progressr 0.8.0 is on CRAN. It comes with some new features:

- A new 'rstudio' handler that reports on progress via the RStudio job interface in RStudio
- `withProgressShiny()` now updates the `detail` part, instead of the `message` part
- In addition to signalling relative amounts of progress, it's now also possible to signal total amounts

If you're curious what **progressr** is about, have a look at my e-Rum 2020 presentation.

Progress updates in RStudio's job interface

If you're using RStudio Console, you can now report on progress in the RStudio's job interface as long as the progress originates from a **progressr**-signalling function. I've shown an example of this in Figure 1.

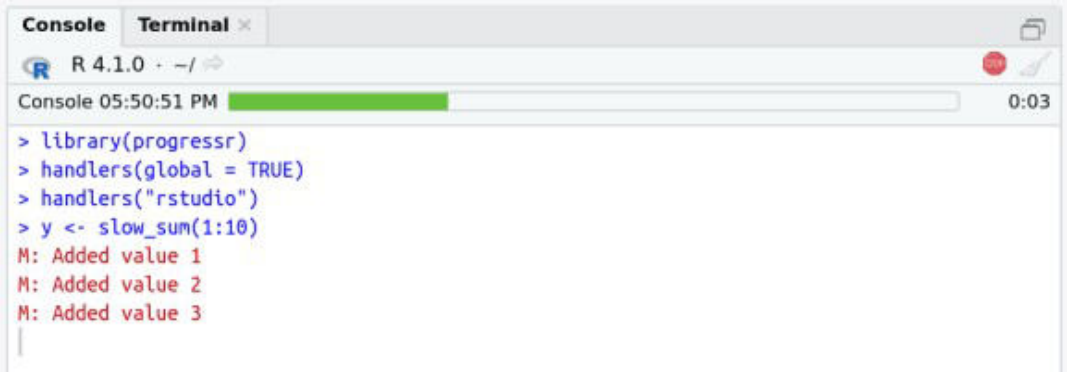

Figure 1: The RStudio job interface can show progress bars. In this screenshot, we see a green progress bar at 30% that has run for 0:03 (seconds) with a title to the left saying 'Console 05:50:51 PM' showing when the progress started.

To try this yourself, run the below in the RStudio Console.

```
library(progressr)
handlers(global = TRUE)
handlers("rstudio")
y <- slow_sum(1:10)
```

The progress bar disappears when the calculation completes.

Tweaks to withProgressShiny()

The `withProgressShiny()` function, which is a **progressr**-aware version of `withProgress()`, gained argument `inputs`. It defaults to `inputs = list(message = NULL, detail = "message")`, which says that a progress message should update the 'detail' part of the Shiny progress panel. For example,

```
X <- 1:10
withProgressShiny(message = "Calculation in progress",
                  detail = "Starting ...",
                  value = 0, {
  p <- progressor(along = X)
  y <- lapply(X, FUN=function(x) {
    Sys.sleep(0.25)
    p(sprintf("x=%d", x))
  })
})
```

will start out as in the left panel of Figure 2, and, as soon as the first progress signal is received, the

‘detail’ part is updated with `x=1` as shown in the right panel.

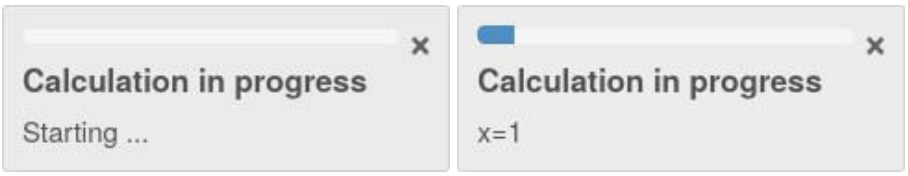

Figure 2: A Shiny progress panel that start out with the 'message' part displaying "Calculation in progress" and the 'detail' part displaying "Starting ..." (left), and whose 'detail' part is updated to "x=1" (right) as soon the first progress update comes in.

Prior to this new release, the default behavior was to update the ‘message’ part of the Shiny progress panel. To revert to the old behavior, set argument `inputs` as in:

```
X <- 1:10
withProgressShiny(message = "Starting ...",
                  detail = "Calculation in progress",
                  value = 0, {
  p <- progressor(along = X)
  y <- lapply(X, FUN=function(x) {
    Sys.sleep(0.25)
    p(sprintf("x=%d", x))
  })
}, inputs = list(message = "message", detail = NULL))
```

This results in what you see in Figure 3. I think that the new behavior, as shown in Figure 2, looks better and makes more sense.

Figure 3: A Shiny progress panel that start out with the 'message' part displaying "Starting ..." and the 'detail' part displaying "Calculation in progress" (left), and whose 'message' part is updated to "x=1" (right) as soon the first progress update comes in.

Update to a specific amount of total progress

When using **progressr**, we start out by creating a progressor function that we then call to signal progress. For example, if we do:

```
my_slow_fun <- function() {
  p <- progressr::progressor(steps = 10)
  count <- 0
  for (i in 1:10) {
    count <- count + 1
    p(sprintf("count=%d", count))
  }
  count
})
```

each call to `p()` corresponds to `p(amount = 1)`, which signals that our function have moved `amount = 1` steps closer to the total amount `steps = 10`. We can take smaller or bigger steps by specifying another `amount`.

In this new version, I've introduced a new beta feature that allows us to signal progress that says where we are in *absolute terms*. With this, we can do things like:

```
my_slow_fun <- function() {
  p <- progressr::progressor(steps = 10)
  count <- 0
  for (i in 1:5) {
    count <- count + 1
    if (runif(1) < 0.5) break
    Sys.sleep(2)
    p(sprintf("count=%d", count))
  }
  ## In case we broke out of the loop early,
  ## make sure to update to 5/10 progress
  p(step = 5)
  for (i in 1:5) {
    count <- count + 1
    Sys.sleep(2)
    p(sprintf("count=%d", count))
  }
  count
}
```

When calling `my_slow_fun()`, we might see progress being reported as:

```
- [-------------------------------------------------]   0%
\ [===>-------------------------------------]  10% count=1
| [=======>-----------------------------------]  20% count=2
\ [==================>--------------------]  50% count=3
...
```

Note how it took a leap from 20% to 50% when `count == 2`. If we run it another again, the move to 50% might happen at another iteration.

Wrapping up

There are also a few bug fixes, which you can read about in NEWS. And a usual, all of this work also when you run in parallel using the future framework.