CASA 4.3 Parallel Processing Framework
- Installation and advance user guide -

Version: 1.2

Status: Development

Prepared By:		
Name	**Organisation**	**Date**
Justo Gonzalez	**ESO**	2014-05-20

Change Record

Version	Date	Affected Section	Remarks
1.0	2014-05-20	All	Initial version
1.1	2014-06-3	4.2 (mpi4py installation) 4.4 (Compiling CASA using MPI compilers) 5.2 (mpi4casa initialization)	- Removed step to generate MPI-enabled version of python interpreter - Added gcwrap options so that SWIG components don't use Python's GIL - Changed the MPISErver/Client rank convention
1.2	2014-10-10	4.3 (Boost MPI) 5.2 (mpi4casa initialization) 6.0 (Packaging and cluster integration)	- Corrected typo in the command to build and install Boost MPI - Added note regarding python binary full path requirement when running in interactive mode - Added a new section for packaging and cluster integration

Table of Contents

1 LIST OF TERMS AND ACRONYMS 4
2 LIST OF RELATED JIRA ISSUES 4
3 INTRODUCTION 5
4 INSTALLATION GUIDE 6
4.1 OPEN MPI 6
4.2 MPI4PY 8
4.3 BOOST MPI 10
4.4 CASA COMPILE CONFIGURATION (CMAKE SETTINGS) 11
5 MPI4CASA 12
5.1 MODULE STRUCTURE 12
5.2 MPI4CASA INITIALIZATION 13
5.3 MPICOMMANDCLIENT LIFE CYCLE 15
5.4 MPICOMMANDCLIENT USAGE 17
5.5 MPICOMMANDCLIENT/SERVER LOGGING 22
5.6 MPICOMMANDCLIENT/SERVER ERROR HANDLING 23
5.7 MPICOMMANDCLIENT BASIC PYTHON EXAMPLES 24
5.8 MPICOMMANDCLIENT CASA TASKS EXAMPLES 31
5.9 MPI4CASA UNIT TEST SUIT 35
6 PACKAGING AND CLUSTER INTEGRATION 36
6.1 STAND-ALONE MPI CASA BUILD 36
6.2 RUN A TORQUE BATCH JOB USING MPI 41

1 List of terms and acronyms

CASA	Common Astronomy Software Applications
MS	Measurement Set
MSs	Group of Measurement Sets
MMS	Multi Measurement Set
MMSs	Group of Multi Measurement Sets
Sub-MS	Sub-Measurement Set (component of a Multi Measurement Set)
Sub-MSs	Group of Sub-Measurement Sets
API	Application Programing Interface
MPI	Message Passing Interface
GUI	Graphical User Interface
HPC	High Performance Computing
stl	C++ Standard Template Library
iPython	Python command shell with advanced interactivity features such as rich history, enhanced introspection, etc
InfiniBand	Network communication link used in HPC applications

2 List of related Jira issues

CAS-5795	MPI-based CASA cluster infrastructure (container ticket, see issues under it)
CAS-5797	Test integration of MPI-based ipcluster with Torque
CAS-5798	Compare the various packages offering python bindings for MPI
CAS-5800	Analysis of schemes to support MPI in the CASA cluster environment
CAS-5801	Analysis of MPI implementations (system wise, apart from the python bindings)

3 Introduction

This is a technical document (advance user and developer guide) describing how to install and set up the necessary packages to use the new MPI-based parallel processing framework for CASA 4.3. It also contains several examples on how to use mpi4casa, the new Python module that enables MPI-based parallelism at the task level. The following list described the package used, and why it is needed:

1. **Open MPI:** One of the advantages of MPI is the possibility to switch from one implementation to another w/o suffering API changes. Despite of this I believe that CASA should provide a preferred MPI implementation together with the release package, and we should also have a standardized development environment, and that means selecting a group of MPI implementations and use them for development and testing. Having said this I've chosen Open MPI for my own development and testing because it is the one of the newest implementations, with extensive development and aims to support all common interconnects including InfiniBand and full MPI-2 compliance. Therefore I suggest to go ahead with it, and study in parallel what advantages / disadvantages come along with other implementations.

2. **mpi4py:** This is basically the MPI Python bindings that allow to use MPI from the Python layer. There are several choices, but mpi4py is the best in terms of performance, completeness and supports communication of arbitrary python objects, thus w/o requiring any serialization. It is necessary because in general CASA is best parallelized from the task level, which is the highest level possible, and thus minimizes the parallelization overhead. This is so for all the tasks that support trivial parallelism, and therefore don't need efficient communication at the C++ level. The exception is imaging, which due to the major/minor cycle structure requires efficient communication among the parallel processes.

3. **boostmpi:** This is the preferred package providing a C++ interface for MPI. It supports direct communication of all the stl library types and containers w/o requiring further serialization, thus is a really good candidate. Furthermore boost has become a standard in itself in the C++ modern development world, and it is a force pushing for many changes in the new C++ standards (e.g. many new features of C++11 are already available in C++99 via boost).

4. **mpi4casa:** This is the new CASA module implementing a complete MPI-based parallel processing framework. It is built on top of mpi4py, and thus honors its name. It uses a Client/Server model, where two completely different CASA environments are loaded depending on the role. The 'main' Client process is basically a normal CASA session, including all the associated GUIs and processes for interactive usage. The other Server processes load the minimal CASA environment w/o GUIs, iPython or any interactivity component, and communicate with the 'main' Client process via MPI.

4 Installation guide

4.1 Open MPI

4.1.1. Download Open MPI:

```
wget http://www.open-mpi.org/software/ompi/v1.6/downloads/openmpi-1.6.5.tar.bz2
```

4.1.2. Unzip and untar package

```
bunzip2 openmpi-1.6.5.tar.bz2
tar -xvf openmpi-1.6.5.tar
cd openmpi-1.6.5
```

4.1.3. Configure enabling thread-safe support, compile and install
NOTE: It is absolutely necessary that you explicitly add the multi-threading support options, because they are not added by default.

```
./configure --enable-mpi-thread-multiple --enable-opal-multi-threads
make all
make install
```

4.1.4. Update the $HOME/.bash_profile file to include the location of the MPI libraries in LD_LIBRARY_PATH

```
vim $HOME/.bash_profile
export LD_LIBRARY_PATH=/usr/local/lib
```

4.1.5. Make sure that MPI compilers and launchers are available in the PATH:

```
MPI C compiler:       /usr/local/bin/mpicc
MPI C++ compiler:     /usr/local/bin/mpicxx
MPI F77 compiler:     /usr/local/bin/mpif77
MPI F90 compiler:     /usr/local/bin/mpif90
Pre MPI 2.0 launcher: /usr/local/bin/mpirun
MPI 2.0 launcher      /usr/local/bin/mpiexec
```

4.1.6. Enable password free ssh access to all the machines running MPI:

```
ssh-keygen -t rsa
$HOME/.ssh/id_rsa.pub | ssh user@hostname 'cat >> $HOME/.ssh/authorized_keys'
```

4.1.7. Make a basic test in the localhost to make sure MPI is properly install

```
mpirun -n 4 ls
```

4.1.8. Prepare a simple hostfile with 2 nodes and 2 slots per node following the Open MPI hostfile syntax:

```
# This is an example hostfile.  Comments begin with #
# The following node is a single processor machine:
foo.example.com
# The following node is a dual-processor machine:
bar.example.com slots=2
# The following node is a quad-processor machine, and
# we absolutely want to disallow over-subscribing it:
yow.example.com slots=4 max-slots=4
```

4.1.9. Make a test using the hostfile from the previous step:

```
[testhpc2@almahpc02 MPI]$ cat hostfile
almahpc02.ads.eso.org slots=2
almahpc01.hq.eso.org slots=2
[testhpc2@almahpc02 MPI]$ mpirun -n 4 -hostfile hostfile hostname
almahpc02
almahpc02
almahpc01
almahpc01
```

4.2 mpi4py

4.2.1. Download mpi4py

```
     wget http://mpi4py.googlecode.com/files/mpi4py-1.3.1.tar.gz
```

4.2.2. Unzip and untar package

```
     tar -xzvf mpi4py-1.3.1.tar.gz
```

4.2.3. Build and install the mpi4py package using the python version shipped with CASA. It will automatically detect the MPI implementation and use it for compilation.

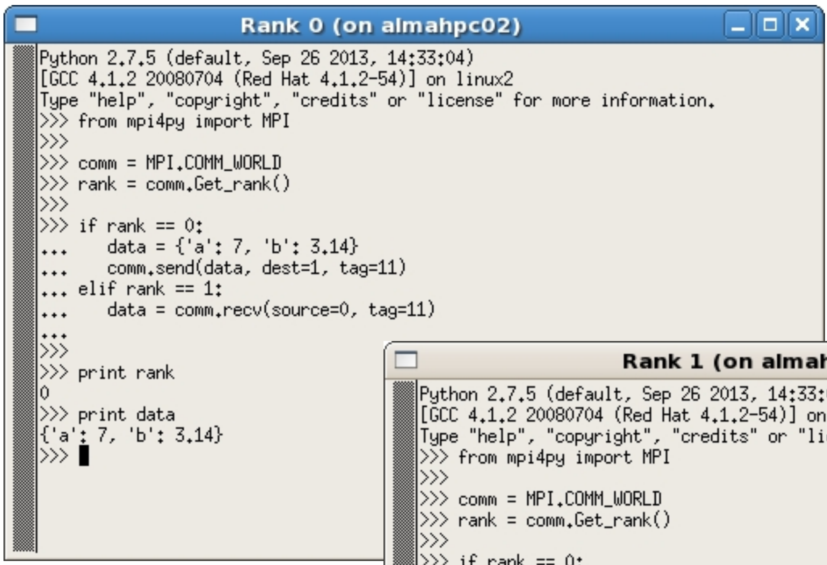
```
     /usr/lib64/casa/01/bin/python setup.py build
     /usr/lib64/casa/01/bin/python setup.py install
```

4.2.4. Run a simple parallel python test using one xterm per python process:

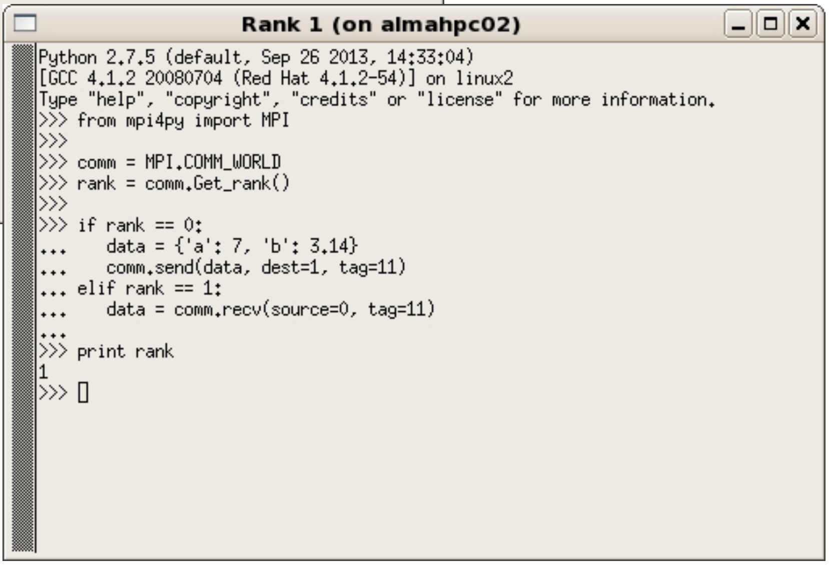
```
     mpirun -n 2 -xterm 0,1 /usr/lib64/casa/01/bin/python
```

4.2.5. Copy and paste the following test code in both xterm python terminals. It basically sends a dictionary from the rank 0 python process to the rank 1 python process. Make sure that the dictionary is received in the rank 1 process by printing it.

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
   data = {'a': 7, 'b': 3.14}
   comm.send(data, dest=1, tag=11)
elif rank == 1:
   data = comm.recv(source=0, tag=11)
```


Rank 0 (on almahpc02)

```
Python 2.7.5 (default, Sep 26 2013, 14:33:04)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-54)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from mpi4py import MPI
>>>
>>> comm = MPI.COMM_WORLD
>>> rank = comm.Get_rank()
>>>
>>> if rank == 0:
...     data = {'a': 7, 'b': 3.14}
...     comm.send(data, dest=1, tag=11)
... elif rank == 1:
...     data = comm.recv(source=0, tag=11)
...
>>>
>>> print rank
0
>>> print data
{'a': 7, 'b': 3.14}
>>> 
```


Rank 1 (on almahpc02)

```
Python 2.7.5 (default, Sep 26 2013, 14:33:04)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-54)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from mpi4py import MPI
>>>
>>> comm = MPI.COMM_WORLD
>>> rank = comm.Get_rank()
>>>
>>> if rank == 0:
...     data = {'a': 7, 'b': 3.14}
...     comm.send(data, dest=1, tag=11)
... elif rank == 1:
...     data = comm.recv(source=0, tag=11)
...
>>> print rank
1
>>> 
```

4.3 Boost MPI

4.3.1. Install bjam (boost setup utility)

```
wget http://sourceforge.net/projects/boost/files/boost-jam/3.1.18/boost-jam-
3.1.18.tgz/download
tar -xzvf boost-jam-3.1.18.tgz
cd boost-jam-3.1.18
./build.sh
cp bin.linuxx86_64/bjam /usr/local/bin/
```

4.3.2. Download and unpack boost:

```
wget
http://sourceforge.net/projects/boost/files/boost/1.41.0/boost_1_41_0.tar.gz/download
tar -xzvf boost_1_41_0.tar.gz
cd boost_1_41_0
```

4.3.3. Edit the boost configuration file to specify the python binary and add the boos::mpi module

```
vim ./tools/build/v2/user-config.jam
=> Add using python : 2.7 : /usr/lib64/casa/01/bin/python ;
=> Add using mpi ;
```

4.3.4. Using bjam configure boost specifying as prefix the location where the CASA boost packages is installed (/usr/lib64/casa/01)

```
bjam --debug-configuration --prefix= /usr/lib64/casa/01 install
```

4.4 CASA compile configuration (cmake settings)

4.4.1. Now all the pieces are ready to compile CASA C++ code with full MPI support. To do it so it only necessary to specify the MPI compilers in the cmake configuration step.

```
cmake -DCMAKE_Fortran_COMPILER=/usr/bin/mpif90 -DCMAKE_C_COMPILER=/usr/bin/mpicc -
DCMAKE_CXX_COMPILER=/usr/bin/mpic++ (... other options ...)
```

This has to be done for all CASA modules (casacore, code, gcwrap, asap, etc) that contain explicit calls to MPI (i.e. includes to mpi.h or boost::mpi).

4.4.2. Apart from using the MPI compilers, it is also necessary to use in the gwrap cmake options a SWIG parameter so that the CASA SWIG components (Python bindings) release the Python's GIL (global interpreter lock), and the MPIClient/Server service threads get proper priority to run their tasks

- Python's GIL is in place for all threads launched at the Python level because access to Python objects is not thread-safe.
- Python's GIL only allows a given thread to use CPU resources if all the other threads are performing I/O operations (read, write, send, receive, etc)
- Python's GIL net effect is that it prevents threads launched at the Python level from using the CPU at the same time, thus only 1 CPU is used regardless of the number of threads.
- Unfortunately it is not possible to bypass Python's GIL even if the different threads access different Python object, thus there is not thread-safe risk.
- Fortunately SWIG has an option ('-threads)' so that the Python objects generated are not affected by the Python's GIL, thus whenever a CASA component call is invoked the Python's GIL is released, and threads at the Python level (MPIClient/Server service threads can run their tasks).

