Parallelisation in CASA 4.0

Report on the work by the ESO HPC group in the 4.0 development cycle (svn r495)

S.Castro, J.A.Gonzalez, D.Petry (lead)
(ESO)

17 October 2012

Contents

1 Introduction

In the CASA 4.0 development cycle, the ESO HPC group was working on debugging and extending the trivial parallelisation of non-imaging tasks in CASA to the point where an entire ALMA data analysis can be carried out on multi-MSs (MMSs). The main goal was to achieve a situation where the data can be partitioned directly after import and then kept an MMS over all calibration steps without having to be repartitioned such that the final calibrated dataset can be fed directly as an MMS into parallel clean (pclean).

This goal was achieved in CASA 4.0: An entire ALMA data analysis can now be carried out using parallelised tasks as demonstrated by the script alma-m100-analysis-hpc-regression.py which is now part of the CASA regressions suite. Many low-level changes were required in the MMS creation and the CASA cluster infrastructure (see section 2 and 3) on the one hand and some in the casacore/ms, the casacore/tables, the code/synthesis, and the code/air_casawvr modules on the other hand.

In addition to working towards the main goal, we put emphasis on proper unit testing for each affected CASA task. A new part of the unit test infrastructure was created which exercises tasks on data in MMS format (see section 9). For some tasks unit tests had to be created from scratch.

This writeup describes the results of our work going along the individual items roughly in the order of the data analysis stream. Essentially all related CASA JIRA tickets are gathered under the two umbrella tickets CAS-4106 and CAS-4372.

The tasks which have been tested to work with MMSs are

- `applycal`
- `bandpass`
- `clean`
- `concat` (results in an output MS, not an MMS)
- `fixplanets`
- `flagmanager`
- `fluxscale`
- `gaincal`
- `gencal`
- `listobs`
- `listpartition`
- `listvis`
- `listhistory`
- `partition` (repartitioning of an MMS is also possible)
- `pclean`
- `plotms`
- `setjy`
- `split` (results in an output MS by default, in an MMS if parameter keepmms=True)
- `flagdata`

- `uvcontsub` (results in an output MMS)
- `virtualconcat` (results in an output MMS)
- `vishead`
- `wvrgcal`

Other tasks *may* also work with MMSs, but this wasn't confirmed with dedicated tests, yet.

Among the tasks mentioned above, the following tasks will work in a parallelised way on MMSs to speed up processing or at least avoid repartitioning:

- `applycal`
- `partition` (repartitioning of an MMS is also possible)
- `setjy` (when parameter usescratch=True)
- `split` (when parameter keepmms=True)
- `flagdata`
- `uvcontsub`

The parallelisation of these tasks still has to be regarded as a prototype which has the required functionality but is not yet optimized. Optimization will have to happen in a second iteration, presumably in development cycle 4.1. Starting points for optimization are the handling of the MS subtables and possible overhead in the operation of simple_cluster.

Preliminary performance tests on a 12-processor workstation with a 1.2 TB solid state disk (SSD) show improvements in the parallelised tasks over a non-parallelised execution on the same system (i.e. also using a SSD!). These improvements depend on cluster configuration and partitioning setup. This is discussed in section 10.

A significant improvement over the non-MMS case can be seen in `applycal`, time averaging in `split`, and when comparing `virtualconcat` with `concat`.

As a next step, a detailed evaluation of the performance of CASA 4.0 on powerful systems with parallel file systems such as the Socorro cluster or the future EU ARC cluster (to become available in November 2012) will have to be carried out.

Suggestions for further work are enumerated in section 11.

2 Work on the MMS structure

The original MMS as it came out of, e.g. partition, was not yet mature. It was spread over two parallel directories and could not easily be moved, renamed, or copied. The voluminous POINTING table was stored in each subMS leading to an MMS data volume much larger than the equivalent MS.

New features in CASA 4.0:

MMS can now be handled like a normal directory: With changes both in casacore (with help by Ger van Diepen) and on the Python level (partitionhelper.py and task_partition.py), the MMS can now be handled like a normal ("monolithic") MS. It can be moved and renamed like any directory. And tasks which are not MMS-aware can process it like a monolithic MS.

