Parallel programming with R and Azure Batch

Embarrasingly Parallel Problems

Also called **perfectly parallel**, these are the most simple cases for parallel computing:

- Similar tasks are repeated many times
  - Iterations in a loop construct, chunks of data
- Calculations are independent of each other
- Little or no manipulation needed to create parallel tasks
- Examples:
  - Independent Monte Carlo simulations
  - Analysis by groups of data

Example: Monte Carlo Simulation, random walk of an asset price

```
getClosingPrice <- function(opening_price=100, mean_change=1.001, volatility=0.01, days=1825) {
    movement <- rnorm(days, mean = mean_change, sd = volatility)
    path <- cumprod(c(opening_price, movement))
    closingPrice <- path[days]
    return(closingPrice)
}
```

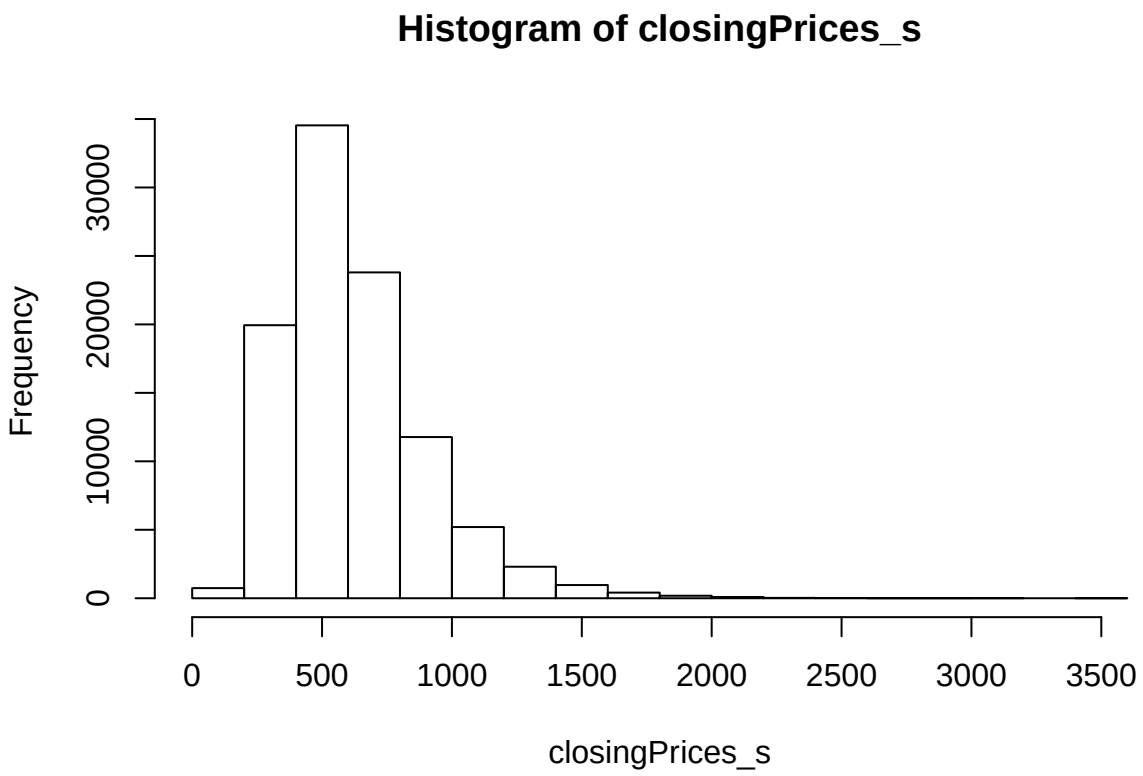
To run multiple independent simulations, we can use one of the many looping constructs that come with R, like `for`, `while` or the `apply` family.