```
cmake -DCMAKE_Fortran_COMPILER=/usr/bin/mpif90 -DCMAKE_C_COMPILER=/usr/bin/mpicc -
DCMAKE_CXX_COMPILER=/usr/bin/mpic++ -DCMAKE_SWIG_FLAGS="-threads" ..
```

5 mpi4casa

5.1 Module structure

The new module to support MPI at the CASA task level is called mpi4casa. It is located under gcwrap/python/scripts/mpi4casa. The following table describes the contents of the module:

MPIEnvironment.py	Static class in charge of Initializing MPI and properly set the MPI-related variables.
MPICommunicator.py	Singleton class, which centralizes all the MPI, calls, using the appropriate targets and tags.
MPICommandClient.py	Singleton class, which provides access to high-level methods to: - Send CASA commands to the remote side (with blocking, non-blocking modes, targeting a single or multiple servers) - Get the response from the remote calls (either return variables or stack traces in case of error)
MPIMonitorClient.py	Singleton class used primarily by MPIMonitorClient, to monitor the state of the remote servers in terms of activity (busy/idle) and responsiveness.
MPICommandServer.py	Counterpart for MPICommandClient: Singleton class with a service thread that checks for incoming MPI messages containing CASA (or in general Python) command requests. It executes them and sends back the response (return code), and back trace in case if error.
MPIMonitorServer.py	Counterpart for MPIMonitorClient: Singleton class with a service thread that checks for incoming MPI messages containing check status requests.
mpi4casapy.py	Runnable python script used to launch the MPI Server environment.
task_wrappers.py	Python script containing the import definitions for the CASA tasks wrappers to be used in the MPI Server environment.
task_macros.py	Python script containing the macro definitions for the CASA tasks wrappers to be used in the MPI Server environment.
test_mpi4casa.py	Unit test suit with more than 60 cases covering: - MPI client/server command cases: * evaluation with return code or execution * blocking and non-blocking modes * single and multiple targets * parameters provided via string or dictionary * error/exception handling * server timeout state handling * server timeout recovery

5.2 mpi4casa initialization

As you can see in the previous section mpi4casa uses a Client/Server execution model:

- **MPIClient:** 'Main' MPI instance with all interactivity features (GUIs, iPython shell, async, etc). That is, a complete normal CASA instance.

- **MPIServer:** Minimal CASA environment without any interactivity features
  - Basically a python script running in the background
  - CASA env. Variables are sent as a dictionary via MPI from MPIClient (e.g. log file, data paths, etc)

- **Convention:** The established convention is that the Client is the process with rank 0, (in MPI rank process rank numbering starts with 0). For this reason:

  - stdin must be redirected only for the process with rank 0

```
mpirun -n 4 -stdin 0 $CASAROT/bin/casapy
```

  - xterm must be initiated only for the process with rank 0

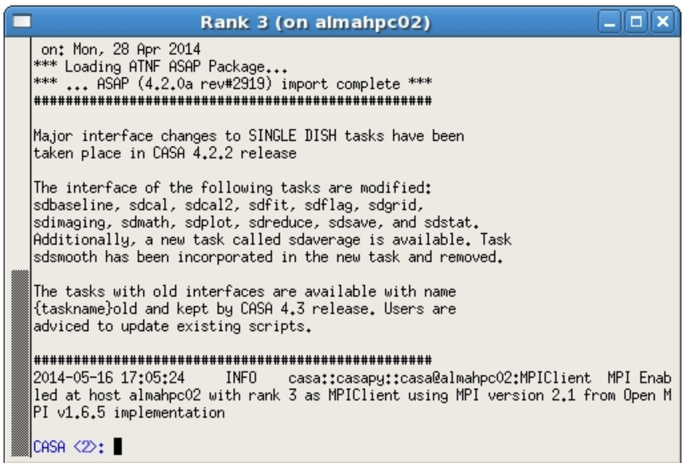
```
mpirun -n 4 -xterm 0 $CASAROT/bin/casapy
```

NOTE: stdin/out redirection from the terminal is not the recommended way to interact with the client due to a miss-formatting that occurs when stdin/out is redirected trough the mpirun process. Instead we recommend to use a dedicated xterm window. However stdin/out redirection from terminal is the only way to interact with the client in MACOX

NOTE: Due to the ssh-MPI interaction it is necessary to specify the full path of the CASA binary when running in interactive mode.

- **Initialization success:** After launching CASA with mpirun, if the initialization is successful you will see the following message:

```
            casa::casapy::casa@almahpc02:MPIClient  MPI Enabled at host almahpc02 with
    rank 3 as MPIClient using MPI version 2.1 from Open MPI v1.6.5 implementation
```



```
 on: Mon, 28 Apr 2014
*** Loading ATNF ASAP Package...
*** ... ASAP (4.2.0a rev#2919) import complete ***
##################################################

Major interface changes to SINGLE DISH tasks have been
taken place in CASA 4.2.2 release

The interface of the following tasks are modified:
sdbaseline, sdcal, sdcal2, sdfit, sdflag, sdgrid,
sdimaging, sdmath, sdplot, sdreduce, sdsave, and sdstat.
Additionally, a new task called sdaverage is available. Task
sdsmooth has been incorporated in the new task and removed.

The tasks with old interfaces are available with name
{taskname}old and kept by CASA 4.3 release. Users are
adviced to update existing scripts.

##################################################
2014-05-16 17:05:24     INFO     casa::casapy::casa@almahpc02:MPIClient  MPI Enab
led at host almahpc02 with rank 3 as MPIClient using MPI version 2.1 from Open M
PI v1.6.5 implementation

CASA <2>: 
```

This message indicates that the MPI version provided by the underlying MPI implementation is correct, and also that thread-safe support is enabled.

Also, if all the processes are deployed in the same machine, you can see via pstree how the processes are deployed from mpirun:

```
[testhpc2@almahpc02 MPI]$ ( mpirun -n 4 -xterm 3 casapy --nologger --log2term &> mpirung.log ) &
[1] 8704
[testhpc2@almahpc02 MPI]$ pstree 8704
mpirun─┬─3*[python]
       └─xterm───python─┬─2*[casaviewer]
                        ├─dbus-daemon
                        ├─ipcontroller
                        ├─python
                        └─3*[{python}]
```

- **Initialization failure:** If the mpi4casa initialization is not sucessfull (e.g.: the underlying MPI implementation does not have thread-safe support) and error message appears and it is not possible to use instantiate MPIClient:

```
          WARN    casa::casapy::casa       Provided MPI implementation (Open MPI v1.6.5)
 is not thread safe configured, maximum thread safe level supported is: MPI THREAD SINGLE

          WARN    casa::casapy::casa+     NOTE: In most MPI implementations thread-safety
 can be enabled at pre-compile, by setting explicit thread-safe configuration options,

          WARN    casa::casapy::casa+    e.g. (MPI 1.6.5) --enable-mpi-thread-multiple
```

5.3 MPICommandClient life cycle

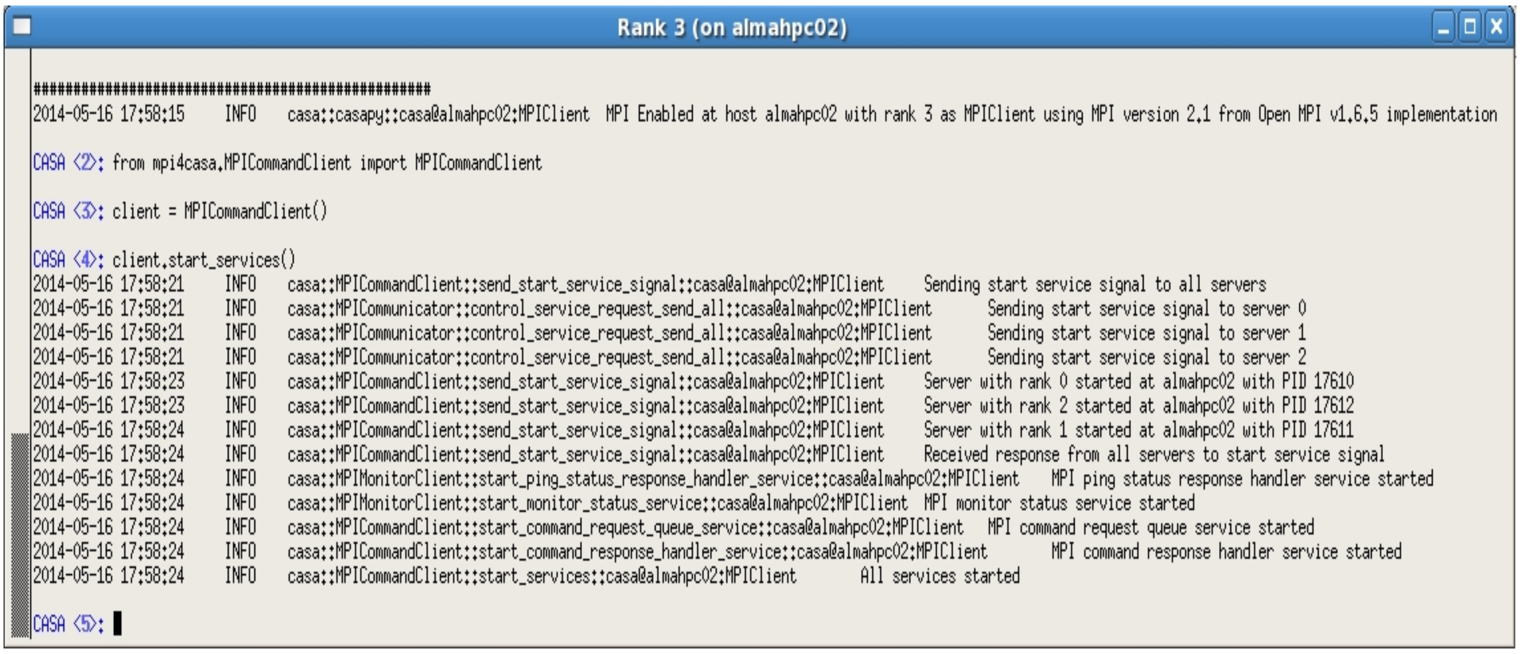
As mentioned previously MPICommandClient creational pattern is a singleton. This design choice is useful to avoid multiple initializations, which can cause resource problems. Nevertheless, the user can control when the Client services are initialized or de-initialized, in order to waste CPU resources when parallelization is not necessary.

NOTE: mpi4py is already integrated with ParallelTaskHelper, so it is not necessary to do anything (in particular initialize MPICommandClient) in order to run tasks in parallel. ParalleTaskHelper re-uses or initializes the MPICommandClient singleton if necessary.

- **MPICommandClient initialization:** MPICommandClient is a singleton, therefore it is necessary to initialize it only once, and all the other instances created afterwards will point to the first one. This is done transparently for the user.

```
CASA <2>: from mpi4casa.MPICommandClient import MPICommandClient

CASA <3>: client = MPICommandClient()

CASA <4>: client.start_services()
```

MPICommandClient will not only initialize the client service, but also send a message to the servers so that they are initialized:



```
##################################################
2014-05-16 17:58:15     INFO    casa::casapy::casa@almahpc02:MPIClient  MPI Enabled at host almahpc02 with rank 3 as MPIClient using MPI version 2.1 from Open MPI v1.6.5 implementation

CASA <2>: from mpi4casa.MPICommandClient import MPICommandClient

CASA <3>: client = MPICommandClient()

CASA <4>: client.start_services()
2014-05-16 17:58:21     INFO    casa::MPICommandClient::send_start_service_signal::casa@almahpc02:MPIClient     Sending start service signal to all servers
2014-05-16 17:58:21     INFO    casa::MPICommunicator::control_service_request_send_all::casa@almahpc02:MPIClient       Sending start service signal to server 0
2014-05-16 17:58:21     INFO    casa::MPICommunicator::control_service_request_send_all::casa@almahpc02:MPIClient       Sending start service signal to server 1
2014-05-16 17:58:21     INFO    casa::MPICommunicator::control_service_request_send_all::casa@almahpc02:MPIClient       Sending start service signal to server 2
2014-05-16 17:58:23     INFO    casa::MPICommandClient::send_start_service_signal::casa@almahpc02:MPIClient     Server with rank 0 started at almahpc02 with PID 17610
2014-05-16 17:58:23     INFO    casa::MPICommandClient::send_start_service_signal::casa@almahpc02:MPIClient     Server with rank 2 started at almahpc02 with PID 17612
2014-05-16 17:58:24     INFO    casa::MPICommandClient::send_start_service_signal::casa@almahpc02:MPIClient     Server with rank 1 started at almahpc02 with PID 17611
2014-05-16 17:58:24     INFO    casa::MPICommandClient::send_start_service_signal::casa@almahpc02:MPIClient     Received response from all servers to start service signal
2014-05-16 17:58:24     INFO    casa::MPIMonitorClient::start_ping_status_response_handler_service::casa@almahpc02:MPIClient    MPI ping status response handler service started
2014-05-16 17:58:24     INFO    casa::MPIMonitorClient::start_monitor_status_service::casa@almahpc02:MPIClient  MPI monitor status service started
2014-05-16 17:58:24     INFO    casa::MPICommandClient::start_command_request_queue_service::casa@almahpc02:MPIClient   MPI command request queue service started
2014-05-16 17:58:24     INFO    casa::MPICommandClient::start_command_response_handler_service::casa@almahpc02:MPIClient        MPI command response handler service started
2014-05-16 17:58:24     INFO    casa::MPICommandClient::start_services::casa@almahpc02:MPIClient        All services started

CASA <5>: 
```

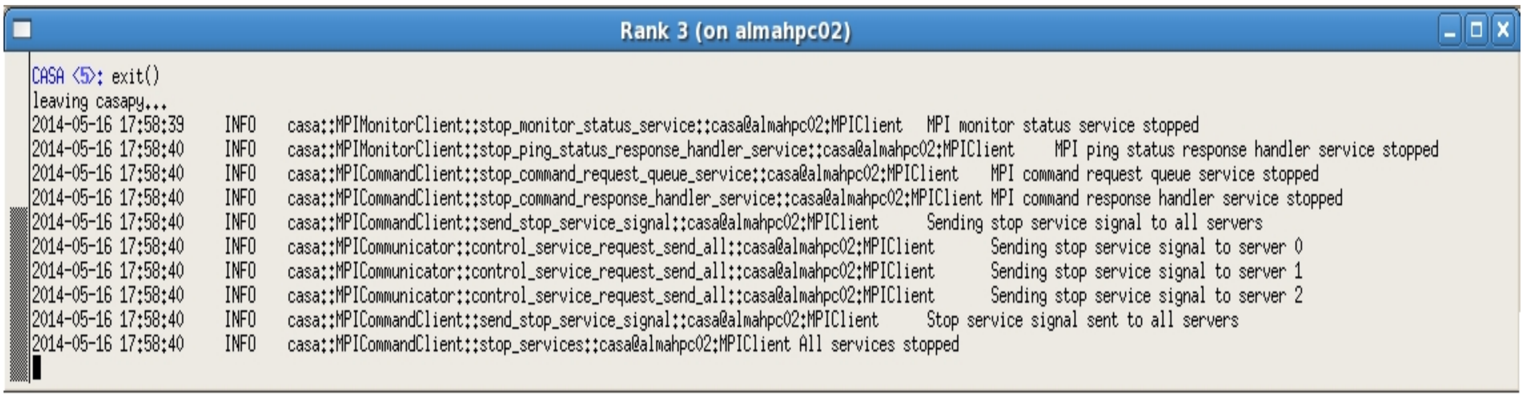
- **MPICommandClient de-initialization:** Since MPICommandClient is a singleton its life cycle is handled carefully in a transparent way for the user, so that the singleton instance is not deleted as long as other Python objects are using it. This is not done trough a __del__ operator but using the atexit module instead, to register an exit function which is executed when python is finalized.

  Nevertheless it is possible to stop the services manually after running all the parallel processing steps

  CAUTION: When MPIClient services are stopped, the entire parallel system is stopped, including finalization of the MPIServer processes. This means that it is not longer possible to use the parallel processing framework after stopping the MPIClient services.

```
CASA <4>: client.stop_services()
```

Otherwise, if the MPIClient services are not stopped manually, the stop function is automatically invoked when Python exists as described before:



```
CASA <5>: exit()
leaving casapy...
2014-05-16 17:58:39     INFO    casa::MPIMonitorClient::stop_monitor_status_service::casa@almahpc02:MPIClient   MPI monitor status service stopped
2014-05-16 17:58:40     INFO    casa::MPIMonitorClient::stop_ping_status_response_handler_service::casa@almahpc02:MPIClient     MPI ping status response handler service stopped
2014-05-16 17:58:40     INFO    casa::MPICommandClient::stop_command_request_queue_service::casa@almahpc02:MPIClient    MPI command request queue service stopped
2014-05-16 17:58:40     INFO    casa::MPICommandClient::stop_command_response_handler_service::casa@almahpc02:MPIClient MPI command response handler service stopped
2014-05-16 17:58:40     INFO    casa::MPICommandClient::send_stop_service_signal::casa@almahpc02:MPIClient      Sending stop service signal to all servers
2014-05-16 17:58:40     INFO    casa::MPICommunicator::control_service_request_send_all::casa@almahpc02:MPIClient       Sending stop service signal to server 0
2014-05-16 17:58:40     INFO    casa::MPICommunicator::control_service_request_send_all::casa@almahpc02:MPIClient       Sending stop service signal to server 1
2014-05-16 17:58:40     INFO    casa::MPICommunicator::control_service_request_send_all::casa@almahpc02:MPIClient       Sending stop service signal to server 2
2014-05-16 17:58:40     INFO    casa::MPICommandClient::send_stop_service_signal::casa@almahpc02:MPIClient      Stop service signal sent to all servers
2014-05-16 17:58:40     INFO    casa::MPICommandClient::stop_services::casa@almahpc02:MPIClient All services stopped
```

5.4 MPICommandClient usage

MPICommandClient has mainly two public methods to interface with the user, in order to send commands to the remote servers, and retrieve the results. Additionally there is a method to retrieve the status of the remote servers, if that is necessary for debugging reasons.

NOTE: mpi4py is already integrated with ParallelTaskHelper, so it is not necessary to do anything (in particular initialize MPICommandClient) in order to run tasks in parallel. ParalleTaskHelper re-uses or initializes the MPICommandClient singleton if necessary.

- def **push_command_request**(*self*,command,block=False,target_server=None,parameters=None)

  This method allows sending CASA/Python commands to the remote servers, so that they are evaluated (with return code) or executed (w/o return code).

  It is possible to call this method in block and non-block mode. When running in non-block mode it will return a list of integers, corresponding to the command request ids, which can be used in a second stage to retrieve the response via the public method 'get_command_response' (explained later in this section). When running in blocking mode it will return a list of dictionaries, one per command request, containing the response parameters.

  It is possible to specify a defined target server or group of target servers, by providing a list of integers, corresponding to the MPI rank of the target servers.

  It is also possible to specify the command parameters via an auxiliary dictionary, and the command will be executed in the remote servers after injecting the variables specified in the parameters dictionary in the python global variables namespace.

- **command**: String containing the Python/CASA command to be executed. The parameters can be specified in two ways:

  - ♦ Within the command in itself, also as strings:

