Performance

Eric Archer

2/12/2018

Performance

In writing code, you are constantly balancing several things. The concept of "performance" could be related to accuracy, speed, stability, readability, or extensibility. In truth, all of these are important, but I would argue that accuracy (getting the right result) comes first. If the code does not deliver what is expected, then it is of no use regardless of how fast it executes or how easy it is to read.

Accuracy is often a direct result of readability. If you write code that is easy to follow and read, the flow of your logic will be obvious and tend to be clear - most importantly, to you!

Even when you've written some code that is performing as expected, you may want to spend some time getting it to run faster. The very first thing you need to do is understand where the bottlenecks are. Depending on how you've structured your code, there are multiple things that could be slower than expected and can be sped up with some restructuring.

Note that everything takes some time, so nothing is truly instantaneous. Recognizing that means that you will only be able to optimize but so much. At some point you will be spending more time re-programming trying to eke out a handful of milliseconds than it would take for all of the runs you could possibly imagine. Stop. You've done enough.

Profiling

But, if you're just getting started in this process, the first step you should do is profile your code. The core R functions for this are `Rprof` and `summaryRprof`. The former is used to start and stop profiling of code that has executed, and the latter summarizes the results of that profiling. In the example below, we'll profile some code that does some permutation of CTD data to produce a bootstrap mean of the temperature at each station.

```
# we're doing 10 replicates and want to return the result in an array
boot.mean <- sapply(1:10, function(i) {
  # read the data in
  df <- read.csv("ctd.csv", stringsAsFactors = FALSE)
  # loop over each value of station
  sapply(unique(df$station), function(st) {
    # identify the rows for this station and the first depth level
    i <- which(df$station == st & df$depth == 1)
    # take a random sample of these rows (with replacement)
    i <- sample(i, length(i), replace = TRUE)
    # return the mean of temperature for these rows
    mean(df$temp[i])
  })
})
ci <- apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))
ci
```

```
      Station.1 Station.10 Station.11 Station.12 Station.13 Station.14
2.5%   16.96571   16.40972   15.91380   16.61540   16.78820   16.57125
97.5%  18.00449   17.00735   16.76437   17.23257   17.40587   17.21197
```

```
      Station.15 Station.16 Station.17 Station.18 Station.19 Station.2
2.5%    16.34681   16.55129   16.18291    15.9694   16.89645  16.49076
97.5%   17.15080   17.14203   16.79884    16.9914   17.56478  16.99815
      Station.20 Station.21 Station.22 Station.23 Station.24 Station.25
2.5%    16.76123   16.51598   16.44913   16.32203   17.23517   16.56966
97.5%   17.25565   17.59090   17.40545   16.89503   17.71079   16.97226
      Station.26 Station.27 Station.28 Station.29 Station.3 Station.30
2.5%    16.76203   16.07651   16.66791   16.27764  16.34010   16.31692
97.5%   16.99571   16.97278   17.49306   17.12553  17.07857   16.78918
      Station.31 Station.32 Station.33 Station.34 Station.35 Station.36
2.5%    15.92600   17.29165   16.23401   15.93970   16.24278   16.10900
97.5%   17.11182   17.74407   16.76567   16.58322   16.84793   17.14575
      Station.37 Station.38 Station.39 Station.4 Station.40 Station.5
2.5%    15.64305   16.45678   16.66183  15.94942   16.88526  15.52572
97.5%   16.48501   17.12169   16.99933  16.62759   17.67809  16.45601
      Station.6 Station.7 Station.8 Station.9
2.5%   16.31252  16.37262  16.33176  16.43304
97.5%  17.07155  17.23546  17.09358  17.35681
```

