LIP IT/Tech Group

S. Schröder & M. Krause

documentation for the new compute cluster TARDIS

Max-Planck-Institut für Bildungsforschung
Max Planck Institute for Human Development

content of this document

1. Changes
2. Introduction
3. What is a Job?
4. Example for FSL
5. Example for Freesurfer
6. MATLAB
   - native
   - compiled
   - SPM
1. Example for gnu-R
2. External ressources

1. Changes

- the current version of this document is 2.0
- we added:
  - generic job description
  - matlab compiler examples
  - fsl examples
  - tardis specs
- we changed:
  - some major typos
  - overall document structure

2. Introduction

- this short manual describes what TARDIS is and some basic steps on how to use the new compute cluster
- it is going to be updated every time we successfully test a new use case
- it is meant to be a guideline and if you need help at any point we will be glad to assist you

- we hope that our efforts will be to your benefit and would appreciate all of your feedback, please contact us at:

grid-admin@mpib-berlin.mpg.de

TARDIS

Tardis, **A** **R**apid **D**istributed **I**nformation **S**ystem

technical facts

- 608 Intel(R) Xeon(R) CPU E5-2670 CPUs inside 32 DELL m620 blade servers (R_{peak} = 5.1 TFLOPS , R_{max} = 12.6 TFLOPS)
- 3.2 TB total amount of memory
- up to 256 GB per node
- 16 TB of direct attached storage
- simple 2-level fat tree network with theoretical 40GB/s bisection bandwidth