```
"flagdata(backup=True)"
```

  - ♦ In the parameters dictionary, using the native types:

```
parameters={backup:True}
```

- **block**: Boolean to control whether command request is executed in blocking or not:

  - ♦ Block=True: It will block until the command is queue, sent to a remote process, executed, and the corresponding response is received.
  - ♦ Block=False: It will not block, just register the command request and return an identifier which can be used later on to get the command response

- **target_server**: List of integers (server ids) which are the target for the command. The ids correspond to the MPI process rank of the target servers

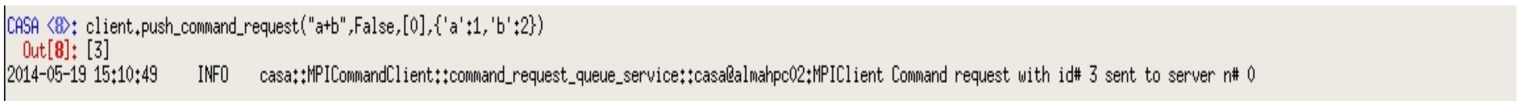
  NOTE: Remember that according with the mpi4casa convention the process with highest rank corresponds to the client, and the other process are the servers. e.g.: In a 4 processors run, the ranks with id 0,1,2 correspond to the servers, and can be used as target_server, and the process with rank 0 corresponds to the client.

  - ♦ target_server=None: The command will be executed by the first available server
  - ♦ target_server=2: The command will be executed by the server n#2 as soon as it is available
  - ♦ target_server=[0,1]: The command will be executed by the servers n #2 and #3 as soon as they are available

- **parameters** (optional): dictionary containing the parameters to be used in the command execution. These parameters will be injected in the local variable space of the server and deleted after execution. Any python (pickable) object is accepted.

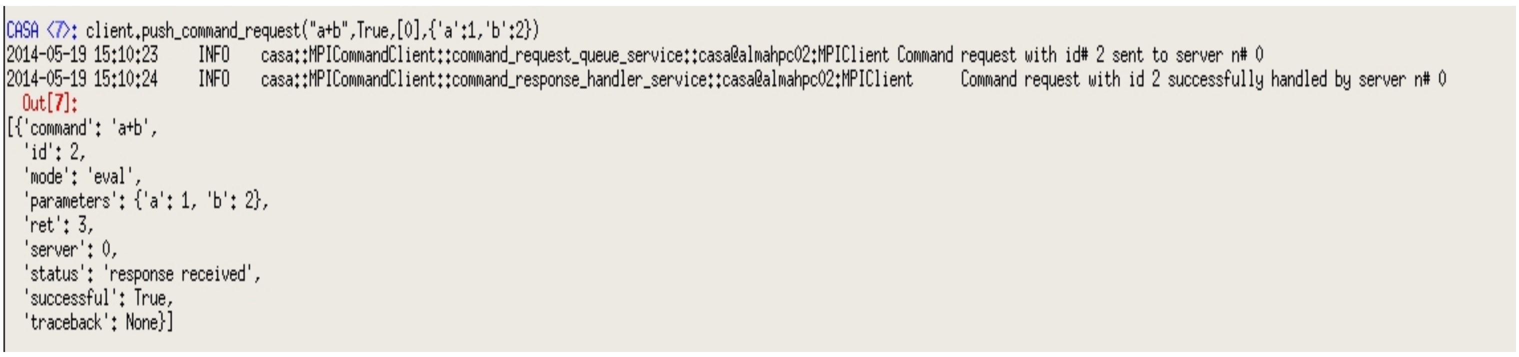
- **returns:**

  - In non-blocking mode: It will return immediately, giving the responses already available as a list of dictionaries, one per command request, containing the response parameters.



```
CASA <8>: client.push_command_request("a+b",False,[0],{'a':1,'b':2})
  Out[8]: [3]
2014-05-19 15:10:49     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 3 sent to server n# 0
```

  - In blocking mode: It will not return until the responses for all the command requests have been received. Then it returns a list of dictionaries, one per command request, containing the response parameters.



```
CASA <7>: client.push_command_request("a+b",True,[0],{'a':1,'b':2})
2014-05-19 15:10:23     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 2 sent to server n# 0
2014-05-19 15:10:24     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 2 successfully handled by server n# 0
  Out[7]:
[{'command': 'a+b',
  'id': 2,
  'mode': 'eval',
  'parameters': {'a': 1, 'b': 2},
  'ret': 3,
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

The command response dictionary contains the following keys:

keyword	type	Description
command	string	CASA/Python command that was executed
id	integer	Identifier assigned to the command request
mode	string	There are two command modes: - eval: The command was executed in eval mode which returns a variable - exec: The command was just executed, w/o returning any variable
parameters	dictionary	Dictionary containing the parameters used for the command execution
ret	(variable)	Variable returned by the command
server	integer	MPI rank of the server process which executed the command request
status	string	Command request status: - timeout: The server assigned to this command requested timeout. - holding queue: This command request is still holding a queue, and has not been sent to the server yet. - request sent: This command has already been sent to a server, and the response has been received.
successful	boolean	Boolean to specify if the command execution was successful or not, where successful means that the command did not throw any exceptions.
traceback	string	When the command is not successful it throws an exception, whose back trace is stored in this parameter.

▪ def **get_command_response**(*self*,command_request_id_list,block=False,verbose=True)

This method to retrieve the response for a given command request id/ids specified as a list of integers as returned by push_command_request in non-blocking mode.

This method also has a blocking and non-blocking modes, where blocking means that the method will not return until all the responses have been received, and non-blocking will return immediately, giving the responses already available as a list of dictionaries, one per command request, containing the response parameters.

The verbose parameter control the information provided when the command responses have not been received yet. When verbose mode is True it will inform about the servers which are running the command requests whose answer has not been received yet.

- **Command Request Id:** List with ids (integers) of the command response to retrieve

- **Block**: Boolean to control whether command request is executed in blocking or not:

  ♦ Block=True: It will block until the response from all command is received

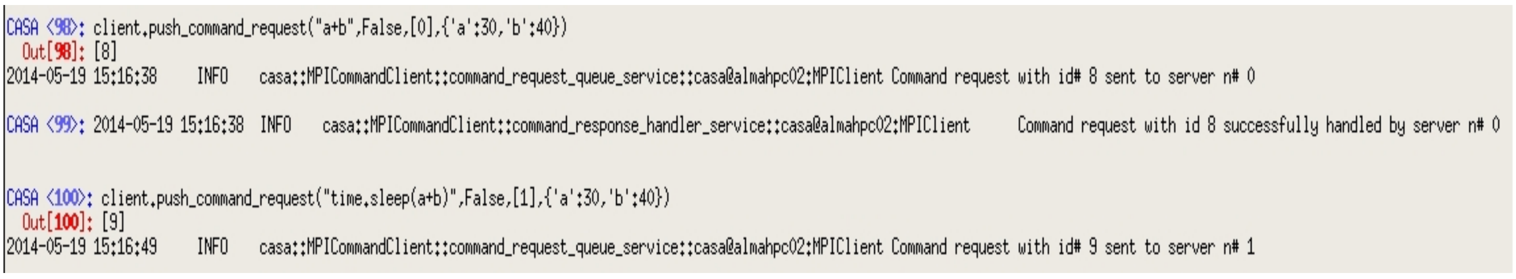
  ♦ Block=False: It will not block, and just return the available responses

  NOTE: If a command was sent to a server which is not responsive at retrieval time, it will not block the return of this method, but notify of the time-out error instead

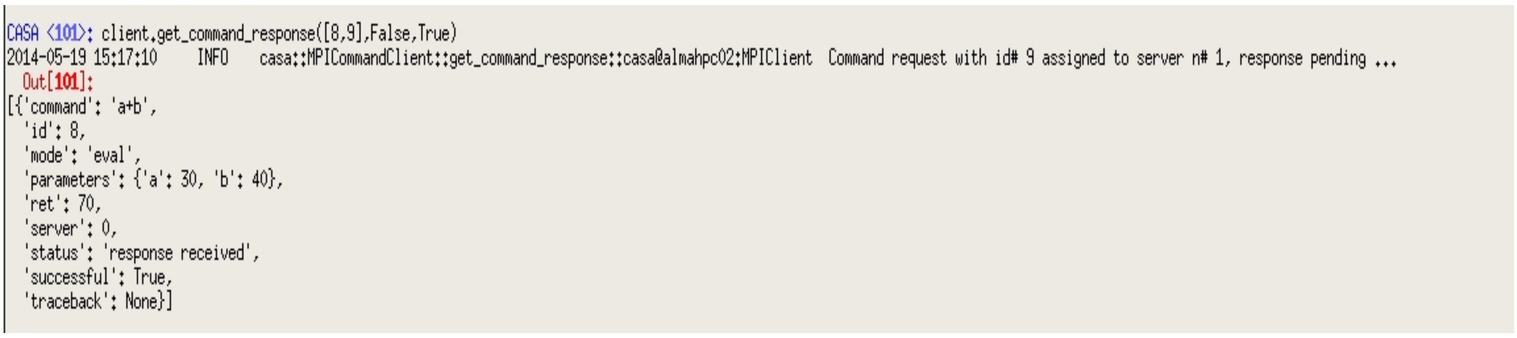
- **verbose:** Boolean to control weather information about command request status is posted in non-blocking mode

- **returns:**

  ♦ In non-blocking mode: It will return immediately, giving the responses already available as a list of dictionaries, one per command request, containing the response parameters.

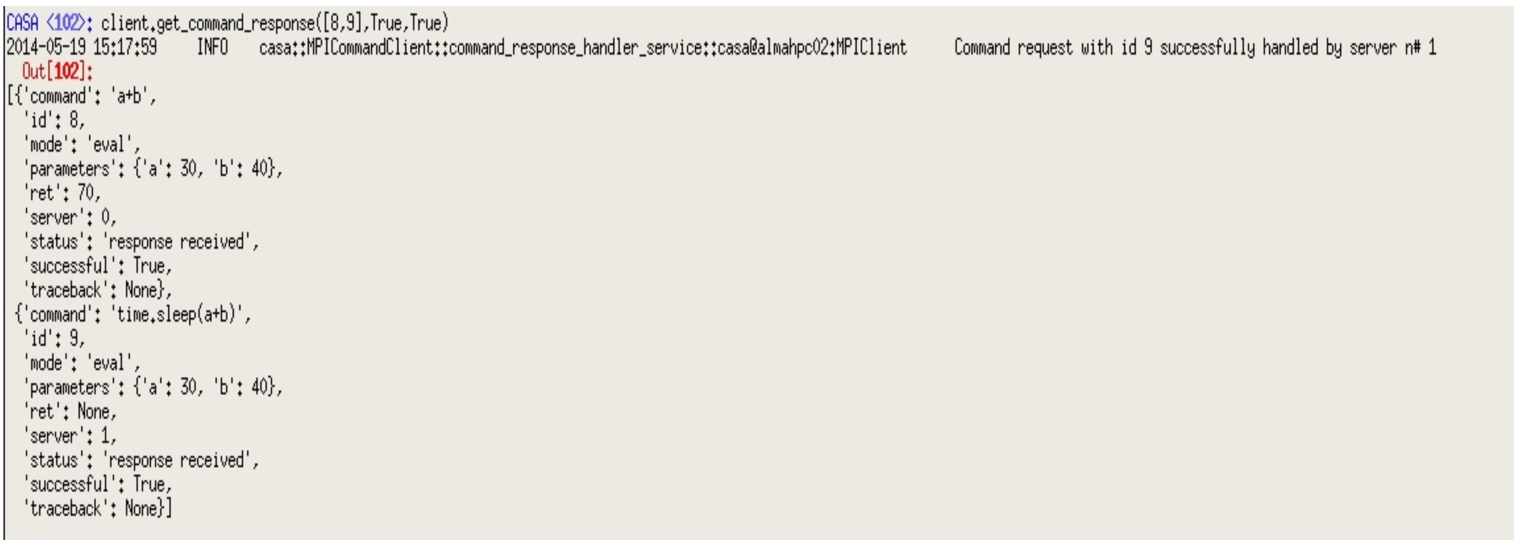


```
CASA <98>: client.push_command_request("a+b",False,[0],{'a':30,'b':40})
  Out[98]: [8]
2014-05-19 15:16:38     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 8 sent to server n# 0

CASA <99>: 2014-05-19 15:16:38  INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 8 successfully handled by server n# 0


CASA <100>: client.push_command_request("time.sleep(a+b)",False,[1],{'a':30,'b':40})
  Out[100]: [9]
2014-05-19 15:16:49     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 9 sent to server n# 1
```



```
CASA <101>: client.get_command_response([8,9],False,True)
2014-05-19 15:17:10     INFO    casa::MPICommandClient::get_command_response::casa@almahpc02:MPIClient  Command request with id# 9 assigned to server n# 1, response pending ...
  Out[101]:
[{'command': 'a+b',
  'id': 8,
  'mode': 'eval',
  'parameters': {'a': 30, 'b': 40},
  'ret': 70,
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

  ♦ In blocking mode: It will not return until the responses for all the command requests have been received. Then it returns a list of dictionaries, one per command request, containing the response parameters.



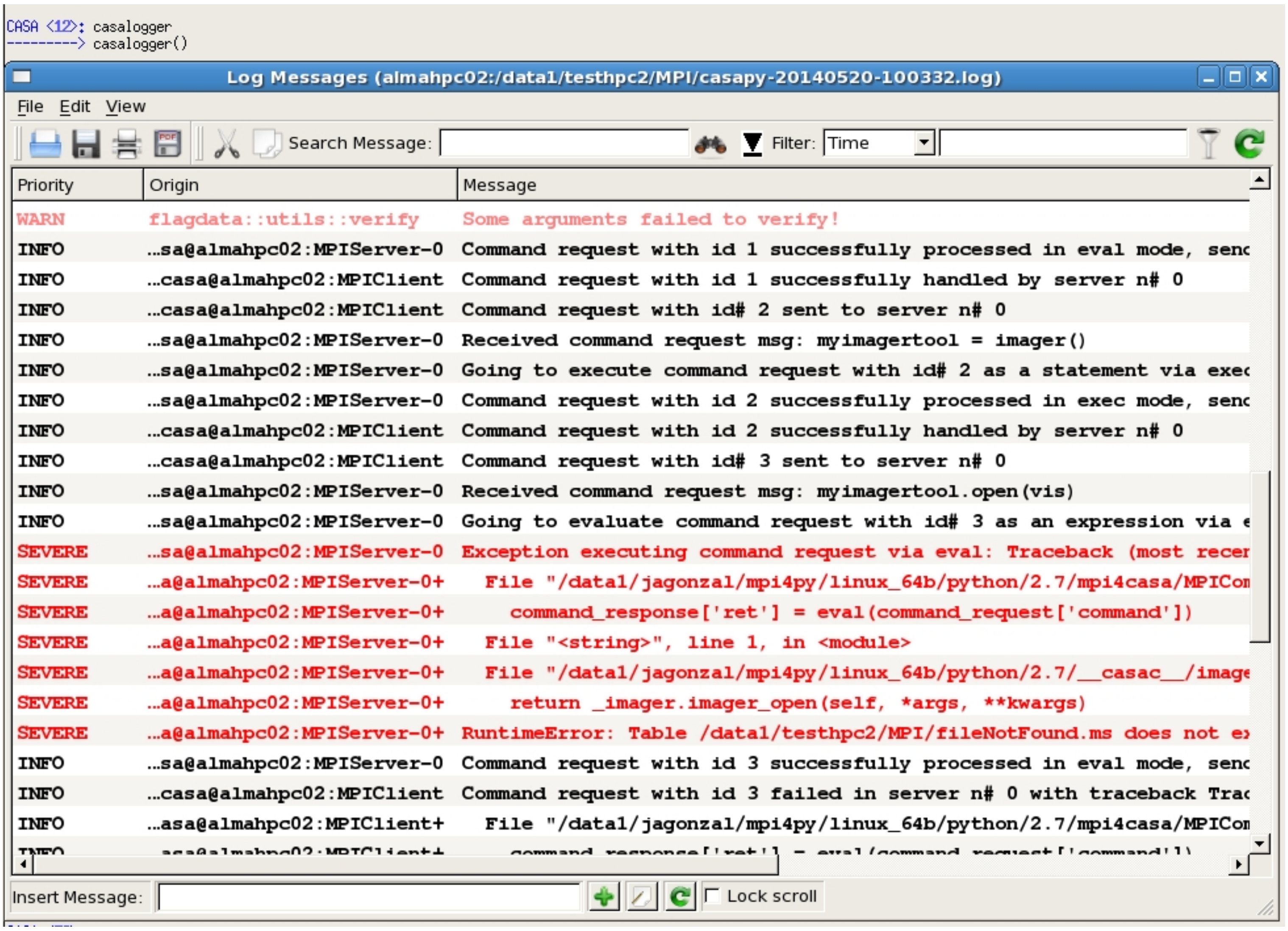
```
CASA <102>: client.get_command_response([8,9],True,True)
2014-05-19 15:17:59     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 9 successfully handled by server n# 1
  Out[102]:
[{'command': 'a+b',
  'id': 8,
  'mode': 'eval',
  'parameters': {'a': 30, 'b': 40},
  'ret': 70,
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None},
 {'command': 'time.sleep(a+b)',
  'id': 9,
  'mode': 'eval',
  'parameters': {'a': 30, 'b': 40},
  'ret': None,
  'server': 1,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

5.5 MPICommandClient/Server logging

In the current version (CASA 4.3) both sides (client and servers) send the logs to the same log file, and it is possible to inspect the server logs in real time, by using the casalog GUI in the client side. However, the server logs will not appear in the Client terminal directly.