Single copy of POINTING and SYSCAL: In order to reduce the volume of the MMS, the POINTING and SYSCAL tables (which are read-only in all use cases and identical for all subMSs) are stored only with the master-subMS and linked into the other subMSs.

Create a .flagversions for the MMS: In a normal data analysis stream, data is imported using importasdm which creates a backup of the flags in a directory parallel to the MS with extension .flagversions. By running partition after importasdm this backup got lost and could not be used later in other parts of the data analysis stream. We introduced in the partition task the capability to create a .flagversions directory for the output MMS which saves the flags present in the MMS at its creation.

3 Work on the cluster infrastructure

We have carried out a complete overhaul in the cluster framework to improve aspects such as configuration, deployment, error handling, communication, monitoring, robustness and security. Additionally, the new version of the cluster infrastructure comes with a set of unit tests that exercises all the aspects above mentioned and proves its compatibility with Mac OSX 10.6/10.7 64bits and RHEL 5.3/5.6 32b/64b.

New cluster configuration file: Defined and implemented a new format for the parallel processing cluster configuration file that allows the user to specify the resources needed per engine together with the percentage of idle resources (RAM and CPU) to be used in each node of the cluster in order to deploy the maximum number of engines possible according to these constrains and set the internal parallelisation variables within each engine (e.g.: OpenMP number of threads) accordingly.

Exceptions and errors handling: Implemented an exception handling mechanism to control the exceptions that occur at the engine level and show them at the controller CASA instance level. The ERROR/SEVERE messages are now analyzed and shown at the controller CASA instance level only when they are relevant to the global execution.

Localhost cluster: The cluster infrastructure can now easily be deployed on localhost without resorting to ssh. Therefore it is not necessary to set a password free access anymore. Additionally, there is a fallback mode when only one engine can be deployed, that automatically bypasses all the cluster infrastructure, and performs the execution locally and sequentially trough the list of subMSs.

Controller/Engine communication: Return variables from the tasks executed in the remote engines are now either consolidated into one single return variable which is forwarded to the controller CASA instance level or grouped together in a subMS/return variable map for further analysis of the results. Additionally it is now also possible to override the input parameters for each task in order to customize the run for each subMS. Finally the communication via ssh between the engines and the controller instances is now more robust by means of proper redirection of stdout and stderr.

Cluster monitoring service: There is a new monitoring service that analyzes the status and resources used per engine and in total per node included in the cluster. The resources monitored are CPU (percentage), RAM (total in MB), I/O (total RW and rate per second) together with some other parameters to identify the job being executed (task name, subMS name and execution time). This information is dumped into an ASCII file, which can be monitored in a separated terminal.

Cluster robustness: The cluster internal state machine comes with a complete overhaul to load the engines environment exactly as a normal CASA instance but w/o the unnecessary services (e.g.: logger) and finalize the cluster gracefully using the native iPython multienginecclient methods and terminating the service threads properly even when the cluster objects are deleted before stopping the cluster.

Security policy: The security policy has been reviewed to avoid interferences between consecutive deployments of the cluster infrastructure that were causing problems in Mac OSX. Now the RSA private keys used in the controller-engine communication are specific to each cluster and are not re-used by any other cluster instance.

Multiple platform support: The new version of the cluster infrastructure comes with complete set of unit tests that exercises all the aspects above mentioned and proves its compatibility with Mac OSX 10.6/10.7 64bits and RHEL 5.3/5.6 32b/64b.

4 Work on flagging

The new flagdata framework (tflagdata, renamed to flagdata in CASA 4.0) is now fully compatible with the MMS format including the following aspects:

New features in CASA 4.0:

Consolidation of return variables: The returning summary dictionaries are now merged in a single dictionary.

Handling of input/output files: The input/outfiles are properly pointed by means of full paths so that the remote engines can located them regardless of their local working directory.

Handling of flagcmd lists: The flagcmd lists are now re-written to adapt them to the remote engines environment.

Handling of neglectable errors: The NullSelection ERRORS that ocurr at the engine level due to empty selections in some subMS are now handled properly and not forwarded to the controller instance.