the classical, sequential workflow

input data on
fileserver or a local copy

simulation/analysis
on office machine

results copied back
to file server

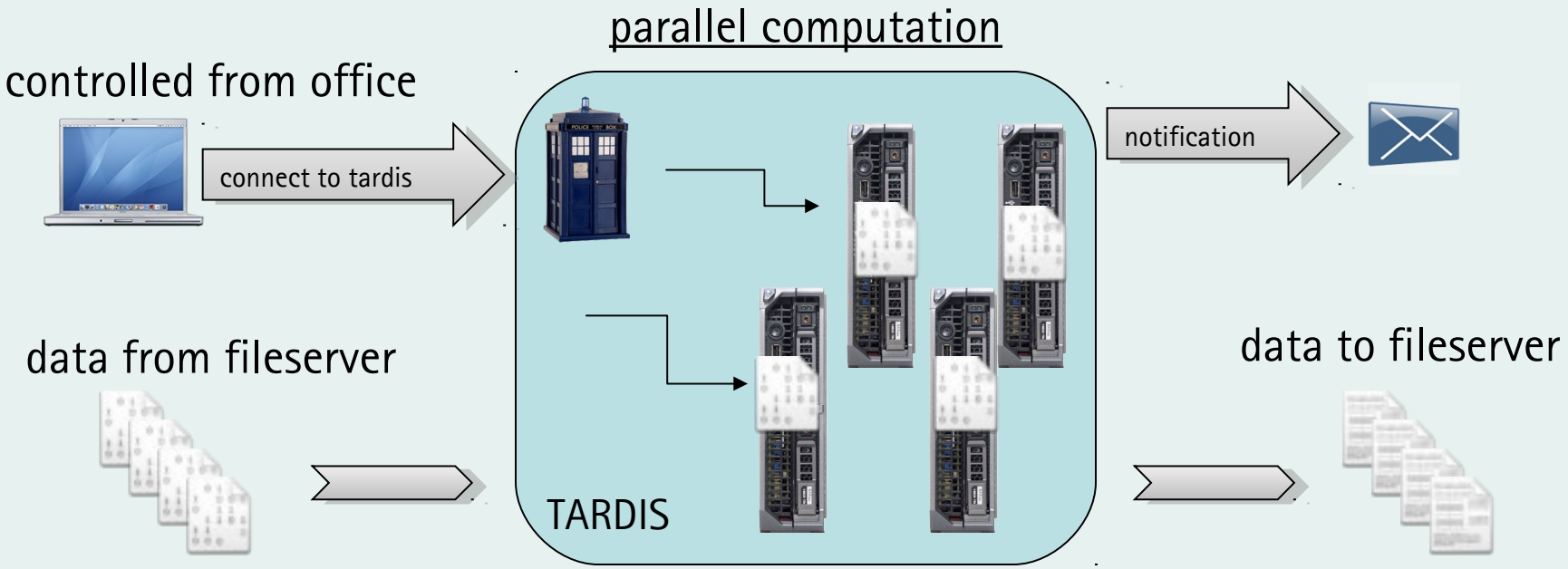
sequential computation

- lots of computations are still done on workstation computers
- requires office machines to be running all the time

→ inefficient, warm, noisy and slow

- not feasible for huge datasets or computationally heavy algorithms

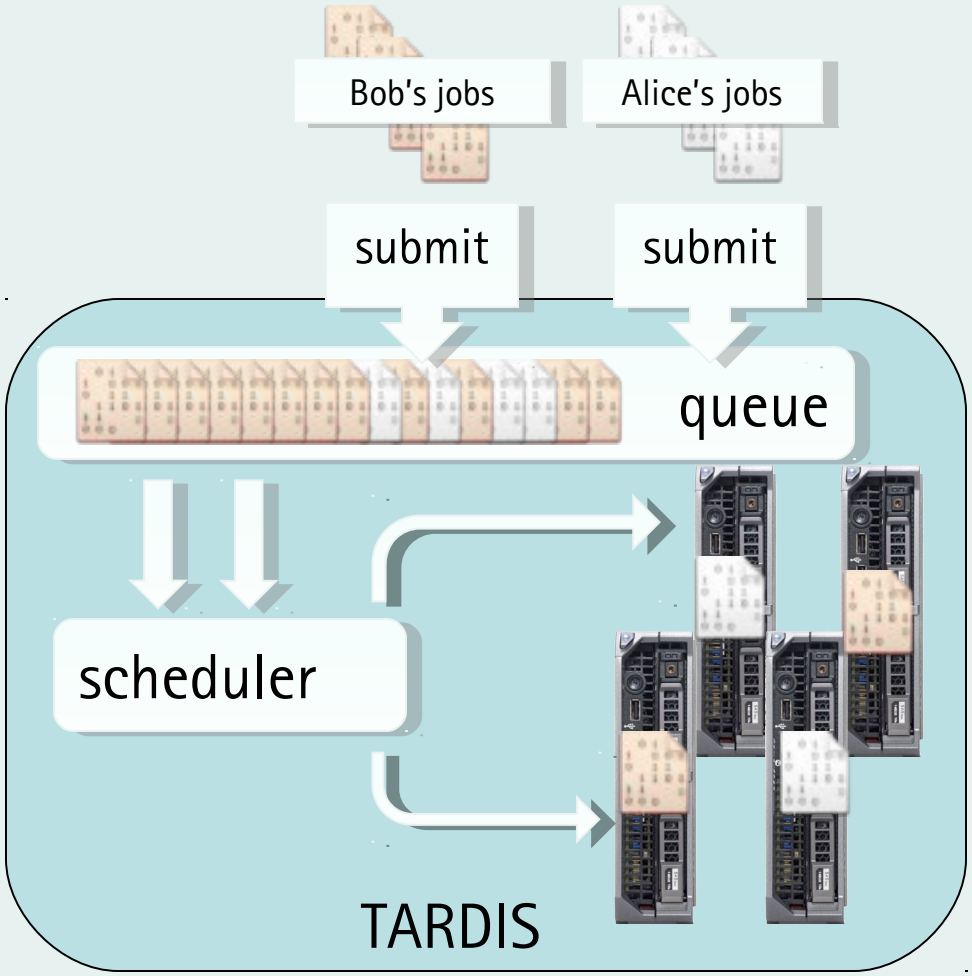
distributed workflow



- delegate computational tasks to a central cluster
- a task can be: a matlab job or a generic job like:
  freesurfer, fsl, R(script), plink, python, java or basically any other kind of program
- you will automatically be notified via e-mail upon completion of your job, also see **http://gridmaster2012/**

resource management

- TARDIS resources are shared among all users
- the scheduler will decide which job to start next
- the scheduling goal is to share the computation time per time frame equally among all users



