lrz

Leibniz Supercomputing Centre

of the Bavarian Academy of Sciences and Humanities

High Performance Data Analytics Using R at LRZ – Part 1

October 2022

Course Information

- The aim of this course is to demonstrate the different ways of using R efficiently and productively on LRZ systems with a focus on parallelization mechanisms and data analytics/machine learning tasks
- It is not an introduction to R itself
- Many of the topics covered in this course are based on issues encountered by users, for which they created tickets at the LRZ Servicedesk
- Also, it assumes you have some prior knowledge and experience in using GNU/Linux and SSH, as well as a good understanding of the LRZ HPC and BDAI Infrastructure (if you attended this week's courses, you are fine)

HPC & BDAI Systems for Bavarian Universities

Data Science Archive (DSA)

Archive and Backup (ABS)

Data Science Storage (DSS)

LRZ Linux Cluster

CoolMUC-2 Teramem CoolMUC-3

LRZ AI Systems

- “Big Data” CPU nodes
- HPE P100 node
- V100 nodes
- DGX-1 P100, DGX-1 V100
- (Multiple DGX A100)

AI

LRZ Compute Cloud

LRZ Compute Cloud
(w/ some GPUs)

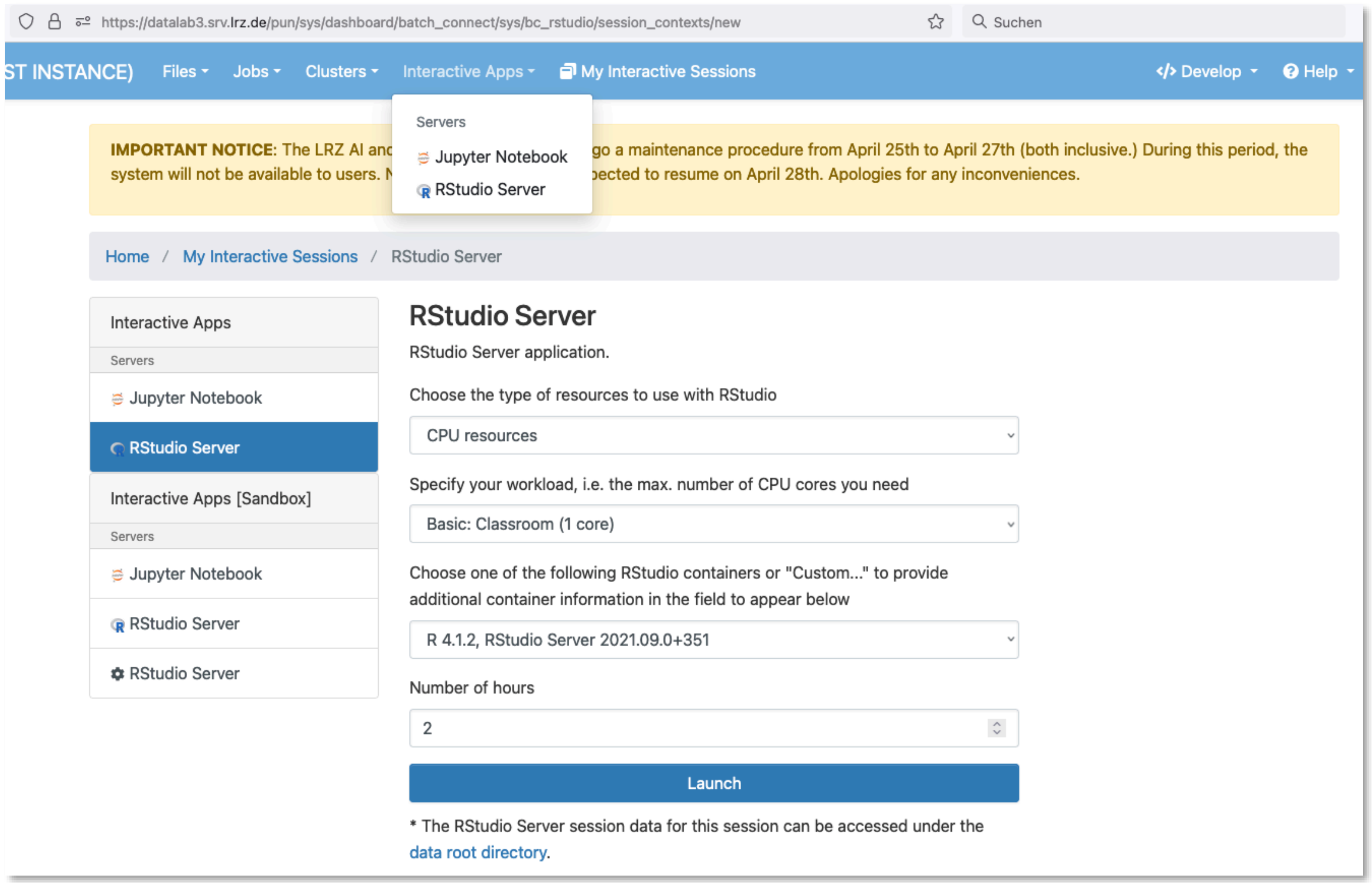
lxlogin[1-4].lrz.de
lxlogin8.lrz.de

datalab2.srv.lrz.de
https://datalab3.srv.lrz.de

https://cc.lrz.de

https://doku.lrz.de/display/PUBLIC/9.+Interactive+Web+Servers+on+the+LRZ+AI+Systems

AI Systems: RStudio Server



IMPORTANT NOTICE: The LRZ AI and … go a maintenance procedure from April 25th to April 27th (both inclusive.) During this period, the system will not be available to users. N… pected to resume on April 28th. Apologies for any inconveniences.

RStudio Server

RStudio Server application.

Choose the type of resources to use with RStudio: CPU resources

Specify your workload, i.e. the max. number of CPU cores you need: Basic: Classroom (1 core)

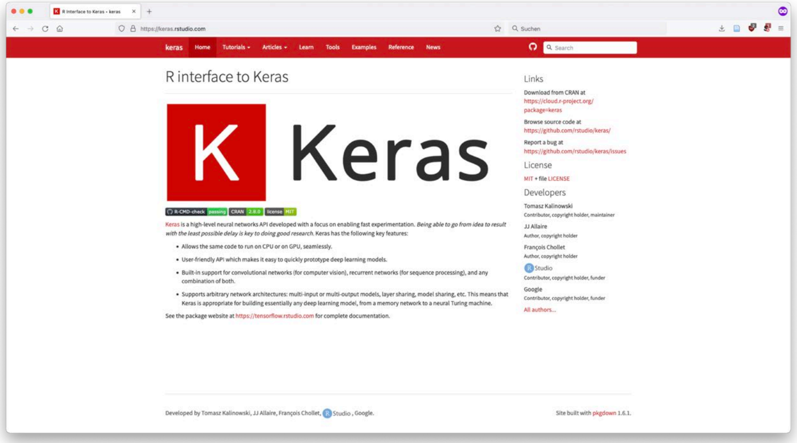
Choose one of the following RStudio containers or "Custom..." to provide additional container information in the field to appear below: R 4.1.2, RStudio Server 2021.09.0+351

Number of hours: 2

Launch

* The RStudio Server session data for this session can be accessed under the data root directory.

R for AI



R interface to Keras

Keras is a high-level neural networks API developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.* Keras has the following key features:

- Allows the same code to run on CPU or on GPU, seamlessly.
- User-friendly API which makes it easy to quickly prototype deep learning models.
- Built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
- Supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, etc. This means that Keras is appropriate for building essentially any deep learning model, from a memory network to a neural Turing machine.

See the package website at https://tensorflow.rstudio.com for complete documentation.

Links

Download from CRAN at https://cloud.r-project.org/package=keras

Browse source code at https://github.com/rstudio/keras/

Report a bug at https://github.com/rstudio/keras/issues

License

MIT + file LICENSE

Developers

Tomasz Kalinowski
Contributor, copyright holder, maintainer

JJ Allaire
Author, copyright holder

François Chollet
Author, copyright holder

RStudio
Contributor, copyright holder, funder

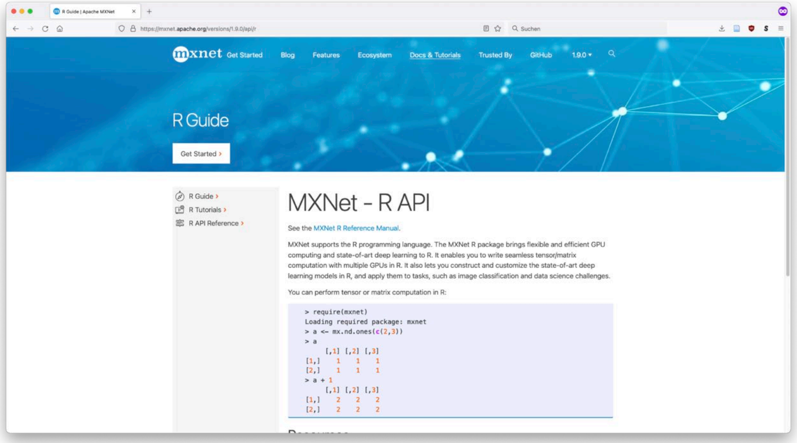
Google
Contributor, copyright holder, funder

All authors...

Developed by Tomasz Kalinowski, JJ Allaire, François Chollet, RStudio, Google.

Site built with pkgdown 1.6.1.

https://keras.rstudio.com/



MXNet - R API

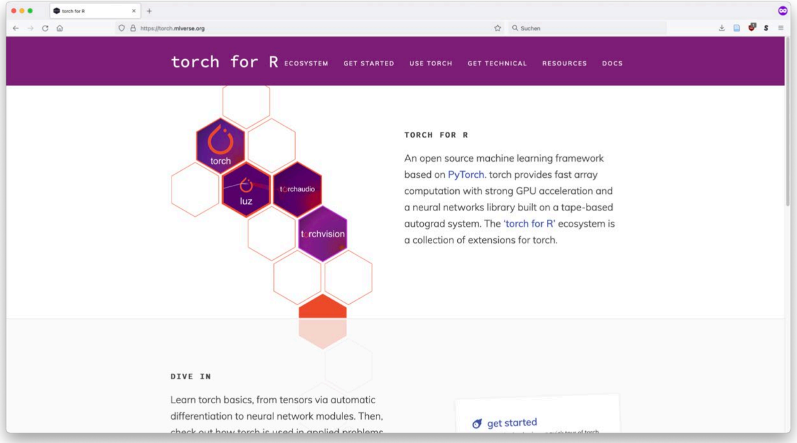
See the MXNet R Reference Manual.