Provided that the logs form all servers are flushed to the same file, the MPI rank of each server has been added to the log origin to facilitate the log analysis. However, this has been done only at the Python level, and in second iteration it will be added to the C++ level. It is a rather trivial step but since it involves compiling CASA with MPI compilers I have decided to postpone this step until we have a proper CASa development environment with MPI enabled.

In future version the plan is that the logger in the server side sends the logs to the client logger, which in turns can send them to the terminal and/or the log file.



```
CASA <12>: casalogger
--------> casalogger()
```

Log Messages (almahpc02:/data1/testhpc2/MPI/casapy-20140520-100332.log)

```
Priority  Origin                          Message
WARN      flagdata::utils::verify         Some arguments failed to verify!
INFO      ...sa@almahpc02:MPIServer-0     Command request with id 1 successfully processed in eval mode, senc
INFO      ...casa@almahpc02:MPIClient     Command request with id 1 successfully handled by server n# 0
INFO      ...casa@almahpc02:MPIClient     Command request with id# 2 sent to server n# 0
INFO      ...sa@almahpc02:MPIServer-0     Received command request msg: myimagertool = imager()
INFO      ...sa@almahpc02:MPIServer-0     Going to execute command request with id# 2 as a statement via exec
INFO      ...sa@almahpc02:MPIServer-0     Command request with id 2 successfully processed in exec mode, senc
INFO      ...casa@almahpc02:MPIClient     Command request with id 2 successfully handled by server n# 0
INFO      ...casa@almahpc02:MPIClient     Command request with id# 3 sent to server n# 0
INFO      ...sa@almahpc02:MPIServer-0     Received command request msg: myimagertool.open(vis)
INFO      ...sa@almahpc02:MPIServer-0     Going to evaluate command request with id# 3 as an expression via e
SEVERE    ...sa@almahpc02:MPIServer-0     Exception executing command request via eval: Traceback (most recer
SEVERE    ...a@almahpc02:MPIServer-0+       File "/data1/jagonzal/mpi4py/linux_64b/python/2.7/mpi4casa/MPICon
SEVERE    ...a@almahpc02:MPIServer-0+         command_response['ret'] = eval(command_request['command'])
SEVERE    ...a@almahpc02:MPIServer-0+       File "<string>", line 1, in <module>
SEVERE    ...a@almahpc02:MPIServer-0+       File "/data1/jagonzal/mpi4py/linux_64b/python/2.7/__casac__/image
SEVERE    ...a@almahpc02:MPIServer-0+         return _imager.imager_open(self, *args, **kwargs)
SEVERE    ...a@almahpc02:MPIServer-0+     RuntimeError: Table /data1/testhpc2/MPI/fileNotFound.ms does not ex
INFO      ...sa@almahpc02:MPIServer-0     Command request with id 3 successfully processed in eval mode, senc
INFO      ...casa@almahpc02:MPIClient     Command request with id 3 failed in server n# 0 with traceback Trac
INFO      ...asa@almahpc02:MPIClient+       File "/data1/jagonzal/mpi4py/linux_64b/python/2.7/mpi4casa/MPICon
INFO      ...asa@almahpc02:MPIClient+         command_response['ret'] = eval(command_request['command'])
```

5.6 MPICommandClient/Server error handling

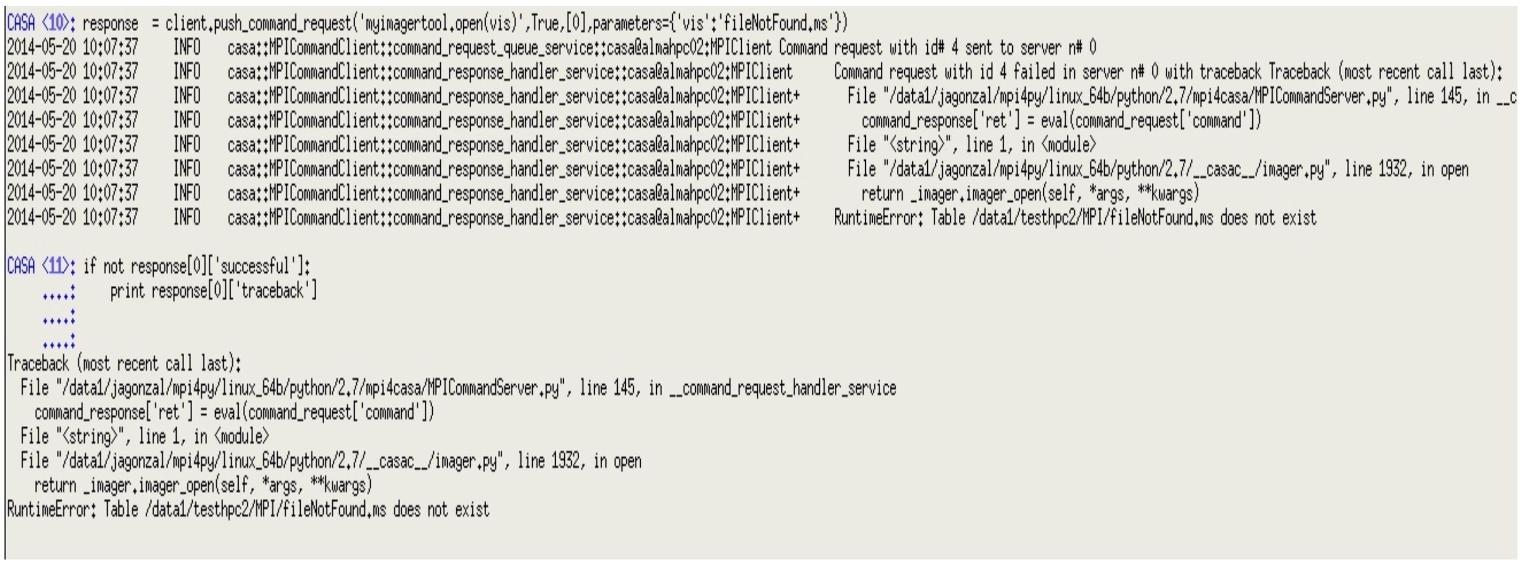
MPICommandClient/Server are ready to handle any python exception that occurs in the server, and report it back to the client side, properly formatted.

When a python/CASA command is not successful (i.e. and exception is thrown), then the command response sets the Boolean field 'successful' to False, and the string field 'traceback' contains the python trace-back already formatted (using the python module traceback).

NOTE: Please notice that in terms of Python a command execution is considered not successful only when it throws an exception. In terms of CASA, some tasks return a Boolean False when the call is not successful, but w/o throwing any exception. In this case, the return variable 'ret' of the command response will contain a Boolean False, but the Boolean field 'successful' will be set to True.

- ex 1.: exception thrown when opening a tool

```
parameters = {'vis':'fileNotFound.ms'}
response  = client.push_command_request('myimagertool.open(vis)',True,[0],parameters)


CASA <11>: if not response[0]['successful']:
     ....:     print response[0]['traceback']
     ....:
     ....:
Traceback (most recent call last):
  File "/data1/jagonzal/mpi4py/linux_64b/python/2.7/mpi4casa/MPICommandServer.py", line
145, in __command_request_handler_service
    command_response['ret'] = eval(command_request['command'])
  File "<string>", line 1, in <module>
  File "/data1/jagonzal/mpi4py/linux_64b/python/2.7/__casac__/imager.py", line 1932, in
open
    return _imager.imager_open(self, *args, **kwargs)
RuntimeError: Table /data1/testhpc2/MPI/fileNotFound.ms does not exist
```

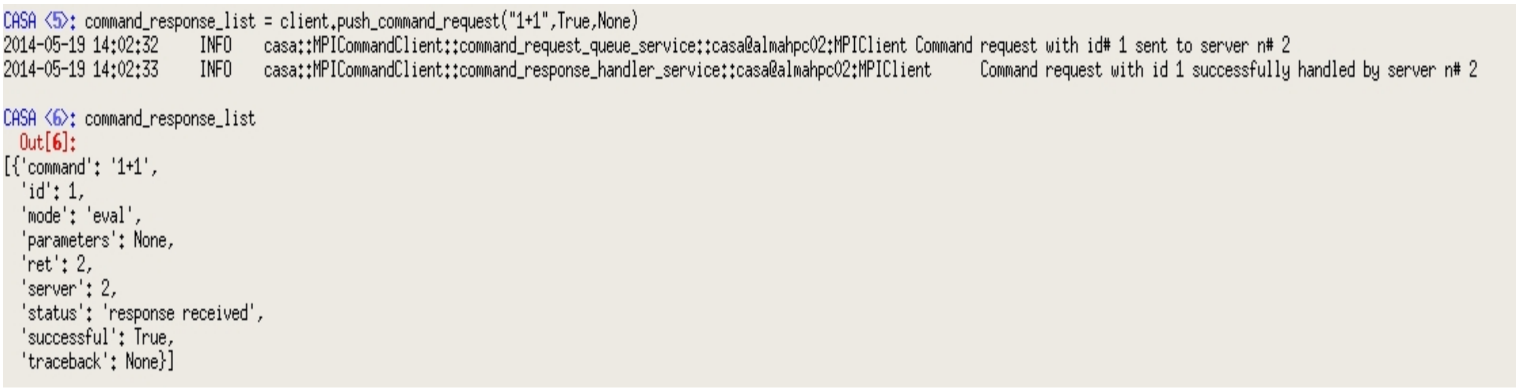


```
CASA <10>: response  = client.push_command_request('myimagertool.open(vis)',True,[0],parameters={'vis':'fileNotFound.ms'})
2014-05-20 10:07:37     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 4 sent to server n# 0
2014-05-20 10:07:37     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 4 failed in server n# 0 with traceback Traceback (most recent call last):
2014-05-20 10:07:37     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient+       File "/data1/jagonzal/mpi4py/linux_64b/python/2.7/mpi4casa/MPICommandServer.py", line 145, in __c
2014-05-20 10:07:37     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient+         command_response['ret'] = eval(command_request['command'])
2014-05-20 10:07:37     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient+       File "<string>", line 1, in <module>
2014-05-20 10:07:37     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient+       File "/data1/jagonzal/mpi4py/linux_64b/python/2.7/__casac__/imager.py", line 1932, in open
2014-05-20 10:07:37     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient+         return _imager.imager_open(self, *args, **kwargs)
2014-05-20 10:07:37     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient+     RuntimeError: Table /data1/testhpc2/MPI/fileNotFound.ms does not exist

CASA <11>: if not response[0]['successful']:
     ....:     print response[0]['traceback']
     ....:
     ....:
Traceback (most recent call last):
  File "/data1/jagonzal/mpi4py/linux_64b/python/2.7/mpi4casa/MPICommandServer.py", line 145, in __command_request_handler_service
    command_response['ret'] = eval(command_request['command'])
  File "<string>", line 1, in <module>
  File "/data1/jagonzal/mpi4py/linux_64b/python/2.7/__casac__/imager.py", line 1932, in open
    return _imager.imager_open(self, *args, **kwargs)
RuntimeError: Table /data1/testhpc2/MPI/fileNotFound.ms does not exist
```

5.7 MPICommandClient basic python examples

- **ex 1:** Blocking mode, string parameters, undefined target server

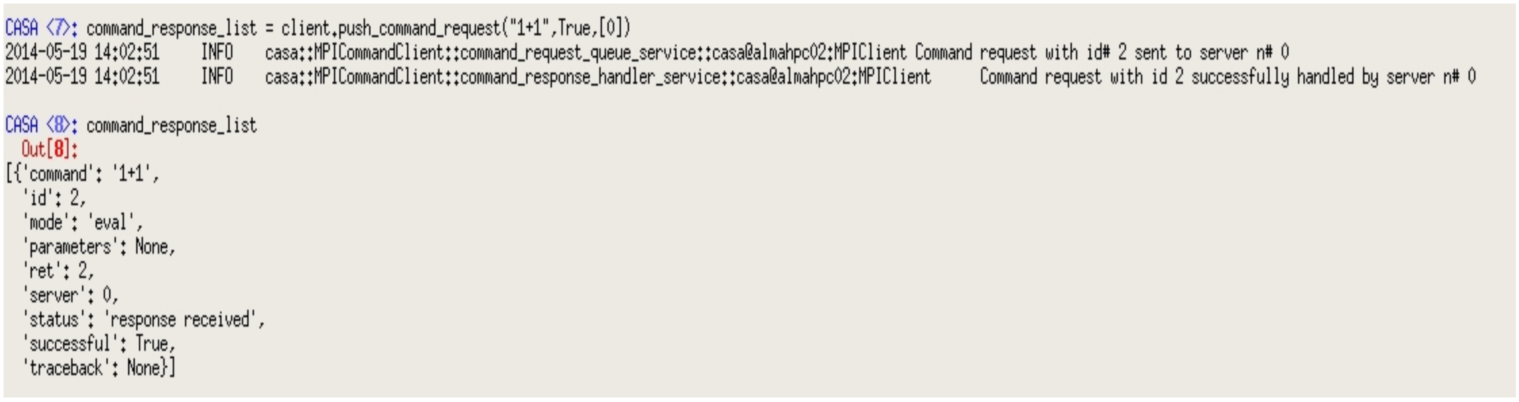
  command_response_list = client.push_command_request(*"1+1"*,True,None)



```
CASA <5>: command_response_list = client.push_command_request("1+1",True,None)
2014-05-19 14:02:32     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 1 sent to server n# 2
2014-05-19 14:02:33     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 1 successfully handled by server n# 2

CASA <6>: command_response_list
  Out[6]:
[{'command': '1+1',
  'id': 1,
  'mode': 'eval',
  'parameters': None,
  'ret': 2,
  'server': 2,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

- **ex 2:** Blocking mode, string parameters, defined target server

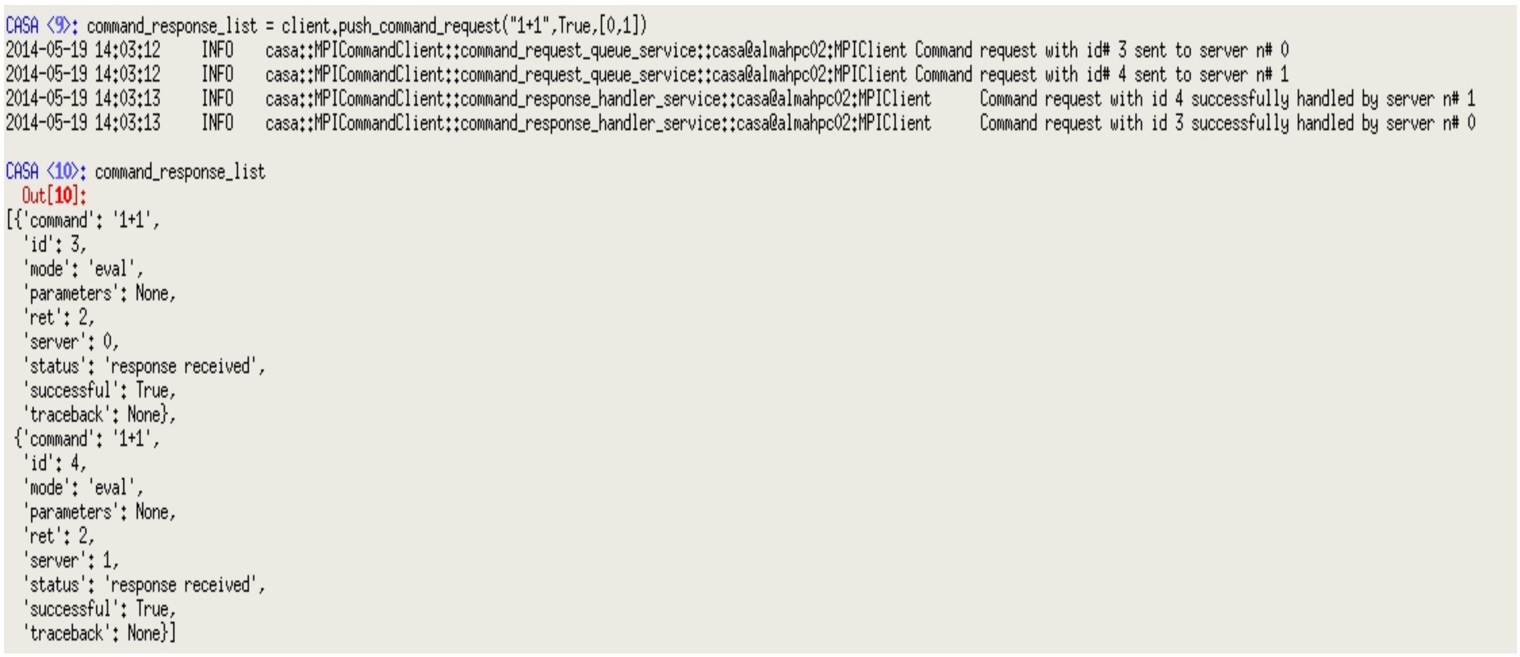
  command_response_list = client.push_command_request(*"1+1"*,True,[0])