Important:
To avoid endless jobs and to assure a theoretical fairness the scheduler has to assume a value for the jobs duration. This is 24h by default. You can however specify your own value.
Your job will be aborted if it exceeds the specified time to prevent blocking of the queue!

entering the TARDIS

(it's bigger on the inside)

- to login we need to add your account to the mpibgrid group (contact us)
- connect to the master (gridmaster2012 or 141.14.164.164 when DNS setup is broken)
- requires SSH/SFTP (secure shell, see external resources
- users are authenticated with their MPIB domain accounts

example:

```
$> ssh gridmaster2012 -l krause
$> scp local-file
krause@gridmaster2012:remote/
...
```

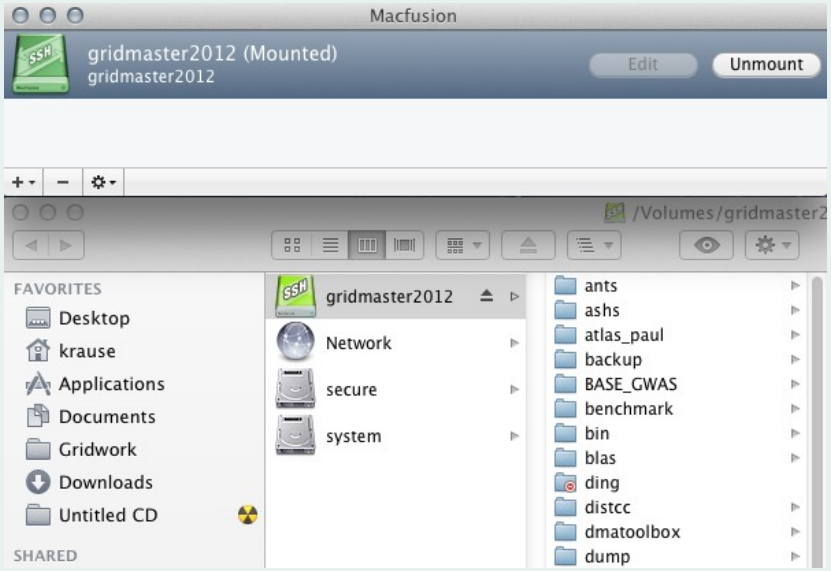
ssh fingerprints (for the paranoids):

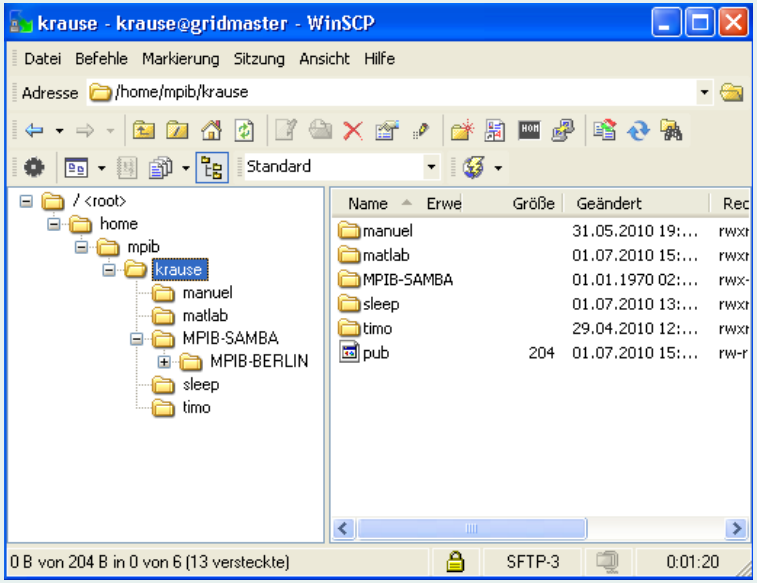
```
rsa: b3:c5:6a:3a:e1:cf:ca:38:57:43:19:1e:fc:45:eb:f4
dsa: 47:3a:37:7f:73:00:9d:c1:8c:9f:f7:bb:8d:fa:67:e6
```