MXNet supports the R programming language. The MXNet R package brings flexible and efficient GPU computing and state-of-art deep learning to R. It enables you to write seamless tensor/matrix computation with multiple GPUs in R. It also lets you construct and customize the state-of-art deep learning models in R, and apply them to tasks, such as image classification and data science challenges.

You can perform tensor or matrix computation in R:

```
> require(mxnet)
Loading required package: mxnet
> a <- mx.nd.ones(c(2,3))
> a
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
> a + 1
     [,1] [,2] [,3]
[1,]    2    2    2
[2,]    2    2    2
```

https://mxnet.apache.org/versions/1.9.0/api/r



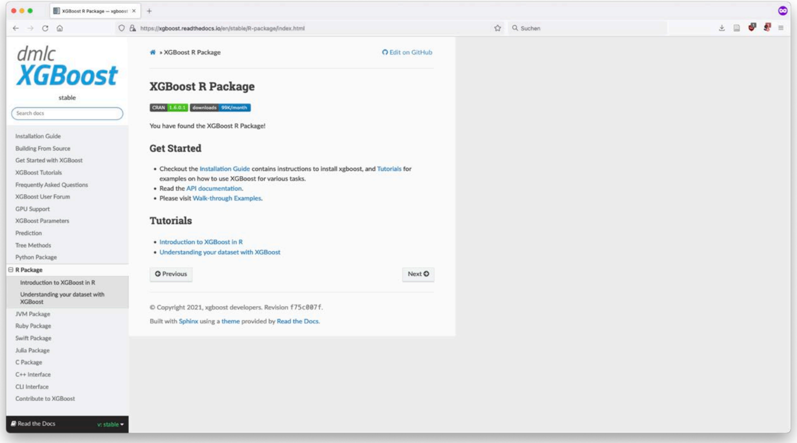
TORCH FOR R

An open source machine learning framework based on PyTorch. torch provides fast array computation with strong GPU acceleration and a neural networks library built on a tape-based autograd system. The 'torch for R' ecosystem is a collection of extensions for torch.

DIVE IN

Learn torch basics, from tensors via automatic differentiation to neural network modules. Then,

https://torch.mlverse.org/



XGBoost R Package

You have found the XGBoost R Package!

Get Started

- Checkout the Installation Guide contains instructions to install xgboost, and Tutorials for examples on how to use XGBoost for various tasks.
- Read the API documentation.
- Please visit Walk-through Examples.

Tutorials

- Introduction to XGBoost in R
- Understanding your dataset with XGBoost

© Copyright 2021, xgboost developers. Revision f75c007f.

Built with Sphinx using a theme provided by Read the Docs.

https://xgboost.readthedocs.io/en/stable/R-package/

… and more

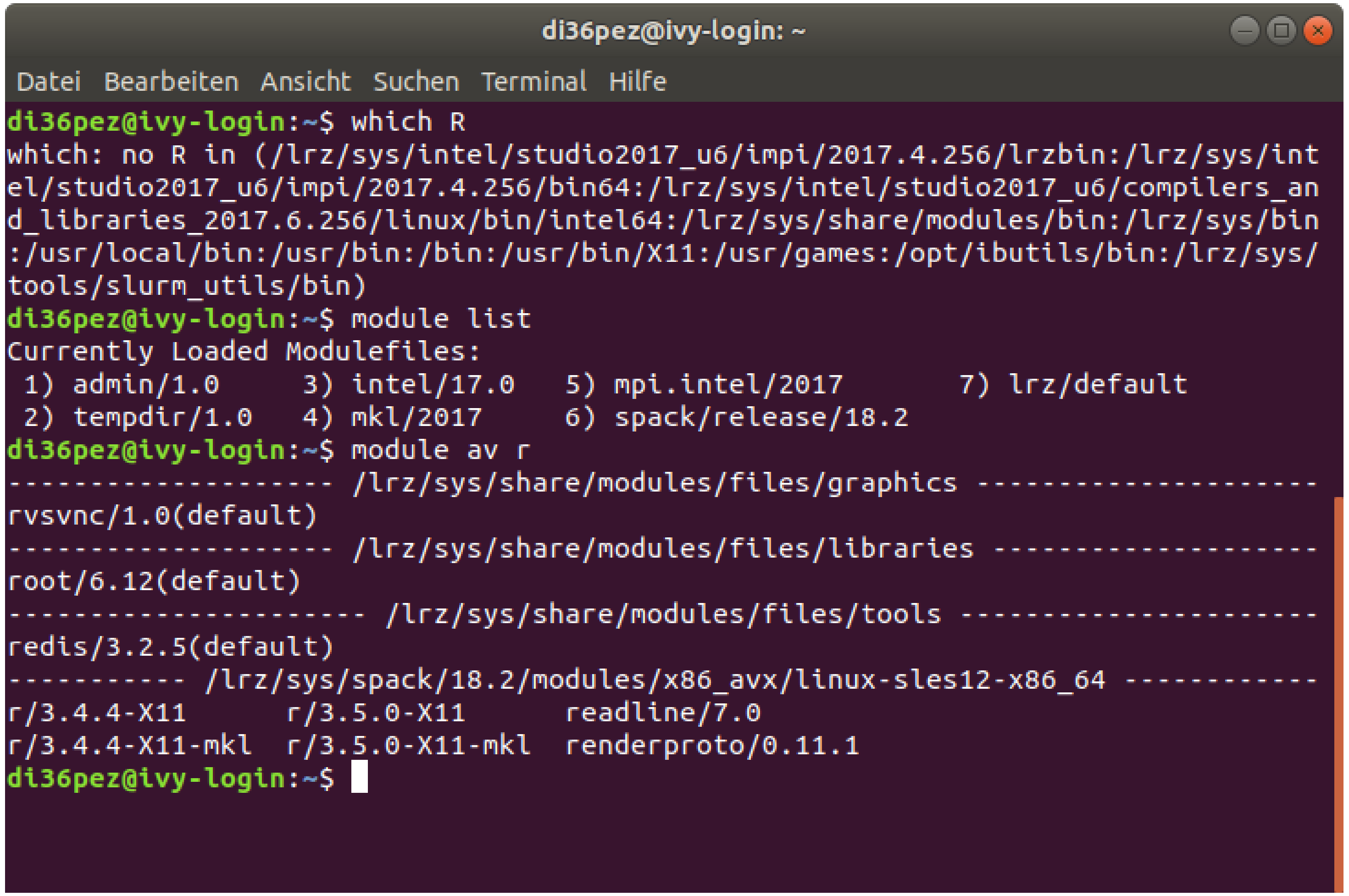
Linux Cluster

- Connect to the CoolMUC-2 segment of the Linux Cluster
- From a terminal application:
  `$ ssh <user>@lxlogin1.lrz.de`
- Alternatives would be lxlogin[2-4].lrz.de for CoolMUC-2 (or lxlogin8.lrz.de for CoolMUC-3)

R Modules

- R is not accessible on the Linux Cluster by default (try: `$ which R`)
- Environment modules allow for the dynamic modification of environment variables
- A (minimal) set of default modules is active after login:
  ```
  $ module list
  ```
- Use the module system to search for different R versions:
  `$ module available r` (or `module av r`)

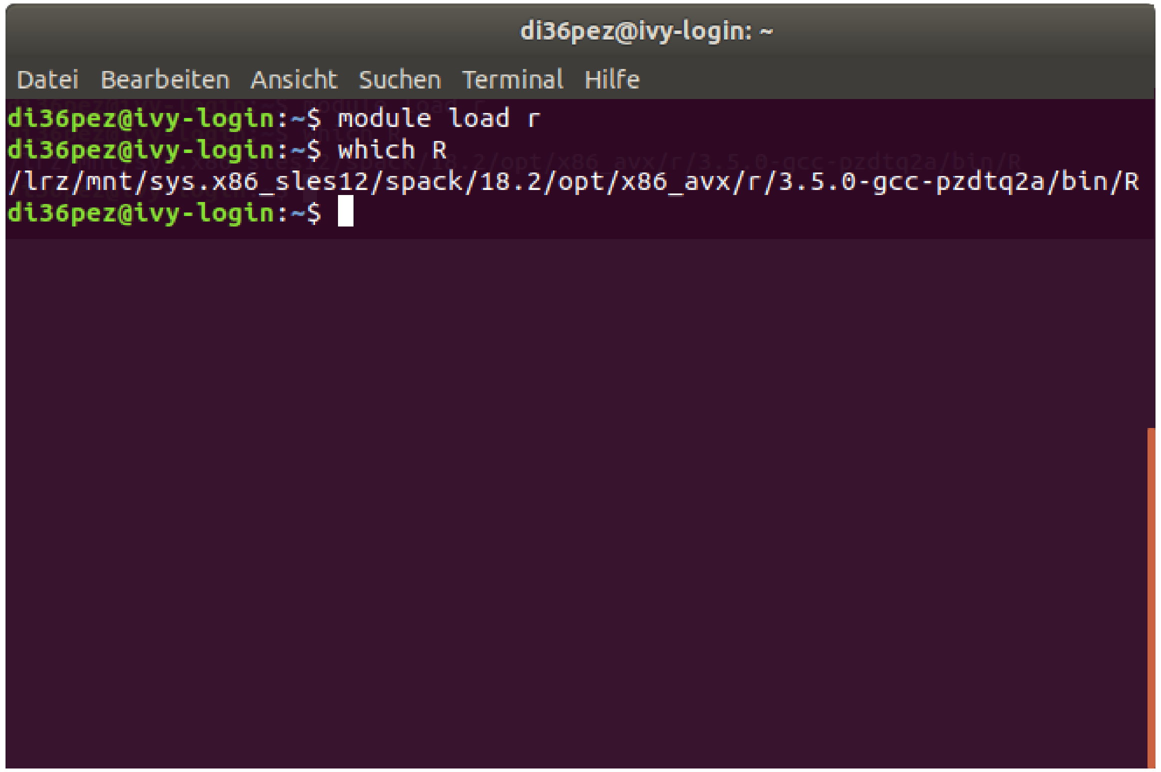
R Modules



```
di36pez@ivy-login:~$ which R
which: no R in (/lrz/sys/intel/studio2017_u6/impi/2017.4.256/lrzbin:/lrz/sys/int
el/studio2017_u6/impi/2017.4.256/bin64:/lrz/sys/intel/studio2017_u6/compilers_an
d_libraries_2017.6.256/linux/bin/intel64:/lrz/sys/share/modules/bin:/lrz/sys/bin
:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games:/opt/ibutils/bin:/lrz/sys/
tools/slurm_utils/bin)
di36pez@ivy-login:~$ module list
Currently Loaded Modulefiles:
 1) admin/1.0     3) intel/17.0   5) mpi.intel/2017       7) lrz/default
 2) tempdir/1.0   4) mkl/2017     6) spack/release/18.2
di36pez@ivy-login:~$ module av r
-------------------- /lrz/sys/share/modules/files/graphics --------------------
rvsvnc/1.0(default)
-------------------- /lrz/sys/share/modules/files/libraries --------------------
root/6.12(default)
---------------------- /lrz/sys/share/modules/files/tools ----------------------
redis/3.2.5(default)
----------- /lrz/sys/spack/18.2/modules/x86_avx/linux-sles12-x86_64 -----------
r/3.4.4-X11       r/3.5.0-X11      readline/7.0
r/3.4.4-X11-mkl  r/3.5.0-X11-mkl  renderproto/0.11.1
di36pez@ivy-login:~$ 
```

R Modules

- (The current default version of) R can be loaded using
  `$ module load r`
- If you need a different version, you have to specify the full name of the module, e.g. “r/3.4.4-gcc8-mkl”



```
di36pez@ivy-login:~$ module load r
di36pez@ivy-login:~$ which R
/lrz/mnt/sys.x86_sles12/spack/18.2/opt/x86_avx/r/3.5.0-gcc-pzdtq2a/bin/R
di36pez@ivy-login:~$
```

R Modules

- We are using the package manager Spack (https://spack.io) to provide applications/modules
- Spack “meta modules” make the (additional) module path(s) available
- By default, the latest LRZ release of Spack is loaded (cf. `$ module list`)
- If in doubt, stick to the final releases set as default (i.e. spack/YY.X.Z)!