Unit tests compatibility: The suit of tflagdata unit tests can now be ran vs. MMS instead of MS automatically (see section 9). Additionally there is a new set of unit tests to cover for the MMS-specific features above mentioned.

5 Work on MMS splitting and concatenation

CASA users are used to being able to operate on MSs with the split and concat tasks, i.e. extract a subset of the data from an MS into a new MS with the possibility to perform averaging or construct a single MS (e.g. for delivery) from a group of individual MSs. Essentially all CASA guides make use of these tasks.

It was therefore necessary to make sure that split and concat (a) work with MMSs and (b) perform as in the case of monolithic MSs. The second point, i.e. speed, is essential because the time which may be gained in parallelisation elsewhere must not be wasted during splitting and concatenation.

New features in CASA 4.0:

New task virtualconcat: The new task virtualconcat uses the new tool method **ms.virtconcatenate()** which in turn uses a new method in the MSConcat class in casacore. It permits the user to concatenate arbitrary datasets (both MSs and MMSs) into a single MMS. When the parameter **keepcopy** is set to True (default False), the input datasets are preserved. Otherwise they are just moved into the new MMS thereby reducing the necessary I/O by a large amount. If the input is already an MMS, it is treated like a group of individual MSs. The subtables of all subMSs of the new output MMS are then reindexed using the normal concat routines.

A detail concerning the creation of the FIELD table needed to be covered in this case: If two fields have the same direction but a different name, e.g. “3C273 phase” and “3C273 bandpass”, then concat in CASA 3.4 would merge them because it would disregard the field name and only

test the direction. With a new parameter **respectname** both in concat and virtualconcat, it is possible to force concat to keep the two fields separate.

New bool parameter "keepmms" in split: Running `split` on an MMS produces by default a monolithic MS. If the user wants to continue the processing with parallel processing, he/she needs to repartition the data. This introduces time-consuming I/O. With the new keepmms functionality, setting `keepmms` to True (default is False) makes split operate on the subMSs of the input MMS individually. The subMSs which result in a non-empty output MS (selection) are then virtually concatenated into the output MMS using ms.virtconcatenate().

To further speed up the processing, split with keepmms=True handles the POINTING table separately if no selection on time or antennas takes place (the majority of the use cases). If need be, also the POINTING table is virtually concatenated using the new tool method **tb.createmultitable()**

6 Work on the calibration tasks

While there was little work required on the Python level making the calibration tasks *run* on MMSs, considerable work went into creating the missing unit tests for these tasks.

Furthermore, the testing of parallel applycal revealed that the VisSet class and related classes were deriving the absolute paths of the main table and MS subtables in an unorthodox way which did not work with the general MMS. This required several fixes in the **synthesis** module.

6.1 gencal

It was not necessary to modify anything in task gencal to make it work with MMSs. This was verified with the current unit tests using the option "–datadir" as described in section 9.

6.2 gaincal

It was not necessary to modify anything in task gaincal to make it work with MMSs. We created new unit tests for this task and verified its validity using the option "–datadir" as described in section 9.

6.3 bandpass

It was not necessary to modify anything in task bandpass to make it work with MMSs. We created new unit tests for this task and verified its validity using the option "–datadir" as described in section 9.

6.4 setjy

The setjy task is now fully compatible with the MMS format in both scratch and scratch-less mode:

Scratch-less mode operation: setjy iterates sequentially in the main controller instance trough the list of subMS performing all the setjy processing individually for each subMS thus setting the model properly in every 'eligible' subMS. Afterwards, there is a new step in setjy to gather the models from the keywords of each individual subMS and copy them to the rest.

Scratch mode operation: setjy uses ParallelTaskHelper to run setjy in parallel for each subMS creating the MODEL_DATA column for all of them in order to ensure table description consistency across the MMS (this is also the behavior for normal MS) although it is not filled for the subMSs that don't have any rows in the data selection range for performance reasons. The underlying tool methods are now parallel-safe by means of using subMS specific names for the temporary model files.

Visibility model header validation: The visibility model header has been validated by checking that all subMSs contain the models for all fields in their keywords with the parameters specified in the setjy run.

