Installation and Processing on Juropa

Here will be some notes on the Juropa installation and how to use the software on Juropa at the Juelich Supercomuting Centre.
Work in progress but useful anyhow. The next section Using Juropa for LOFAR Processing contains the information you need to use the system as of May 2014. The sections below these up to date information are kept for archiving purpose (still useful).

Using Juropa for LOFAR Processing

Here are the most recent information on how to make use of Juropa for LOFAR Processing. Last edit June 2014.

Account

First of all you need an account on the system. The Project leader is Matthias Hoeft and the Project ID is HTB00 (needed for registration). The following website contains all necessary links for allocating computing time in the Jülich Supercomputing Centre (JSC). Click on the link “User Accounts for projects on JUQUEEN, JUROPA,…” and follow the instructions.
http://www.fz-juelich.de/ias/jsc/EN/Expertise/Services/JSConline/ComputingTime/_node.html
german version:
http://www.fz-juelich.de/ias/jsc/DE/Leistungen/Dienstleistungen/JSCOnline/Rechenzeitvergabe/_node.html
Get in contact with Matthias so he can sign your account application and initiate the next steps.

LTA Pipeline Environment Libraries

The following libraries with given versions are installed in the home of user htb003 on Juropa.
/lustre/jhome17/htb00/htb003/ [local and lofar]

Library	Version
bison	2.5
blitz	0.10
boost	1.44.0
cmake	2.8.5
casacore	trunk
casarest	8741
cfitsio	3240
fftw	3.2.2
flex	2.5.35
hdf5	1.8.4
libpng	1.5.6
libpqxx	3.1

Library	Version
log4cplus	1.0.4
matplotlib	1.2.1
m4	1.4.16
monetdb python client	11.15.7
numpy	1.7.1
OpenBLAS	0.2.5
PostgreSQL	9.1.2
pyrap	trunk
python	2.7.5
scons	1.3.0
setuptools	0.6c11
wcslib	4.4.4
unittest-xml-reporting	1.5.0
pyfits	3.1.2
pywcs	1.10.2
scipy	0.12.0
argparse	1.2.1
libiberty	
LOFAR	1.16

Additional software for post processing requested by users:

Package	Version
SIP	4.15.1
PyQt4	4.10.3
iPython	1.1.0
casapy	41.0.24668

LTA Installation on Juropa

The operating system is:

```
SUSE Linux Enterprise Server 11 (x86_64)
VERSION = 11
PATCHLEVEL = 1
```

With kernel version:

```
Linux 2.6.32.59-0.3-default x86_64 GNU/Linux
```

The current working installation is in:

```
/lustre/jhome17/htb00/htb003/LOFAR-R14-P275
```

Some things have to be Changed in order to compile and run everything on Juropa.

General Compile settings

There are unresolved issues with older versions. Tests with gcc4.3.4 and gcc4.4.6 gave the error:

```
File "[install_dir]/lofar/release/lib/python2.6/site-
packages/lofar/parmdb/__init__.py", line 112, in getDefValues
return self._getDefValues (parmnamepattern)
TypeError: No to_python (by-value) converter found for C++ type:
casa::Record
```

The compiler suite has to be changed to gcc4.6.3. You do this with the command:

```
module load GCC/4.6.3
```

You also have to load the gsl module

```
module load gsl
```

Set environment variables:

```
export CC=/usr/local/gcc/gcc-4.6.3/bin/gcc
export CXX=/usr/local/gcc/gcc-4.6.3/bin/g++
```

Using gcc4.6.3 gives the error

```
  File "[install_dir]/local/lib/python2.6/site-packages/pyrap.tables-
trunk_r332-py2.6-linux-x86_64.egg/pyrap/tables/table.py", line 1023, in
addcols
    self._addcols (tdesc, dminfo, addtoparent)
TypeError: No registered converter was able to produce a C++ rvalue of type
int from this Python object of type numpy.int32
```

This can be corrected by changing the order of the “pyrap.tables” import in the node script “imager_prepare.py”