```
CASA <7>: command_response_list = client.push_command_request("1+1",True,[0])
2014-05-19 14:02:51     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 2 sent to server n# 0
2014-05-19 14:02:51     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 2 successfully handled by server n# 0

CASA <8>: command_response_list
  Out[8]:
[{'command': '1+1',
  'id': 2,
  'mode': 'eval',
  'parameters': None,
  'ret': 2,
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

- **ex 3:** Blocking mode, string parameters, multiple target server

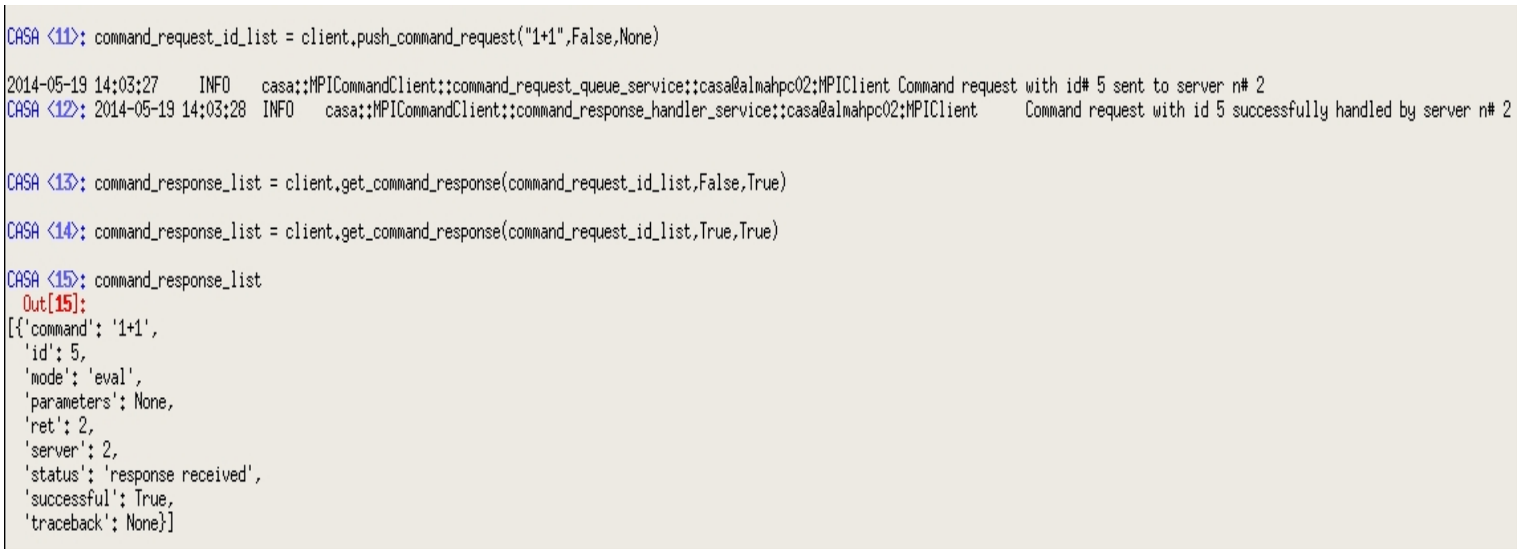
```
command_response_list = client.push_command_request("1+1",True,[0,1])
```



```
CASA <9>: command_response_list = client.push_command_request("1+1",True,[0,1])
2014-05-19 14:03:12     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 3 sent to server n# 0
2014-05-19 14:03:12     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 4 sent to server n# 1
2014-05-19 14:03:13     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 4 successfully handled by server n# 1
2014-05-19 14:03:13     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 3 successfully handled by server n# 0

CASA <10>: command_response_list
  Out[10]:
[{'command': '1+1',
  'id': 3,
  'mode': 'eval',
  'parameters': None,
  'ret': 2,
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None},
 {'command': '1+1',
  'id': 4,
  'mode': 'eval',
  'parameters': None,
  'ret': 2,
  'server': 1,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

- **ex 4:** Non-Blocking mode, string parameters, undefined target server

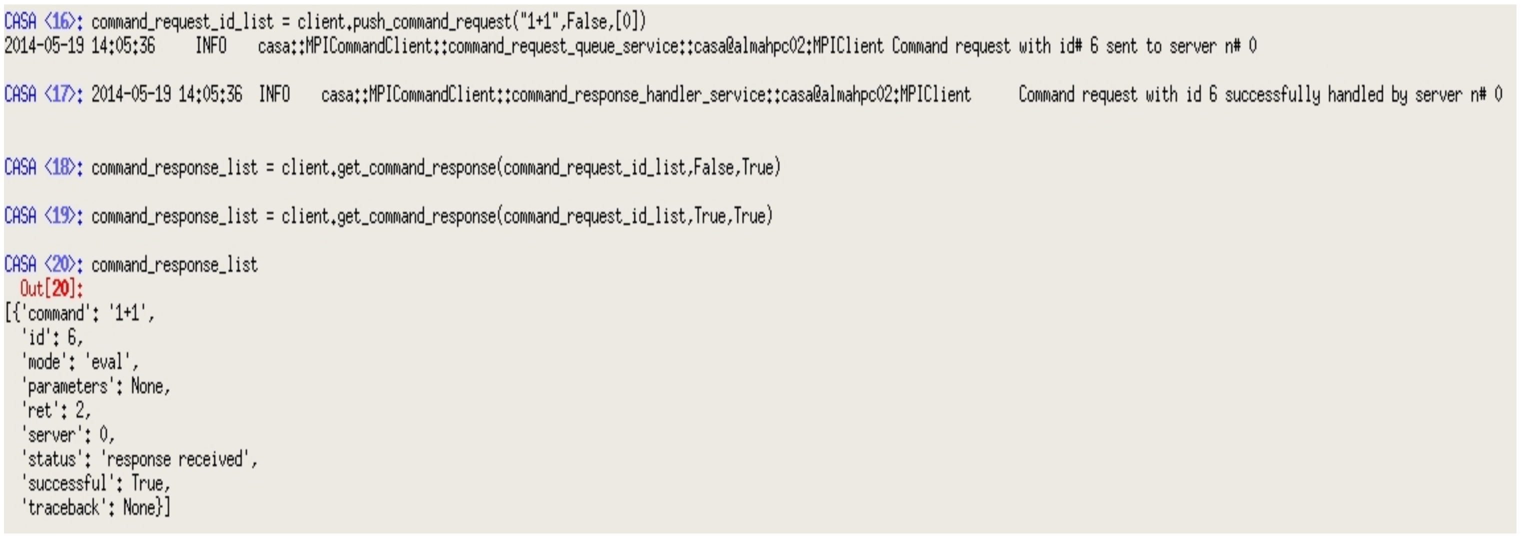
```
command_request_id_list = self.client.push_command_request("1+1",False,None)
# Try to get responses before time in non-blocking more
command_response_list = client.get_command_response(command_request_id_list,False,True)
# Get response in blocking mode
command_response_list = client.get_command_response(command_request_id_list,True,True)
```



```
CASA <11>: command_request_id_list = client.push_command_request("1+1",False,None)

2014-05-19 14:03:27     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 5 sent to server n# 2
CASA <12>: 2014-05-19 14:03:28  INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 5 successfully handled by server n# 2


CASA <13>: command_response_list = client.get_command_response(command_request_id_list,False,True)

CASA <14>: command_response_list = client.get_command_response(command_request_id_list,True,True)

CASA <15>: command_response_list
  Out[15]:
[{'command': '1+1',
  'id': 5,
  'mode': 'eval',
  'parameters': None,
  'ret': 2,
  'server': 2,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

- **ex 5:** Non-Blocking mode, string parameters, defined target server

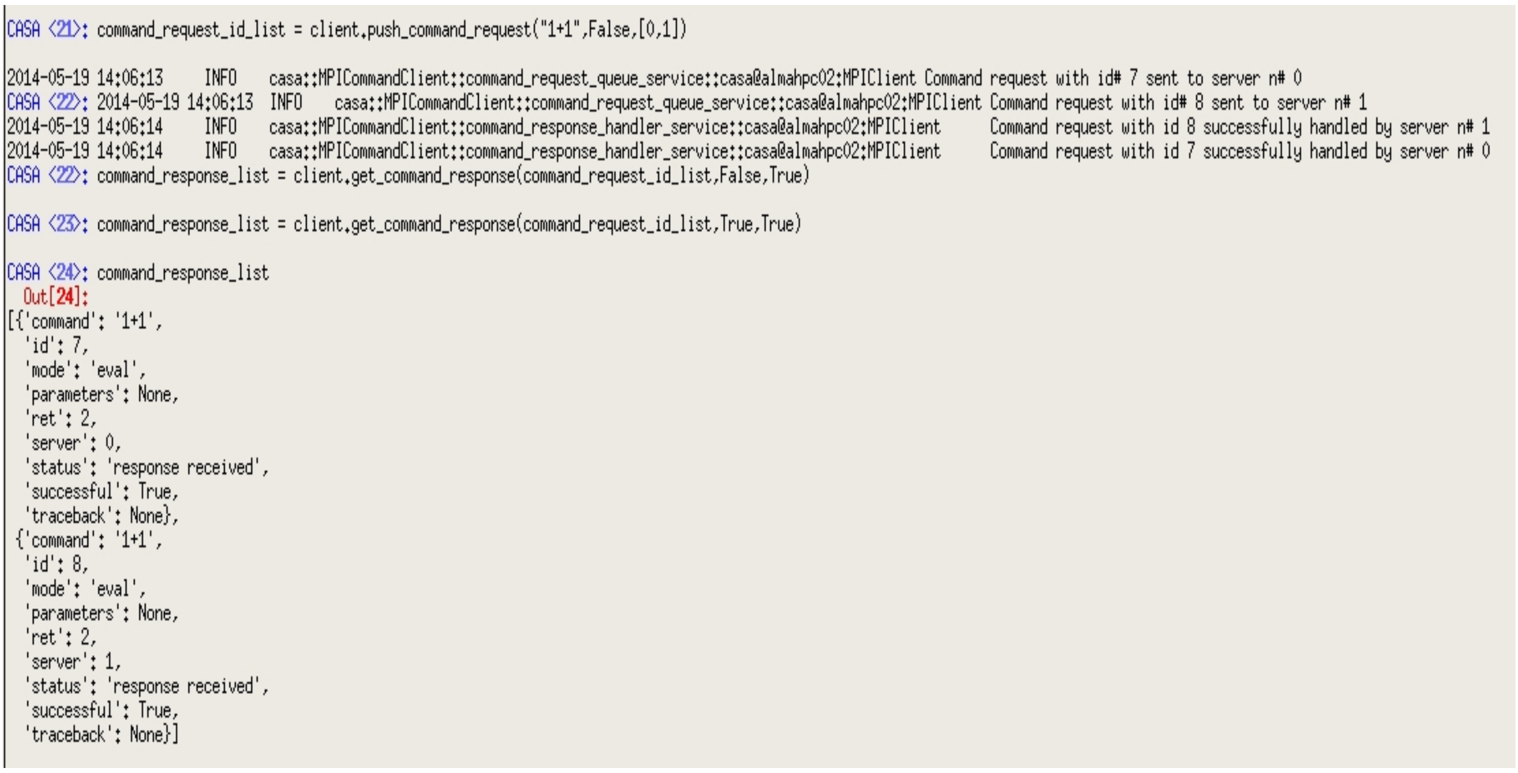
```
command_request_id_list = self.client.push_command_request("1+1",False,[0])
# Try to get responses before time in non-blocking more
command_response_list = self.client.get_command_response(command_request_id_list,False,True)
# Get response in blocking mode
command_response_list
```



```
CASA <16>: command_request_id_list = client.push_command_request("1+1",False,[0])
2014-05-19 14:05:36     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 6 sent to server n# 0

CASA <17>: 2014-05-19 14:05:36  INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 6 successfully handled by server n# 0


CASA <18>: command_response_list = client.get_command_response(command_request_id_list,False,True)

CASA <19>: command_response_list = client.get_command_response(command_request_id_list,True,True)

CASA <20>: command_response_list
  Out[20]:
[{'command': '1+1',
  'id': 6,
  'mode': 'eval',
  'parameters': None,
  'ret': 2,
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

- **ex 6:** Non-Blocking mode, string parameters, multiple target servers

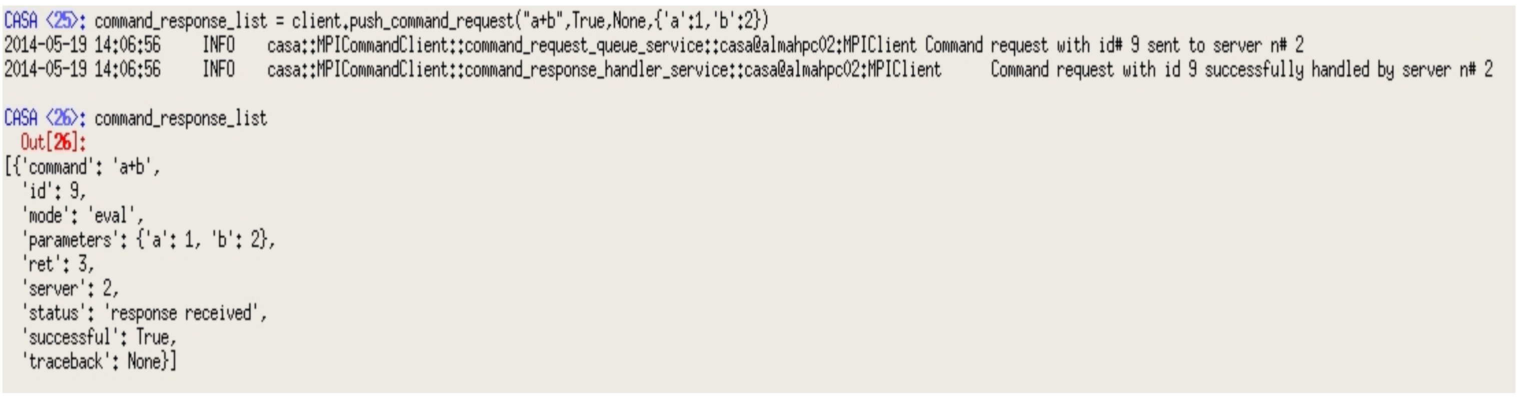
```
command_request_id_list = self.client.push_command_request("1+1",False,[0,1])
# Try to get responses before time in non-blocking more
command_response_list = self.client.get_command_response(command_request_id_list,False,True)
# Get response in blocking mode
command_response_list = self.client.get_command_response(command_request_id_list,True,True)
```



```
CASA <21>: command_request_id_list = client.push_command_request("1+1",False,[0,1])

2014-05-19 14:06:13     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 7 sent to server n# 0
CASA <22>: 2014-05-19 14:06:13  INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 8 sent to server n# 1
2014-05-19 14:06:14     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 8 successfully handled by server n# 1
2014-05-19 14:06:14     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 7 successfully handled by server n# 0
CASA <22>: command_response_list = client.get_command_response(command_request_id_list,False,True)

CASA <23>: command_response_list = client.get_command_response(command_request_id_list,True,True)

CASA <24>: command_response_list
  Out[24]:
[{'command': '1+1',
  'id': 7,
  'mode': 'eval',
  'parameters': None,
  'ret': 2,
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None},
 {'command': '1+1',
  'id': 8,
  'mode': 'eval',
  'parameters': None,
  'ret': 2,
  'server': 1,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