MODEL_DATA column validation: The MODEL_DATA column has been validated by checking that all the subMSs with rows in the data selection range have their values aligned with the parameters specified in the setjy run and additionally by checking that the subMSs without rows in the data selection range have their values set to the default.

6.5 fluxscale

It was not necessary to modify anything in task fluxscale to make it work with MMSs. We created new unit tests for this task and verified its validity using the option "–datadir" as described in section 9.

6.6 applycal

Was superficially already MMS-capable in CASA 3.4 but needed work in the underlying code in the synthesis module (see above). Furthermore, the following work was done:

Handling of negligible errors: The NullSelection ERRORS that occur at the engine level, due to empty selections in some subMS are now handled properly, and not forwarded to the controller CASA instance. Additionally, they are internally handled to make sure that all sub-tables are properly closed.

Data columns and table description consistency: The CORRECTED_DATA column is always created to ensure table description consistency, although it is not filled in for the subMSs that don't have any rows in the data selection range, for performance reasons. Additionally, the re-initialization methods have been modified, in order not to create the MODEL_DATA column, after recovering from the NullSelection errors, that occur at the subMS level.

MMS vs MS validation: The CORRECTED_DATA column produced by an input MMS has been validated vs the result obtained with the equivalent MS, when using bcal, gcal and fluxscale cal tables.

7 Work on uvcontsub

The uvcontsub task was parallelised for the MMS case. The unit test was updated to also exercise the case of MMS input.

Input MMS compatibility: It can process an input MMS in parallel, extracting the continuum spectrum for each subMS in a remote engine individually.

Output MMS result: The resulting cont and contsub files from all the subMSs are gathered, and virtually concatenated to present them in MMS format. In both cases, the POINTING table is handled specially in order to ensure that there is only one full copy of it in the output MMSs and that there is no time wasted on making intermediate copies.

MMS vs MS validation: These results have been validated by comparing the output cont/contsub files produced by and MMS with the cont/contsub files produced by the equivalent MS.

8 Work on wvrgcal

The wvrgcal task in CASA 4.0 uses the code contained in the almawvr library version 1.2 together with the actual executable "wvrgcal" compiled from source files in the code/air_casawvr module.

When testing on MMSs, it became apparent that the code in code/air_casawvr was making the assumption that the rows of the MS main table are sorted in time. Since this is not generally true, neither for an MMS nor for an MS, a time sorted row index needed to be introduced. These changes concerned four separate parts of the code. They are in r21246.

The modified code in code/air_casawvr has been sent to Bojan Nikolic to be merged back to the Cambridge repository.

9 Testing

The existing regressions and unit tests of CASA 3.4 did not exercise any of the new HPC-related functionality. Also the unit test for partition was not yet adequate. For some of the modified tasks, unit tests did not even exist. In order to ensure that the HPC-related modifications would not break any old functionality, missing unit tests had to be created. Once these existed, a way to repeat these same unit tests with input in MMS format had to be found. This required the creation of some additional test infrastructure to handle MMS as described below. An additional requirement we set for ourselves was that we avoid to bloat the data repository with MMS copies of the already stored MSs.

New features in CASA 4.0:

New CASA task "listpartition": Similarly to the task "listobs", "listpartition" can list the properties of an MMS, as well as an MS. For each subMS of the MMS, it lists in the logger or save to a file, the available scans, spws, number of channels and the size in disk. There is also a boolean parameter "createdict" that returns a Python dictionary for manipulation. See *help listpartition* inside casapy for more information.

Many functions that are used by listpartition are made available for the users in the Python script partitionhelper.py. It contains functions to easy the comparison of the structure of MMSs and MSs. These functions are heavily used in test_listpartition.py and test_partition.py. A couple of other functions to make more general table comparisons is made available in the script testhelper.py. Both, testhelper.py and partitionhelper.py can be imported inside casapy.