Change from:

```
from __future__ import with_statement
import sys
import shutil
import os
import subprocess
import copy
from lofarpipe.support.pipelinelogging import CatchLog4CPlus
from lofarpipe.support.pipelinelogging import log_time
from lofarpipe.support.utilities import patch_parset
from lofarpipe.support.utilities import catch_segfaults
from lofarpipe.support.lofarnode import  LOFARnodeTCP
from lofarpipe.support.utilities import create_directory
from lofarpipe.support.data_map import DataMap
```

```
from lofarpipe.support.subprocessgroup import SubProcessGroup

import pyrap.tables as pt
```

to

```
from __future__ import with_statement
import sys
import shutil
import os
import subprocess
import copy
import pyrap.tables as pt
from lofarpipe.support.pipelinelogging import CatchLog4CPlus
from lofarpipe.support.pipelinelogging import log_time
from lofarpipe.support.utilities import patch_parset
from lofarpipe.support.utilities import catch_segfaults
from lofarpipe.support.lofarnode import  LOFARnodeTCP
from lofarpipe.support.utilities import create_directory
from lofarpipe.support.data_map import DataMap
from lofarpipe.support.subprocessgroup import SubProcessGroup
```

Libiberty

The change of the compiler suite brings additional problems. The system paths change to custom locations which prevents the finding of the correct version of the library “libiberty.a”. The one found is missing the compiler flag -fPIC so it can be linked dynamically. You have to compile it yourself (or copy the correct one from the old path ←- needs checking).

PYRAP

Pyrap will not compile because it will not find the glibc libraries as they are not in the standard location anymore. To correct this you have to edit the pyrap SConsript in pyrap/libpyrap/trunk from

```
env = Environment(ENV = { 'PATH' : os.environ[ 'PATH' ],
                          'HOME' : os.environ[ 'HOME' ]
                          },
```

to

```
env = Environment(ENV = os.environ,
```

Note: you have to use the trunk version and not the latest release because otherwise pyrap tables wont work. Also add the the compile option –enable-rpath. Use “batchbuild-trunk.py” for installation instead of “batchbuild.py”.

FFTW

For the fftw library you have to add the option “–enable-threads” for the compiler.

SCIPY

For scipy add the environment variable UMFPACK=“None”.

Blitz

Version 0.9 did not compile. Version 0.10 works fine.

CMake

Tried using the system installed CMake which is also version 2.8.5. There are Problems with some “find” scripts (LAPACK,BLAS). I remember casacore and Lofar would not install as they should, forgot the details though as the self installed version does what it should. Maybe one should look into it in the future because what CMake to use should not matter.

Casacore

Need to use the trunk version instead of release 1.5 for functionality. Using the buildscript options, Casacore is missing support for hdf5 which is needed for the final steps of the imaging pipeline. Added hdf5 support as well as OpenMP and Threads by hand in the ccmake interface. ToDo: add options to buildscript

LOFAR

The system installed libpng was hiding the selfcompiled version which lead to problems during program execution. The proper path to libpng library and include dir have to be set in cmake. Same goes for the selfcompiled libiberty. ToDo: add options in cmake.build file (at least for libpng since its part of standard installation).
The variants file GNU.cmake has to be edited to set the path to the new compiler version

```
# GNU compiler suite
set(GNU_COMPILERS GNU_C GNU_CXX GNU_Fortran GNU_ASM)
set(GNU_C         /usr/local/gcc/gcc-4.6.3/bin/gcc)      # GNU C compiler
set(GNU_CXX       /usr/local/gcc/gcc-4.6.3/bin/g++)      # GNU C++ compiler
set(GNU_Fortran   /usr/local/gcc/gcc-4.6.3/bin/gfortran) # GNU Fortran
compiler
set(GNU_ASM       /usr/local/gcc/gcc-4.6.3/bin/gcc)      # GNU assembler
```