- **ex 7:** Blocking mode, dictionary parameters, undefined target server

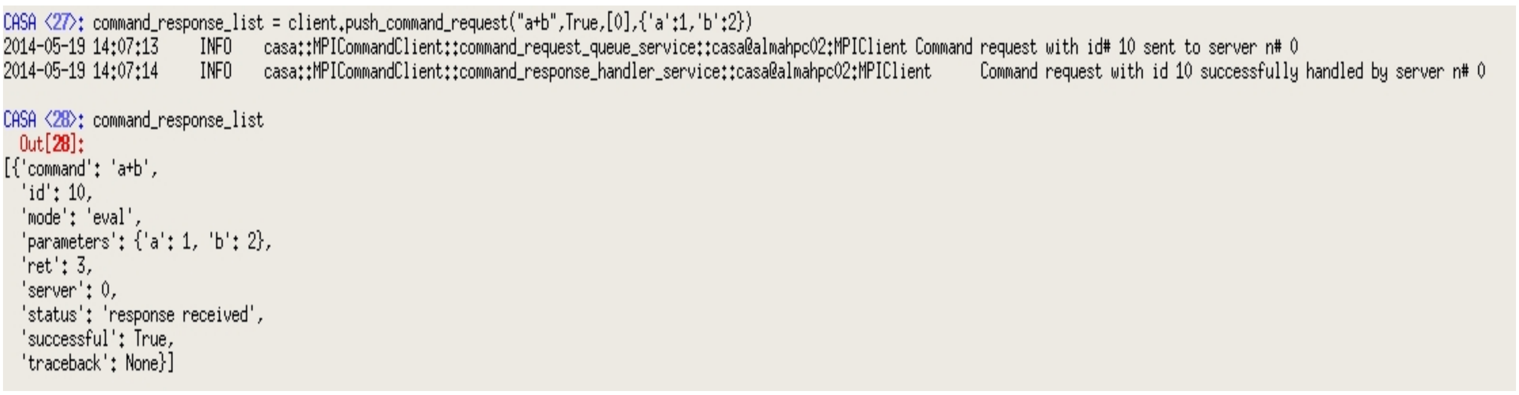
```
command_response_list = self.client.push_command_request("a+b",True,None,{'a':1,'b':2})
```



```
CASA <25>: command_response_list = client.push_command_request("a+b",True,None,{'a':1,'b':2})
2014-05-19 14:06:56     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 9 sent to server n# 2
2014-05-19 14:06:56     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 9 successfully handled by server n# 2

CASA <26>: command_response_list
  Out[26]:
[{'command': 'a+b',
  'id': 9,
  'mode': 'eval',
  'parameters': {'a': 1, 'b': 2},
  'ret': 3,
  'server': 2,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

- **ex 8:** Blocking mode, dictionary parameters, defined target server

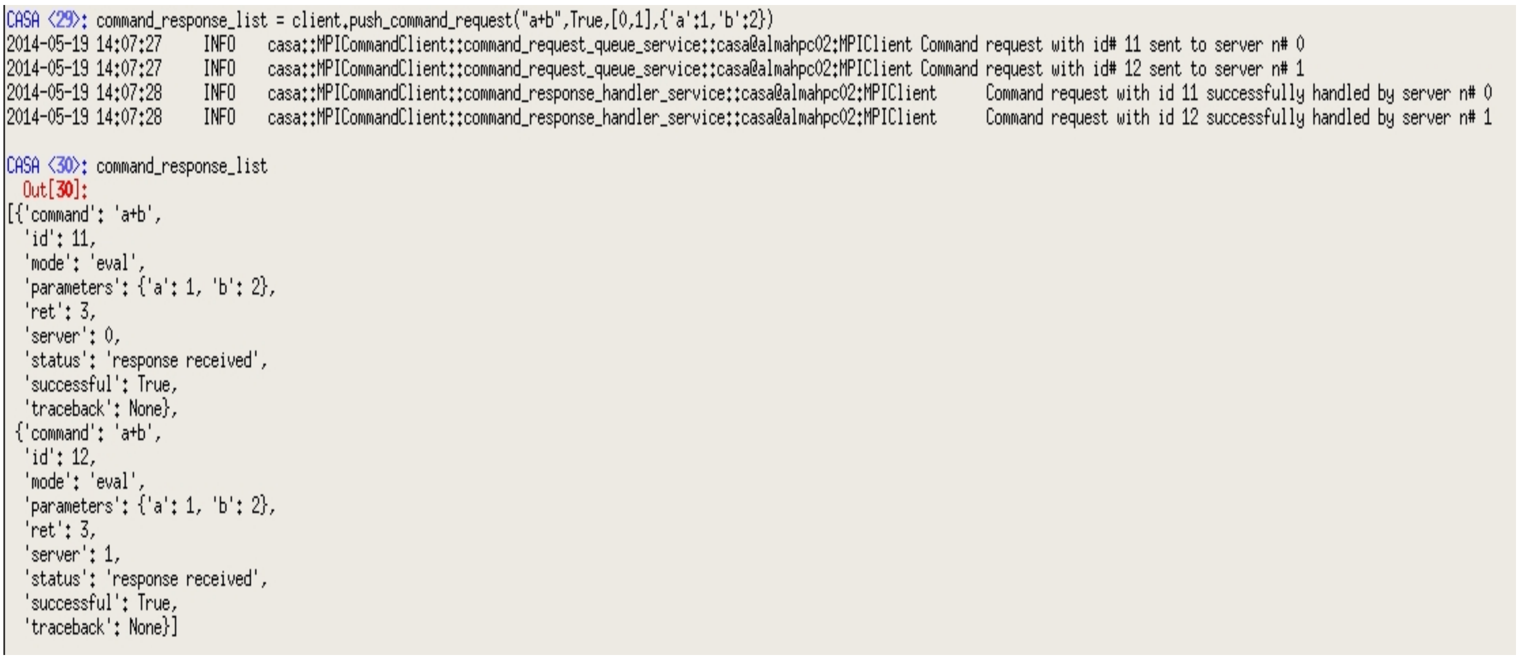
command_response_list = *self*.client.push_command_request(*"a+b"*,True,[0],{*'a'*:1,*'b'*:2})



```
CASA <27>: command_response_list = client.push_command_request("a+b",True,[0],{'a':1,'b':2})
2014-05-19 14:07:13     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 10 sent to server n# 0
2014-05-19 14:07:14     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 10 successfully handled by server n# 0

CASA <28>: command_response_list
  Out[28]:
[{'command': 'a+b',
  'id': 10,
  'mode': 'eval',
  'parameters': {'a': 1, 'b': 2},
  'ret': 3,
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

- **ex 9:** Blocking mode, dictionary parameters, multiple target server

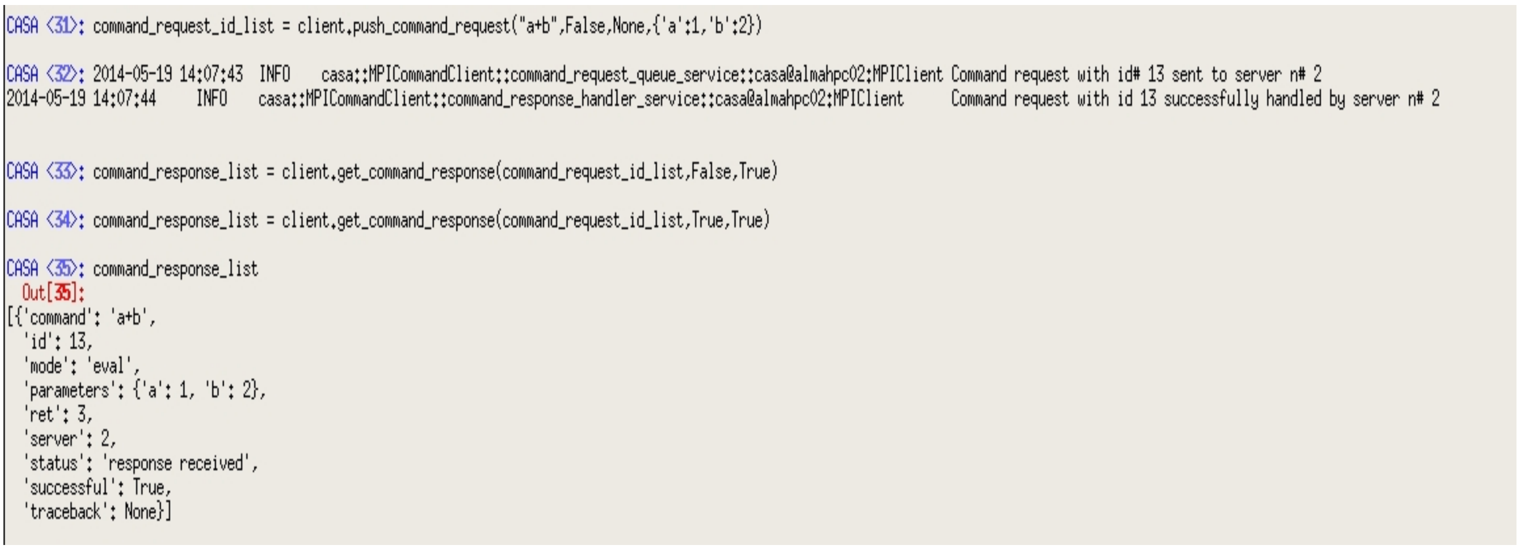
command_response_list = *self*.client.push_command_request(*"a+b"*,True,[0,1],{*'a'*:1,*'b'*:2})



```
CASA <29>: command_response_list = client.push_command_request("a+b",True,[0,1],{'a':1,'b':2})
2014-05-19 14:07:27     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 11 sent to server n# 0
2014-05-19 14:07:27     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 12 sent to server n# 1
2014-05-19 14:07:28     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 11 successfully handled by server n# 0
2014-05-19 14:07:28     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 12 successfully handled by server n# 1

CASA <30>: command_response_list
  Out[30]:
[{'command': 'a+b',
  'id': 11,
  'mode': 'eval',
  'parameters': {'a': 1, 'b': 2},
  'ret': 3,
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None},
 {'command': 'a+b',
  'id': 12,
  'mode': 'eval',
  'parameters': {'a': 1, 'b': 2},
  'ret': 3,
  'server': 1,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

- **ex 10:** Non-Blocking mode, dictionary parameters, undefined target server

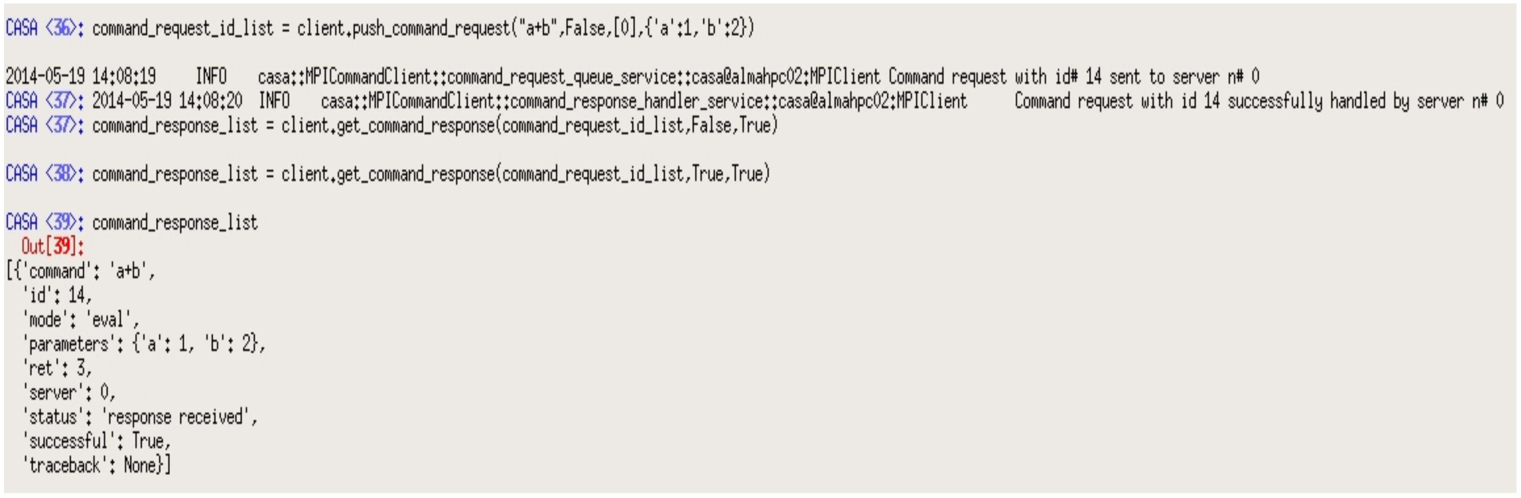
```
command_request_id_list = self.client.push_command_request("a+b",False,None,{'a':1,'b':2})
# Try to get responses before time in non-blocking more
command_response_list = self.client.get_command_response(command_request_id_list,False,True)
# Get response in blocking mode
command_response_list = self.client.get_command_response(command_request_id_list,True,True)
```



```
CASA <31>: command_request_id_list = client.push_command_request("a+b",False,None,{'a':1,'b':2})

CASA <32>: 2014-05-19 14:07:43  INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 13 sent to server n# 2
2014-05-19 14:07:44     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 13 successfully handled by server n# 2


CASA <33>: command_response_list = client.get_command_response(command_request_id_list,False,True)

CASA <34>: command_response_list = client.get_command_response(command_request_id_list,True,True)

CASA <35>: command_response_list
  Out[35]:
[{'command': 'a+b',
  'id': 13,
  'mode': 'eval',
  'parameters': {'a': 1, 'b': 2},
  'ret': 3,
  'server': 2,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

- **ex 11:** Non-Blocking mode, dictionary parameters, defined target server

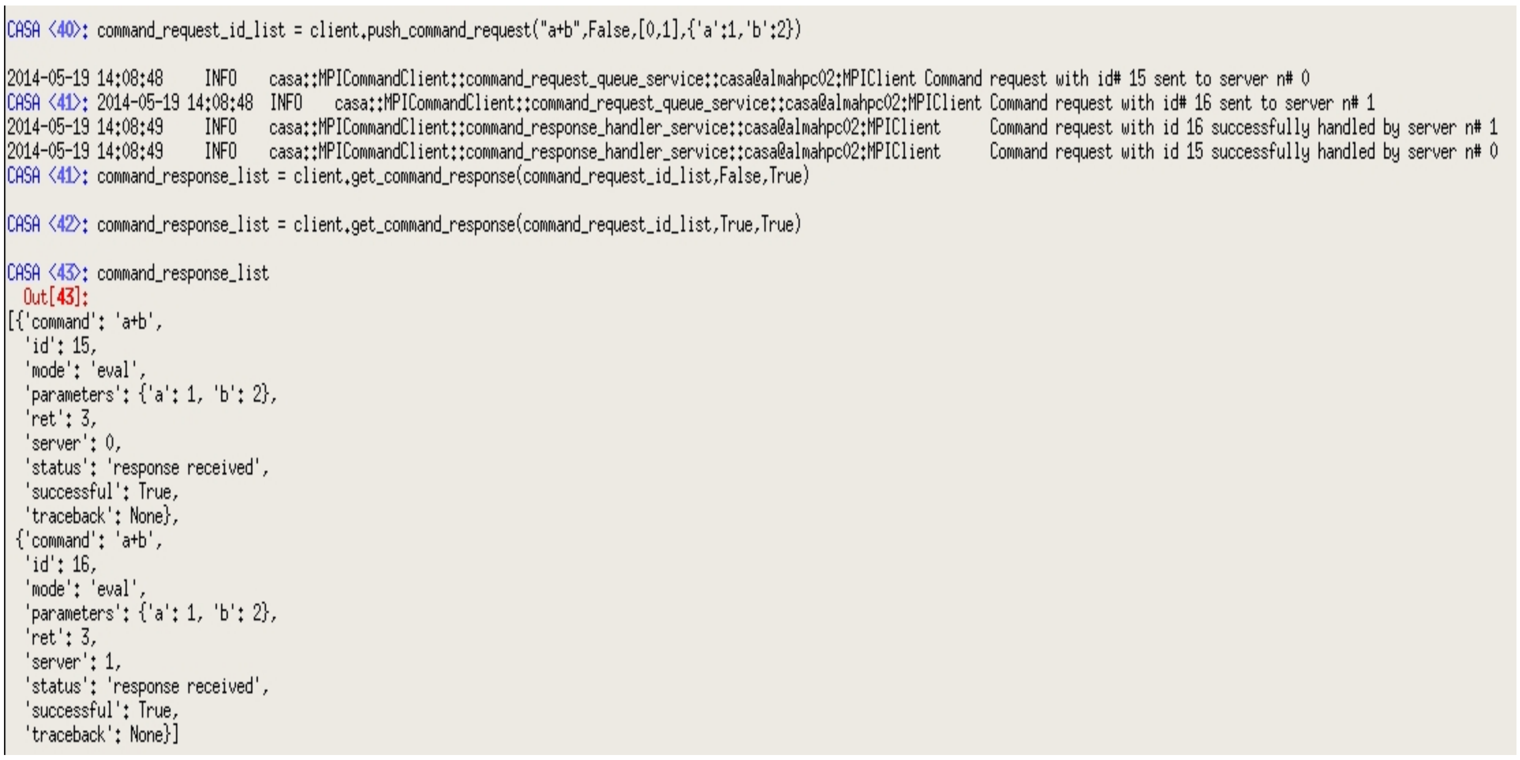
```
command_request_id_list = self.client.push_command_request("a+b",False,[0],{'a':1,'b':2})
# Try to get responses before time in non-blocking more
command_response_list = self.client.get_command_response(command_request_id_list,False,True)
# Get response in blocking mode
command_response_list = self.client.get_command_response(command_request_id_list,True,True)
```



```
CASA <36>: command_request_id_list = client.push_command_request("a+b",False,[0],{'a':1,'b':2})

2014-05-19 14:08:19     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 14 sent to server n# 0
CASA <37>: 2014-05-19 14:08:20  INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 14 successfully handled by server n# 0
CASA <37>: command_response_list = client.get_command_response(command_request_id_list,False,True)

CASA <38>: command_response_list = client.get_command_response(command_request_id_list,True,True)

CASA <39>: command_response_list
  Out[39]:
[{'command': 'a+b',
  'id': 14,
  'mode': 'eval',
  'parameters': {'a': 1, 'b': 2},
  'ret': 3,
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

- **ex 12:** Non-Blocking mode, dictionary parameters, multiple target server