Let's first see what parts of this are taking the most time and how much:

```r
# Start profiling
Rprof()

# Run code
boot.mean <- sapply(1:10, function(i) {
  df <- read.csv("ctd.csv", stringsAsFactors = FALSE)
  sapply(unique(df$station), function(st) {
    i <- which(df$station == st & df$depth == 1)
    i <- sample(i, length(i), replace = TRUE)
    mean(df$temp[i])
  })
})
ci <- apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))

# Stop profiling
Rprof(NULL)

# Examine profile summary
summaryRprof()
```

```
$by.self
                   self.time self.pct total.time total.pct
"scan"                  5.92    83.38       5.92     83.38
"which"                 0.66     9.30       0.76     10.70
".External2"            0.38     5.35       0.38      5.35
"&"                     0.06     0.85       0.06      0.85
"<Anonymous>"           0.02     0.28       0.02      0.28
"=="                    0.02     0.28       0.02      0.28
"close.connection"      0.02     0.28       0.02      0.28
"mean"                  0.02     0.28       0.02      0.28

$by.total
                     total.time total.pct self.time self.pct
"block_exec"               7.10    100.00      0.00     0.00
```

```
"call_block"                7.10    100.00      0.00     0.00
"doTryCatch"                7.10    100.00      0.00     0.00
"eval"                      7.10    100.00      0.00     0.00
"evaluate_call"             7.10    100.00      0.00     0.00
"evaluate::evaluate"        7.10    100.00      0.00     0.00
"evaluate"                  7.10    100.00      0.00     0.00
"FUN"                       7.10    100.00      0.00     0.00
"handle"                    7.10    100.00      0.00     0.00
"in_dir"                    7.10    100.00      0.00     0.00
"knitr::knit"               7.10    100.00      0.00     0.00
"lapply"                    7.10    100.00      0.00     0.00
"process_file"              7.10    100.00      0.00     0.00
"process_group.block"       7.10    100.00      0.00     0.00
"process_group"             7.10    100.00      0.00     0.00
"rmarkdown::render"         7.10    100.00      0.00     0.00
"sapply"                    7.10    100.00      0.00     0.00
"timing_fn"                 7.10    100.00      0.00     0.00
"try"                       7.10    100.00      0.00     0.00
"tryCatch"                  7.10    100.00      0.00     0.00
"tryCatchList"              7.10    100.00      0.00     0.00
"tryCatchOne"               7.10    100.00      0.00     0.00
"withCallingHandlers"       7.10    100.00      0.00     0.00
"withVisible"               7.10    100.00      0.00     0.00
"read.csv"                  6.32     89.01      0.00     0.00
"read.table"                6.32     89.01      0.00     0.00
"scan"                      5.92     83.38      5.92    83.38
"which"                     0.76     10.70      0.66     9.30
".External2"                0.38      5.35      0.38     5.35
"type.convert"              0.38      5.35      0.00     0.00
"&"                         0.06      0.85      0.06     0.85
"<Anonymous>"               0.02      0.28      0.02     0.28
"=="                        0.02      0.28      0.02     0.28
"close.connection"          0.02      0.28      0.02     0.28
"mean"                      0.02      0.28      0.02     0.28
"[[.data.frame"             0.02      0.28      0.00     0.00
"[["                        0.02      0.28      0.00     0.00
"$.data.frame"              0.02      0.28      0.00     0.00
"$"                         0.02      0.28      0.00     0.00
"close"                     0.02      0.28      0.00     0.00

$sample.interval
[1] 0.02

$sampling.time
[1] 7.1
```