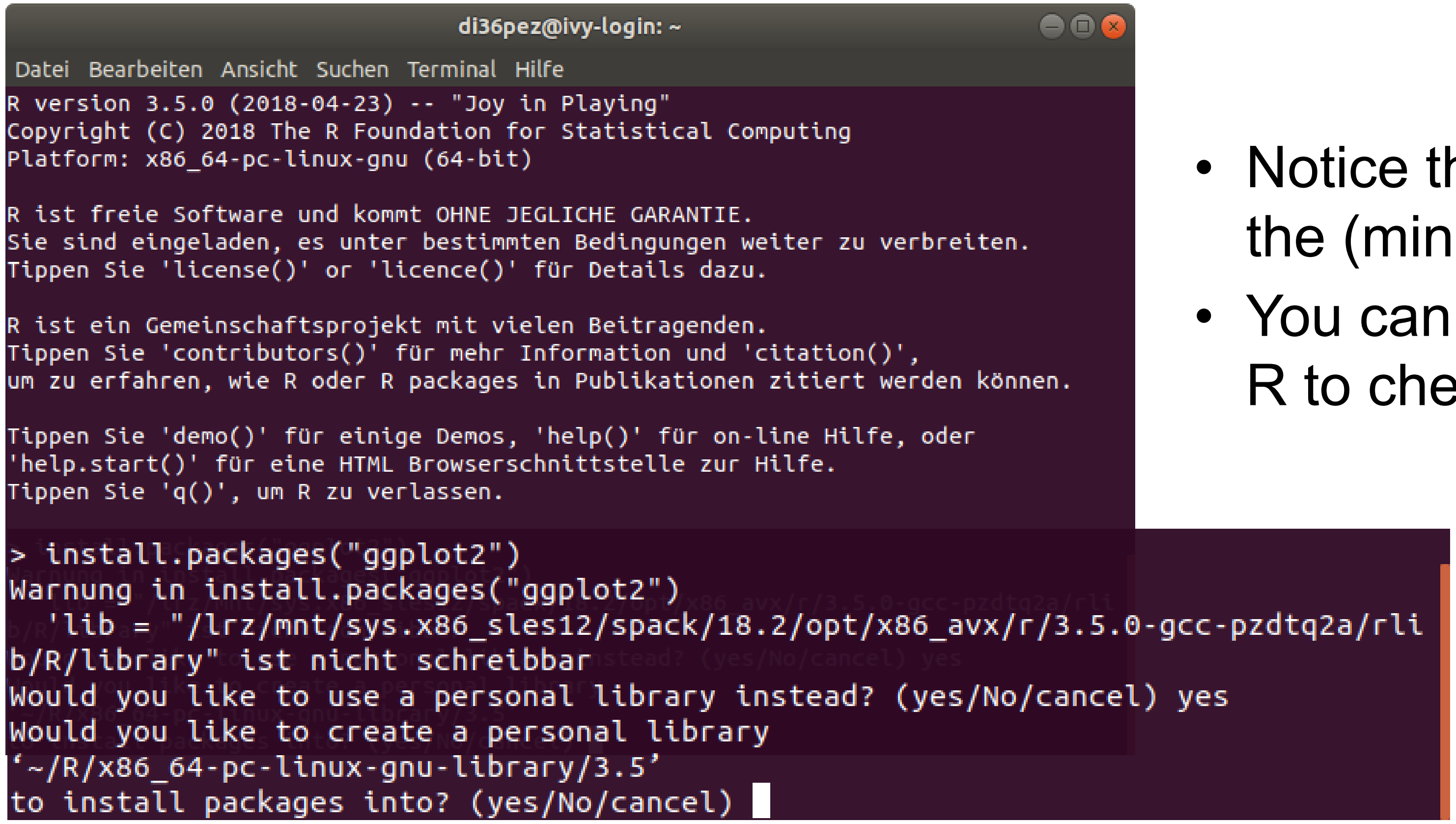
R Package Management

- All R packages are installed into libraries – these are (just) directories in the file system with subdirectories for each installed package
- The default installation of R comes with a single library (if defined, $R_HOME/library) usually containing the standard and recommended packages (in RStudio, this is called the System Library)
- On a multiuser system, regular users may not add/install packages directly into this library (but administrators can)
- On the Linux Cluster we only provide the standard set of base packages in this central location

R Package Management

- Individual users can have (one or more) additional, personal libraries (called User Library in RStudio)
- The path for this library directory can be specified by the environment variable $R_LIBS_USER (amongst others)
- If this is not defined, R will ask you to create a personal package library when installing packages for the first time…

R Package Management

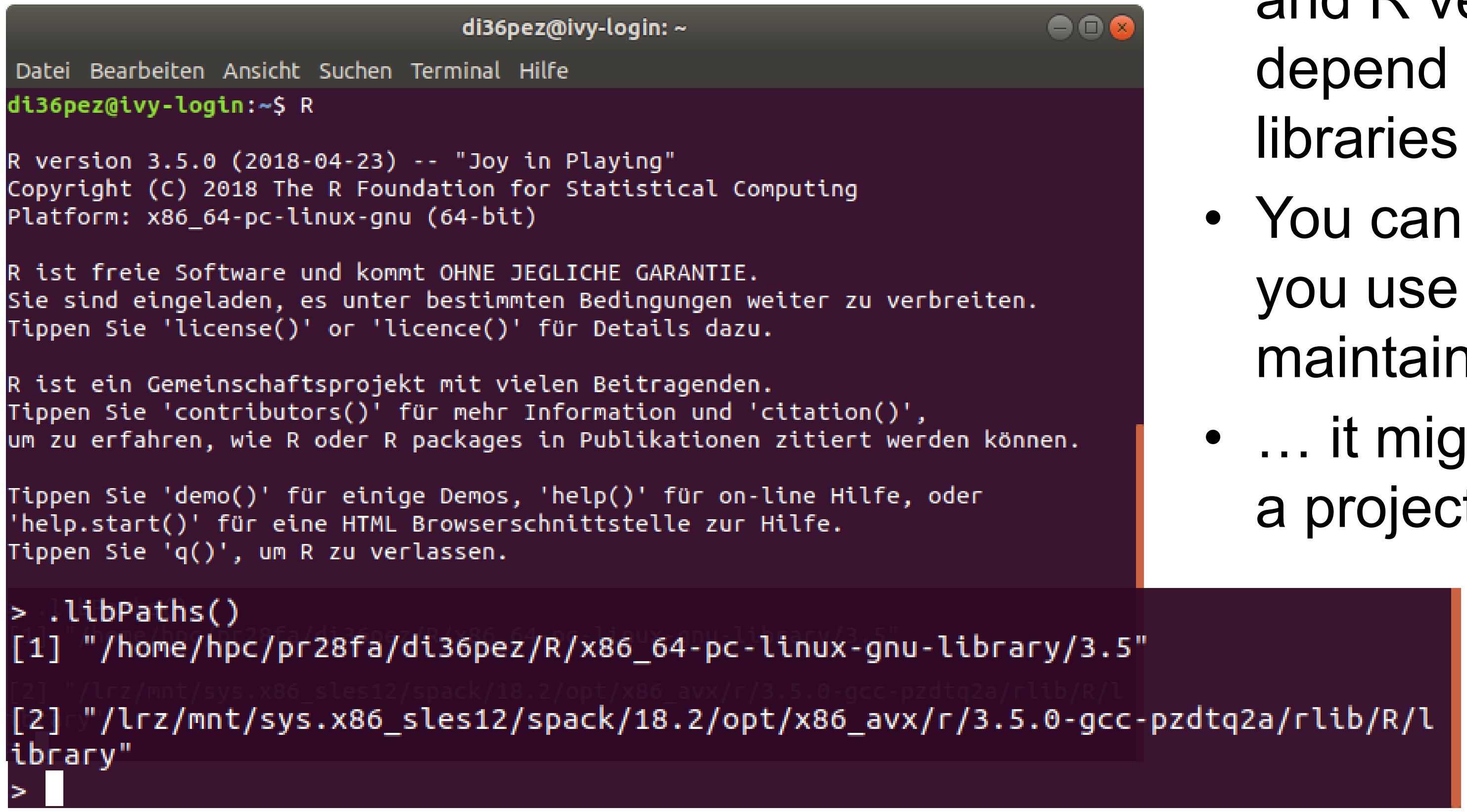


```
R version 3.5.0 (2018-04-23) -- "Joy in Playing"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R ist freie Software und kommt OHNE JEGLICHE GARANTIE.
Sie sind eingeladen, es unter bestimmten Bedingungen weiter zu verbreiten.
Tippen Sie 'license()' or 'licence()' für Details dazu.

R ist ein Gemeinschaftsprojekt mit vielen Beitragenden.
Tippen Sie 'contributors()' für mehr Information und 'citation()',
um zu erfahren, wie R oder R packages in Publikationen zitiert werden können.

Tippen Sie 'demo()' für einige Demos, 'help()' für on-line Hilfe, oder
'help.start()' für eine HTML Browserschnittstelle zur Hilfe.
Tippen Sie 'q()', um R zu verlassen.
```

```
> install.packages("ggplot2")
Warnung in install.packages("ggplot2")
  'lib = "/lrz/mnt/sys.x86_sles12/spack/18.2/opt/x86_avx/r/3.5.0-gcc-pzdtq2a/rli
b/R/library" ist nicht schreibbar
Would you like to use a personal library instead? (yes/No/cancel) yes
Would you like to create a personal library
'~/R/x86_64-pc-linux-gnu-library/3.5'
to install packages into? (yes/No/cancel)
```