```
command_request_id_list = self.client.push_command_request("a+b",False,[0,1],{'a':1,'b':2})
# Try to get responses before time in non-blocking more
command_response_list = self.client.get_command_response(command_request_id_list,False,True)
# Get response in blocking mode
command_response_list = self.client.get_command_response(command_request_id_list,True,True)
```



```
CASA <40>: command_request_id_list = client.push_command_request("a+b",False,[0,1],{'a':1,'b':2})

2014-05-19 14:08:48     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 15 sent to server n# 0
CASA <41>: 2014-05-19 14:08:48  INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 16 sent to server n# 1
2014-05-19 14:08:49     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 16 successfully handled by server n# 1
2014-05-19 14:08:49     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 15 successfully handled by server n# 0
CASA <41>: command_response_list = client.get_command_response(command_request_id_list,False,True)

CASA <42>: command_response_list = client.get_command_response(command_request_id_list,True,True)

CASA <43>: command_response_list
  Out[43]:
[{'command': 'a+b',
  'id': 15,
  'mode': 'eval',
  'parameters': {'a': 1, 'b': 2},
  'ret': 3,
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None},
 {'command': 'a+b',
  'id': 16,
  'mode': 'eval',
  'parameters': {'a': 1, 'b': 2},
  'ret': 3,
  'server': 1,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

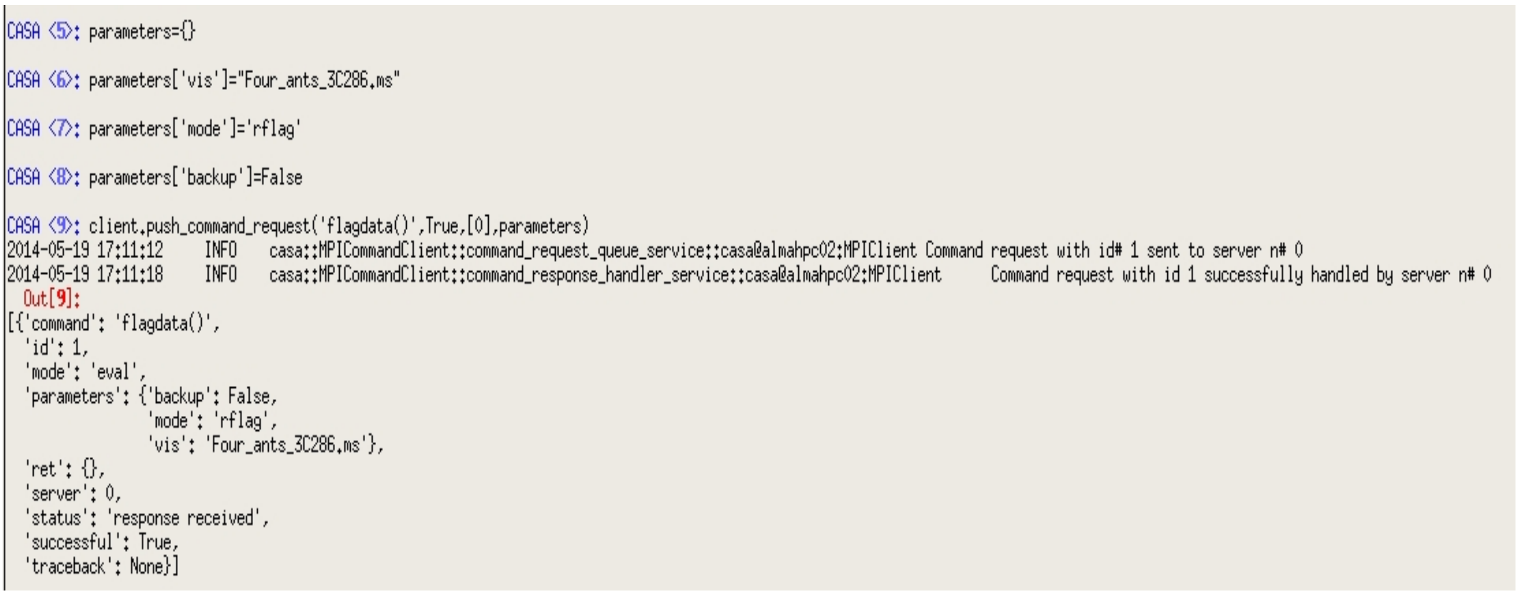
5.8 MPICommandClient CASA tasks examples

Executing a CASA task command does not differ at all from executing a basic python command, but for the sake of completeness I have prepared some basic examples here:

NOTE: mpi4py is already integrated with ParallelTaskHelper, so it is not necessary to do anything (in particular initialize MPICommandClient) in order to run tasks in parallel. ParalleTaskHelper re-uses or initializes the MPICommandClient singleton if necessary.

- ex 1.: flagdata rflag, string parameters, blocking mode, defined server

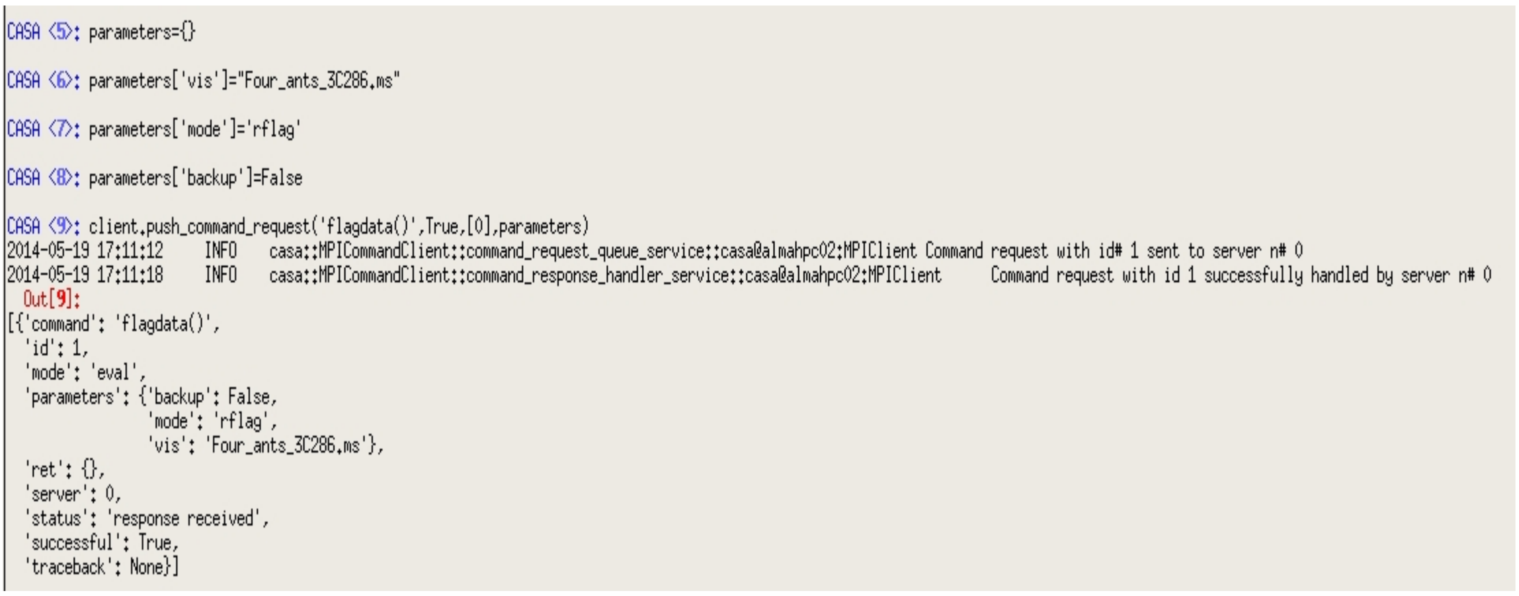
```
client.push_command_request("flagdata(vis='Four_ants_3C286.ms',mode='summary')",True,[1])
```



```
CASA <5>: parameters={}

CASA <6>: parameters['vis']="Four_ants_3C286.ms"

CASA <7>: parameters['mode']='rflag'

CASA <8>: parameters['backup']=False

CASA <9>: client.push_command_request('flagdata()',True,[0],parameters)
2014-05-19 17:11:12     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 1 sent to server n# 0
2014-05-19 17:11:18     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 1 successfully handled by server n# 0
  Out[9]:
[{'command': 'flagdata()',
  'id': 1,
  'mode': 'eval',
  'parameters': {'backup': False,
                 'mode': 'rflag',
                 'vis': 'Four_ants_3C286.ms'},
  'ret': {},
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

- ex 2.: flagdata summary, dict parameters, blocking mode, defined server

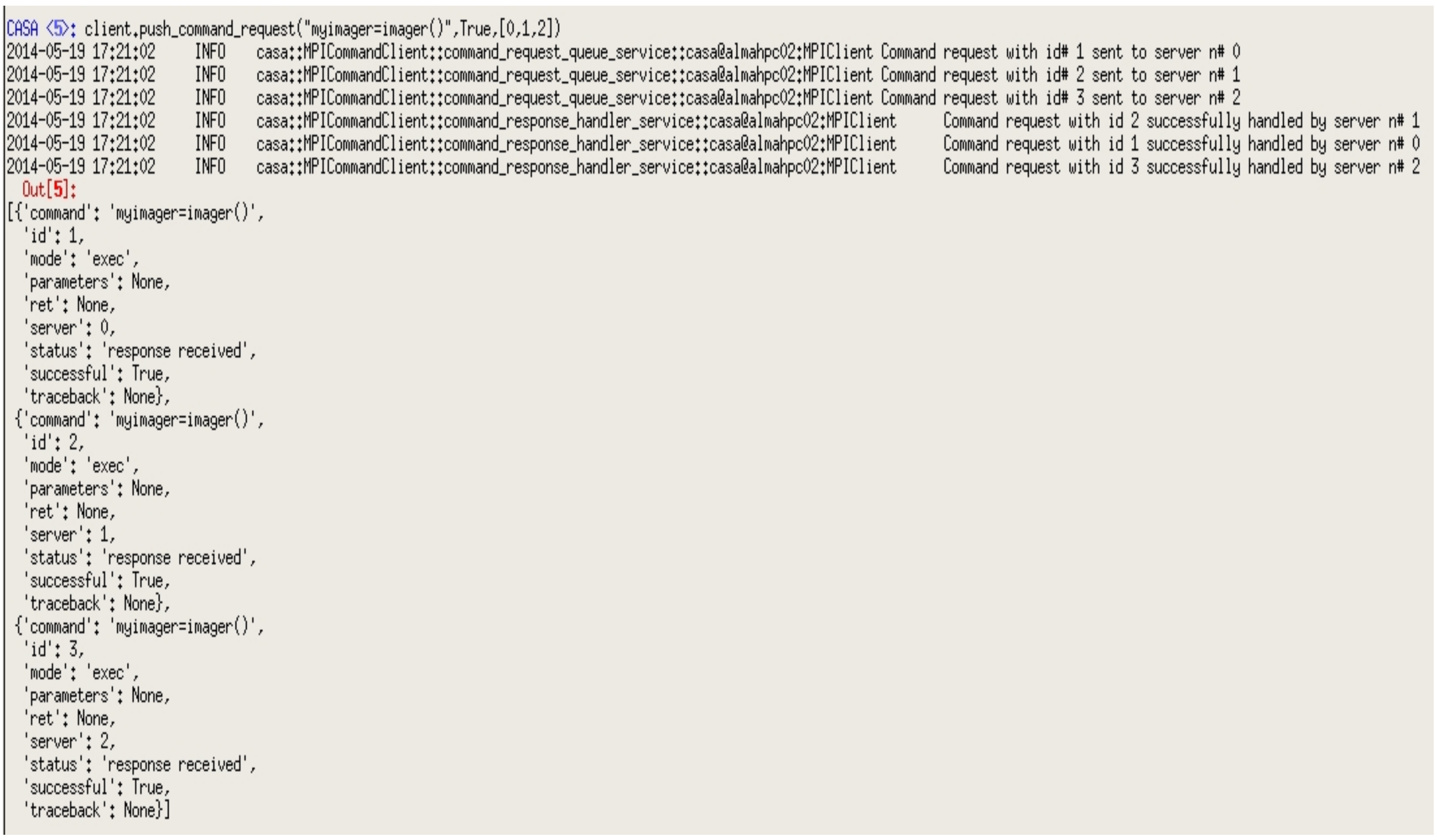
```
parameters = {'vis':'Four_ants_3C286.ms','mode':'rflag','flagbackup':False}
client.push_command_request('flagdata()',True,[0],parameters)
```



```
CASA <5>: parameters={}

CASA <6>: parameters['vis']="Four_ants_3C286.ms"

CASA <7>: parameters['mode']='rflag'

CASA <8>: parameters['backup']=False

CASA <9>: client.push_command_request('flagdata()',True,[0],parameters)
2014-05-19 17:11:12     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 1 sent to server n# 0
2014-05-19 17:11:18     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 1 successfully handled by server n# 0
  Out[9]:
[{'command': 'flagdata()',
  'id': 1,
  'mode': 'eval',
  'parameters': {'backup': False,
                 'mode': 'rflag',
                 'vis': 'Four_ants_3C286.ms'},
  'ret': {},
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

- ex 3.: create an imager tool in each remote server, blocking mode

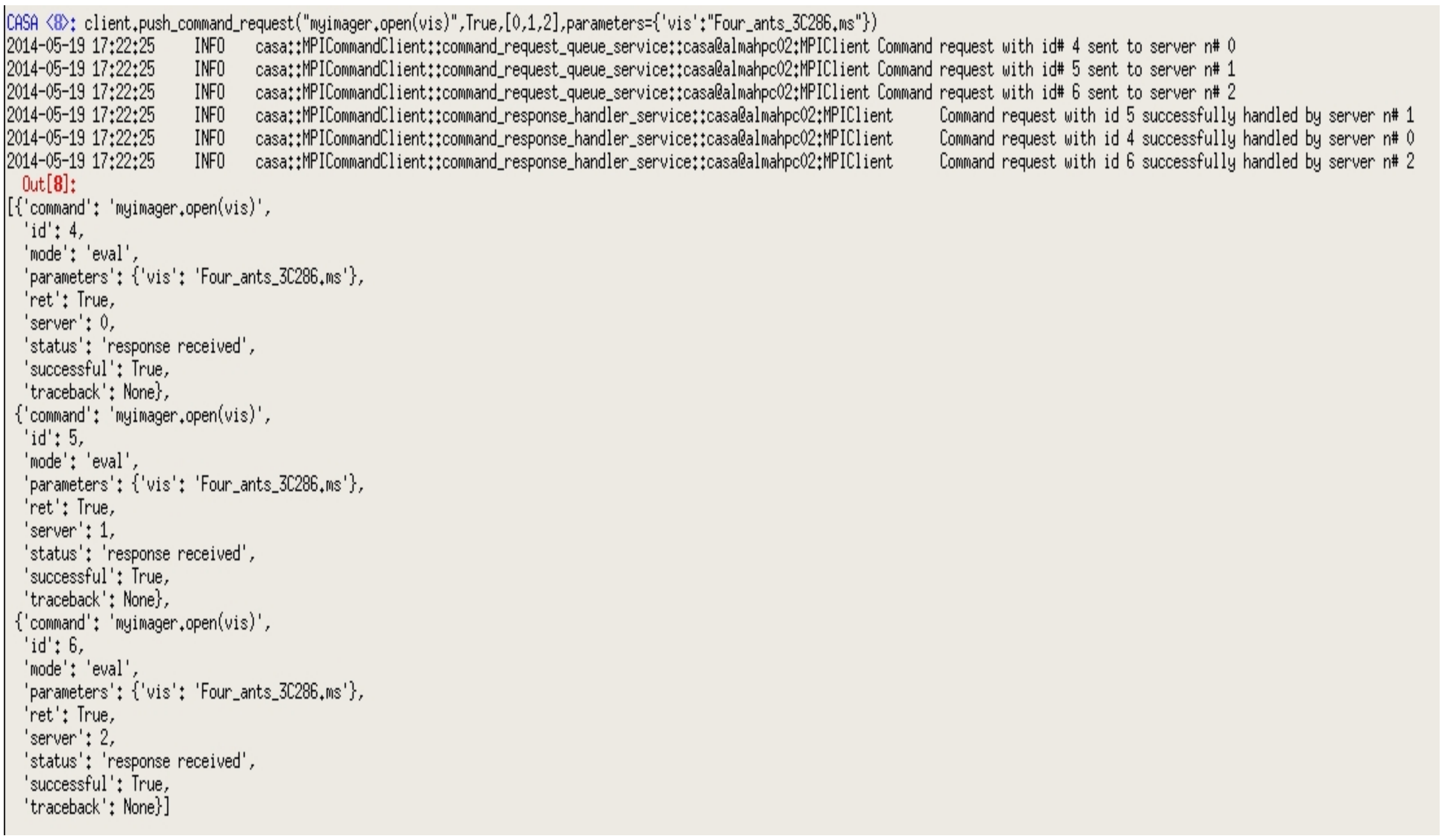
```
client.push_command_request('myimager=imager()',True,[0,1,2])
```



```
CASA <5>: client.push_command_request("myimager=imager()",True,[0,1,2])
2014-05-19 17:21:02     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 1 sent to server n# 0
2014-05-19 17:21:02     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 2 sent to server n# 1
2014-05-19 17:21:02     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 3 sent to server n# 2
2014-05-19 17:21:02     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 2 successfully handled by server n# 1
2014-05-19 17:21:02     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 1 successfully handled by server n# 0
2014-05-19 17:21:02     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 3 successfully handled by server n# 2
  Out[5]:
[{'command': 'myimager=imager()',
  'id': 1,
  'mode': 'exec',
  'parameters': None,
  'ret': None,
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None},
 {'command': 'myimager=imager()',
  'id': 2,
  'mode': 'exec',
  'parameters': None,
  'ret': None,
  'server': 1,
  'status': 'response received',
  'successful': True,
  'traceback': None},
 {'command': 'myimager=imager()',
  'id': 3,
  'mode': 'exec',
  'parameters': None,
  'ret': None,
  'server': 2,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

- ex 4.: open newly created imager tool in each remote server, dict parameters, blocking mode