New option "–datadir" in runUnitTest.py: The unit test framework got a new option called "–datadir" which sets an environmental variable called TEST_DATADIR, pointing to a directory where the data is found. Tests can be modified apriori to read this variable to access data from an alternate location, otherwise they work normally and use the path given internally in the tests. This way, with a few new lines, any test can run using data from another directory. It can be used to run the tests using MMSs or for any other general use that needs different data sets. The unit tests manual linked from the CASA Index wiki page contains an example on how to use this option. See also (http://www.eso.org/ scastro/ALMA/CASAUnitTests.htm). Or any of these tests in the repository can be used as an example: test_bandpass.py, test_split.py, test_tflagdata.py, and many others.

As a result of our work to include the "–datadir" option in runUnitTest.py, a few bugs had to be fixed in the unit tests framework. The script was also made more robust with a new parsing of argument variables.

Easy way to create MMSs for the unit tests: Following the introduction of the "–datadir" option in the unit tests framework, there was the need to easily create MMSs for the tests. We created two new scripts for this purpose. The main class "convertToMMS", added to the file partitionhelper.py, can be used to manually create MMSs from a directory containing normal

MSs, such as the directories in the data repository under data/regression/unittest. The script runs using the default values of partition, except that "datacolumn" is set to 'all'. It tries to create an MMS for every MS in the input directory. It skips non-MS directories such as cal tables. If partition succeeds, the script creates a link to every other directory or file in the output directory. This script might fail if run on single dish MS because the "datacolumn" needs to be set in partition. In order to run this script inside casapy, do the following:

```
import partitionhelper as ph
datapath='/opt/casa/data/regression/unittest/gaincal'
ph.convertToMMS(inpdir=datapath,createmslink=True)
```

This will create MMSs for all MSs in the gaincal directory. The MMSs are created inside a directory "mmsdir" in the working directory, unless set otherwise. The "createmslink" option will create a symbolic link to the new MMS with an extension ".ms". Type ph.convertToMMS() for further options to run this script.

The second script created for this project creates MMSs automatically for some CASA tasks. The available tasks are the ones that have been verified to work with MMS, as described throughout this document. The script "make_mmsdata.py" runs outside casapy. For each of the tasks known by the script, the convertToMMS() class is called up to create MMSs in the "unittest_mms" output directory, saved locally. The script has options to ignore some tasks or run only a few of them. See below a few examples on how to use it. The script is installed under the path-to-casa-installation/python/2.6/regressions/admin/.

```
# Get help to run the script;
casapy --nogui --log2term -c <path>/make_mmsdata.py

# See the list of tasks that the script will create tests for;
casapy --nogui --log2term -c <path>/make_mmsdata.py --list

# Create MMSs for all listed tasks;
casapy --nogui --log2term -c <path>/make_mmsdata.py --all

# Do not create MMSs for the given tasks;
casapy --nogui --log2term -c <path>/make_mmsdata.py -i listobs split
```

With these two scripts and the use of the "–datadir" in runUnitTest.py, one can run any task test (modified for this purpose) and verify that it works with MMSs.

New set of cluster-specific unit tests: The new version of the cluster infrastructure comes with a set of unit tests that exercises all the new features (see section 3) and proves its compatibility with Mac OSX 10.6/10.7 64bits, and RHEL 5.3/5.6 32b/64b.

10 First performance measurements

The available test machine was a HP Z600 workstation with 12 processors and 12 GB RAM, a SATA disk (2 TB), and a Solid State Disk (OCZ VeloDrive, 1.2 TB).

The benchmark scripts are available from the CASA repository:

`alma-m100-analysis-regression.py`

and

`alma-m100-analysis-hpc-regression.py`

The first script was already available for CASA 3.4. It executes a complete ALMA data analysis from data import to image analysis. The second script, "alma-m100-analysis-hpc-regression.py" does the same except that after `importasdm`, there is a call to `partition` which converts the imported MS to an MMS. The rest of the analysis is then carried out based on MMSs and at the end clean or pclean are used.

The timing of each step of the script is measured via a timing function defined at the beginning of the scripts.

Table 1 shows the results using CASA stable r21422.

All tests were carried out in a working directory on the solid state disk. The differences in execution time for the individual steps are purely due to the parallelisation and, in the first step, due to the additional partitioning.