- Notice the suggested path – it is specific to the (minor) version of R!
- You can use the .libPaths() function within R to check the current library directories…

R Package Management



```
di36pez@ivy-login:~$ R

R version 3.5.0 (2018-04-23) -- "Joy in Playing"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R ist freie Software und kommt OHNE JEGLICHE GARANTIE.
Sie sind eingeladen, es unter bestimmten Bedingungen weiter zu verbreiten.
Tippen Sie 'license()' or 'licence()' für Details dazu.

R ist ein Gemeinschaftsprojekt mit vielen Beitragenden.
Tippen Sie 'contributors()' für mehr Information und 'citation()',
um zu erfahren, wie R oder R packages in Publikationen zitiert werden können.

Tippen Sie 'demo()' für einige Demos, 'help()' für on-line Hilfe, oder
'help.start()' für eine HTML Browserschnittstelle zur Hilfe.
Tippen Sie 'q()', um R zu verlassen.

> .libPaths()
[1] "/home/hpc/pr28fa/di36pez/R/x86_64-pc-linux-gnu-library/3.5"
[2] "/lrz/mnt/sys.x86_sles12/spack/18.2/opt/x86_avx/r/3.5.0-gcc-pzdtq2a/rlib/R/l
ibrary"
> 
```

- So, subject to the system/cluster segment and R version you‘re using, you will depend on different system and user libraries
- You can always control the R packages you use (and their versions) by maintaining your user library…
- … it might even be beneficial to do this in a project-specific manner.

R Package Management

- A challenge: on GNU/Linux (most) „add-on“ R packages will be compiled from source
- This requires compilers, tools and additional dependencies available on the system
- For general compatibility use (a recent version) of the GNU Compiler Collection GCC to compile add-on packages. Setting this up has been automatized when loading the latest R modules.
  This is essentially equivalent to the following module commands:
  ```
  module unload intel-mpi
  module unload intel-oneapi-compilers
  module load gcc
  module load r
  ```
- If you miss any dependencies, make sure to check the available modules!
- And, as always: if you encounter any problems, please talk to us!

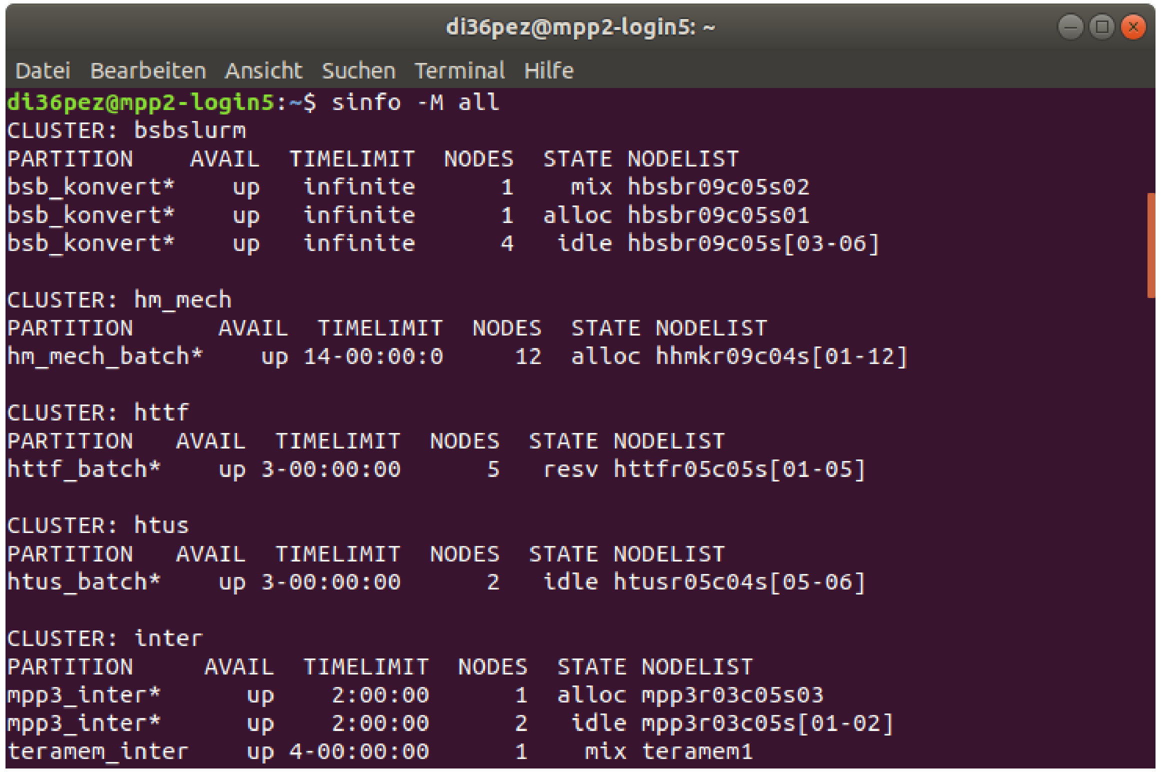
R Package Management

- Optional:
  there are package managers which can be run as user applications and may provide additional dependency requirements
- They manage R and (many of) its packages „from the outside“
- For this, you could take a look at Spack (https://spack.io) or conda (https://conda.io)

Slurm Workload Manager

- Slurm is a job scheduler:
  - Allocates access to resources (time, memory, nodes/cores)
  - Provides framework for starting, executing, and monitoring work
  - Manages queue of pending jobs (enforcing “fair share” policy)
- Use the `sinfo` command to get information about the available clusters
  `$ sinfo --clusters=all` or, shortened:
  `$ sinfo –M all`

Slurm Workload Manager



```
di36pez@mpp2-login5:~$ sinfo -M all
CLUSTER: bsbslurm
PARTITION    AVAIL  TIMELIMIT  NODES  STATE NODELIST
bsb_konvert*    up   infinite      1    mix hbsbr09c05s02
bsb_konvert*    up   infinite      1  alloc hbsbr09c05s01
bsb_konvert*    up   infinite      4   idle hbsbr09c05s[03-06]

CLUSTER: hm_mech
PARTITION      AVAIL  TIMELIMIT  NODES  STATE NODELIST
hm_mech_batch*    up 14-00:00:0     12  alloc hhmkr09c04s[01-12]

CLUSTER: httf
PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
httf_batch*    up 3-00:00:00      5   resv httfr05c05s[01-05]

CLUSTER: htus
PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
htus_batch*    up 3-00:00:00      2   idle htusr05c04s[05-06]

CLUSTER: inter
PARTITION     AVAIL  TIMELIMIT  NODES  STATE NODELIST
mpp3_inter*      up    2:00:00      1  alloc mpp3r03c05s03
mpp3_inter*      up    2:00:00      2   idle mpp3r03c05s[01-02]
teramem_inter    up 4-00:00:00      1    mix teramem1
```

- Look for the cluster segments
  - inter (allows for interactive usage)
  - cm2 (the main CoolMUC-2 cluster)
  - serial (shared nodes for serial jobs)
- What is their current status?
- Get information about a specific cluster segment, e.g.
  `$ sinfo -M inter` or
  `$ sinfo -M cm2`

CoolMUC-2 Overview

Slurm Cluster	Slurm Partition	Node Range	Slurm Job Settings
cm2	cm2_large	25-64	`--clusters=cm2` `--partition=cm2_large` `--qos=cm2_large`
	cm2_std	3-24	`--clusters=cm2` `--partition=cm2_std` `--qos=cm2_std`
cm2_tiny	cm2_tiny	1-4	`--clusters=cm2_tiny`
serial	serial_std	1	`--clusters=serial` `--partition=serial_std` `--mem=<memory_per_node>MB`
	serial_long	1	`--clusters=serial` `--partition=serial_long` `--mem=<memory_per_node>MB`
inter	cm2_inter	1-4	`--clusters=inter` `--partition=cm2_inter`
	teramem_inter	1	`--clusters=inter` `--partition=teramem_inter`

For additional details see https://doku.lrz.de/display/PUBLIC/Job+Processing+on+the+Linux-Cluster

Interactive R Session

- The inter cluster can be used for interactive resource allocation:
```
$ salloc -p cm2_inter -N 1
```
- Using this shell, you can e.g. run R interactively on this node (if the R module is loaded):
```
$ R
```

Interactive R Session

```
user@cm2login1:~$ salloc -p cm2_inter -N 1
salloc: Granted job allocation 159945
user@i22r07c05s11:~$ module load r
user@i22r07c05s11:~$ R

R version 3.6.3 (2020-02-29) -- "Holding the Windsock"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

[...]

> library(parallel)
> detectCores()
[1] 56
>
```

Job Processing

- For production jobs, you want to prepare and submit batch scripts
- They tell Slurm about the resources you need and the scripts/programs you want to run…

Job Processing

```
#!/bin/bash
#SBATCH --clusters=cm2_tiny
#SBATCH --nodes=1

module load slurm_setup

module load r

Rscript myscript.R
```

- A very minimal example of a job script (not necessarily recommended, but working in some cases), requesting
  - a single, exclusive node (with 28 cores)
  - of the cm2_tiny partition/cluster, part of
  - the CoolMUC-2 system
- Submit this job script to the queue:
  $ sbatch <myjob.sh>

Job Processing

```
#!/bin/bash
#SBATCH -o /dss/dsshome1/.../.../myjob.%j.%N.out
#SBATCH -D /dss/dsshome1/.../.../workdir
#SBATCH -J jobname
#SBATCH --get-user-env
#SBATCH --clusters=cm2
#SBATCH --partition=cm2_std
#SBATCH --nodes=3
#SBATCH --mail-type=end
#SBATCH --mail-user=xyz@xyz.de
#SBATCH --export=NONE
#SBATCH --time=08:00:00

# module load slurm_setup
# module unload intel-mpi
# module load openmpi

module load r

mpirun R –f myscript.R
```

- A more practical example…
  - defining custom output file(s)
  - setting a working directory
  - assigning a job name
  - configuring mail notifications
  - managing the environment
  - limiting walltime explicitly
- See documentation for more details: https://doku.lrz.de/x/AgaVAg