We use `replicate` to run repeatedly in a single thread:

```
n <- 100000
start_s <- Sys.time()
closingPrices_s <- replicate(n, getClosingPrice())
end_s <- Sys.time()
difftime(end_s, start_s)
```

```
## Time difference of 16.21148 secs
```

```
hist(closingPrices_s)
```



The package `foreach`

The foreach looping construct

The package foreach introduces a looping construct that supports parallel execution

The `foreach` loop looks similar to a `for` loop, but is implemented using a binary operator, called `%do%`. Unlike `for`, it returns a value

```
library(foreach)
foreach(i=1:5) %do% sprintf("Hello from iteration %s", i)
```

```
## [[1]]
## [1] "Hello from iteration 1"
##
## [[2]]
## [1] "Hello from iteration 2"
##
## [[3]]
## [1] "Hello from iteration 3"
##
## [[4]]
## [1] "Hello from iteration 4"
##
## [[5]]
## [1] "Hello from iteration 5"
```

The loop `foreach` with the operator `%do%` runs locally and single threaded, pretty much like `for`. This is essentially used for intermediate local tests.

`foreach` comes with another operator, `%dopar%` that runs iterations in parallel. In order to do so, we need to register a parallel backend

Registering a parallel backend

There are multiple packages that implement functionality to create and register parallel backend clusters:

- `foreach::registerDoSEQ` explicitly register the default sequential backend
- `doParallel::registerDoParallel` local cluster via library(parallel)
- `doFuture::registerDoFuture` HPC with schedulers
- `future::makeClusterMPI` Message Passing Interface (MPI) cluster
- `doAzureParallel::registerDoAzureParallel`

Example: local cluster with `doParallel`:

```
library(doParallel)
```

```
## Loading required package: iterators
```

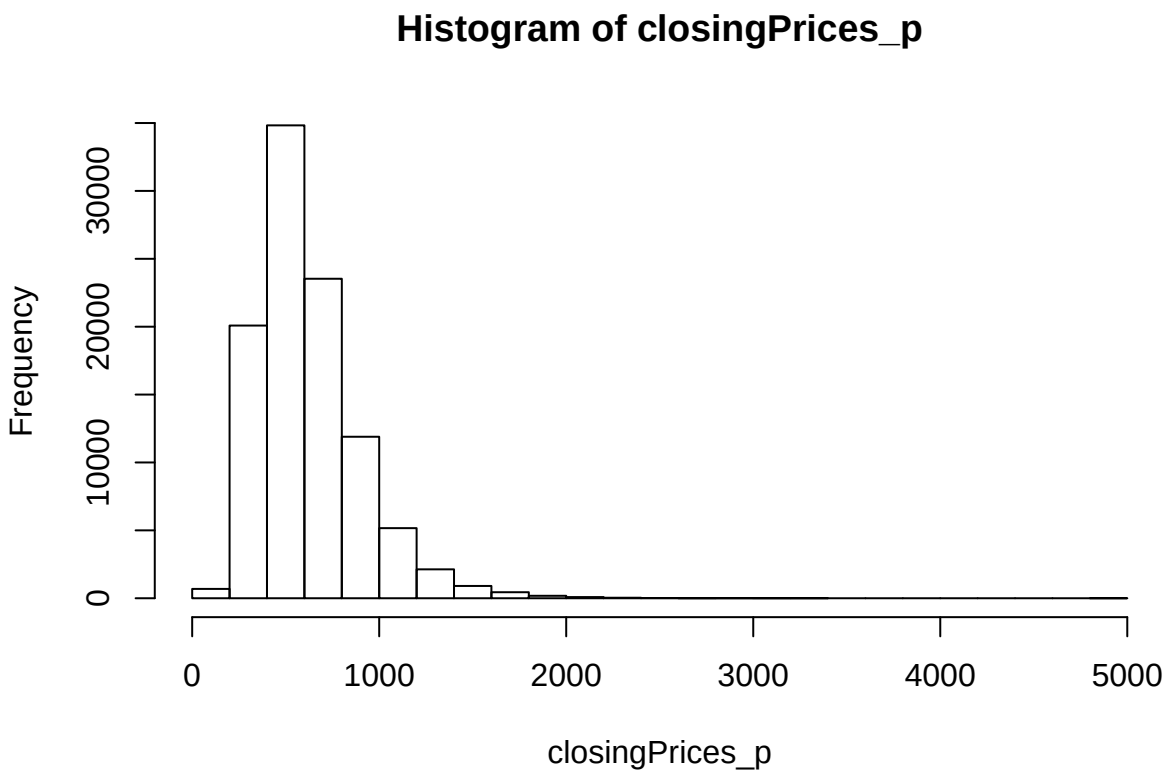
```
## Loading required package: parallel
```

```
local_cluster <- parallel::makeCluster(4)
registerDoParallel(local_cluster)

n <- 100000
iterations <- 4
start_p <- Sys.time()
closingPrices_p <- foreach(i = seq(iterations), .combine = "c") %dopar% {
    replicate(n/iterations, getClosingPrice())
}
end_p <- Sys.time()
difftime(end_p, start_p)
```

```
## Time difference of 6.34193 secs
```

```
hist(closingPrices_p)
```



