R future

css: bootstrap.min.css width: 1440 height: 900

Non blocking, parallel assignment in R

EdinbR, 21/09/2016

Guillaume Devailly, [@G_Devailly](https://twitter.com/G_Devailly)

Figure 1: The Roslin Institute

R future

Future is an R package by Henrik Bengtsson, available on CRAN.

```
> install.packages("future")
> library(future)
> plan(multiprocess) # more on that later
```

It introduces yet another assignment operator: `%<-%`

%<-% is yet another assignment operator:

```
> a %<-% 42
> a
[1] 42
> b %<-% c(rep(a, 4), 43)
> b
[1] 42 42 42 42 43
```

%<-% is non blocking (1/2):

First, let's create slow functions:

```
> slow <- function(myFunc, by_seconds = 3) {
+     return(
+         function(...) {
+             Sys.sleep(by_seconds)
+             myFunc(...)
+         }
+     )
+ }
>
> slow_rnorm <- slow(rnorm)
> t0 <- Sys.time()
>
> slow_rnorm(4)
[1] -1.361  0.476  0.120  2.372
>
> Sys.time() - t0
Time difference of 3.03 secs
```

%<-% is non blocking (2/2):

Without %<-%:

```
> t0 <- Sys.time()
> a <- slow_rnorm(2)
>
> Sys.time() - t0
Time difference of 3 secs
>
> a
[1]  1.08 -1.09
```

With %<-%:

```
> t0 <- Sys.time()
> b %<-% slow_rnorm(2)
>
> Sys.time() - t0
Time difference of 0.007 secs
>
> b
[1]  0.398 -0.714
```

%<-% is not magic:

It create a *future*, a variable that will be available in the future. The task is **not** magically optimised, it runs in the background. When the variable is needed, the R (main) process will wait until the *future* is resolved.

```
> t0 <- Sys.time()
> b %<-% slow_rnorm(2)
>
> Sys.time() - t0
Time difference of 0.006 secs
>
> b
[1] -1.637  0.853
>
> Sys.time() - t0
Time difference of 3.01 secs
```

futures can be run in parallel:

Standard assignment:

```
> t0 <- Sys.time()
> x1 <- slow_rnorm(2)
> x2 <- slow_rnorm(2)
>
> Sys.time() - t0
Time difference of 6.01 secs
>
> list(x1, x2)
[[1]]
[1]  0.613 -1.303

[[2]]
[1]  1.13 -2.16
>
> Sys.time() - t0
Time difference of 6.01 secs
```

Assignment with future:

```
> t0 <- Sys.time()
> x3 %<-% slow_rnorm(2)
> x4 %<-% slow_rnorm(2)
>
> Sys.time() - t0
Time difference of 0.063 secs
>
> list(x3, x4)
[[1]]
```

```
[1]  0.965 -0.109

[[2]]
[1]  0.602 -0.985
>
> Sys.time() - t0
Time difference of 3.06 secs
```

future allows easy parallelisation of heterogeneous tasks (1/3)

```
> myMat <- matrix(1:6, ncol = 2)
> slow_apply <- slow(apply)
```

Standard assignment:

```
> t0 <- Sys.time()
> myMean  <- slow_apply(myMat, 1, mean)
> mySd    <- slow_apply(myMat, 1, sd)
> myRnorm <- slow_rnorm(3)
> myRunif <- slow(runif)(3)
>
> data.frame(myMean, mySd, myRnorm, myRunif)
  myMean mySd myRnorm myRunif
1    2.5 2.12   -0.69  0.0578
2    3.5 2.12   -1.23  0.2029
3    4.5 2.12   -1.34  0.6304
>
> Sys.time() - t0
Time difference of 12 secs
```

future allows easy parallelisation of heterogenous tasks (2/3)

Parallelization with *future*:

```
> t0 <- Sys.time()
> myMean  %<-% slow_apply(myMat, 1, mean)
> mySd    %<-% slow_apply(myMat, 1, sd)
> myRnorm %<-% slow_rnorm(3)
> myRunif %<-% slow(runif)(3)
>
> data.frame(myMean, mySd, myRnorm, myRunif)
  myMean mySd myRnorm myRunif
1    2.5 2.12   0.096   0.838
2    3.5 2.12  -1.745   0.112
3    4.5 2.12  -1.606   0.530
>
> Sys.time() - t0
Time difference of 6.03 secs
```

future allows easy parallelisation of heterogenous tasks (3/3)