We can see that a majority of the time was taken by reading the file ("scan", "read.table", "read.csv"). This makes sense because it is doing that every iteration of the loop in the `sapply` function. Let's be smarter and change the code so that file reading is happening only once:

```
# Start profiling
Rprof()

# Run code
df <- read.csv("ctd.csv", stringsAsFactors = FALSE)
```

```r
boot.mean <- sapply(1:10, function(i) {
  sapply(unique(df$station), function(st) {
    i <- which(df$station == st & df$depth == 1)
    i <- sample(i, length(i), replace = TRUE)
    mean(df$temp[i])
  })
})
ci <- apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))

# Stop profiling
Rprof(NULL)

# Examine profile summary
summaryRprof()
```

```
$by.self
             self.time self.pct total.time total.pct
"scan"            0.64    55.17       0.64     55.17
"which"           0.44    37.93       0.48     41.38
".External2"      0.04     3.45       0.04      3.45
"&"               0.02     1.72       0.02      1.72
"=="              0.02     1.72       0.02      1.72

$by.total
                    total.time total.pct self.time self.pct
"block_exec"              1.16    100.00      0.00     0.00
"call_block"              1.16    100.00      0.00     0.00
"doTryCatch"              1.16    100.00      0.00     0.00
"eval"                    1.16    100.00      0.00     0.00
"evaluate_call"           1.16    100.00      0.00     0.00
"evaluate::evaluate"      1.16    100.00      0.00     0.00
"evaluate"                1.16    100.00      0.00     0.00
"handle"                  1.16    100.00      0.00     0.00
"in_dir"                  1.16    100.00      0.00     0.00
"knitr::knit"             1.16    100.00      0.00     0.00
"process_file"            1.16    100.00      0.00     0.00
"process_group.block"     1.16    100.00      0.00     0.00
"process_group"           1.16    100.00      0.00     0.00
"rmarkdown::render"       1.16    100.00      0.00     0.00
"timing_fn"               1.16    100.00      0.00     0.00
"try"                     1.16    100.00      0.00     0.00
"tryCatch"                1.16    100.00      0.00     0.00
"tryCatchList"            1.16    100.00      0.00     0.00
"tryCatchOne"             1.16    100.00      0.00     0.00
"withCallingHandlers"     1.16    100.00      0.00     0.00
"withVisible"             1.16    100.00      0.00     0.00
"read.csv"                0.68     58.62      0.00     0.00
"read.table"              0.68     58.62      0.00     0.00
"scan"                    0.64     55.17      0.64    55.17
"which"                   0.48     41.38      0.44    37.93
"FUN"                     0.48     41.38      0.00     0.00
"lapply"                  0.48     41.38      0.00     0.00
"sapply"                  0.48     41.38      0.00     0.00
".External2"              0.04      3.45      0.04     3.45
```

```
"type.convert"              0.04      3.45      0.00     0.00
"&"                         0.02      1.72      0.02     1.72
"=="                        0.02      1.72      0.02     1.72

$sample.interval
[1] 0.02

$sampling.time
[1] 1.16
```

Notice that our total time decreased, but file reading is still taking the most time. Since this is on its own line, we can't really make this any faster, but lets profile just the nested `sapply` lines:

```
df <- read.csv("ctd.csv", stringsAsFactors = FALSE)

# Start profiling
Rprof()

# Run code
boot.mean <- sapply(1:10, function(i) {
  sapply(unique(df$station), function(st) {
    i <- which(df$station == st & df$depth == 1)
    i <- sample(i, length(i), replace = TRUE)
    mean(df$temp[i])
  })
})
ci <- apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))

# Stop profiling
Rprof(NULL)

# Examine profile summary
summaryRprof()
```

```
$by.self
        self.time self.pct total.time total.pct
"which"      0.42    87.50       0.48    100.00
"=="         0.04     8.33       0.04      8.33
"$"          0.02     4.17       0.02      4.17

$by.total
                     total.time total.pct self.time self.pct
"which"                    0.48    100.00      0.42    87.50
"block_exec"               0.48    100.00      0.00     0.00
"call_block"               0.48    100.00      0.00     0.00
"doTryCatch"               0.48    100.00      0.00     0.00
"eval"                     0.48    100.00      0.00     0.00
"evaluate_call"            0.48    100.00      0.00     0.00
"evaluate::evaluate"       0.48    100.00      0.00     0.00
"evaluate"                 0.48    100.00      0.00     0.00
"FUN"                      0.48    100.00      0.00     0.00
"handle"                   0.48    100.00      0.00     0.00
"in_dir"                   0.48    100.00      0.00     0.00
"knitr::knit"              0.48    100.00      0.00     0.00
"lapply"                   0.48    100.00      0.00     0.00
```

```
"process_file"              0.48    100.00      0.00     0.00
"process_group.block"       0.48    100.00      0.00     0.00
"process_group"             0.48    100.00      0.00     0.00
"rmarkdown::render"         0.48    100.00      0.00     0.00
"sapply"                    0.48    100.00      0.00     0.00
"timing_fn"                 0.48    100.00      0.00     0.00
"try"                       0.48    100.00      0.00     0.00
"tryCatch"                  0.48    100.00      0.00     0.00
"tryCatchList"              0.48    100.00      0.00     0.00
"tryCatchOne"               0.48    100.00      0.00     0.00
"withCallingHandlers"       0.48    100.00      0.00     0.00
"withVisible"               0.48    100.00      0.00     0.00
"=="                        0.04      8.33      0.04     8.33
"$"                         0.02      4.17      0.02     4.17

$sample.interval
[1] 0.02

$sampling.time
[1] 0.48
```