Combining results

`foreach` returns a combination of the result of each iteration. By default, the results are combined into a list as long as the number of iterations. This behaviour can be changed using the parameter `.combine`

In the previous example we specify `.combine = "c"`, hence the results are combined into a vector.

We can as well combine the into columns or rows of a matrix:

```
foreach(i = 1:4, .combine = "cbind") %dopar% rnorm(5)
```

```
##        result.1    result.2   result.3   result.4
## [1,]  0.4957689 -1.86612092 -0.2213148 -1.1856193
## [2,] -1.8866283 -1.44088053 -0.1643084  0.2563306
## [3,]  1.0922320  0.01895137 -0.1697880 -0.7931275
## [4,] -0.8648588 -0.12153623  1.4780140  0.5342533
## [5,] -0.4580880 -0.52581216 -0.6025481  0.1228652
```

```
foreach(i = 1:4, .combine = "rbind") %dopar% rnorm(5)
```

```
##                 [,1]       [,2]      [,3]        [,4]       [,5]
## result.1  0.06691699 0.06710229 0.4917640 -2.68496090  0.4085436
## result.2 -0.51153087 0.11154627 0.3620306 -0.53325143 -0.5376506
## result.3  0.86582640 2.01563657 1.7487483  1.58379611  0.1682133
## result.4  0.64751964 0.95840705 0.3102912  0.03288353  1.3146414
```

Azure setup

We will now use the package `doAzureParallel` to interact with an Azure subscription.

You will need to have a valid Azure subscription with access to the resources we will use in this example: an Azure Batch account and a storage account.

First thing is to install the package and its dependency `rAzureBatch`

```
devtools::install_github("Azure/rAzureBatch", ref = "v0.7.0")
```

```
## Skipping install of 'rAzureBatch' from a github remote, the SHA1 (e05403ad) has not changed since las
##   Use `force = TRUE` to force installation
```

```
devtools::install_github("Azure/doAzureParallel", ref = "v0.8.0")
```

```
## Skipping install of 'doAzureParallel' from a github remote, the SHA1 (9e9b4942) has not changed since
##   Use `force = TRUE` to force installation
```

```
library(doAzureParallel)
```

```
##
## Attaching package: 'doAzureParallel'
```

```
## The following objects are masked from 'package:parallel':
##
##     makeCluster, stopCluster
```

Azure credentials

Generate a credentials configuration file (template)

```
generateCredentialsConfig("credentials.json")
```

Edit the file with your own Azure credentials:

- Batch account name
- Batch account key: Primary access key
- Batch account URL
- Storage account name
- Storage account key

You find the above in the *Keys* section of your Batch and Storage accounts in the Azure portal

Next, set your credentials in the current R session.

```
setCredentials("credentials.json")
```

```
## ============================================================================================
## Batch Account: miraibatch
## Batch Account Url: https://miraibatch.northeurope.batch.azure.com
## Storage Account: razureinterfaces
## Storage Account Url: https://razureinterfaces.blob.core.windows.net
## ============================================================================================
## Your credentials have been successfully set.
```

If successful, you can see a new option with information of your Azure account.

```
getOption("az_config")
```

Azure Batch pool

Generate a cluster configuration file (template)

```
generateClusterConfig("cluster.json")
```

Edit the cluster configuration file with your desired configuration.