SSH support

Logging into compute nodes via ssh is not permitted on the system. Subprocesses have to be started on the one rented compute for now via shell or mpiexec command. Distribution to multiple nodes is in the works.

lofarpipe/support/remotecommand.py has to be edited to circumvent ssh for locahost job spawning (svn diff, see extra section)

File copy

Since the Juropa cluster uses a shared filesystem every data should be (read: HAS to be) present at job start to not waste computing time. The login nodes are supposed to be used for job preparation and analysis afterwards. So we do not need to copy data to the working directory (quota is limited!). The change for that is in lofarpipe/recipes/nodes/imager_prepare.py (svn diff, see extra section).

Imaging Pipeline

Because the datacopy to the working directory will not be done automatically the data has to present in your working directory set in pipeline.cfg plus subfolder jobname. Somthing like working_dir/imaging_pipeline/subbands
In lofarpipe/recipes/nodes/imager_prepare.py in the call to rfi_console the “indirect_read” option has to be removed because of insufficient write access on the target machine (some folder you are not supposed to use as normal user)

GSM Database

A local version of the GSM Database has to be used. At the moment it is running on Juropa02 but has to be restarted after downtimes or after the demon used up its 30min wallclock time. The database is in the home folder of user zdv596 and can be started with

```
cd /lustre/jhome9/lofar/zdv596
monetdbd start gsm
monetdb start gsm
```

monetdb is installed in /lustre/jhome9/lofar/zdv596/LOFAR-Release-1_14/local/bin
How to install a local GSM Database take a look at this
http://www.lofar.org/wiki/doku.php?id=lta:software_stack_installation#gsm_database_installation

gsmutils.py

The changes made during release 1.14 (after initial release) break the functionality of the database access on Juropa. You can revert back to the initial release of 1.14 or use the fix mentioned below. The error is as follows:

```
ERROR:node.jj29l09.imager_create_dbs:expected_fluxes_in_fov raise exception:
GDK reported error.
!BATfetchjoin(tmpr_2277,tmp_4347) does not hit always (|bn|=0 != 46216=|l|)
=> can't use fetchjoin.

ERROR:node.jj29l09.imager_create_dbs:failed creating skymodel
Traceback (most recent call last):
  File "/lustre/jhome17/htb00/htb003/LOFAR-R14-
P275/lofar/release/lib/python2.7/site-
packages/lofarpipe/recipes/nodes/imager_create_dbs.py", line 470, in
<module>
    _jobid, _jobhost, _jobport).run_with_stored_arguments())
  File "/lustre/jhome17/htb00/htb003/LOFAR-R14-
P275/lofar/release/lib/python2.7/site-
packages/lofarpipe/support/lofarnode.py", line 85, in
run_with_stored_arguments
    returnvalue = self.run_with_logging(*self.arguments)
  File "/lustre/jhome17/htb00/htb003/LOFAR-R14-
P275/lofar/release/lib/python2.7/site-
packages/lofarpipe/support/lofarnode.py", line 59, in run_with_logging
    return self.run(*args)
  File "/lustre/jhome17/htb00/htb003/LOFAR-R14-
P275/lofar/release/lib/python2.7/site-
packages/lofarpipe/recipes/nodes/imager_create_dbs.py", line 71, in run
    monet_db_password, assoc_theta)
TypeError: 'int' object is not iterable
```

Bart Scheers provided a temporary fix for this issue. You need to change the configuration of your database.

```
Stop gsm database set nthreads property to 1:

monetdb stop gsm
monetdb set nthreads=1 gsm
monetdb start gsm
```