Job Management and Accounting

- Submit a job:
```
$ sbatch myjob.sh
```
- Query status of your jobs:
```
$ squeue -M <cluster> -u <user>
```
- Approximate start time of pending jobs:
```
$ squeue -M <cluster> -u <user> --start
```
- Abort a job:
```
$ scancel -M <cluster> <jobid>
```
- Get accounting data for (past) jobs:
```
$ sacct -X -M <cluster> [-S <YYYY-MM-DD>] -u <user>
```

Potential Pitfalls

- Jobs get aborted (by Slurm) if they use more resources than specified
  -> you need to estimate memory and runtime requirements
  - Estimate memory requirements from a (single, local) serial run, extrapolate if needed (use e.g. your system monitor or the “top” command line tool)
  - Provide some “buffer” for runtime
- Queuing times can be long
  - Use “sinfo” to find less busy cluster segments
  - Smaller, less demanding jobs generally start faster
    -> you can benefit from accurate resource estimation

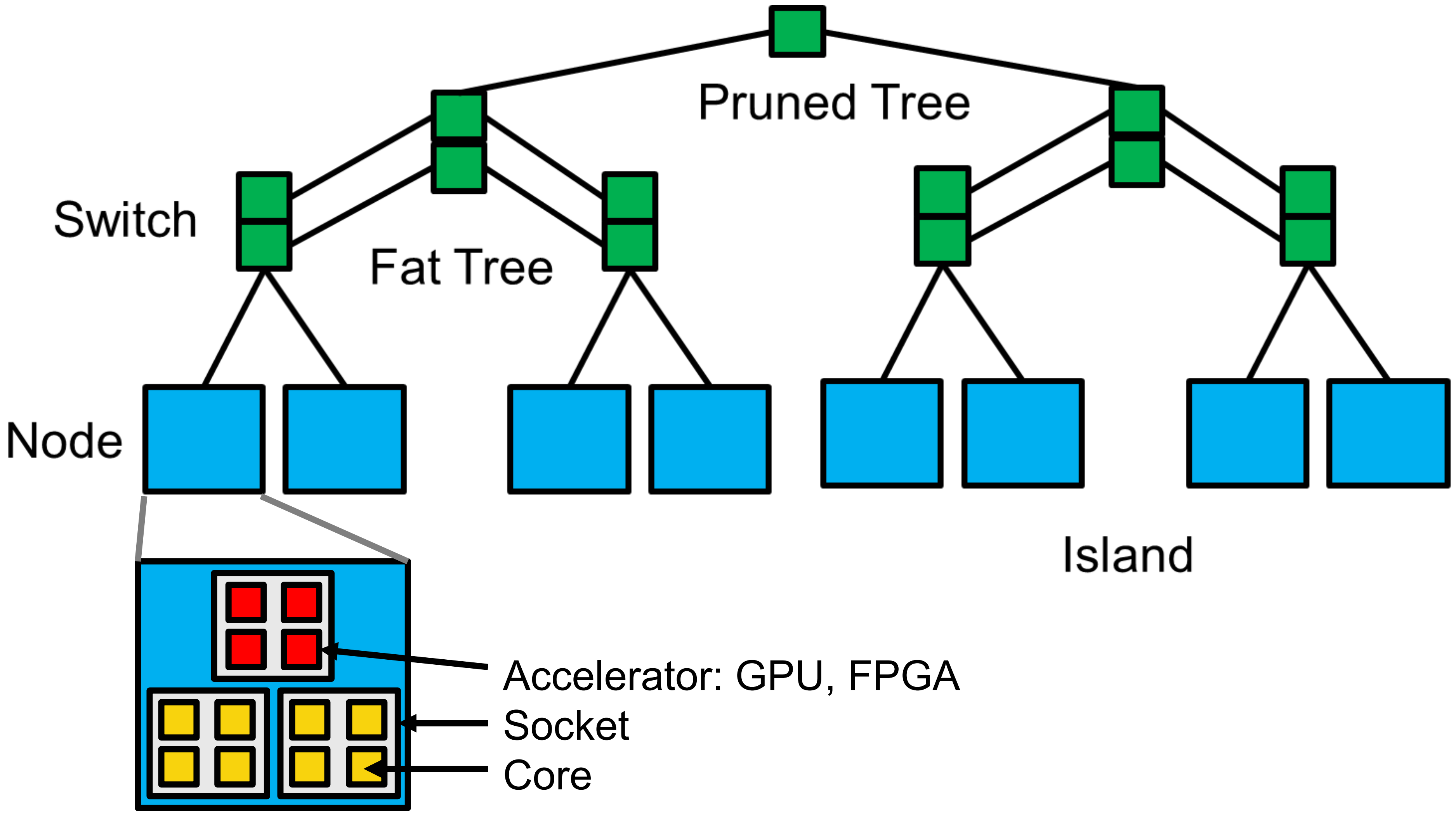
Potential Pitfalls

- Debugging can be inconvenient
- The time interval between changes in the R code and seeing results/getting feedback is longer than usual
- The compute environment (compute nodes of the cluster) and the development/test environments (local, login or interactive nodes) are usually not exactly the same
  - Debug as much as possible in a serial fashion
  - Prepare small jobs and test them interactively (using “salloc”/”srun”)

High Performance Data Analytics Using R at LRZ – Part 2

2022-04-22 | J. Albert-von der Gönna

HPC/AI Cluster Systems



Parallelization

Motivation:

- You have a lot of (more or less) indepen-dent tasks or

- You want to accelerate a single complex task -> it might be possible to turn the single complex task into many smaller (more or less) independent tasks

…and you have access to a (massively parallel) supercomputer/multiuser cluster!

Parallelization Scenario: Embarrassingly/Pleasingly Parallel

- many independent processes (10 - 100.000)
- individual task (list) for each process
- private memory for each process
- no communication between processes
- results are stored separately on a (large) storage medium

Parallelization Scenario: Worker Queue

- many independent processes (10 - 100.000)
- central task scheduler (database)
- private memory for each process
- results are sent back to task scheduler
- re-scheduling of failed tasks possible

Parallelization Scenario: Shared Memory

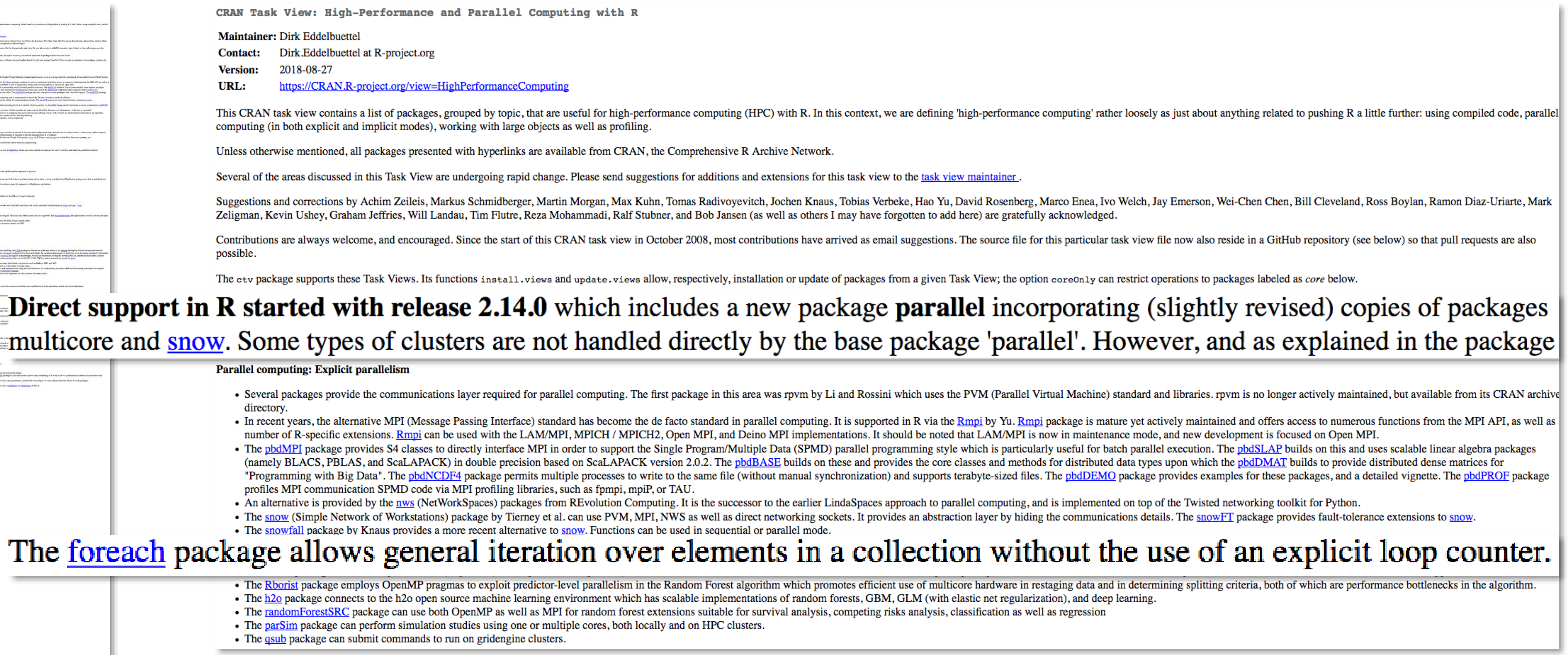
- a few processes working closely together (10-100)
- single task list (script/program)
- shared memory
  (cache coherent non-uniform memory architecture aka ccNUMA)
- results are kept in shared memory

Parallelization Scenario: Message Passing

- many independent processes
  (10 - 100.000)
- one task list (script/program) for all processes
- each process can (in principle) talk to every other process
- private memory
- needs communication strategy in order to scale (area of optimization, e.g. nearest neighbor communication)
- beware of deadlocks!

https://cran.r-project.org/web/views/HighPerformanceComputing.html

CRAN Task View: High-Performance and Parallel Computing



CRAN Task View: High-Performance and Parallel Computing with R

Maintainer:	Dirk Eddelbuettel
Contact:	Dirk.Eddelbuettel at R-project.org
Version:	2018-08-27
URL:	https://CRAN.R-project.org/view=HighPerformanceComputing

This CRAN task view contains a list of packages, grouped by topic, that are useful for high-performance computing (HPC) with R. In this context, we are defining 'high-performance computing' rather loosely as just about anything related to pushing R a little further: using compiled code, parallel computing (in both explicit and implicit modes), working with large objects as well as profiling.

Unless otherwise mentioned, all packages presented with hyperlinks are available from CRAN, the Comprehensive R Archive Network.

Several of the areas discussed in this Task View are undergoing rapid change. Please send suggestions for additions and extensions for this task view to the task view maintainer .