Now it seems like `which` is taking most of the time, so lets focus on that. If we remember that we don't have to use which. We can just index with the base logical vector. Since we still need to randomly sample temperature with replacement, we'll extract that column and do the sampling on it:

```
df <- read.csv("ctd.csv", stringsAsFactors = FALSE)

# Start profiling
Rprof()

# Run code
boot.mean <- sapply(1:10, function(i) {
  sapply(unique(df$station), function(st) {
    temp <- df$temp[df$station == st & df$depth == 1]
    temp <- sample(temp, length(temp), replace = TRUE)
    mean(temp)
  })
})
ci <- apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))

# Stop profiling
Rprof(NULL)

# Examine profile summary
summaryRprof()
```

```
$by.self
      self.time self.pct total.time total.pct
"FUN"      0.42     91.3       0.46     100.0
"=="       0.04      8.7       0.04       8.7

$by.total
                     total.time total.pct self.time self.pct
"FUN"                      0.46     100.0      0.42     91.3
"block_exec"               0.46     100.0      0.00      0.0
```

```
"call_block"                0.46     100.0      0.00      0.0
"doTryCatch"                0.46     100.0      0.00      0.0
"eval"                      0.46     100.0      0.00      0.0
"evaluate_call"             0.46     100.0      0.00      0.0
"evaluate::evaluate"        0.46     100.0      0.00      0.0
"evaluate"                  0.46     100.0      0.00      0.0
"handle"                    0.46     100.0      0.00      0.0
"in_dir"                    0.46     100.0      0.00      0.0
"knitr::knit"               0.46     100.0      0.00      0.0
"lapply"                    0.46     100.0      0.00      0.0
"process_file"              0.46     100.0      0.00      0.0
"process_group.block"       0.46     100.0      0.00      0.0
"process_group"             0.46     100.0      0.00      0.0
"rmarkdown::render"         0.46     100.0      0.00      0.0
"sapply"                    0.46     100.0      0.00      0.0
"timing_fn"                 0.46     100.0      0.00      0.0
"try"                       0.46     100.0      0.00      0.0
"tryCatch"                  0.46     100.0      0.00      0.0
"tryCatchList"              0.46     100.0      0.00      0.0
"tryCatchOne"               0.46     100.0      0.00      0.0
"withCallingHandlers"       0.46     100.0      0.00      0.0
"withVisible"               0.46     100.0      0.00      0.0
"=="                        0.04       8.7      0.04      8.7

$sample.interval
[1] 0.02

$sampling.time
[1] 0.46
```

We can now see that we're spending a non-negligble amount of time in the logical operators, `==` and `&`. Since we're always interested in the first depth class (`depth == 1`), let's extract that early on:

```
df <- read.csv("ctd.csv", stringsAsFactors = FALSE)

# Start profiling
Rprof()

# Run code
df.1 <- df[df$depth == 1, ]
boot.mean <- sapply(1:10, function(i) {
  sapply(unique(df.1$station), function(st) {
    temp <- df.1$temp[df.1$station == st]
    temp <- sample(temp, length(temp), replace = TRUE)
    mean(temp)
  })
})
ci <- apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))

# Stop profiling
Rprof(NULL)

# Examine profile summary
summaryRprof()
```

```
$by.self
             self.time self.pct total.time total.pct
"sample.int"      0.02      100       0.02       100

$by.total
                      total.time total.pct self.time self.pct
"sample.int"                0.02       100      0.02      100
"block_exec"                0.02       100      0.00        0
"call_block"                0.02       100      0.00        0
"doTryCatch"                0.02       100      0.00        0
"eval"                      0.02       100      0.00        0
"evaluate_call"             0.02       100      0.00        0
"evaluate::evaluate"        0.02       100      0.00        0
"evaluate"                  0.02       100      0.00        0
"FUN"                       0.02       100      0.00        0
"handle"                    0.02       100      0.00        0
"in_dir"                    0.02       100      0.00        0
"knitr::knit"               0.02       100      0.00        0
"lapply"                    0.02       100      0.00        0
"process_file"              0.02       100      0.00        0
"process_group.block"       0.02       100      0.00        0
"process_group"             0.02       100      0.00        0
"rmarkdown::render"         0.02       100      0.00        0
"sample"                    0.02       100      0.00        0
"sapply"                    0.02       100      0.00        0
"timing_fn"                 0.02       100      0.00        0
"try"                       0.02       100      0.00        0
"tryCatch"                  0.02       100      0.00        0
"tryCatchList"              0.02       100      0.00        0
"tryCatchOne"               0.02       100      0.00        0
"withCallingHandlers"       0.02       100      0.00        0
"withVisible"               0.02       100      0.00        0

$sample.interval
[1] 0.02

$sampling.time
[1] 0.02
```