Create your cluster if it does not exist; this takes a few minutes. Alternatively get your cluster configuration from the Batch account

```
# Get configuration from Azure
cluster <- getCluster("raipool")
```

```
##
## nodes:
##   idle:                5
##   creating:            0
##   starting:            0
##   waitingforstarttask: 0
##   starttaskfailed:     0
##   preempted:           0
##   running:             0
##   other:               0
## Your cluster has been registered.
## Dedicated Node Count: 5
## Low Priority Node Count: 0
```

```
# Create new cluster from configuration file
cluster <- makeCluster("cluster.json")
```

```
## ========================================================================================================
## Name: raipool
## Configuration:
##   Docker Image: rocker/tidyverse:latest
##   MaxTasksPerNode: 2
##   Node Size: Standard_D2_v2
## Scale:
##   Autoscale Formula: QUEUE
##   Dedicated:
##       Min: 5
##       Max: 5
##   Low Priority:
##       Min: 0
##       Max: 0
## ========================================================================================================
```

```
## Warning in self$client$extractAzureResponse(response, content): Conflict (HTTP 409).
```

```
## The specified cluster 'raipool' already exists. Cluster 'raipool' will be used.
## Your cluster has been registered.
## Dedicated Node Count: 5
## Low Priority Node Count: 0
```

Beware of open issue #330 in package `doAzureParallel`. Registering a cluster that has been obtained with `getCluster` results in a not obvious misconfiguration of the cluster. In current version, this issue can be worked around by creating the cluster (again) before registering. That is, running the chunk above. The cluster will in fact not be re-created, but the metadata will be fixed.

Finally, register your Azure cluster as the parallel backend for `foreach`

```
registerDoAzureParallel(cluster)
```

Check that you the cluster is available

```
# Number of execution workers currently registered in the doPar backend
getDoParWorkers()
```

```
## [1] 10
```

```
# Hello world
foreach(i=1:5) %dopar% sprintf("Hello World from node %s", i)
```

```
## ===========================================================================================
## Id: job20200916132742
## chunkSize: 1
## enableCloudCombine: TRUE
## errorHandling: stop
## wait: TRUE
## autoDeleteJob: TRUE
## ===========================================================================================
## Submitting tasks (1/5)Submitting tasks (2/5)Submitting tasks (3/5)Submitting tasks (4/5)Submitting ta
## Submitting merge task. . .
## Waiting for tasks to complete. . .
## | Progress: 0.00% (0/5) | Running: 0 | Queued: 0 | Completed: 0 | Failed: 0 || Progress: 100.00% (5/5
## Tasks have completed. Merging results.. Completed.
```

```
## $`1`
## [1] "Hello World from node 1"
##
## $`2`
## [1] "Hello World from node 2"
##
## $`3`
## [1] "Hello World from node 3"
##
## $`4`
## [1] "Hello World from node 4"
##
## $`5`
## [1] "Hello World from node 5"
```

Parallel computing on Azure Batch

Parallel Random Forest

Let's take random forest as an example of an operation that can take a while to execute. Let's say our inputs are the matrix x, and the factor y:

```
x <- matrix(runif(500), 100)
y <- gl(2, 50)
```

Lets create a random forest model with 1000 trees. We will plit up the problem into 5 pieces, with the `ntree` argument set to 200. The package `randomForest` comes with a function called `combine` that combines the resulting `randomForest` objects.

We first run it locally.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
rf <- foreach(ntree = rep(10000, 5), .combine = combine) %do%
    randomForest(x, y, ntree = ntree)
rf
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = ntree)
##                Type of random forest: classification
##                      Number of trees: 50000
## No. of variables tried at each split: 2
```

Now lets run the same on the registered parallel backend. What about the package `randomForest`? Is it in the Azure Batch pool?

The following block will throw an error

```
rf <- foreach(ntree = rep(10000, 5), .combine = combine, .errorhandling = "pass") %dopar%
    randomForest(x, y, ntree = ntree)
```