Varying the number of engines deployed between 6 and 10 and the number of subMSs in the partitioning from 6 to 32 showed that for the test machine, a setup of 8 engines and 8 subMSs is optimal. In this case, all subMSs can still be processed in one parallel iteration while each of the engines has sufficient RAM and the SSD can provide enough throughput. On a different machine, the optimal setup will be different.

Table 1: Performance comparison of **CASA stable r21422** with default cluster setup running an ALMA analysis on science verification data (taken in 2011) non-parallelised (non-HPC) and parallelised (HPC) using **8 subMSs partitioned by scan** and **8 engines** on a HP Z600 workstation with 12 processors, 12 GB RAM, and a 1.2 TB solid state disk.

Step	non-HPC version		HPC version		Speedup	Step description
	Time used (s)	Time Fraction (%)	Time used (s)	Time Fraction (%)		
0	391.62	9.31	391.62	9.96		Data import
0b	0.00	0.00	230.16	5,85		Partitioning
1	2.59	0.06	3.84	0.09	0.67	Generate antenna position cal tables
2	1.03	0.02	4.07	0.10	0.26	Generate tsys cal tables
3	21.35	0.50	25.53	0.64	0.84	Correct the Titan position
4	156.36	3.71	97.90	2.48	**1.60**	Apriori flagging
5	178.76	4.25	177.98	4.52	1.00	Generate WVR cal tables
6	80.62	1.91	92.51	2.35	0.87	Generate delay calibration tables
7	1447.59	34.43	868.08	22.07	**1.67**	Apply antpos, wvr, tsys, and delay tables
8	404.57	9.62	360.04	9.15	1.12	Split off non-wvr spws and save flags
9	94.52	2.24	136.44	3.46	0.69	Flagging
10	231.26	5.50	225.07	5.72	1.03	Rebin to a reduced resolution of 10 km/s
11	5.83	0.13	10.97	0.27	0.53	Fast phase-only gaincal for bandpass
12	10.32	0.24	14.42	0.36	0.72	Bandpass
13	9.75	0.23	31.16	0.79	0.31	Setjy
14	48.02	1.14	57.02	1.45	0.84	Fast phase-only gaincal
15	23.48	0.55	31.85	0.81	0.74	Slow phase-only gaincal
16	27.22	0.64	35.96	0.91	0.76	Slow amp and phase gaincal
17	1.37	0.03	2.64	0.06	0.52	Fluxscale
18	158.51	3.77	105.30	2.67	**1.51**	Applycal
calib.	*3294.86*	*78.37*	*2902.56*	*73.81*	*1.14*	All import and calibration steps
19	29.01	0.69	35.15	0.89	0.83	Test image of the secondary phase cal
20	197.19	4.69	232.62	5.91	0.85	Test image of the primary phase cal
21	57.36	1.36	83.69	2.12	0.69	Test image of Titan
22	29.30	0.69	44.89	1.14	0.65	Split off calibrated M100 data
23	16.34	0.38	14.48	0.36	1.13	Concatenate M100 data (virtual in HPC case)
24	32.79	0.78	12.51	0.31	**2.62**	Average concatenated M100 data in time
25	251.12	5.97	287.26	7.30	0.87	Continuum image of M100
26	23.79	0.56	27.53	0.70	0.86	Determine and subtract continuum
27	1.12	0.02	4.35	0.11	0.26	Test image of central field
28	266.56	6.34	270.24	6.87	0.99	Clean line cube mosaic
29	3.21	0.07	15.61	0.39	0.21	Make moment maps
30	1.39	0.03	1.39	0.03	1.00	Verification of the regression results
total	4204.11	100.00	3932.43	100.00	1.07	