This is considerably faster than before. Let's see where (if) there are other bottlenecks now by increasing the number of replicates from 10 to 1000. Also, because the total time is getting smaller, let's decrease the sampling interval to 0.01.

```r
df <- read.csv("ctd.csv", stringsAsFactors = FALSE)

# Start profiling
Rprof(interval = 0.01)

# Run code
df.1 <- df[df$depth == 1, ]
boot.mean <- sapply(1:1000, function(i) {
  sapply(unique(df.1$station), function(st) {
    temp <- df.1$temp[df.1$station == st]
    temp <- sample(temp, length(temp), replace = TRUE)
    mean(temp)
```

```
  })
})
ci <- apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))

# Stop profiling
Rprof(NULL)

# Examine profile summary
summaryRprof()
```

```
$by.self
                 self.time self.pct total.time total.pct
"FUN"                 1.21    62.69       1.93    100.00
"sample.int"          0.11     5.70       0.11      5.70
"[[.data.frame"       0.10     5.18       0.21     10.88
"sample"              0.08     4.15       0.19      9.84
"%in%"                0.06     3.11       0.10      5.18
"mean"                0.06     3.11       0.09      4.66
"$"                   0.05     2.59       0.34     17.62
"unique.default"      0.05     2.59       0.05      2.59
"$.data.frame"        0.04     2.07       0.29     15.03
"[["                  0.04     2.07       0.25     12.95
"lapply"              0.03     1.55       1.92     99.48
"mean.default"        0.03     1.55       0.03      1.55
"names"               0.02     1.04       0.02      1.04
"sys.call"            0.02     1.04       0.02      1.04
"unique"              0.01     0.52       0.06      3.11
"<Anonymous>"         0.01     0.52       0.01      0.52
"simplify2array"      0.01     0.52       0.01      0.52

$by.total
                     total.time total.pct self.time self.pct
"FUN"                      1.93    100.00      1.21    62.69
"block_exec"               1.93    100.00      0.00     0.00
"call_block"               1.93    100.00      0.00     0.00
"doTryCatch"               1.93    100.00      0.00     0.00
"eval"                     1.93    100.00      0.00     0.00
"evaluate_call"            1.93    100.00      0.00     0.00
"evaluate::evaluate"       1.93    100.00      0.00     0.00
"evaluate"                 1.93    100.00      0.00     0.00
"handle"                   1.93    100.00      0.00     0.00
"in_dir"                   1.93    100.00      0.00     0.00
"knitr::knit"              1.93    100.00      0.00     0.00
"process_file"             1.93    100.00      0.00     0.00
"process_group.block"      1.93    100.00      0.00     0.00
"process_group"            1.93    100.00      0.00     0.00
"rmarkdown::render"        1.93    100.00      0.00     0.00
"timing_fn"                1.93    100.00      0.00     0.00
"try"                      1.93    100.00      0.00     0.00
"tryCatch"                 1.93    100.00      0.00     0.00
"tryCatchList"             1.93    100.00      0.00     0.00
"tryCatchOne"              1.93    100.00      0.00     0.00
"withCallingHandlers"      1.93    100.00      0.00     0.00
"withVisible"              1.93    100.00      0.00     0.00
```

```
"lapply"                    1.92     99.48      0.03     1.55
"sapply"                    1.92     99.48      0.00     0.00
"$"                         0.34     17.62      0.05     2.59
"$.data.frame"              0.29     15.03      0.04     2.07
"[["                        0.25     12.95      0.04     2.07
"[[.data.frame"             0.21     10.88      0.10     5.18
"sample"                    0.19      9.84      0.08     4.15
"sample.int"                0.11      5.70      0.11     5.70
"%in%"                      0.10      5.18      0.06     3.11
"mean"                      0.09      4.66      0.06     3.11
"unique"                    0.06      3.11      0.01     0.52
"unique.default"            0.05      2.59      0.05     2.59
"mean.default"              0.03      1.55      0.03     1.55
"names"                     0.02      1.04      0.02     1.04
"sys.call"                  0.02      1.04      0.02     1.04
"<Anonymous>"               0.01      0.52      0.01     0.52
"simplify2array"            0.01      0.52      0.01     0.52
"apply"                     0.01      0.52      0.00     0.00
"quantile.default"          0.01      0.52      0.00     0.00
"sort.default"              0.01      0.52      0.00     0.00
"sort.int"                  0.01      0.52      0.00     0.00
"sort"                      0.01      0.52      0.00     0.00

$sample.interval
[1] 0.01

$sampling.time
[1] 1.93
```

