BACHTOOLS

Janek Thomas[1], **Bernd Bischl**[1], **Michel Lang**[2]
[1]Computational Statistics, LMU
[2]TU Dortmund

Section 1

Introduction and General Batch Computing

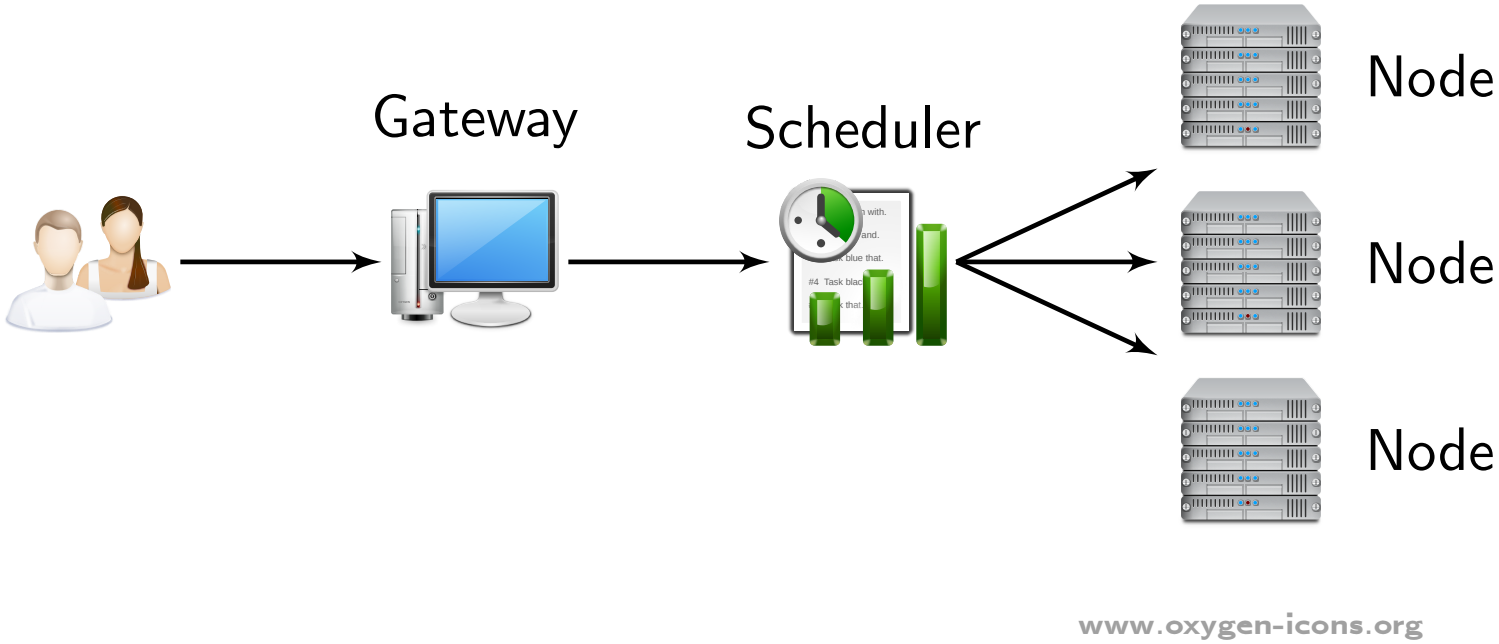
Naive batch computing I

Computing on multicore machines (non-cluster)

- Prepare standalone script(s) that run your jobs, save results at end
- Parameters must be hard coded or retrieved through commandline
- Login on a machine per SSH
- Start job(s) with `R CMD BATCH myscript1.R`, combine this with nohup, screen or tmux
- Start remaining jobs when resources get available (argh...)
- Check manually for completion / errors (argh again...)
- Write script to collect results

No automation, no resource management or fair share,
neither extensible nor scalable.
— Don't do it this way —

High Performance Computing (HPC) clusters I



- User log onto the gateway server (master or head node)
- Network of multiple computing nodes, managed by the scheduler
- Scheduler orchestrates the computation and organizes queues to fairly distribute compuation times among users
- Nodes either share a file system or a some form of file staging

Manual working on a batch system I

- You have to specify
  (A) Resource specifications (number CPUs, number of tasks, expected runtime and memory)
  (B) Which cluster use, e.g., serial, mpp2 or hugemem.
  (C) Command to execute (e.g. `R CMD BATCH <myscript.R>`)
- Specs passed to CLI tools either directly as arguments or encoded in a shell script
- Check status of jobs via CLI tools (e.g. `squeue`)
- Write script to collect results

Usual workflow on a batch system I

- **Unroll** your `R` **loop(s)** so that your script computes a single iteration
- Write a **script that writes** `R` **scripts** for each iteration setting the iteration counter(s) at the beginning
- Write a **script that writes job description files** for each `R` script
- Write a **script that submits** your job description files
- **Crawl** through file system checking for existence of results or log files
- Write a **script that combines** your scattered result files