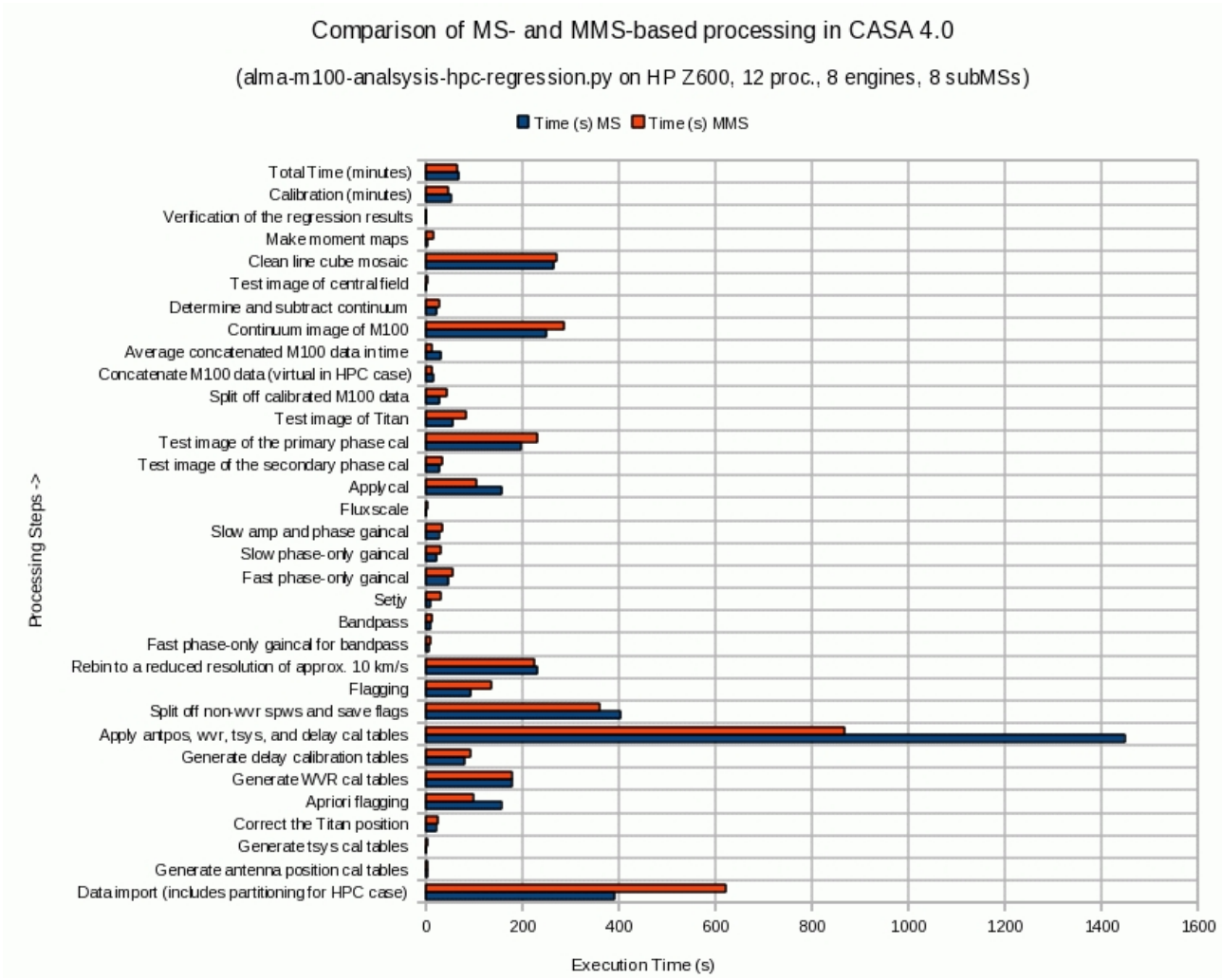

Figure 1: The data presented in Table 1 as a bar chart showing the performance measurements obtained as described in the text with **CASA stable r21422** on a HP Z600 workstation (12 GB RAM, 12 processors) with a 1.2 TB solid state disk.

11 Open issues and suggestions for next steps in development cycle 4.1

11.1 Detailed performance evaluation

The most important open issue is a detailed study of how the performance of all relevant tasks is different on MSs and MMSs in order to determine the optimal tuning of the analysis system. This study should be carried out on the intended final systems, i.e. clusters with parallel file systems but on the side also on SSD-equipped high-power workstations.

For the study, also a regression with a larger, more recent dataset should be created and used. This is already in preparation based on a ALMA QA2 calibration script for a 24-antenna dataset of ca. 53 GB size.

The study should be careful not to tune the script to the best analysis path ex-post, i.e. in order to be realistic, a path has to be followed which includes exploratory steps as would be present when *unknown* data is analysed for the first time.