- always copy your data to your home directory and delete it afterwards (if we exceed the global limit we will have to enforce per user quotas)
- the disk space on the gridmaster is temporary space only
- it is intended for high throughput, there is no backup of your temporary data

entering the TARDIS

- there are graphical tools for all operating systems to simplify data transfer
- on Linux use your default file manager with the URL ssh://gridmaster2012/
- on Windows use: http://winscp.net/eng/index.php
- on MacOS we recommend macfusion, which requires
  - http://osxfuse.github.com/
  - http://macfusionapp.org/





3. What is a job?

- a job is a computational task that can be submitted to a queue
- job files consist of *meta-data* and actual *tasks/commands*
- tasks can be any programs you would otherwise run in a terminal
- you have to provide an estimated running time in the meta data section, otherwise 24h will be set by default
- jobs are terminated if they exceed this value
- see external resources for most meta data definitions

3. What is a job?

example job file

```
####### generic job file <job1.pbs> #######
## this is a comment
##   meta data section
## job name
#PBS -N name
##expected running time:
#PBS -l walltime=10:00:00
## addition resource requirements for the job

## your actual computational task
cd project-location/
./program-name -option1 –option2 arg1 arg2 ...
```

3. What is a job?

- prepare your input data on the login node (ssh, winscp, macfusion)
- all compute nodes share the same directories
- submit your job files to the queue with the command „qsub“
- all compute nodes share the same directories
- check their status with „qstat“
- see external resources for more information about qsub et al.

Example:

```
$> qsub job1.pbs
321.master.tardis.mpib-berlin.mpg.de
$> qsub job2.pbs
322.master.tardis.mpib-berlin.mpg.de
$> qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
321.master.tardis...      sleep            krause                 0 R default

322.master.tardis...      sleep            krause                 0 R default
```

3. What is a job?

- a common pattern for job submission is a loop over a set of input data
- for each input file a temporary *jobfile* will be created and submitted

example automatic submission script: **createjobs.sh**

```
#!/bin/bash

SUBJECTS="01 02 03 04 05 06"

for subject in $SUBJECTS; do
    echo "#PBS -N task-$subject"      >> jobfile
    echo "#PBS -l walltime=48:0:0"    >> jobfile
    echo "#PBS -m n"                  >> jobfile
    echo "cd $PBS_O_DIR/$subject"     >> jobfile

    echo "some-program -i $subject"   >> jobfile
    qsub jobfile  # submit job
    rm jobfile    # clean up temporary file
done
```

4. FSL on the cluster

- we successfully tested two approaches for using FSL on the cluster

1. FSL inherent parallelization

   (tbss* , randomise_parallel, bedpostx)

1. using the before mentioned loop to run feat on a number of design files

- at the moment there is FSL 4.1 and 5.0 available

4.1 FSL inherent parallelization

- we had/have to manually fix most fsl programs to use the qsub command and do things in parallel
- for a list of programs see /usr/share/fsl/overlay on gridmaster2012
- to use those programs, setup FSL and then update your path to prefer the files in the fsl overlay
- afterwards use FSL programs as you would on your workstation

Example:

```
# set up FSL
<user>@gridmaster2012:~> export FSLDIR=/usr/share/fsl/5.0/
<user>@gridmaster2012:~> source $FSLDIR/etc/fslconf/fsl.sh
<user>@gridmaster2012:~> export PATH=/usr/share/fsl/overlay:$PATH
# run FSL programs
<user>@gridmaster2012:~> cd mytbss
<user>@gridmaster2012:mytbss> tbss_1_preproc *.gz # wait...
<user>@gridmaster2012:mytbss> tbss_2 -T # or even -n
...
```