- Found a bug in your code? Write a **script that kills** all running jobs, fix the bug, submit everything again
- Some jobs have hit the wall time? Write a **script that finds** out which jobs you need to resubmit with weaker constraints
- Want to try your model on another data set or using other parameters? Eventually **start from scratch**, it might get ugly

Conclusions and further remarks I

+ They are pretty fast!
+ Many statistical tasks are embarrassingly parallel
- Job description files needed
- We cannot control when jobs are started.
- Jobs cannot really communicate, except by writing stuff on disk (or we have to allocate multiple cores and use something like MPI)
- Requesting many nodes at once increases time spend in queue
- Auxiliary scripts to create files and submit jobs necessary
- Functions to collect results can get complicated and lengthy
- If some jobs fail (e.g, singularities), debugging is awful

Section 2

The batchtools Package

Packages I

batchtools

- Basic infrastructure to communicate with a high performance cluster
- Tailored around Map-Reduce paradigm
- Can be incorporated into other packages
- Supported via `parallelMap` and `BiocParallel`
- Additional abstraction for "applying algorithms on problems"
- Assists the user in conducting comprehensive computer experiments
- Successor package (and combination) of `BatchJobs` and `BatchExperiments`.

batchtools – Features I

- Basic infrastructure to communicate with batch systems from within `R`
- Complete control over the batch system from within `R`: submit, supervise, kill
- Persistent state of computation for experiments
- `R` code independent from the underlying batch system
- Reproducibility in distributed environments, even if the architecture changes
- Convenient result collection capabilities
- Debugging tools

Supported Systems I

Real batch systems:

- Torque/PBS based systems
- Sun Grid Engine / Oracle Grid Engine
- Load Sharing Facility (LSF)
- SLURM
- DockerSwarm

Other modes:

- Interactive: Jobs executed in current interactive `R` session
- Multicore: local multicore execution with spawned processes
- SSH: distributed computing on loosely connected machines which are accessible via SSH (makeshift cluster)

Links and references I

- `https://github.com/mllg/batchtools`
  - Installation infos
  - R documentation
  - Vignettes
  - Issue tracker
  - Recent development version in git
- Paper:
  M Lang, B Bischl, D Surmann - The Journal of Open Source Software, 2017:
  "batchtools: Tools for R to work on batch systems"
  Available on project page

(1) Create a registry I

- Object used to access and exchange informations: file paths, job parameters, computational events, ...
- All information is stored in a single, portable directory
- Initialization of a new registry:

```
> library(batchtools)
> reg = makeRegistry(
+ file.dir = "~/project",          # accessible on all nodes
+ seed = 1                         # initial seed for first job
+ )
```

- `loadRegistry(dir)` to resume working with an existing registry

(2) Define Jobs I

batchMap

- Like `lapply` or `mapply`
- $(x_1, x_2) \times (y_1, y_2) \to (f(x_1, y_1), f(x_2, y_2))$
- 10 Jobs to calculate $1 + 9$, $2 + 8$, $\ldots$, $9 + 1$

```
> map = function(i, j)  i +j
> ids = batchMap(fun = map, i = 1:9, j = 9:1, reg = reg)
```

- Stores function on file system
- Creates jobs as rows in a `data.table`
- Parameters also serialized into the `data.table` for fast access
- All jobs get unique positive integers as IDs
- `reg =` can be omitted in most cases. See `getDefaultRegistry`.

(3) Subset Jobs I

- Query job IDs by computational status: `find*` functions `findSubmitted`, `findRunning`, `findDone`, ...
- Query job IDs by parameters: `findJobs(pars)`

```
> findJobs(j==1)
> findNotSubmitted()
> findDone()
```

- Set operations on ID `data.tables`: `merge`
- `data.table` of IDs can be passed to basically all functions interacting with the batch system

(4) Submit Jobs I

submitJobs

- Creates R script files and job description files on the fly
- Resources can be provided as named list

```
> # 1 hour maximal execution time, about 2 GB of RAM
> res = list(walltime = 60*60, memory = 2000)
>
> # ... and submit
> submitJobs(resources = res)
```

- Submits all jobs per default
- Subsets of jobs can be providing as `data.table` or `vector`

```
>   submitJobs(ids = 1:5, ressources = res)
```

(5) Supervise and debug I

- Quick overview of what is going on: `getStatus()`

```
Status for jobs:  10
Submitted:        10 (100.0%)
Started:          10 (100.0%)
Errors:            0 (  0.0%)
Running:           2 ( 20.0%)
Expired:           0 (  0.0%)
Done:              8 ( 80.0%)
Time: min=1.50s avg=5.20s max=8.80s
```

- Display log files with a customizable pager (less, vi, ...): `showLog(findErrors()[1])`
- You can also `grepLogs(pattern)`
- Found a bug? `killJobs(findRunning())`
- Run a job in the current `R` session: `testJob(id)`

(6) Collect results I

Reduce

```
> # combine in numeric vector
> reduceResults(ids = findDone(), init = numeric(0),
+               fun = function(aggr, job, res) c(aggr, res))
```

- Convenience wrappers around `reduceResults`: `reduceResults[DataTable|List]`
- Simple loading

```
> loadResult(id = 1)
```