Here we see that we're spending most of our recoverable time in sample. It isn't easy to make that or simplify that step as we have to do this random sampling. The next items down have to do with indexing the data frame (`[.data.frame`). Lets try to handle that by doing some extraction ahead of time and working with vectors rather than data.frames:

```
df <- read.csv("ctd.csv", stringsAsFactors = FALSE)

# Start profiling
Rprof(interval = 0.01)

# Run code
df.1 <- df[df$depth == 1, ]
temp <- df.1$temp
station <- df.1$station

boot.mean <- sapply(1:1000, function(i) {
  sapply(unique(station), function(st) {
    st.temp <- temp[station == st]
    st.temp <- sample(st.temp, length(st.temp), replace = TRUE)
    mean(st.temp)
  })
})
ci <- apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))

# Stop profiling
Rprof(NULL)
```

```r
# Examine profile summary
summaryRprof()
```

```
$by.self
                 self.time self.pct total.time total.pct
"FUN"                 1.08    76.06       1.42    100.00
"unique.default"      0.10     7.04       0.10      7.04
"sample.int"          0.09     6.34       0.10      7.04
"sample"              0.05     3.52       0.15     10.56
"mean"                0.03     2.11       0.05      3.52
"mean.default"        0.02     1.41       0.02      1.41
"lapply"              0.01     0.70       1.42    100.00
"unique"              0.01     0.70       0.11      7.75
"$"                   0.01     0.70       0.01      0.70
"length"              0.01     0.70       0.01      0.70
"unlist"              0.01     0.70       0.01      0.70

$by.total
                      total.time total.pct self.time self.pct
"FUN"                       1.42    100.00      1.08    76.06
"lapply"                    1.42    100.00      0.01     0.70
"block_exec"                1.42    100.00      0.00     0.00
"call_block"                1.42    100.00      0.00     0.00
"doTryCatch"                1.42    100.00      0.00     0.00
"eval"                      1.42    100.00      0.00     0.00
"evaluate_call"             1.42    100.00      0.00     0.00
"evaluate::evaluate"        1.42    100.00      0.00     0.00
"evaluate"                  1.42    100.00      0.00     0.00
"handle"                    1.42    100.00      0.00     0.00
"in_dir"                    1.42    100.00      0.00     0.00
"knitr::knit"               1.42    100.00      0.00     0.00
"process_file"              1.42    100.00      0.00     0.00
"process_group.block"       1.42    100.00      0.00     0.00
"process_group"             1.42    100.00      0.00     0.00
"rmarkdown::render"         1.42    100.00      0.00     0.00
"sapply"                    1.42    100.00      0.00     0.00
"timing_fn"                 1.42    100.00      0.00     0.00
"try"                       1.42    100.00      0.00     0.00
"tryCatch"                  1.42    100.00      0.00     0.00
"tryCatchList"              1.42    100.00      0.00     0.00
"tryCatchOne"               1.42    100.00      0.00     0.00
"withCallingHandlers"       1.42    100.00      0.00     0.00
"withVisible"               1.42    100.00      0.00     0.00
"sample"                    0.15     10.56      0.05     3.52
"unique"                    0.11      7.75      0.01     0.70
"unique.default"            0.10      7.04      0.10     7.04
"sample.int"                0.10      7.04      0.09     6.34
"mean"                      0.05      3.52      0.03     2.11
"mean.default"              0.02      1.41      0.02     1.41
"$"                         0.01      0.70      0.01     0.70
"length"                    0.01      0.70      0.01     0.70
"unlist"                    0.01      0.70      0.01     0.70
"cb$putconst"               0.01      0.70      0.00     0.00
```

```
"cmp"                       0.01      0.70      0.00      0.00
"cmpCall"                   0.01      0.70      0.00      0.00
"cmpCallArgs"               0.01      0.70      0.00      0.00
"cmpCallSymFun"             0.01      0.70      0.00      0.00
"cmpfun"                    0.01      0.70      0.00      0.00
"cmpSym"                    0.01      0.70      0.00      0.00
"compiler:::tryCmpfun"      0.01      0.70      0.00      0.00
"genCode"                   0.01      0.70      0.00      0.00
"h"                         0.01      0.70      0.00      0.00
"simplify2array"            0.01      0.70      0.00      0.00
"tryInline"                 0.01      0.70      0.00      0.00

$sample.interval
[1] 0.01

$sampling.time
[1] 1.42
```