4.2 FSL design files

- suppose you work on a number of design files
- first you will have to copy your analysis directory to the grid

  this includes the design files and all your specific input data
- you will then use a simple loop script around all your design files and submit them
- once the cluster is finished you can copy back your results
- it is probably necessary to mass change the fsf files to match standard images and data paths (you can do this in your editor, or as shown with the powerful sed)

4.2 FSL design files

- fix the usage of standard files

```
<user>@gridmaster2012:~workdir/> grep share file1.fsf
file1.fsf:set fmri(regstandard) "/usr/share/fsl/data/standard/MNI152_T1_1mm_brain“
```

- you might have similar lines in your designfiles
- now have a look at the available standard files and pick one

```
<user>@gridmaster2012:~workdir/> ls /usr/share/data/
bangor-cerebellar-atlas  jhu-dti-whitematter-atlas   oxford-thalamic-connectivity-
atlas
fsl-mni152-templates     juelich-histological-atlas  talairach-daemon-atlas
harvard-oxford-atlases   mni-structural-atlas
<user>@gridmaster2012:~workdir/> ls /usr/share/data/fsl-mini152-templates/
avg152T1_brain.nii.gz                 MNI152_T1_1mm.nii.gz
[...]
```

- correct the path in all designfiles using *sed*

```
~> sed "s#share/fsl/data/standard#share/data/fsl-mini152-templates#" -i *.fsf
```

- to replace all occurances of the old path by the new path

4.2 FSL design files

- once the design files are prepared, you can submit your jobs according to the subject loop pattern shown before with *createjobs.sh*

```
#!/bin/bash

# this sets up a list of all designfiles in the current directory
INPUTS=$(ls designfiles/*.fsf)

# loop through all designfiles, create a job file and submit it
for DESIGNFILE in $INPUTS; do
     # standard settings
     echo "#PBS -l walltime=24:00:00" >> jobfile
     # get mail notification when the job ends or aborts
     # chose "-m n" to get no mail at all
     echo "#PBS -m ae" >> jobfile
     # chose a name for the job
     echo "#PBS -N FSL_$FILE" >> jobfile
     # load the system specific fsl settings
     echo ". /etc/fsl/5.0/fsl.sh" >> jobfile
     echo "cd $PBS_O_WORKDIR" >> jobfile
     # actually run the analysis with the program "feat" on the current design file
     echo "feat $FILE" >> jobfile
     # submit the current job definition
     qsub jobfile
     rm -f jobfile
done
```

4.2 FSL design files

- copy this createjobs.sh script to the cluster and start the analysis

```
<user>@gridmaster2012:~workdir/> ls
designfiles/  data/  createjobs.sh
<user>@gridmaster2012:~workdir/> bash createjobs.sh
101.gridmaster2012
102.gridmaster2012
103.gridmaster2012
104.gridmaster2012
[...]
```

- when successful, this will create a number of jobs on the cluster with a unique job id (101,102, … in this example)

4.2 FSL design files

- check your mail for errors
- get the current status of your jobs with qstat or have a look at the website for a static 5min snapshot of qstat (http://gridmaster2012.mpib-berlin.mpg.de)

qstat:

```
<user>@gridmaster2012:~> qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
101.gridmaster2012        FSL_f1.fsf       <user>                 0 R default

102.gridmaster2012        FSL_f2.fsf       <user>                 0 R default

[...]
```

- have a coffee or two...

5. Freesurfer

- most of the time freesurfer can be parallelized with the *subject loop pattern* and the *recon-all* command, you can find an example script on the next page

- a second example demonstrates a somewhat complex situation of how we can implement dependencies with qsub
- it will probably need some tweaking on your side, but the basic scripts can be found here:

http://gridmaster2012/misc/fsf-long-1.sh
http://gridmaster2012/misc/fsf-long-2.sh
http://gridmaster2012/misc/fsf-long-3.sh