Parallelization with parallel:

```
> library(parallel)
> t0 <- Sys.time()
> myCommands <- c(
+     myMean  = "slow_apply(myMat, 1, mean)",
+     mySd    = "slow_apply(myMat, 1, sd)",
+     myRnorm = "slow_rnorm(3)",
+     myRunif = "slow(runif)(3)"
+ )
>
> as.data.frame(mclapply( # not parallel on windows, but should work elsewhere
+     myCommands,
+     function(x) eval(parse(text = x))
+     # mc.cores = 4L <- add this on non windows OS
+ ))
  myMean mySd myRnorm myRunif
1    2.5 2.12  -0.170   0.798
2    3.5 2.12   0.445   0.267
3    4.5 2.12   0.628   0.445
>
> Sys.time() - t0
Time difference of 12 secs
```

A *future_mclapply* draft function (1/3)

```
> future_mclapply <- function(myList, myFunction) {
+     if(is.vector(myList)) myList <- as.list(myList) # the function will work on vectors to
+     for(i in seq_along(myList)) {
+         command <- paste0(
+             "x",
+             i,
+             " %<-% do.call(",
+             deparse(substitute(myFunction)),
+             ",",
+             deparse(substitute(myList[i])),
+             ")"
+         )
+         eval(parse(text = command))
+     }
+     outputVars <- paste(
+         paste0("x", seq_along(myList)),
+         collapse = ", "
+     )
+     output <- eval(parse(text = paste0("list(", outputVars, ")")))
+     names(output) <- names(myList)
+     return(output)
+ }
```

A *future_mclapply* draft function (2/3)

```
> t0 <- Sys.time()
>
> future_mclapply(
+     list(1:5, 6:10, pi),
+     slow(mean)
+ )
[[1]]
[1] 3

[[2]]
[1] 8

[[3]]
[1] 3.14
>
> Sys.time() - t0
Time difference of 3.03 secs
```

A *future_mclapply* function (3/3)

Or the hidden function (may change / disappear in future version of the package):

```
> t0 <- Sys.time()
>
> future:::flapply(
+     list(1:5, 6:10, pi),
+     slow(mean)
+ )
[[1]]
[1] 3

[[2]]
[1] 8

[[3]]
[1] 3.14
>
> Sys.time() - t0
Time difference of 3.02 secs
```

One package, many plans

- `plan(multiprocess)`: parallel, non blocking
- `plan(eager)`: non parallel, blocking (default)
- `plan(lazy)`: non parallel, non blocking
- `plan(cluster, workers = c("n1", "n2", "n3"))`: run in a cluster, non blocking. Nice gestion of the global variables.

- see also future.BatchJobs for more cluster plans.

Write package using *future*, users will choose how to run it by changing the `plan`.

Limiting the number of cores in `plan(multiprocess)` (1/2)

```
> availableCores()
system
     4
>
> plan(multiprocess(workers = 1 + 3)) # 1 main  + 3 background
> t0 <- Sys.time()
> x1 %<-% slow_rnorm(1)
> x2 %<-% slow_rnorm(1)
> x3 %<-% slow_rnorm(1)
>
>
> Sys.time() - t0
Time difference of 0.022 secs
>
> c(x1, x2, x3)
[1]  1.9745  0.0521 -1.1159
>
> Sys.time() - t0
Time difference of 3.02 secs
```

Limiting the number of cores in `plan(multiprocess)` (2/2)

```
> plan(multiprocess(workers = 1 + 2)) # 1 main  + 2 background
> t0 <- Sys.time()
> x1 %<-% slow_rnorm(1)
> x2 %<-% slow_rnorm(1)
> x3 %<-% slow_rnorm(1) # no more background process available, blocks the main process
>
> Sys.time() - t0
Time difference of 3.63 secs
>
> c(x1, x2, x3)
[1] 0.225 0.424 1.041
>
> Sys.time() - t0
Time difference of 6.64 secs
```

Miscellaneous

- Nested features are supported, see this vignette
- Check if a future is resolved:

```
> x1 %<-% slow_rnorm(3)
> f <- futureOf(x1)
> resolved(f)
[1] FALSE
```

- No easy way to stop an unresolved future at the moment (killing the process?)

Links

- The package: (https://cran.r-project.org/web/packages/future/index.html)
- The comprehensive vignette:https://cran.r-project.org/web/packages/future/vignettes/future-1-overview.html
- slides from useR2016 by Henrik Bengtsson.
- `doFuture` package: alternative to `doMC`, `doParallel`, `doMPI`, and `doSNOW`.