Suggestions and corrections by Achim Zeileis, Markus Schmidberger, Martin Morgan, Max Kuhn, Tomas Radivoyevitch, Jochen Knaus, Tobias Verbeke, Hao Yu, David Rosenberg, Marco Enea, Ivo Welch, Jay Emerson, Wei-Chen Chen, Bill Cleveland, Ross Boylan, Ramon Diaz-Uriarte, Mark Zeligman, Kevin Ushey, Graham Jeffries, Will Landau, Tim Flutre, Reza Mohammadi, Ralf Stubner, and Bob Jansen (as well as others I may have forgotten to add here) are gratefully acknowledged.

Contributions are always welcome, and encouraged. Since the start of this CRAN task view in October 2008, most contributions have arrived as email suggestions. The source file for this particular task view file now also reside in a GitHub repository (see below) so that pull requests are also possible.

The `ctv` package supports these Task Views. Its functions `install.views` and `update.views` allow, respectively, installation or update of packages from a given Task View; the option `coreOnly` can restrict operations to packages labeled as *core* below.

Direct support in R started with release 2.14.0 which includes a new package **parallel** incorporating (slightly revised) copies of packages multicore and snow. Some types of clusters are not handled directly by the base package 'parallel'. However, and as explained in the package

Parallel computing: Explicit parallelism

- Several packages provide the communications layer required for parallel computing. The first package in this area was rpvm by Li and Rossini which uses the PVM (Parallel Virtual Machine) standard and libraries. rpvm is no longer actively maintained, but available from its CRAN archive directory.
- In recent years, the alternative MPI (Message Passing Interface) standard has become the de facto standard in parallel computing. It is supported in R via the Rmpi by Yu. Rmpi package is mature yet actively maintained and offers access to numerous functions from the MPI API, as well as number of R-specific extensions. Rmpi can be used with the LAM/MPI, MPICH / MPICH2, Open MPI, and Deino MPI implementations. It should be noted that LAM/MPI is now in maintenance mode, and new development is focused on Open MPI.
- The pbdMPI package provides S4 classes to directly interface MPI in order to support the Single Program/Multiple Data (SPMD) parallel programming style which is particularly useful for batch parallel execution. The pbdSLAP builds on this and uses scalable linear algebra packages (namely BLACS, PBLAS, and ScaLAPACK) in double precision based on ScaLAPACK version 2.0.2. The pbdBASE builds on these and provides the core classes and methods for distributed data types upon which the pbdDMAT builds to provide distributed dense matrices for "Programming with Big Data". The pbdNCDF4 package permits multiple processes to write to the same file (without manual synchronization) and supports terabyte-sized files. The pbdDEMO package provides examples for these packages, and a detailed vignette. The pbdPROF package profiles MPI communication SPMD code via MPI profiling libraries, such as fpmpi, mpiP, or TAU.
- An alternative is provided by the nws (NetWorkSpaces) packages from REvolution Computing. It is the successor to the earlier LindaSpaces approach to parallel computing, and is implemented on top of the Twisted networking toolkit for Python.
- The snow (Simple Network of Workstations) package by Tierney et al. can use PVM, MPI, NWS as well as direct networking sockets. It provides an abstraction layer by hiding the communications details. The snowFT package provides fault-tolerance extensions to snow.
- The snowfall package by Knaus provides a more recent alternative to snow. Functions can be used in sequential or parallel mode.

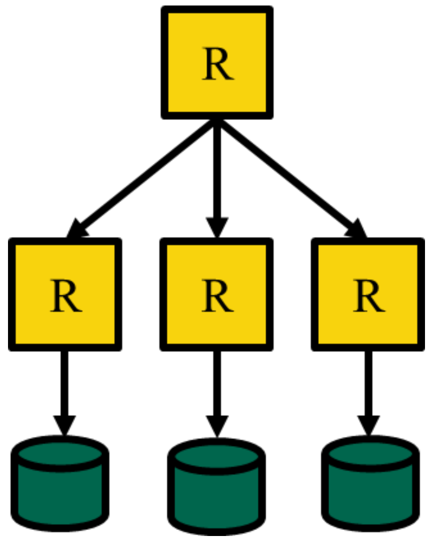
The foreach package allows general iteration over elements in a collection without the use of an explicit loop counter.

- The Rborist package employs OpenMP pragmas to exploit predictor-level parallelism in the Random Forest algorithm which promotes efficient use of multicore hardware in restaging data and in determining splitting criteria, both of which are performance bottlenecks in the algorithm.
- The h2o package connects to the h2o open source machine learning environment which has scalable implementations of random forests, GBM, GLM (with elastic net regularization), and deep learning.
- The randomForestSRC package can use both OpenMP as well as MPI for random forest extensions suitable for survival analysis, competing risks analysis, classification as well as regression
- The parSim package can perform simulation studies using one or multiple cores, both locally and on HPC clusters.
- The qsub package can submit commands to run on gridengine clusters.

(Explicit) Parallelization Using R

- Embarrassingly/pleasingly parallel (independent processes):
  - basic approach: start as many R processes as you need in the shell with different scripts

Parallelization Using R: Embarrassingly/Pleasingly Parallel



Embarrassingly/Pleasingly Parallel

```
$ R –f script.R &
```

Parallelization Using R: Embarrassingly/Pleasingly Parallel

- Use the command line to start your R process (in the background):
  ```
  $ Rscript script0.R &
  ```
- If you do this repeatedly, the resulting R processes will be distributed by the OS to different cores (subject to availability):
  ```
  $ Rscript script1.R &
  $ Rscript script2.R &
  $ Rscript script3.R & …
  ```
- To further automate this procedure, you could write a bash script (run_all_R_scripts.sh) containing these commands and then run this single script:
  ```
  $ bash run_all_R_scripts.sh &
  ```

- Typically, do not start more processes than cores!
- Do not use the (cluster) login nodes for this (e.g. request an interactive shell instead)!

https://callr.r-lib.org/

There is, of course, also an R package to achieve this: callr

- "It is sometimes useful to perform a computation in a separate R process, without affecting the current R process at all. This package does exactly that."

- Use r() to run an R function in a new R process.

```
library(callr)
r(function() var(iris[, 1:4]))

#>              Sepal.Length Sepal.Width Petal.Length Petal.Width
#> Sepal.Length    0.6856935  -0.0424340    1.2743154   0.5162707
#> Sepal.Width    -0.0424340   0.1899794   -0.3296564  -0.1216394
#> Petal.Length    1.2743154  -0.3296564    3.1162779   1.2956094
#> Petal.Width     0.5162707  -0.1216394    1.2956094   0.5810063
```

Parallelization Using R: Embarrassingly/Pleasingly Parallel

- Let’s look at a toy problem:

```
for(i in 1:20) sum(sort(runif(1e7)))
```

- Add a time measurement:

```
system.time(for(i in 1:20) sum(sort(runif(1e7))))
```

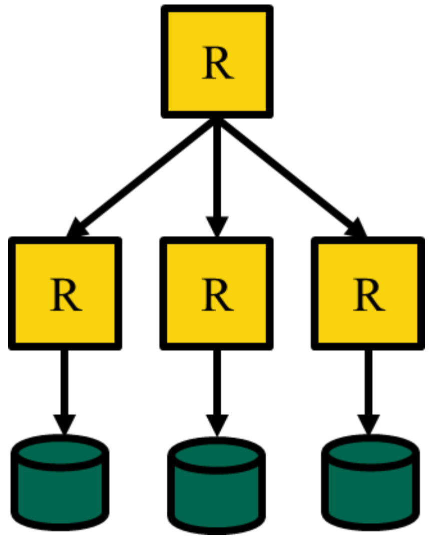
- You might also be familiar with alternatives like the following:

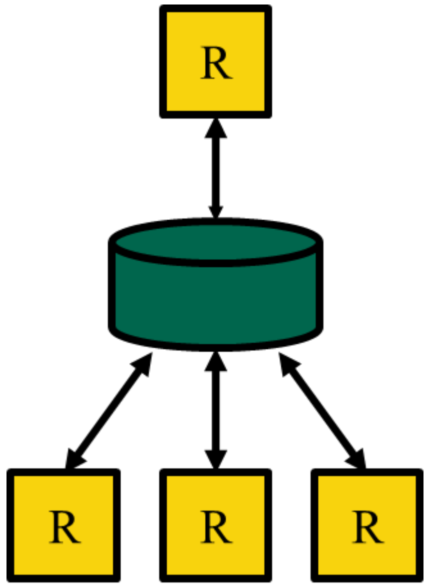
```
lapply(1:20, function(x) sum(sort(runif(1e7))))
```

(Explicit) Parallelization Using R

- Embarrassingly/pleasingly parallel (independent processes):
  - basic approach: start as many R processes as you need in the shell with different scripts
- Worker Queue (weak coupling, shared file system or database):
  - a main process (with access to a database/shared file system) coordinates several R processes, potentially on different compute nodes (e.g. batchtools, rredis/doRedis)

Parallelization Using R: Worker Queue





Embarrassingly/Pleasingly Parallel

```
$ R -f script.R &
```

Shared file system or database

rredis/doRedis, batchtools, clustermq

Parallelization Using R: rredis/doRedis

Redis is an open source, fast, persistent, networked database with many features, among them a blocking queue-like data structure (Redis “lists”). This feature makes Redis useful as a lightweight back end for parallel computing.

A Redis server has to be set up as part of the cluster (e.g. on a login node) or even somewhere else, containing the problem description(s). Worker processes connect to this server and tasks are assigned to them.