(1) http://surfer.nmr.mgh.harvard.edu/fswiki/LongitudinalProcessing

5. Freesurfer basic example

- do a time consuming (up to 20h) recon-all run for a number of subjects

```
#!/bin/bash

# define input data
SUBJECTS=ID{1..99} # ID1, ID2, ... , ID99
# loop
for subject in $SUBJECTS; do
  # meta-data
  echo "#PBS -N $subject" >> jobfile
  echo "#PBS -l walltime=120:00:00" >> jobfile
  echo "#PBS -m n" >> jobfile

  # actual freesurfer commands
  echo 'cd $PBS_O_WORKDIR' >> jobfile # go to current location
  echo "export FREESURFER_HOME=/opt/freesurfer/5.3.0/" >> jobfile
  echo 'source $FREESURFER_HOME/SetUpFreeSurfer.sh' >> jobfile
  echo "export SUBJECTS_DIR='./'" >> jobfile
  echo "recon-all -i ${subject}.nii -subjid ${subject}" >> jobfile

  # submit and clean temporary file
  qsub jobfile
  rm jobfile
done
```

5. Freesurfer advanced example

- for this advanced example we will use the Longitudinal Stream analysis workflow as described in the fsl-wiki [1] and move it to the grid
- we assume you already know how to prepare your data and what a jobfile is for (see the FSL example section for details)
- the analysis consists of 3 steps:
  - cross-sectionally processing
  - template creation
  - longitudinal processing
- those steps are data dependent and have to be serialized in that specific order!
- for performance and convenience reasons all processing tasks should be submitted at a single point in time
- therefore a dependency chain has to be created

(1) http://surfer.nmr.mgh.harvard.edu/fswiki/LongitudinalProcessing

5. Freesurfer advanced example

In order to create the dependencies in step 2 and step 3 we will need a naming convention for the subjects. This example assumes that your subject data will reside in directories named:

studyprefixsequencedIDTimestampID

where:

- studyprefix - some string
- sequenceID - ascending number to identify the subject
- TimestampID - a single small letter, alphabetically ordered

In each example script you will find a VARIABLE block that looks like this:

You will have to change the PREFIX variable accordingly.

```
############
# VARIABLES
PREFIX="study"
############
```

6. MATLAB

- just like FSL there are two ways to parallelize MATLAB tasks, each with considerable up- and downsides

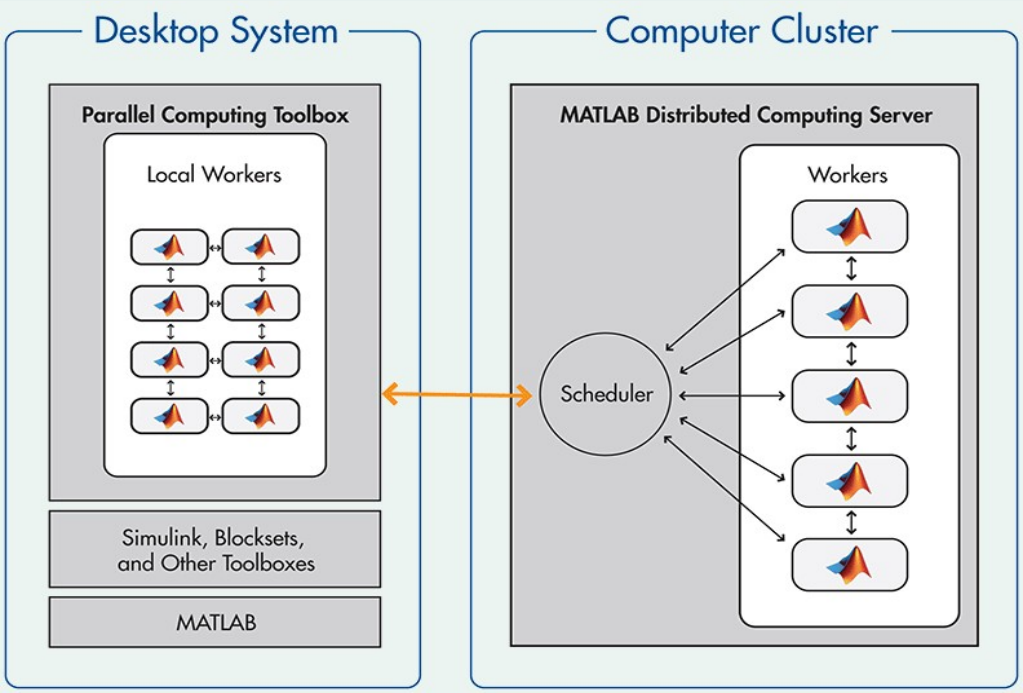
1) MATLAB's parallel computing toolbox job interface
+ easy to use (http://ww.mathworks.de/products/parallel-computing/index.html)
+ already used in a number of 3rd party toolboxes
- does not play well with existing cluster software
- very limited number of licenses (= number of jobs)