```
parameters={'vis':'Four_ants_3C286.ms'}
client.push_command_request('myimager.open(vis)',True,[0,1,2],parameters)
```



```
CASA <8>: client.push_command_request("myimager.open(vis)",True,[0,1,2],parameters={'vis':"Four_ants_3C286.ms"})
2014-05-19 17:22:25     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 4 sent to server n# 0
2014-05-19 17:22:25     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 5 sent to server n# 1
2014-05-19 17:22:25     INFO    casa::MPICommandClient::command_request_queue_service::casa@almahpc02:MPIClient Command request with id# 6 sent to server n# 2
2014-05-19 17:22:25     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 5 successfully handled by server n# 1
2014-05-19 17:22:25     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 4 successfully handled by server n# 0
2014-05-19 17:22:25     INFO    casa::MPICommandClient::command_response_handler_service::casa@almahpc02:MPIClient      Command request with id 6 successfully handled by server n# 2
  Out[8]:
[{'command': 'myimager.open(vis)',
  'id': 4,
  'mode': 'eval',
  'parameters': {'vis': 'Four_ants_3C286.ms'},
  'ret': True,
  'server': 0,
  'status': 'response received',
  'successful': True,
  'traceback': None},
 {'command': 'myimager.open(vis)',
  'id': 5,
  'mode': 'eval',
  'parameters': {'vis': 'Four_ants_3C286.ms'},
  'ret': True,
  'server': 1,
  'status': 'response received',
  'successful': True,
  'traceback': None},
 {'command': 'myimager.open(vis)',
  'id': 6,
  'mode': 'eval',
  'parameters': {'vis': 'Four_ants_3C286.ms'},
  'ret': True,
  'server': 2,
  'status': 'response received',
  'successful': True,
  'traceback': None}]
```

5.9 mpi4casa unit test suit

mpi4casa has a complete unit test suit, with more than 60 tests, which can be launched using CASA's unit test tools, specifying a configurable number of parallel processors, or even specifying a distributed environment via hostfile.

```
mpirun -n 4 casapy --nologger --log2term -c runUnitTest.py test_mpi4casa
```

```
mpirun -n 4 -hostfile host casapy --nologger --log2term -c runUnitTest.py test_mpi4casa
```

The unit test suit covers all combinations of MPICommandClient/Server usage:

- blocking and non-blocking mode
- execution mode
  - with return variable (eval)
  - w/o return variable (exec)
- target server specification
  - undefined server
  - defined server
  - multiple servers
  - busy/iddle server
- parameters specification
  - Via command string
  - Via parameter dictionary
- error handling cases:
  - ImportError
  - NameError
  - ZeroDivisionError
  - TypeError
  - IndexError
  - KeyError
- Server timeout handling
  - Server timeout
  - Assignment to timeout server
  - Timeout recovery
- Singleton behavior
  - Invalid MPICommandClient/Server instantiation
  - MPICommandClient destruction

It also covers integration via ParallelTaskHelper with the following tasks:

- flagdata
- applycal
- uvcontsub
- setjy

6 Packaging and cluster integration

During the CASA technical forum conferences it was decided to choose OpenMPI as a preferred MPI implementation for CASA. Following up this decision we have prepared a build, packaging and integration procedure to run CASA in the standard NRAO/ESO/NAOJ clusters which currently are configured as follows:

- RHEL 6.X 64b OS
- Access to a Lustre shared file system via an Infiniband connection as
- Use Torque as resource manager, in particular for dispatching batch jobs.

One of the advantages of OpenMPI is that it aims to support all interconnects including Infiniband transparently, and also uses a hostfile format compatible with that produced automatically by Torque, so it is not necessary to create a custom hostfile like in the case of the previous iPython -based clusters.

6.1 Stand-alone MPI CASA build

Section 4 describes how to install and configure all the necessary packages to enable MPI in CASA, however, it assumes root access to the system, and this condition cannot be guaranteed in the general case.

The following procedure aims to prepare a stand-alone version of CASA together with all the needed packages to bypass the default system OpenMPI configuration (in case it exists) and compile both, auxiliary packages (like mpi4py and Boost MPI) and the CASA source coda against our local OpenMPI installation.

For a reference the 3rd party package versions are described as follow:

Package name	Version	Usage
Open MPI	1.6.5	MPI Implementation
bjam	3.1.8	Boost build manager
boost (including boost MPI)	1.41.0	C++ MPI binding
mpi4py	1.3.1	Python MPI binding

All the steps described below are contained in the following scripts available to download:

- Script to build CASA using a custom configured version of OpenMPI
  - build-casa-source-with-mpi-packages.sh

- Scripts to prepare a stand-alone package from the previous step:
  - pack-casa-3rd-party-non-mpi-packages.sh
  - findreplace.py

- Example of cluster.req and qsub.sh scripts to commit a CASA MPI batch job to Torque:
  - cluster.req
  - qsub.sh

6.1.1 Prepare environment and build area

NOTE: The installation path must be identical in the build and target machine.

```
# Custom configuration
export BUILD_CORES=8
export ARCH="linux_64b"
export REPO="trunk"
export WORKAREA="/lustre/user/mpi"

# Download CASA repositories
mkdir $WORKAREA
cd $WORKAREA
svn co https://svn.cv.nrao.edu/svn/casa/trunk
export BASECAMP="$WORKAREA/$REPO"

# Dowload CASA data repository
export DATA="$BASECAMP/data"
mkdir $DATA
cd $DATA
svn co https://svn.cv.nrao.edu/svn/casa-data/trunk/ephemerides
svn co https://svn.cv.nrao.edu/svn/casa-data/trunk/geodetic

# Prepare build areas
mkdir $BASECAMP/packages
mkdir $BASECAMP/build
mkdir $BASECAMP/casacore/build
mkdir $BASECAMP/code/build
mkdir $BASECAMP/gcwrap/build
mkdir $BASECAMP/asap/build
export BUILDROOT="$BASECAMP/$ARCH"

# Update path
export PATH="$BUILDROOT/bin:$PATH"
export LD_LIBRARY_PATH="$BUILDROOT/lib:$LD_LIBRARY_PATH"
```

6.1.2 Build 3rd party packages and install them under $CASAPATH

```
# Open MPI 1.6.5
cd $BASECAMP/packages
wget wget http://www.open-mpi.org/software/ompi/v1.6/downloads/openmpi-1.6.5.tar.bz2
bunzip2 openmpi-1.6.5.tar.bz2
tar -xvf openmpi-1.6.5.tar
cd openmpi-1.6.5
./configure --enable-mpi-thread-multiple --enable-opal-multi-threads --prefix=$BUILDROOT
make all -j$BUILD_CORES
make install

# bjam 3.1.18
cd $BASECAMP/packages
wget http://sourceforge.net/projects/boost/files/boost-jam/3.1.18/boost-jam-3.1.18.tgz
tar -xzvf boost-jam-3.1.18.tgz
cd boost-jam-3.1.18
./build.sh
cp bin.linuxx86_64/bjam $BUILDROOT/bin/

# boost 1.41.0
cd $BASECAMP/packages
wget http://sourceforge.net/projects/boost/files/boost/1.41.0/boost_1_41_0.tar.gz
tar -xzvf boost_1_41_0.tar.gz
cd boost_1_41_0
echo "using python : 2.7 : /usr/lib64/casa/01/bin/python ;" >> ./tools/build/v2/user-
config.jam
echo "using mpi ;" >> ./tools/build/v2/user-config.jam
bjam --debug-configuration --prefix=$BUILDROOT install

# mpi4py 1.3.1
cd $BASECAMP/packages
wget http://mpi4py.googlecode.com/files/mpi4py-1.3.1.tar.gz
tar -xzvf mpi4py-1.3.1.tar.gz
cd mpi4py-1.3.1
/usr/lib64/casa/01/bin/python setup.py build
/usr/lib64/casa/01/bin/python setup.py install --prefix=$BUILDROOT
```

6.1.3 Build CASA

NOTE: ASAP CMake settings don't work with Open MPI Fortran compiler in the section to find Fortran libs, therefore ASAP Fortran code is not compiled using the OpenMPI compilers for the time being

```
# casacore
cd $BASECAMP/casacore/build
cmake -DCMAKE_INSTALL_PREFIX=$BUILDROOT -DCMAKE_Fortran_COMPILER=$BUILDROOT/bin/mpif90 -
DCMAKE_C_COMPILER=$BUILDROOT/bin/mpicc -DCMAKE_CXX_COMPILER=$BUILDROOT/bin/mpic++ ..
make all -j$BUILD_CORES
make install

# code
cd $BASECAMP/code/build
cmake -DCMAKE_INSTALL_PREFIX=$BUILDROOT -DCMAKE_Fortran_COMPILER=$BUILDROOT/bin/mpif90 -
DCMAKE_C_COMPILER=$BUILDROOT/bin/mpicc -DCMAKE_CXX_COMPILER=$BUILDROOT/bin/mpic++ -
DBOOST_ROOT=$BUILDROOT -DUSE_ALMAWVR=ON ..
make all -j$BUILD_CORES

# Source casainit
source $BASECAMP/casainit.sh

# gcwrap
cd $BASECAMP/gcwrap/build
cmake -DCMAKE_INSTALL_PREFIX=$BUILDROOT -DCMAKE_Fortran_COMPILER=$BUILDROOT/bin/mpif90 -
DCMAKE_C_COMPILER=$BUILDROOT/bin/mpicc -DCMAKE_CXX_COMPILER=$BUILDROOT/bin/mpic++ -
DCMAKE_SWIG_FLAGS="-threads" -DBOOST_ROOT=$BUILDROOT ..
make all -j$BUILD_CORES

# asap
cd $BASECAMP/asap/build
cmake -DCMAKE_INSTALL_PREFIX=$BUILDROOT -DCMAKE_C_COMPILER=$BUILDROOT/bin/mpicc -
DCMAKE_CXX_COMPILER=$BUILDROOT/bin/mpic++ -DBOOST_ROOT=$BUILDROOT ..
make all -j$BUILD_CORES
```

6.1.4 Create link to include mpi4py in CASA PYTHON container

NOTE: Python prefix convention is different from that of CASA, therefore this step has to be done manually instead of at mpi4py configuration time

```
export PYTHONPATH="$BUILDROOT/python/2.7"
ln -s $BUILDROOT/lib/python2.7/site-packages/mpi4py $PYTHONPATH
```

6.1.5 Pack non-MPI 3rd party packages together with CASA under $CASAROOT

NOTE: This procedure is the same that the imaging team follows to test a CASA development build in the cluster environment. It basically copies the system libraries under $CASAROOT, but also it is necessary to replace some hard-coded paths. We have extended it to include some optional packages like ALMA WVR calibration package.

NOTE: The findreplace.py script, which is used in this step, is the original one from the imaging team and can be downloaded from the links provided in introduction of the 6.1 sections. It has to be located in the folder from where this procedure is executed.

```
ROOT="/lustre/user/mpi/trunk"
ARCH="linux_64b"
PYBIN="bin/python2.7"

FINDREPLACE="findreplace.py"
CASA3PARTYSTR="/usr/lib64/casa/01"

export EL6PATH=$ROOT/$ARCH


STEPS="1 2 3 4 5 6 7"

for step in $STEPS
do
# STEP (1)
    if [ $step == '1' ];
    then
    cp -ar /usr/lib64/casa/01/lib/* $EL6PATH/lib
    cp -a /usr/lib64/casa/01/bin/* $EL6PATH/bin
    cp -a /usr/lib64/libwcs* $EL6PATH/lib/
    cp -a /usr/lib64/libpgsbox* $EL6PATH/lib
    cp -a /usr/lib64/libaatm.* $EL6PATH/lib
    cp -a /usr/lib64/libxerces-c.* $EL6PATH/lib
    cp -a /usr/lib64/libcfitsio.* $EL6PATH/lib
    cp -a /usr/lib64/libalmawvr* $EL6PATH/lib
    fi

# STEP (2)
    if [ $step == '2' ];
    then
# (2)  Modify $EL6PATH/bin/casapy (for Linux, not Mac)   to change the  "casa3party"
variable from "/usr/lib64/casa/01" to $EL6PATH
    sed -e "s|"$CASA3PARTYSTR"|"$EL6PATH"|g" $EL6PATH/bin/casapy >| /tmp/casapy.$USER
    cp /tmp/casapy.$USER $EL6PATH/bin/casapy
    fi


# STEP (3)
    if [ $step == '3' ];
    then
# (3)  export LD_LIBRARY_PATH=$EL6PATH/lib:LD_LIBRARY_PATH
    export LD_LIBRARY_PATH=$EL6PATH/lib:$LD_LIBRARY_PATH
    fi

# STEP (4) Modify   findreplace.py to put the new python2.7 path in...
    if [ $step == '4' ];
    then
    CMD="s|HUUUHAAA|"$ROOT\/$ARCH\/$PYBIN"|g"
    sed -e $CMD $FINDREPLACE  >| /tmp/findreplace.py.$USER
    fi

# STEP (5)  cd $EL6PATH
```

```
    if [ $step == '5' ];
    then
    cd $EL6PATH; python /tmp/findreplace.py.$USER    #(or from wherever it is)
    fi

# STEP (6)  cd $EL6PATH/bin

    if [ $step == '6' ];
    then
# ( This step is required just once.... for the stuff copied in (1) )
# -------------------------------------------
    cd $EL6PATH/bin
    chmod +x *
    fi


# STEP (7)
    if [ $step == '7' ];
    then
#To run from a different machine :
    source $ROOT/casainit.sh
    export LD_LIBRARY_PATH=$ROOT/$ARCH/lib:$LD_LIBRARY_PATH
    echo $LD_LIBRARY_PATH
    fi
#casapy
done

# --------------------------------------------
# After rebuilding using the build scripts, we need to repeat only (2).
# --------------------------------------------
```

6.2 Run a Torque batch job using MPI

As described previously OpenMPI uses a hostfile format compatible with that produced automatically by Torque, so it is not necessary to create a custom hostfile like in the case of the previous iPython -based clusters.

In particular, after submitting a batch job request, Torque sets an environmental variable named $PBS_NODEFILE, which points to a file with the list of nodes reserved by Torque to run the submitted batch job.

When more than one core per host is available then the corresponding hostname is repeated as many times as cores available for the job. This format is natively understood by OpenMPI and therefore it is necessary to only point OpenMPI to the $PBS_NODEFILE using the --hostfile option.

This, together with the mpirun launcher has to be used as a command which is stored in a bash script which Torque finds via the $COMMAND variable in the cluster.req request file.

The only precautions needed are the following:

- The installation path must be identical in the build and target machine.
- We are using a local installation of the system libraries (including OpenMPI) which are packed together with the CASA libraries under $CASAROOT. Therefore the .bashrc file of the user running the Torque batch job must set the environment properly to use these.

➢ Example of .bashrc user environment settings

```
CASAROOT="/lustre/user/mpi/trunk/linux_64bit
source $CASAROOT/casainit.sh
export LD_LIBRARY_PATH="$CASAROOT/lib:$LD_LIBRARY_PATH"
```

➢ Example of cluster.req file to submit a job to Torque

```
#
# Required settings
#
MEMORY="8gb"
WORK_DIR="/lustre/user/workarea"
COMMAND="/lustre/user/workarea/qsub.sh"

#
# Optional settings
#
NUM_NODES="4"              # default is 1
NUM_CORES="2"              # default is 1
STDOUT="run.log"           # file relative to WORK_DIR.  default is no output
STDERR="run.log"           # file relative to WORK_DIR.  default is no output
```

➢ Example of qsub.sh file containing the command to run

```
mpirun -hostfile $PBS_NODEFILE casapy --nologger --log2term --nogui -c "alma-m100-
analysis-hpc-regression.py"
```

NOTE: Notice that it is not necessary to specify the number of MPI processes because this information is already contained in the $PBS_NODEFILE file generated automatically by Torque.

➢ Command to submit a job to Torque

```
submit -f cluster.req
```