This is a very flexible and dynamic approach, as workers can basically run wherever you want (as long as they can connect to the server). When running on the cluster, you have to deal with resource allocation separately (via the Slurm workload manager) and potential firewall access restrictions.

https://mllg.github.io/batchtools/

Parallelization Using R: batchtools

"batchtools provides a parallel implementation of Map for high performance computing systems managed by schedulers like Slurm, …

- all relevant batch system operations (submitting, listing, killing) are either handled internally or abstracted via simple R functions
- with a well-defined interface, the source is independent from the underlying batch system - prototype locally, deploy on any high performance cluster"

i.e. a (interactive) R process is used in combination with the shared file system and the workload manager of the cluster to distribute workloads across nodes https://mschubert.github.io/clustermq/

Parallelization Using R: clustermq

- Allows to send function calls as jobs on a computing cluster with a minimal interface provided by the Q() function

```
library(clustermq)
fx = function(x) x * 2

Q(fx, x=1:3, n_jobs=1)
```

- All calculations are load-balanced, i.e. workers that get their jobs done faster will also receive more function calls to work on
- Computations are done entirely on the network and without any temporary files on network-mounted storage

https://mschubert.github.io/clustermq/

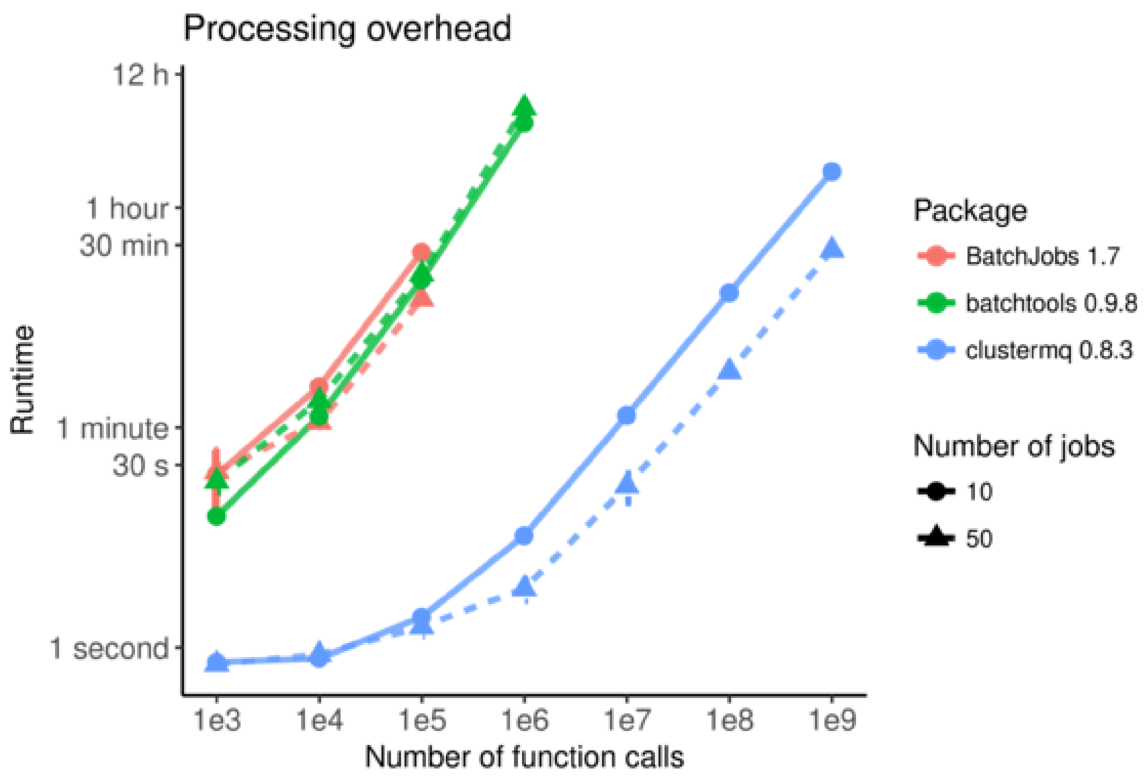
Parallelization Using R: clustermq

Use clustermq if you want:

- a one-line solution to run cluster jobs with minimal setup
- access cluster functions from your local machine (e.g. using RStudio) via SSH
- fast processing of many function calls without network storage I/O

Use batchtools if you:

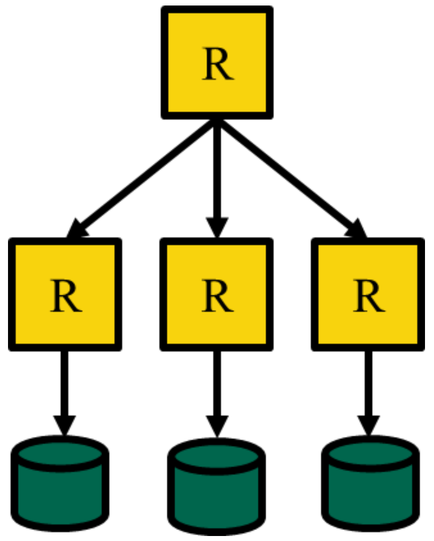
- want to use a mature and well-tested package
- don’t mind that arguments to every call are written to/read from disc
- don’t mind there’s no load-balancing at run-time



(Explicit) Parallelization Using R

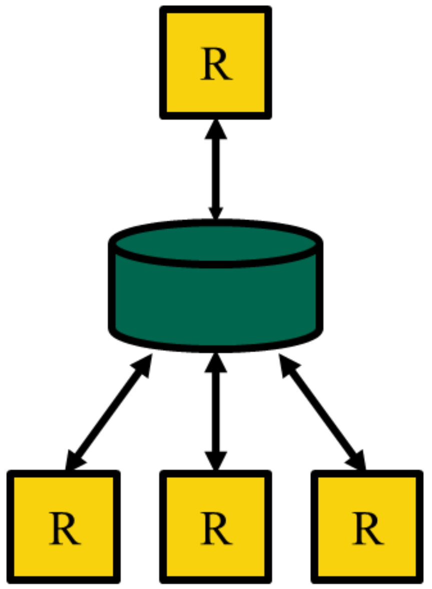
- Embarrassingly/pleasingly parallel (independent processes):
  - basic approach: start as many R processes as you need in the shell with different scripts
- Worker Queue (weak coupling, shared file system or database):
  - a main process (with access to a database/shared file system) coordinates several R processes, potentially on different compute nodes (e.g. batchtools, rredis/doRedis)
- Shared Memory (strong coupling):
  - one R process spawns sub-processes on a single node with many cores (e.g. parallel/doParallel; formerly multicore/doMC, snow/doSNOW)

Parallelization Using R: Shared Memory



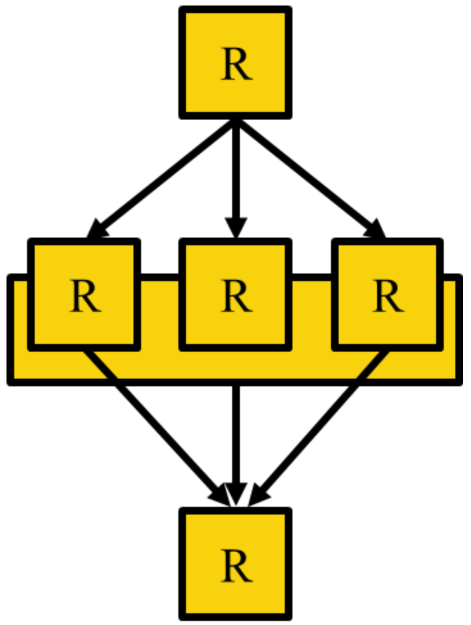
Embarrassingly/Pleasingly Parallel

```
$ R -f script.R &
```



Shared file system or database

rredis/doRedis, batchtools, clustermq



Shared memory

parallel/doParallel

Shared Memory Parallelization: Multithreading with doParallel

- As seen earlier, the for loop construct in R:

```
for(i in 1:20) sum(sort(runif(1e7)))
  # serial execution/single thread
```

- “The foreach package provides a new looping construct for executing R code repeatedly. […] it supports parallel execution, that is, it can execute those repeated operations on multiple processors/cores on your computer, or on multiple nodes of a cluster.”

```
library(foreach)
foreach(i = 1:20) %do% sum(sort(runif(1e7)))  # serial execution

foreach(i = 1:20) %dopar% sum(sort(runif(1e7)))
  # multithread execution (?)
```

Shared Memory Parallelization: Multithreading with doParallel

- This is were the "do-backends" (aka %dopar% adapters, e.g. doParallel) come into play…
- By creating/registering a cluster, foreach's %dopar% operator can rely on these parallel resources, e.g. using parallel's multicore-like functionality ("forking"):

```
library(foreach)
library(doParallel)
registerDoParallel(cores=4)
  # define number of cores, this enables multicore-functionality
  # (preferred on GNU/Linux, but won't work on Windows)
foreach(i = 1:20) %dopar% sum(sort(runif(1e7)))
```

Shared Memory Parallelization: Multithreading with doParallel

- The procedure is similar for snow-like functionality:

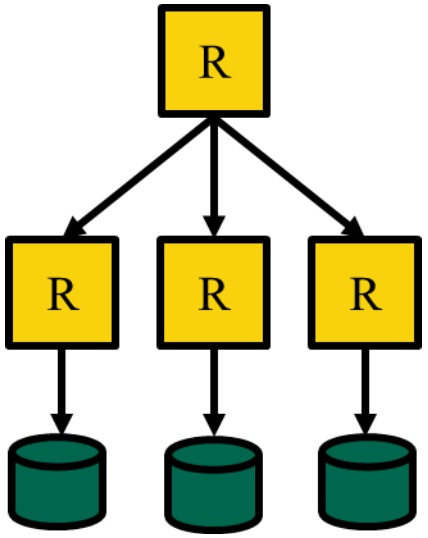
```
library(foreach)
library(doParallel)
cluster.object <- makePSOCKcluster(4)
registerDoParallel(cluster.object)
foreach(i = 1:20) %dopar% sum(sort(runif(1e7)))
stopCluster(cluster.object)
```

- This uses Rscript to launch further copies of R (on the same host or optionally elsewhere; in the latter case, hostnames need to be provided)
- [parallel, by relying on snow, also allows to create MPI-clusters (makeMPIcluster()-function) but Rmpi/doMPI is usually recommended to be used instead]

(Explicit) Parallelization Using R

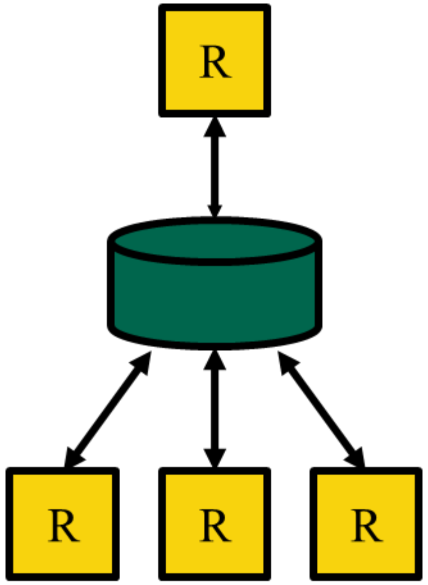
- Embarrassingly/pleasingly parallel (independent processes):
  - basic approach: start as many R processes as you need in the shell with different scripts
- Worker Queue (weak coupling, shared file system or database):
  - a main process (with access to a database/shared file system) coordinates several R processes, potentially on different compute nodes (e.g. batchtools, rredis/doRedis)
- Shared Memory (strong coupling):
  - one R process spawns sub-processes on a single node with many cores (e.g. parallel/doParallel; formerly multicore/doMC, snow/doSNOW)
- Message Passing (strong coupling):
  - several R processes talk to each other (across different nodes) by passing messages (e.g. Rmpi/doMPI), this also allows for a (single) main and (multiple) workers model