1) using a combination of MATLAB Compiler and native qsub
+ submitted jobs *not* bound to licenses
- probably a little more time consuming job preparation
- only a few compiler licenses itself available

6.1 MATLAB parallel computing toolbox

- MATLAB (tested versions: 2010b and 2012a) provides its own job management interface as a part of the parallel computing toolbox
- use built-in functions `createJob()` and `createTask()` for asynchronous jobs
- `parfor()` and distributed arrays for synchronous tasks

I. for MATLAB 2012a download and run installer.m from
http://gridmaster2012/matlab/
II. prepare input data on the master or use data dependencies
III. have a look at the very basic asynchronous example at:
http://gridmaster2012/matlab/example/



6.2 compiled MATLAB jobs

- it is possible to "translate" (package) matlab code to standalone binaries that in turn can be treated as classical jobs
- we had success with SPM and some simple MATLAB functions already
- usually all you have to do is
  - transform your main function to accept parameters
  - compile with `mcc -m func.m -a dependency1/ -a dependency2/`
  - wrap in job file and submit as

```
./run_func /opt/matlab/interactive parameter_list
```

- at the moment there is only 1 matlab compiler license available and it will be blocked for 30min once used - we are trying to fix that
- see detailed example on the next pages

6.2 compiled MATLAB jobs example

- suppose you have a main function that iterates over a combination of parameters where the most inner loop body takes at least a couple of minutes to process

```
function main()

outfile=fopen('output','w+')
for i = 1:100
  for j = 0:0.1:2
      temp = map(some_array, @some_func_1(i, j) )
      write(outfile, reduce(temp, @some_func_2) )
  end
end

function some_func(i,j)
[...]

function some_fun2()
[...]
```

6.2 compiled MATLAB jobs example

- what we can obviously do here is factor out both loops and compute each loop body in parallel (asynchronous with qsub)
- hence we would create a new main function called `new_main.m` that writes to its own output file and start this main function by passing to it the current pair of *i* and *j*
- we then compile the the new function: `mcc -m new_main.m`
- and submit it with the common qsub loop pattern

```
#!/bin/bash

for i in {1..100}; do
  for j in $(seq 0 0.1 2) ; do
    # qsub meta parameters go here
    echo "./run_new_main.sh /opt/matlab/interactive $i $j" >> jobfile
    qsub jobfile
    rm -f jobfile
  done
done
```

6.2 compiled MATLAB jobs example

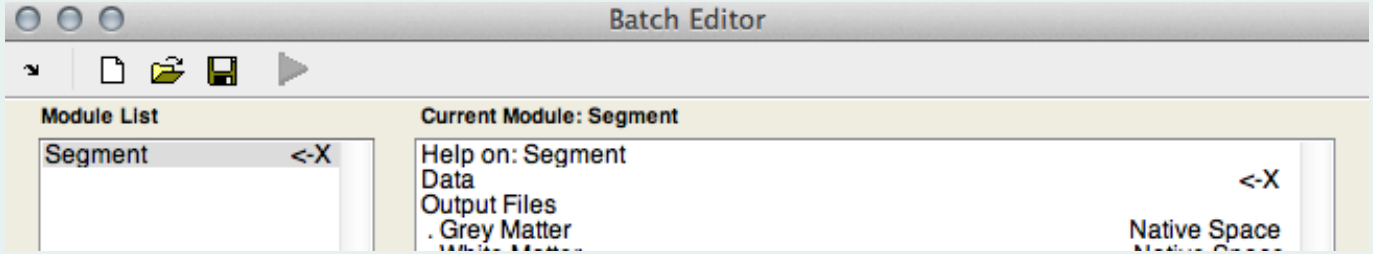
- the new main function would look something like this

```
function main(i, j)

# avoid name clash for output file
outfile=fopen([ 'output_',i,'_', j] ,'w+')
temp = map(some_array, @some_func_1(i, j) )
write(outfile, reduce(temp, @some_func_2) )

function some_func(i,j)
[...]

function some_fun2()
[...]
```