11.2 CASA tasks not yet tested with MMSs

The following tasks still need to be tested to work with MMS input. Tasks marked with a "(p)" can be parallelised.

fixvis (p)

cvel (p)

hanningsmooth (p)

ft (p?)

exportuvfits

visstat

plotxy

polcal

delmod

11.3 Improving the efficiency of simple_cluster

Startup and job distribution in simple_cluster still seem to take too long. This has to be measured in detail in order to determine if the cluster administration overhead could still be reduced.

11.4 Improving the efficiency and usefulness of logging

Presently, the terminal and logger output for some of the parallelised tasks is still either too terse or too confusing.

Also, it is not clear if the computing power consumed by the logging process is significant and whether it can be reduced.

11.5 MMS-capable CASA tasks still to be parallelised

Among the tasks which are already capable of processing an MMS but are not actually parallelised, there is mainly one task which would still be worth to be parallelised: `flagmanager`. This task's execution time scales with MS size. It can take several minutes for a typical ALMA MS. Parallelisation therefore makes sense. It would involve to also have partitioned flagversion tables.

11.6 Automatic cluster setup

The present heuristics take as input the number of processors and the amount of RAM and derive the OpenMP setup and the number of engines to be deployed using the requirement of 512 MB per processor and 90% of the total RAM and 90% of all the processors. This simple algorithm will need to be tested and refined in order to make sure that typical computers are not overloaded and at least one user interaction can still take place when the cluster is running.

Also, there are indications that `pclean` may need a different setup than the calibration-related tasks. **Fast setup switching** may have to be enabled.

The fast setup switching could be achieved by adding a new method in the cluster framework to dynamically specify the number or engines to use without having to re-deploy the cluster. This will make an improvement when the cluster is deployed in a single machine, because one can reduce the number of engines to be used depending on the resources needed per engine, which typically depend on the task (for instance reduce the number of engines to be used when running pclean).

11.7 Partition improvements

Presently, the subMSs are not generally sorted in time. If the partition axis is TIME, then it would be good to have the subMS numbering correspond to the time order. Tasks which access the main table monolithically will otherwise have a non-optimal sort order.

11.8 Lazy importasdm

Ger van Diepen, Michel Caillat and DP are working on a prototype for a ALMA Storage Manager which will be capable of making the ALMA (and EVLA) visibility BLOBs from the ASDM directly readable (not writable) from the MS DATA column.

If this can be achieved and is reasonably fast, this would have two advantages:

1. In a MMS-based analysis, there would not have to be an intermediate MS written at import. `importasdm` would just write an MS with a DATA column pointing to the ASDM BLOBS and `partition` would then write this data into a MMS with a normal storage manager. This would save 30% storage space w.r.t. the present MMS case (ASDM + MS + MMS) and potentially also 30% of the computing time (the first MS is written much more quickly).

2. The lazy import would enable faster access to the metadata (listobs) of an ASDM within CASA.

11.9 Optimize special cases of split usage

In ALMA data analysis, there are four common uses of `split`.

a) Make copy of CORRECTED_DATA into a new MS: This is a trivial operation which could potentially be sped up by making the copy of CORRECTED_DATA on the operating system level and working with *renaming* the copied column to DATA.

b) Make copy of CORRECTED_DATA into a new MS selecting only certain SPWs: This is less trivial unless the partitioning already has done the selection for us. In that case, the problem could be reduced to selecting subMSs and performing the operation described under (a) on them.

c) Time averaging: This is elegantly handled in the new `split` with keepmms=True. There is indeed a speed-up of a factor approx. 2.

d) Spectral averaging: Changes the SPECTRAL_WINDOW table and thus requires a change of those tables. This is done with a virtual concat in the present split which is too time consuming. More clever handling of the subtables would speed this up.

If these cases could be improved (case (c) is already fine), split on MMS would become sufficiently fast.

11.10 Optimize uvcontsub

`uvcontsub` presently uses a direct call to task virtualconcat to recombine the created new subMSs. This entails some overhead and should be replaced by calls to tool methods making use of the knowledge about the subtable properties.