Parallelization Using R: Message Passing



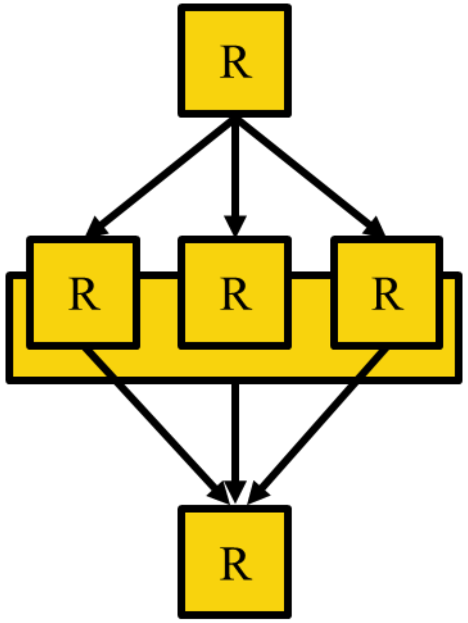
Embarrassingly/Pleasingly Parallel

```
$ R -f script.R &
```



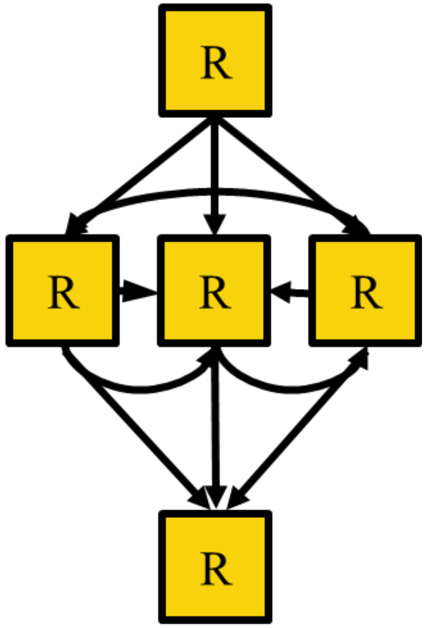
Shared file system or database

rredis/doRedis, batchtools, clustermq



Shared memory

parallel/doParallel



Message Passing

Rmpi/doMPI

Message Passing with doMPI

- To execute a doMPI script on multiple compute nodes a “message passing environment” needs to be set up, i.e. the R interpreter needs to be executed using a command such as `mpirun` (i.e. `mpirun R –f script.R`)

- Then, the already familiar „do-back end“-pattern is put to use within R:

```
library(foreach)
library(doMPI)
cluster.object <- startMPIcluster()
registerDoMPI(cluster.object)
foreach(i = 1:20) %dopar% sum(sort(runif(1e7)))
closeCluster(cluster.object)
```

More on foreach()

- clustermq::register_dopar_cmq(...) registers clustermq as foreach parallel handler

- use times() for simple repetitions:
```
times(10) %do% sum(sort(runif(1e7)))
```
- foreach is a function with several arguments…
```
foreach(i = 1:10, .combine = c, …) %do% sth()  #  process results
    as they get generated, e.g. c(), cbind(), list(), sum(), ...
```
- … evaluates iterators…
```
foreach(i = iter(input)) %do% sth()  # see package iterators
foreach(i = irnorm(100)) %do% sth()
```
- … and provides additional operators:
```
foreach(i = 1:10) %:% when(cond) %do%  sth()  # nesting operator
    and condition cf. Python’s list comprehensions
```

More parallel

- parallel provides parallel replacements of lapply and related functions (as have snow and multicore):
  - multicore-like: e.g. mclapply(1:10, function(x) sum(sort(runif(1e7)))),
    mcmapply (x, FUN, ...), mcMap(FUN, …)
  - snow-like: clusterApply(cl, x, fun, ...), e.g. parLapply(cl, x, FUN, ...)

https://github.com/HenrikBengtsson/parallelly

parallelly: Enhancing the parallel package

- The parallelly package provides functions that enhance the parallel package.
- Showcase: parallelly::availableCores() gives the number of CPU cores available to your R process as given by R options and environment variables, including those set by job schedulers on high-performance compute (HPC) clusters. If R runs under 'cgroups' or a Linux container, then their settings are acknowledges too – vs. parallel::detectCores()

```
$ salloc -p cm2_inter -c 2
$ R
> library(parallel)
> detectCores()
[1] 56
> library(parallelly)
> availableCores()
Slurm     2
```

- The functions and features of parallely are written to be backward compatible with the parallel package, such that they may be incorporated there later.

Even More parallel: Futures/Promises

- Constructs for synchronizing program execution. Describe objects that act as proxies for a result, which is yet unknown (because the computation is incomplete)
- Send command to background and return handle:
  handle <- mcparallel(some_expensive_function)
- Collect result at later point:
  result <- mccollect(handle)

Futures/Promises

```
> system.time(sum(sort(runif(1e7))))
       user  system elapsed
      1.581   0.112   1.700

> system.time(sapply(1:20, function(x) sum(sort(runif(1e7)))))
        user  system elapsed
      28.875   2.998  31.883

> library(parallel)
> h <- mcparallel(sapply(1:20, function(x) sum(sort(runif(1e7)))))
> mccollect(h, wait = FALSE)
NULL
# wait approx. 30 seconds for job to finish
> mccollect(h, wait = FALSE)
[1] 5000214 4999121 5001166 …
```

https://github.com/HenrikBengtsson/future

Futures/Promises

- Package future tries to unify the previous approaches:
  “The purpose of this package is to provide a lightweight and unified Future API for sequential and parallel processing of R expressions via futures. […] Because of its unified API, there is no need to modify any code in order switch from sequential on the local machine to, say, distributed processing on a remote compute cluster. ”

- Implicit:

```
v %<-% { expr }  # future assignment , creates a future and a
    promise to its value (instead of regular assignment <-)
```

- Explicit:

```
f <- future({ expr })  # creates a future
v <- value(f)  # gets the value of the future
    (blocks if not yet resolved)
```

https://github.com/HenrikBengtsson/future

Futures/Promises

- Function plan() allows the user to plan the future, i.e. it specifies how futures are resolved

- For example: plan(sequential) and more

```
> library("future")
> plan(sequential)
> v %<-% {
+   cat("Hello world!\n")
+   3.14
+ }
> v
Hello world!
[1] 3.14
```

https://github.com/HenrikBengtsson/future

Futures/Promises

Name	OSes	Description
synchronous:		*non-parallel:*
sequential	all	sequentially and in the current R process
asynchronous:		*parallel:*
multisession	all	background R sessions (on current machine)
multicore	not Windows	forked R processes (on current machine)
cluster	all	external R sessions on current, local, and/or remote machines

- The future.callr package provides future backends that evaluates futures in a background R process utilizing the callr package - they work similarly to multisession futures but have a few advantages (e.g. more than 125 parallel R processes)
- Package future.batchtools provides an implementation of the Future API on top of the batchtools package, i.e. it allows to process futures (as defined by the future package) on HPC infrastructure

https://github.com/HenrikBengtsson/doFuture

doFuture: future and foreach

- Package doFuture provides a %dopar% adapter for the foreach package such that any type of future (that is supported by the Future API of the future package) can be used for asynchronous (parallel/distributed) or synchronous (sequential) processing.
- Example:

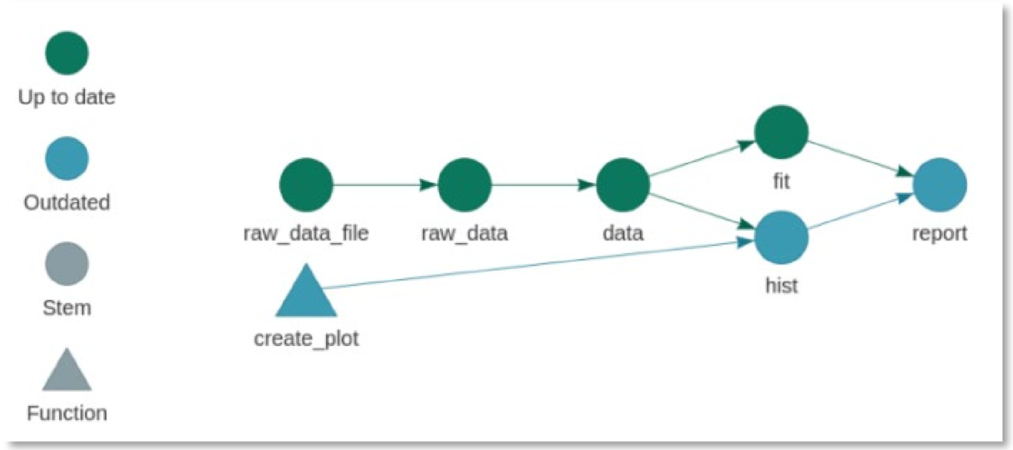
```
library(doFuture)
registerDoFuture()
plan(multicore)
foreach(i = 1:20) %dopar% sum(sort(runif(1e7)))
```

- Look out for the use of foreach and the possibility to register all these different back ends in other R packages, e.g. plyr uses foreach as parallel backend and BiocParallel supports any %dopar% adapter as well!

https://docs.ropensci.org/targets/

… and beyond: the targets package

- The targets package is a Make-like pipeline toolkit for statistics and data science in R
  - maintain a reproducible workflow without repeating yourself
  - skip costly runtime for tasks that are already up to date
  - run the necessary computation with implicit parallel computing
  - abstract files as R objects
- A fully up-to-date targets pipeline is tangible evidence that the output aligns with the code and data, which substantiates trust in the results



https://docs.ropensci.org/targets/

The targets package

- targets supports high-performance computing with the tar_make_clustermq() and tar_make_future() functions (using the future.batchtools backend)
- These functions are like tar_make(), but they allow multiple targets to run simultaneously over parallel workers. Again, these workers can be processes on your local machine, or they can be jobs on a computing cluster.
- tar_make_clustermq() uses persistent workers. That means all the parallel processes launch together as soon as there is a target to build, and all the processes keep running until the pipeline winds down.
- tar_make_future() runs transient workers. That means each target gets its own worker which initializes when the target begins and terminates when the target ends.

Conclusion

- Parallel programming is here to stay (for the foreseeable future).
- Know your hardware…
- … and the possibilities of your software/programming environment.
- Applying proper (high level) abstractions (foreach, futures, targets…) to fully utilize the features of modern CPUs/GPUs and supercomputing infrastructure will allow you to write fast and scalable programs.