- inline functions will be available to the compiled program
- external toolboxes and libraries have to be manually appended to the compiled archive by specifiying the "-a" option of mcc (see `help mcc`)

6.3 SPM with MATLAB

- Statistical Parametric Mapping (SPM) is already using the compiler approach and plays well with the cluster
- the setup is split into two parts
  1. preparing and saving batch files on your workstation (preferably with the SPM gui) and copying the files and data dependencies over to the cluster



- note, that you might still need to change some input data paths/directory names since they are different on the cluster
- to avoid that use relative path names where possible

6.3 SPM with MATLAB

1. run the batch files with the pre compiled spm8 on the cluster with the loop pattern like this

```
#!/bin/bash

cd <some-matlab-project>
for b in *batches.m ; do
  echo "#PBS -m n "                    >> jobfile
  echo "#PBS -j oe "                   >> jobfile
  echo "#PBS -o $HOME/logs/"           >> jobfile
  # more PBS stuff...

  echo 'cd $PBS_O_WORKDIR'             >> jobfile
  echo "run_spm8.sh /opt/matlab/interactive batch $b" >> jobfile
  qsub jobfile
  rm -f jobfile
done
```

- there will be no gui available, so make sure the batches contain all the necessary settings
- you can testrun a single batch on the master node before submitting all jobs:

```
run_spm8.sh /opt/matlab/interactive batch some-batch-file.m
```

7 gnu-R

- gnu-R is easily scritpable and most simulations are embarassingly suited for parallel execution
- suppose you want to run a simulation for a big number of iterations (similar to the matlab example) and your main code looks like this

```
library(somelib)
source("mylib")

n <- 1000

for( i in seq(1,n) ) {
  r = simulation(i)
  write(r, file="output", append=TRUE)
}
```

- this R file is scriptable with Rscript, all we have to do is move the for loop outside and read the current value of i

7 gnu-R

- the resulting R function would then look something like this

```
library(somelib)
source("mylib")

n <- as.integer( commandArgs(TRUE)[1])

for( i in seq(1,n) ) {
  r = simulation(i)
  write(r, file=append("output_",i), append=TRUE)
}
```

- passing *i* as an input parameter and saving the output in an unique file
- this script can then be submitted with a common loop

7 gnu-R

- final qsub script, assuming the R script is called simulation.R

```
#!/bin/bash

cd <some-R-project>
for i in {1..1000} ; do
  echo "#PBS -m n "                    >> jobfile
  echo "#PBS -j oe "                   >> jobfile
  echo "#PBS -o $HOME/logs/"           >> jobfile
  # more PBS stuff...

  echo 'cd $PBS_O_WORKDIR'             >> jobfile
  echo "Rscript simulation.R "         >> jobfile
  qsub jobfile
  rm -f jobfile
done
```

8. external resources

- we already created two minimal documentations (cheat sheets) for the following program (family) that might come in handy for a quick reference
  - `ssh/scp` http://gridmaster2012/misc/ssh_cheat.pdf
  - `qsub` http://gridmaster2012/misc/qsub_cheat.pdf
- if you would like to see cheat sheets for additional programs, just let us know!

thank you for reading

any questions? do not hesitate to contact us at:
grid-admin@mpib-berlin.mpg.de