```
## ===========================================================================================
## Id: job20200916132812
## chunkSize: 1
## enableCloudCombine: TRUE
## errorHandling: pass
## wait: TRUE
## autoDeleteJob: TRUE
## ===========================================================================================
## Submitting tasks (1/5)Submitting tasks (2/5)Submitting tasks (3/5)Submitting tasks (4/5)Submitting ta
## Submitting merge task. . .
## Waiting for tasks to complete. . .
## | Progress: 0.00% (0/5) | Running: 0 | Queued: 0 | Completed: 0 | Failed: 0 || Progress: 100.00% (5/5
## Tasks have completed. Merging results... Completed.
## error calling combine function:
## <simpleError in fun(result.1, result.2): Argument must be a list of randomForest objects>
```

```
rf
```

```
## NULL
```

Diagnostics

`doAzureParallel` offers minimum functionality for diagnostics

```
setVerbose(TRUE)
setHttpTraffic(TRUE)
rf <- foreach(ntree = rep(200, 5), .combine = combine) %dopar%
    randomForest(x, y, ntree = ntree)
```

Some metadata can be fetched from Azure, but for details of the error we would have to The Azure Portal or Azure Batch explorer.

```
doAzureParallel::getJob("job20200915212934")
```

The doAzureParallel troubleshooting documentation in github can be helpful. One can also enable verbose logging

Monitoring and Managing the Azure environment

`doAzureParallel` offers very limited functionality for monitoring and managing the Azure environment. The Azure Portal or specific tools like Azure Batch Explorer and Azure Storate Explorer (or custom tools using Azure APIs) should be used together for monitoring and management.

Special remark should be made about the status of the Azure Batch cluster. This cannot be changed from the R session with `doAzureParallel`. The cluster nodes are created in status "Enable" (Idle), and thus they are a running cost until they are disabled, or deleted. It is possible to delete a cluster with `doAzureParallel`

R runtime environment in Azure Batch

Installing packages for your parallel runs

Our last call to Azure Batch failed because the package `randomForest` was being used, but was not available on the R runtime environment on Azure Batch. It can be installed on the fly using the `foreach` parameter `.packages`

```
rf <- foreach(ntree = rep(10000, 10), .combine = combine, .packages = "randomForest") %dopar%
    randomForest(x, y, ntree = ntree)
```

```
## ===========================================================================================
## Id: job20200916132839
## chunkSize: 1
## enableCloudCombine: TRUE
## packages:
##  randomForest;
## errorHandling: stop
## wait: TRUE
## autoDeleteJob: TRUE
## ===========================================================================================
## Submitting tasks (1/10)Submitting tasks (2/10)Submitting tasks (3/10)Submitting tasks (4/10)Submitti
## Submitting merge task. . .
## Job Preparation Status: Package(s) being installed.
## Waiting for tasks to complete. . .
## | Progress: 0.00% (0/10) | Running: 10 | Queued: 0 | Completed: 0 | Failed: 0 || Progress: 100.00% (
## Tasks have completed. Merging results..... Completed.
```