Configuration again I

`.batchtools.conf.R`

```
> cluster.functions = makeClusterFunctionsSlurm("~/slurm_lmulrz.tmpl", array.jobs = FALSE)
> default.resources = list(walltime = 300L, memory = 512L, ntasks = 1L, ncpus = 1L,
+   nodes = 1L, clusters = "serial")
>
> max.concurrent.jobs = 1000L
```

Demo of basic usage I

```
> unlink("~/project", recursive = TRUE)
>
> library(batchtools)
> reg = makeRegistry(
+   file.dir = "~/project",           # path to store everything
+   seed = 1                          # initial seed for first job
+ )
>
> f = function(i, j) i+j
> ids = batchMap(fun = f, i = 1:9, j = 9:1)
>
> findJobs(expr = j == 1)
> findNotSubmitted()
> findDone()
```

Demo of basic usage II

```
> resources = list(walltime = 5*60, memory = 512)
> submitJobs(resources = resources)
>
> getStatus()
> waitForJobs()
> getStatus()
>
> # a peek into the database
> getJobTable(1:2)
>
> # get results
> loadResult(1)
> reduceResultsList()
> getJobTable()[reduceResultsDataTable()]
```

Section 3

Abstractions for computer experiments

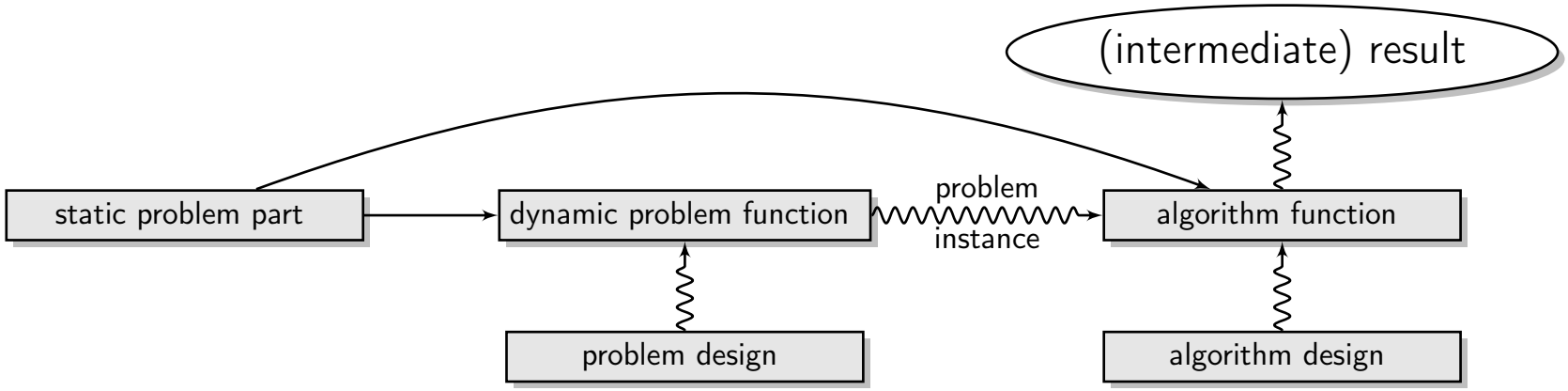
Experiments in batchtools I

- Intended as abstraction for typical statistical tasks:

  Applying algorithms on problems

- More aimed at the end user
- Convenient for simulation studies, comparison and benchmark experiments, sensitivity analysis, ...
- Workflow differs only in job definition
- Scenarios:
  - Compare machine learning algorithms on many data sets
  - Compare one/many estimation procedure(s) on simulated data
  - Compare optimizers on objective functions
  - ...

Abstraction of Computer Experiments I



- Problem definition split into static and dynamic part
  - Static: immutable R objects: matrix, data frames, ...
  - Dynamic: Arbitrary R function: transformations of static part, extraction of data from external sources, ...
- Parametrization through specifying experimental designs for both problems and algorithms
- Each step automatically seeded, random seeds stored in a database

Experiment definition steps I

1. Add problems to registry: `addProblem`
   - Efficient storage: Separation of static (`data`) and dynamic (`instance`) problem parts.
2. Add algorithms to registry: `addAlgorithm`
   - Problem instance gets passed to algorithm
   - Can be connected with an experimental design (function parameters)
   - Return value will be saved on the file system
3. Add experiments to registry: `addExperiments`
   - Experiment: problem instance + algorithm + algorithm parameters
   - Job: Experiment + replication number

Section 4

Wrap up and outlook

What you get I

- Reproducibility: Every computation is seeded, seeds are stored in a `data.table`
- Portability: Data, algorithms, results and job information reside in a single directory
- Extensibility: Add more problems or algorithms, try different parameters or increase the replication numbers at any computational state
- Exchangeability: Share your file directory to allow others to extend your study with their data sets and algorithms

- Greatly simplifies the work with batch systems
- Interactively control batch systems from within R (no shell required)
- Do reproducible research
- Exchange code and results with others

`github.com/mllg/batchtools`

Demo with mlr and OpenML