That made a noticeable improvement. It also highlights something else we're doing repeatedly - using `unique` to get the unique station names. Lets do that earlier.

```
df <- read.csv("ctd.csv", stringsAsFactors = FALSE)

# Start profiling
Rprof(interval = 0.01)

# Run code
df.1 <- df[df$depth == 1, ]
temp <- df.1$temp
station <- df.1$station
st.names <- unique(station)

boot.mean <- sapply(1:1000, function(i) {
  sapply(st.names, function(st) {
    st.temp <- temp[station == st]
    st.temp <- sample(st.temp, length(st.temp), replace = TRUE)
    mean(st.temp)
  })
})
ci <- apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))

# Stop profiling
Rprof(NULL)

# Examine profile summary
summaryRprof()
```

```
$by.self
                  self.time self.pct total.time total.pct
"FUN"                  1.03    75.74       1.36    100.00
"mean"                 0.12     8.82       0.14     10.29
"sample.int"           0.10     7.35       0.10      7.35
"sample"               0.05     3.68       0.15     11.03
"lapply"               0.01     0.74       1.35     99.26
"mean.default"         0.01     0.74       0.02      1.47
```

```
"length"                0.01     0.74       0.01      0.74
"quantile.default"      0.01     0.74       0.01      0.74
"unique"                0.01     0.74       0.01      0.74
"unlist"                0.01     0.74       0.01      0.74

$by.total
                      total.time total.pct self.time self.pct
"FUN"                       1.36    100.00      1.03    75.74
"block_exec"                1.36    100.00      0.00     0.00
"call_block"                1.36    100.00      0.00     0.00
"doTryCatch"                1.36    100.00      0.00     0.00
"eval"                      1.36    100.00      0.00     0.00
"evaluate_call"             1.36    100.00      0.00     0.00
"evaluate::evaluate"        1.36    100.00      0.00     0.00
"evaluate"                  1.36    100.00      0.00     0.00
"handle"                    1.36    100.00      0.00     0.00
"in_dir"                    1.36    100.00      0.00     0.00
"knitr::knit"               1.36    100.00      0.00     0.00
"process_file"              1.36    100.00      0.00     0.00
"process_group.block"       1.36    100.00      0.00     0.00
"process_group"             1.36    100.00      0.00     0.00
"rmarkdown::render"         1.36    100.00      0.00     0.00
"timing_fn"                 1.36    100.00      0.00     0.00
"try"                       1.36    100.00      0.00     0.00
"tryCatch"                  1.36    100.00      0.00     0.00
"tryCatchList"              1.36    100.00      0.00     0.00
"tryCatchOne"               1.36    100.00      0.00     0.00
"withCallingHandlers"       1.36    100.00      0.00     0.00
"withVisible"               1.36    100.00      0.00     0.00
"lapply"                    1.35     99.26      0.01     0.74
"sapply"                    1.35     99.26      0.00     0.00
"sample"                    0.15     11.03      0.05     3.68
"mean"                      0.14     10.29      0.12     8.82
"sample.int"                0.10      7.35      0.10     7.35
"mean.default"              0.02      1.47      0.01     0.74
"simplify2array"            0.02      1.47      0.00     0.00
"length"                    0.01      0.74      0.01     0.74
"quantile.default"          0.01      0.74      0.01     0.74
"unique"                    0.01      0.74      0.01     0.74
"unlist"                    0.01      0.74      0.01     0.74
"apply"                     0.01      0.74      0.00     0.00

$sample.interval
[1] 0.01

$sampling.time
[1] 1.36
```