imager_prepare.py

Prevent datacopy when working on local host only. No “indirect_read” supported on Juropa.
SVN diff for lofarpipe/recipes/nodes/imager_prepare.py

```
Index: CEP/Pipeline/recipes/sip/nodes/imager_prepare.py
===================================================================
--- CEP/Pipeline/recipes/sip/nodes/imager_prepare.py    (revision 25127)
+++ CEP/Pipeline/recipes/sip/nodes/imager_prepare.py    (working copy)
@@ -10,6 +10,7 @@
 import os
 import subprocess
 import copy
```

```
+import pyrap.tables as pt
 from lofarpipe.support.pipelinelogging import CatchLog4CPlus
 from lofarpipe.support.pipelinelogging import log_time
 from lofarpipe.support.utilities import patch_parset
@@ -19,7 +20,7 @@
 from lofarpipe.support.data_map import DataMap
 from lofarpipe.support.subprocessgroup import SubProcessGroup

-import pyrap.tables as pt
+#import pyrap.tables as pt

 # Some constant settings for the recipe
 _time_slice_dir_name = "time_slices"
@@ -140,37 +141,44 @@
             if input_item.skip == True:
                 exit_status = 1 #

-            # construct copy command
-            command = ["rsync", "-r", "{0}:{1}".format(
-                           input_item.host, input_item.file),
-                              "{0}".format(processed_ms_dir)]
+
+           self.logger.debug(input_item.host)
+           self.logger.debug(self.host)
+            # skip the copy if machine is the same (execution on
localhost).
+           # make sure data is in the correct directory. for now:
working_dir/trunk_imager_regression/subbands
+           if input_item.host != "localhost":
+
+                       # construct copy command
+               command = ["rsync", "-r", "{0}:{1}".format(
+                           input_item.host, input_item.file),
+                           "{0}".format(processed_ms_dir)]

-            self.logger.debug("executing: " + " ".join(command))
+               self.logger.debug("executing: " + " ".join(command))

-            # Spawn a subprocess and connect the pipes
-            # The copy step is performed 720 at once in that case which
might
-            # saturate the cluster.
-            copy_process = subprocess.Popen(
-                        command,
-                        stdin=subprocess.PIPE,
-                        stdout=subprocess.PIPE,
-                        stderr=subprocess.PIPE)
+               # Spawn a subprocess and connect the pipes
+               # The copy step is performed 720 at once in that case which
might
```

```
+               # saturate the cluster.
+               copy_process = subprocess.Popen(
+                               command,
+                               stdin=subprocess.PIPE,
+                               stdout=subprocess.PIPE,
+                               stderr=subprocess.PIPE)

-            # Wait for finish of copy inside the loop: enforce single tread
-            # copy
-            (stdoutdata, stderrdata) = copy_process.communicate()
+               # Wait for finish of copy inside the loop: enforce single
tread
+               # copy
+               (stdoutdata, stderrdata) = copy_process.communicate()

-            exit_status = copy_process.returncode
+               exit_status = copy_process.returncode

             #if copy failed log the missing file and update the skip fields
-            if  exit_status != 0:
-                input_item.skip = True
-                copied_item.skip = True
-                self.logger.warning(
+                if  exit_status != 0:
+                       input_item.skip = True
+                       copied_item.skip = True
+                       self.logger.warning(
                             "Failed loading file:
{0}".format(input_item.file))
-                self.logger.warning(stderrdata)
+                       self.logger.warning(stderrdata)

-            self.logger.debug(stdoutdata)
+                       self.logger.debug(stdoutdata)

         return copied_ms_map

@@ -298,7 +306,8 @@

                 # construct copy command
                 self.logger.info(time_slice)
-                command = [rficonsole_executable, "-indirect-read",
+                command = [rficonsole_executable,
+                           ## "-indirect-read",
                            time_slice]
                 self.logger.info("executing rficonsole command:
{0}".format(
                             " ".join(command)))
```

