number of system calls

The external monitoring task allows for collection of system calls per pipeline and CASA task. We are currently modifying the profiling framework to also record the total timings for each type of call, enabling a deeper and more precise analysis of the impact of system calls in the execution timings.

The plot below shows an example of number of system calls stacked by type of calls for an imaging pipeline run.



The cause of the large number of open() and close() calls is not immediately apparent and warrant further examination. Precise timings are necessary to demonstrate whether those calls contribute significantly to run time as well as to show that the small number of fsync() calls can, in certain circumstances, dominate the total pipeline runtime.

6 CONCLUSIONS AND FUTURE STEPS

This is the first report on ALMA pipeline benchmarking and profiling. We intend to issue periodic reports as the work progresses and goes deeper in the analysis of the performance of the ALMA pipeline. In this initial stage of the work, we have shown what kind of information can be obtained through use of this framework and methodology. Initial timing and memory footprint data were reported and analyzed, showing particular features of CASA and pipeline performance where it

may be interesting to conduct a deeper analysis, for example the imaging pipeline memory footprint and execution time for 2017.1.00236.S.

Statistical analysis of the timings reduced to a common basis shows that most of the tasks contribution to runtime is relatively small and little impacted by the differences in size and shape of our data sample. The largest contributors to runtime and the tasks for which the timings are more sensitive to different data size and shape were identified and are currently under investigation, to be described in detail in the upcoming report(s).

The imaging memory footprints were shown to behave as expected, being mostly a function of the image size and number of channels. The effect of setting 'system.resources.memory' in the ~/.casarc file to limit the amount of memory used for imaging was also investigated and shown to be honored but may not work entirely as expected. Further examination of runtime limits, particularly in the context of parallel executions will be a major focus of further profiling. The calibration memory footprints are currently under analysis and will also be part of subsequent reports, as well as detailed memory data for CASA and pipeline tasks.

A large amount of data is produced by the profiling framework and some has yet to be analyzed for the pipeline runs reported in this document. The analysis presented here will be expanded through the use of information from data for IO shapes (number of files per IO size ranges), number of file descriptors and system calls, as well as overall node information on cpu and memory usage and data rates.

In addition to more detailed analysis of identified elements, we also plan to run the following tests:

- All the tests reported here (possibly a representative subset of the tests) vs 2, 4, 8 and 16 way single node and broader multi-node parallelization via both MPI and OpenMP and as a function of casarc memory limiting parameters
- A selected subset of tests picked from serial runs with interesting I/O characteristics will be run on different I/O subsystems (Lustre, HDD, SSD, RAID & NVMe devices)
- A selected subset of tests will be run on a higher clocked CPU systems (depends on availability)
- Repeat tests with suggested data sets for better parameter coverage, particularly ones at the large end of their representative scale (e.g. number of pointings, number of channels).

We recommend that the highest priority tests be the parallelization vs memory constraint tests to better support upcoming pipeline operations but welcome suggestions as to which tests would be most beneficial to operations.

APPENDIX I

The following table shows the present coverage of the parameter space and the corresponding data references. Project names were pulled from the (ALMA cycle 6) CASA pipeline test coverage list, names in blue text also exist in the NAASC internal benchmarking list[1]. Project names with yellow fill are 7m array and green is 12m array.

Scale	Size (GB)	# EBs	# Pointings	# Targets	# FDMs
Small	2015.1.00665.S	2015.1.00665.S	2015.1.00665.S	2015.1.00665.S	2015.1.00665.S
	2016.1.00859.S	2016.1.00859.S	2016.1.00859.S	2016.1.00859.S	2016.1.00859.S
		2017.1.00236.S	2017.1.00236.S	2017.1.00236.S	
				2017.1.00271.S	
			2017.1.00661.S	2017.1.00661.S	
		2017.1.00717.S	2017.1.00717.S	2017.1.00717.S	
		2017.1.00855.S	2017.1.00855.S	2017.1.00855.S	2017.1.00855.S
		2017.1.00884.S	2017.1.00884.S	2017.1.00884.S	2017.1.00884.S
				2017.1.00983.S	
	2017.1.00984.S[2]			2017.1.00984.S[2]	2017.1.00984.S[2]
	2017.1.01053.S	2017.1.01053.S	2017.1.01053.S	2017.1.01053.S	
		2017.1.01085.S	2017.1.01085.S	2017.1.01085.S	
			2017.1.01214.S	2017.1.01214.S	2017.1.01214.S
Medium	2017.1.00019.S				
	2017.1.00236.S				2017.1.00236.S
	2017.1.00271.S				
	2017.1.00661.S				2017.1.00661.S
	2017.1.00855.S				
	2017.1.00884.S				
		2017.1.00983.S			2017.1.00983.S
		2017.1.00984.S[2]	2017.1.00984.S[2]		
	2017.1.01085.S				2017.1.01085.S
	2017.1.01214.S	2017.1.01214.S			
Large		2017.1.00271.S	2017.1.00271.S		
		2017.1.00661.S			
	2017.1.00717.S				2017.1.00717.S
	2017.1.00983.S		2017.1.00983.S		

[1] https://safe.nrao.edu/wiki/bin/view/HPC/ALMABenchmarking

[2] This dataset was abandoned as the execution of the calibration pipeline fails with the baseline version of CASA that was used for all the data processing reported here (casa-release-5.4.0-68).

The ranges corresponding to ‘small’, ‘medium’ and ‘large’ for each parameter are the following:

	sample ranges			Range from identified datasets	
	Small	**Medium**	**Large**	**Min**	**Max**
Size (GB)	1-10	11-100	101-1000	2	157
# EBs	1-2	3-4	5-8	1	9
# Pointings	1-5	6-25	26-125	1	95
# Targets	1-5	6-25	26-125	1	2
# FDMs	1-2	3-4	5-8	1	8

As shown in the table, we have reasonable coverage of the parameter space, except for the number of targets, for which we still need datasets that would cover the ‘medium’ and ‘large’ ranges (as well as define the appropriate ranges for that axis).