```
rf
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = ntree)
##                Type of random forest: classification
##                      Number of trees: 1e+05
## No. of variables tried at each split: 2
```

Note that every time we execute the above chunk, a new installation of the package `randomForest` is triggered.

Runtime environment of the `doAzureParallel` cluster

The R code that we ship to run on the Azure Batch cluster runs on a docker container. This ensures a stable runtime environment: all jobs running on the same node run on a new container. The jobs are independent from each other, thus the past jobs history does not affect new jobs, even if previous jobs did things like installing new packages.

A docker image is specified at the time of creating the cluster, in the file `cluster.json`. The default image is "rocker/tidyverse:latest".

Customizing the docker image for the R runtime

Installing packages at runtime, like in the last example, is not a convenient practice. Recurrent packages should rather be added to the docker image used for all containers. `doAzureParallel` makes this easy by allowing specification of packages in `cluster.json`. Various sources are allowed: cran, github, bioconductor.

It is also possible to specify a custom container image altogether.

Accessing the Azure Storage

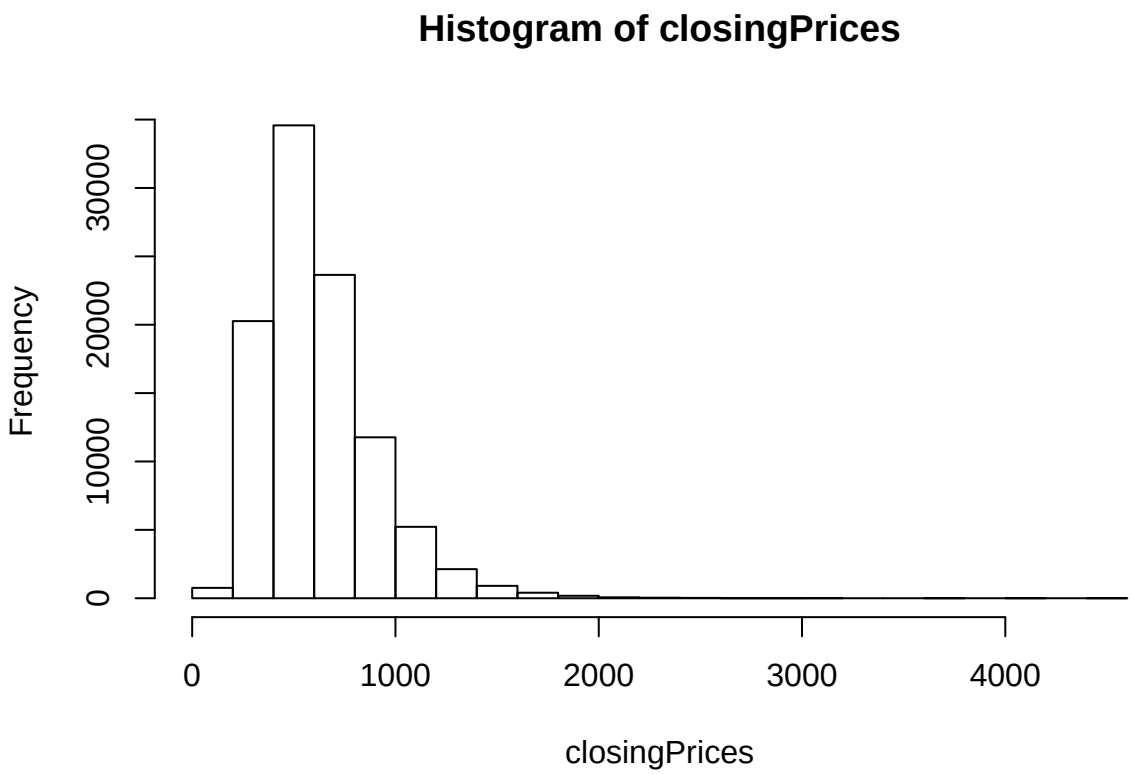
Fetching files from the cluster

Lets modify the closing prices example and write on disk results of each iteration

```
n <- 100000
iterations <- 10
closingPrices <- foreach(i = seq(iterations), .combine = "c") %dopar% {
    results <- replicate(n/iterations, getClosingPrice())
    results_file <- sprintf("results_iteration_%s.csv", i)
    write.csv(results, results_file, row.names = FALSE)
    return(results)
}
```

```
## ===============================================================================================
## Id: job20200916132947
## chunkSize: 1
## enableCloudCombine: TRUE
## errorHandling: stop
## wait: TRUE
## autoDeleteJob: TRUE
## ===============================================================================================
## Submitting tasks (1/10)Submitting tasks (2/10)Submitting tasks (3/10)Submitting tasks (4/10)Submitti
## Submitting merge task. . .
## Waiting for tasks to complete. . .
## | Progress: 0.00% (0/10) | Running: 0 | Queued: 0 | Completed: 0 | Failed: 0 || Progress: 100.00% (1
## Tasks have completed. Merging results.. Completed.
```

```
hist(closingPrices)
```



Now lets download the files generated.

```
getJobFile(jobId = "job20200916105340", taskId = 1, filePath = results_file, downloadPath = "results_fi
```

Mounting Azure File Shares

Distributed computing

References

- Microsoft Azure Batch documentation
- Package foreach. Vignettes
- Package doParallel. Vignettes
- Package doFuture. Vignettes
- Package doAzureFuture. Github repository
- Package rAzureBatch. Github repository