remotecommand.py

Extra Path variable for remote systems where python is not installed in the same place as on the master node.
Prevent ssh commands entirely as they are not supported on Juropa. Just a switch for localhost. SVN diff:

```
Index: CEP/Pipeline/framework/lofarpipe/support/remotecommand.py
===================================================================
--- CEP/Pipeline/framework/lofarpipe/support/remotecommand.py   (revision
25127)
+++ CEP/Pipeline/framework/lofarpipe/support/remotecommand.py   (working
copy)
@@ -111,13 +111,29 @@
     process.kill = lambda : os.kill(process.pid, signal.SIGTERM)
     return process

+def run_via_local(logger, command, arguments):
+    commandstring = ["/bin/sh","-c"]
+    for arg in arguments:
+        command = command + " " + str(arg)
+    commandstring.append(command)
+    process = spawn_process(commandstring, logger)
+    process.kill = lambda : os.kill(process.pid, signal.SIGKILL)
+    return process
+
 def run_via_ssh(logger, host, command, environment, arguments):
     """
     Dispatch a remote command via SSH.

     We return a Popen object pointing at the SSH session, to which we add a
     kill method for shutting down the connection if required.
+
+    hack/
+    if host is localhost run without ssh
+    /hack
     """
+    if host == "localhost":
+        logger.debug("Running command locally")
+        return run_via_local(logger, command, arguments)
     logger.debug("Dispatching command to %s with ssh" % host)
     ssh_cmd = ["ssh", "-n", "-tt", "-x", host, "--", "/bin/sh", "-c"]

@@ -214,6 +230,7 @@
                 self.host,
                 self.command,
                 {
+                   "PATH": os.environ.get('PATH'),
                    "PYTHONPATH": os.environ.get('PYTHONPATH'),
                    "LD_LIBRARY_PATH": os.environ.get('LD_LIBRARY_PATH')
```

```
                    },
```

copier.py

The copy process of the Intrument files used in the target pipeline has to be changed because rsync is not supported between nodes. Change to a simple copy command.
recipes/nodes/copier.py

```
53,56c53
<         if source_node=="localhost":
<             command = ["cp", "-
r","{0}".format(source_path),"{0}".format(target_path)]
<         else:
<             command = ["rsync", "-r",
---
>         command = ["rsync", "-r",
```

parset.py

Changed the “output_dir” in “patch_parset” to a directory visible from all nodes.
Should maybe be changed to the working directory?!

Notes on Processing at the Juropa Cluster

This is going to be the Wiki page for the Lofar Software installation at the Juelich Supercomupting Centre.I will update this page as the installation progresses.

Acquiring Data

Take a look at this site on how to get the data from the LTA
http://www.lofar.org/operations/doku.php?id=public:lta_howto
To download data from the web you need the full filename. You can look those up in the catalog
http://lofar.target.rug.nl/Lofar
The Juelich Http download server is here
https://lofar-download.fz-juelich.de/
For Sara
https://lofar-download.grid.sara.nl/
If you want to do a direct srm copy you need a Grid Certificate.

German Grid Certificate

To get direct srm copy access to the LTA storage you need a Grid Certificate.

http://dgi-2.d-grid.de/zertifikate.php

SRM Copy from Juropa

There are two methods.
One:
Follow this Walkthrough to generate a proxy for your srm download
http://www.lofar.org/operations/doku.php?id=public:srmclientinstallation#walkthrough
If you get the message “No user credentials found” you might need to convert your userkey.pem to a different format with this command

```
openssl rsa -des3 -in userkey.pem -out userkey.pem
```

Make a backup before you do this.
Two:
You need to load the ltools module and execute the given command to activate the environment to use “srmcp”

```
module load ltools
. /usr/local/lroot/etc/env.sh
```

Then you can generate a proxy with the command

```
voms-proxy-init --voms lofar:/lofar/user -vomses $HOME/jlite-0.2/etc/vomses
```

Notice that “jlite” from method one has to be installed for the correct vomses file. When you invoke the srmcp command do it with the option “-srm_protocol_version=2 -server_mode=passive -webservice_path=srm/managerv2”.
If you want more output add the “-debug” option as well.

