Using R in Triton

Aalto University
School of Science

R in Triton: General Information

- R is a very popular language for statistics, bioinformatics etc.
- R has a large collection of libraries provided by R CRAN
  See: https://cran.r-project.org/
- In Triton R is provided through Modules environment
- R is a vectorized language and it utilizes available linear algebra libraries (OpenBLAS,IntelMKL etc.). All versions of R in Triton are compiled against these libraries.
- All modules have a large set of libraries pre-installed by admins. We can add libraries to the installations or you can install them yourself (more on that later).

R in Triton: Running R programs from scripts

- On desktops most users use R through IDE like Rstudio
- In Triton due to the queue system one needs to run R programs from scripts
- Easiest way to do this is to use the Rscript command:

```
Rscript script.R
```

- Another option is to use R CMD BATCH, but it has few caveats

R in Triton: Running R programs from scripts

By default Rscript does few things differently to R CMD BATCH:

- It does not save nor restore a R environment
(--no-restore and –no-save for R CMD BATCH)
- Output is produced to stdout instead of script.Rout
(you got slurm output anyways)
- R startup jargon is skipped
(--slave for R CMD BATCH)
- It speeds up startup by skipping the load of some default packages. You can set the packages to be loaded through:

```
Rscript --default-packages=methods,utils,stats script.R
```

R in Triton: Example R program

- Example available in https://github.com/AaltoScienceIT/triton-examples
- Single-CPU R example:

```
#!/bin/bash
#SBATCH -p short
#SBATCH -t 00:20:00
#SBATCH --ntasks=1
#SBATCH --mem=3G
#SBATCH -o serialR.out

module load R

echo 'Running a simple serial R example:'

srun Rscript serialR.R
```

R in Triton: Example R program

- serialR.R

```
# Run simple cross-validation method with caret and knn
# https://github.com/tobigithub/caret-machine-learning
# Tobias Kind (2015)

# Single example, no cross-validation
  require(caret); data(BloodBrain); set.seed(123);
  fit1 <- train(bbbDescr, logBBB, "knn"); fit1

# cross-validation example with method boot
  require(caret); data(BloodBrain); set.seed(123);
  tc <- trainControl(method="boot")
  fit1 <- train(bbbDescr, logBBB, trControl=tc, method="knn");  fit1


### END
```

R in Triton: Installing libraries

- Many libraries are already included in the modules and we can add even more
- If you want to install your own, there are few caveats

1. R by default uses ~/R/R.version$platform-library/x.y
   This is a problem as /home is a small and slow NFS.
   Easiest solution is to create a folder Rlibs to $WRKDIR and write
   R_LIBS=/path/to/work/dir/Rlibs
   to .Renviron
2. Libraries installed with one version of R do not necessarily work with other version of R
   → Keep the same module version! (save collection)

R in Triton: Parallel R

Trivially parallel:

- You can access SLURM_ARRAY_TASK_ID environment variable from R with System.getenv(“SLURM_ARRAY_TASK_ID”)
- Example:

```
myfunc <- function(x) {
        message(x)
}

myfunc(System.getenv(“SLURM_ARRAY_TASK_ID”))
```

Aalto University
School of Science

R in Triton: Parallel R

Different R packages allow for multiprocessor action for independent tasks:

- parallel-package has mclapply and cluster constructs with parLapply for parallel *apply
- foreach and doParallel can parallelize for-loops
- Rmpi,snow,snowfall can be used for parallelism across nodes

For dependent tasks:

- Rcpp can be used to create C code that can use OpenMP

R in Triton: Parallel R example

- Script is quite similar:

```
#!/bin/bash
#SBATCH -p short
#SBATCH -t 00:20:00
#SBATCH --ntasks=1
#SBATCH --mem=3G
#SBATCH -o serialR.out


module load R

echo 'Running a simple serial R example:'

srun Rscript serialR.R
```

```
#!/bin/bash
#SBATCH -p short
#SBATCH -t 00:20:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --mem=8G
#SBATCH -o parallelR.out

module load R

echo 'Running parallel R example:'

srun Rscript parallelR.R
```

R in Triton: Parallel R example

- R can get the number of CPUs from the environment:

```
cores <- as.integer(Sys.getenv("SLURM_CPUS_PER_TASK"))
```

- This is then used when calling mclapply, parLapply etc. to define the number of workers used

Any questions?