Running the Software

(already outdated, will get updates when Lofar v1.16 is installed (week of 2.9.13 maybe?!))
Currently the software is beeing tested on the Juropa system. The Pipelines are working in single node modus. That means the subbands are computed in serial. One should submit multiple jobs with less subbands.
The software is available in the home directory of user htb003. The root path of the install is

/lustre/jhome17/htb00/htb003

You can find the lofar software in “lofar/release”. The environment you need is loaded with the script “variables_lofar.sh”

In addition you might need a copy of the measurement data
/lustre/jhome17/htb00/htb003/dataCEP
Put it in your home directory and point to it in a file .casarc (just contains:“measures.directory: [yourhome]/dataCEP”)

If you require access to the GlobalSkyModel database, there is a copy of the database from the CEP Cluster (hopefully) running on the Juropa login node jj28l02. Access the databse “gsm” on port 51000 with user “gsm” and pass “msss”

How to keep the measurement and gsm data up to date and distributed has to be discussed

You can now run and test the executables on the login node from “lofar/release/bin” or run python scripts (your own or pipeline scripts in “local/lib/python2.7/site-packages/lofarpipe/recipes”).

To run your jobs on the compute nodes you first have to setup and submit a job via the batch system. A detailed description can be found on the Juropa homepage
'http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUROPA/UserInfo/QuickIntroduction.html'

Here is a simple example of the procedure. Basically you use two scripts. One to configure the job and one to setup the environment for your program and run it.
Job configuration is pretty basic right now because we can only utilize one node per job. Do not get confused by the use of comments '#'. The '#' in front of MSUB commands is necessary for the command to be recognized from the Moab batch system.
You submit the job with the command “msub [yourscript]”. Check your status with “showq -u 'username' ”. To see the whole machine with a gui try the “llview” program.

Contents of 'lofarmsub.sh':

```
#!/bin/bash -x
#MSUB -N Lofar-test
# just the name
#MSUB -l nodes=1:ppn=16
#MSUB -l walltime=00:30:00
#MSUB -e error.txt
# if keyword omitted : default is submitting directory
#MSUB -o output.txt
# if keyword omitted : default is submitting directory
#MSUB -M your@mail.de
#Mailadress
#MSUB -m eab
#send mail on end, abort, begin
./lofarCalibratorPipelinePy2.7.sh
```

The walltime is the time your job will be running on the machine. If it is to low and the job is not finished it will be killed. Is it to high your job might have to wait longer in queue but only the real computing time will be booked.
The maximum walltime is 24h.
The number of nodes has to stay at 1 for the time being. You can experiment with ppn (process per node) which is used for openmp enabled programs.
It is best to name the log files error and output with some job specific parameters and maybe the date.
You can choose to have mails send to you about the status of your job.

Contents of 'lofarCalibratorPipelinePy2.7.sh':

```
#/bin/sh!
```

```
#start of jobscript
export OMP_NUM_THREADS=16
#
#
export PYTHONPATH=/lustre/jhome17/htb00/htb003/local/lib/python2.7/site-
packages:$PYTHONPATH
export
PYTHONPATH=/lustre/jhome17/htb00/htb003/lofar/release/lib/python2.7/site-
packages:$PYTHONPATH
#
export PATH=/lustre/jhome17/htb00/htb003/local/bin:$PATH
export PATH=/lustre/jhome17/htb00/htb003/lofar/release/bin:$PATH
export PATH=/lustre/jhome17/htb00/htb003/lofar/release/sbin:$PATH
#
export
LD_LIBRARY_PATH=/lustre/jhome17/htb00/htb003/lofar/release/lib:$LD_LIBRARY_P
ATH
export
LD_LIBRARY_PATH=/lustre/jhome17/htb00/htb003/lofar/release/lib64:$LD_LIBRARY
_PATH
export
LD_LIBRARY_PATH=/lustre/jhome17/htb00/htb003/local/lib:$LD_LIBRARY_PATH
export
LD_LIBRARY_PATH=/lustre/jhome17/htb00/htb003/local/lib64:$LD_LIBRARY_PATH
#
export LOFARROOT=/lustre/jhome17/htb00/htb003
#
module load gsl
module load GCC/4.6.3
#
/lustre/jhome17/htb00/htb003/lofar/release/bin/msss_target_pipeline.py
/lustre/jhome17/htb00/htb003/pipeline_tests/Pipeline/target_pipeline/Observa
tion64406 -c
/lustre/jhome17/htb00/htb003/lofar/release/share/pipeline/pipeline.cfg --job
target_test_omp16_descfile -d
```

Simply replace the pipeline call with the command you want to run in your job. Example of Alexanders bbs test: /lustre/jhome9/lofar/zdv596/LOFAR-Release-1_14/lofar/release/bin/calibrate-stand-alone -v -n -f L104244_SB200_uv.dppp.MS BBS.parset skymodel.parset One important remark for your working directory. Use the Filesystem mounted under $WORK for your data and jobs.
From the Juropa home page:
$WORK
File system for large temporary files with high I/O bandwidth demands (scratch file system). No backup of files residing here. Files not used for more than 28 days will be automatically deleted!

Jobs in parallel

You can start one job for every independent piece of data. You can use your old scripts and the pipeline scripts but every subtask will be processed in serial on one node. So typically you only

allocate one node for your jobs.
To circumvent this, start the subprocesses in the python scripts in a different manner. Use the mpiexec command to start your subprocess. The Parastation MPI Demon will then allocate free resources to your subprocess when available. For this behavior the environment variable PSI_WAIT has to be set. This means you can allocate the partition you want to work on with more than one node. Run your script and whenever you use a subprocess call use mpiexec with number of processes equal to one (np=1).
You can have up to 16 processes per node (eight cpus + smt mode). How many of these processes are allocated to your np=1 option depends on the number of threads you want to have for openMP. So for OMP_NUM_THREADS=4 you will be able to run 4 subprocesses on one node. With OMP_NUM_THREADS=16 one subprocess per node and with OMP_NUM_THREADS=1 you will have 16.
As an example lets look at a part of a script from Andreas (run_NDPPPs.py):(in progress)
The snippet shows the the subprocess call with subprocess.Popen and how their return is handled. What has to be executed is in the list of tupels “cmds”. Where the first entry is the executable and the second a temporary parset file. Hence the os.remove after the return. The process is put into a list of processes and the function returns when this list is empty.

```
    while True:
        while cmds and len(processes) < max_task:
            task = cmds.pop()
            print time.asctime()," : ",list2cmdline(task)
            processes.append([Popen(task,env=myenv),task[1]])
            if waittime:
                break
        for p in processes:
            if done(p[0]):
                if success(p[0]):
                    os.remove(p[1])
                    processes.remove(p)
                else:
                    fail()
        if not processes and not cmds:
            break
        else:
            time.sleep(sleeptime)
```

To use multiple nodes on Juropa the command that is passed to popen has to be changed in the following way. The first argument is the executable followed by the arguments. The argument for “/bin/sh” has to be passed as one string and not as additional argument in the list. In this example the command we want to run consists of the executable and its argument written as tupels in “cmds”. The mpiexec is executed on one available slot “-np=1” which has the number of processes you specified with OMP_NUM_THREADS. The argument “-x” passes all environment variables to the process executed with mpiexec. Then we wait while there are elements left in the list of processes until all have returned. With the env variable PSI_WAIT=1 we can call more mpiexec than we have available slots. The mpi demon will handle the execution for us.

```
    for task in cmds:
        command = ["mpiexec","-x","-np=1","/bin/sh", "-c", "hostname &&
"+task[0]+" "+task[1]]
        print command
        processes.append([Popen(command,env=myenv),task[1]])
```

```
    while True:
        for p in processes:
            if done(p[0]):
                if success(p[0]):
                    os.remove(p[1])
                    processes.remove(p)
                else:
                    print "Error in: ",p[1]
                    os.remove(p[1])
                    processes.remove(p)

        if not processes:
            break
```

I hope these information are sufficient for some first tests and experiments.
Good luck and let me know of any problems and feel free to give some feedback.