Looping

That produced a minor, but useful improvement in speed. As we look through the rest of timings, we can see that there aren't a lot more savings to get. Most of the time is being taken by `sapply`, which we need in order to do the looping. We can use another member of the `apply` family to do grouped iterations: `tapply`:

```r
df <- read.csv("ctd.csv", stringsAsFactors = FALSE)

# Start profiling
Rprof(interval = 0.01)

# Run code
df.1 <- df[df$depth == 1, ]
temp <- df.1$temp
station <- df.1$station
st.names <- unique(station)

boot.mean <- sapply(1:1000, function(i) {
  tapply(temp, station, function(st.temp) {
    st.temp <- sample(st.temp, length(st.temp), replace = TRUE)
    mean(st.temp)
  })
})
ci <- apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))

# Stop profiling
Rprof(NULL)

# Examine profile summary
summaryRprof()
```

```
$by.self
                 self.time self.pct total.time total.pct
"sample"              0.13    20.63       0.27     42.86
"sample.int"          0.13    20.63       0.14     22.22
"factor"              0.12    19.05       0.17     26.98
"split.default"       0.05     7.94       0.05      7.94
"FUN"                 0.04     6.35       0.63    100.00
"mean"                0.04     6.35       0.05      7.94
"unique.default"      0.04     6.35       0.04      6.35
"eval"                0.01     1.59       0.63    100.00
"lapply"              0.01     1.59       0.62     98.41
"tapply"              0.01     1.59       0.61     96.83
"("                   0.01     1.59       0.01      1.59
"as.integer"          0.01     1.59       0.01      1.59
"exists"              0.01     1.59       0.01      1.59
"mean.default"        0.01     1.59       0.01      1.59
"pmax"                0.01     1.59       0.01      1.59

$by.total
                      total.time total.pct self.time self.pct
"FUN"                       0.63    100.00      0.04     6.35
"eval"                      0.63    100.00      0.01     1.59
"block_exec"                0.63    100.00      0.00     0.00
"call_block"                0.63    100.00      0.00     0.00
"doTryCatch"                0.63    100.00      0.00     0.00
"evaluate_call"             0.63    100.00      0.00     0.00
"evaluate::evaluate"        0.63    100.00      0.00     0.00
"evaluate"                  0.63    100.00      0.00     0.00
"handle"                    0.63    100.00      0.00     0.00
```

```
"in_dir"                    0.63    100.00      0.00     0.00
"knitr::knit"               0.63    100.00      0.00     0.00
"process_file"              0.63    100.00      0.00     0.00
"process_group.block"       0.63    100.00      0.00     0.00
"process_group"             0.63    100.00      0.00     0.00
"rmarkdown::render"         0.63    100.00      0.00     0.00
"timing_fn"                 0.63    100.00      0.00     0.00
"try"                       0.63    100.00      0.00     0.00
"tryCatch"                  0.63    100.00      0.00     0.00
"tryCatchList"              0.63    100.00      0.00     0.00
"tryCatchOne"               0.63    100.00      0.00     0.00
"withCallingHandlers"       0.63    100.00      0.00     0.00
"withVisible"               0.63    100.00      0.00     0.00
"lapply"                    0.62     98.41      0.01     1.59
"sapply"                    0.62     98.41      0.00     0.00
"tapply"                    0.61     96.83      0.01     1.59
"sample"                    0.27     42.86      0.13    20.63
"factor"                    0.17     26.98      0.12    19.05
"sample.int"                0.14     22.22      0.13    20.63
"split.default"             0.05      7.94      0.05     7.94
"mean"                      0.05      7.94      0.04     6.35
"split"                     0.05      7.94      0.00     0.00
"unique.default"            0.04      6.35      0.04     6.35
"unique"                    0.04      6.35      0.00     0.00
"("                         0.01      1.59      0.01     1.59
"as.integer"                0.01      1.59      0.01     1.59
"exists"                    0.01      1.59      0.01     1.59
"mean.default"              0.01      1.59      0.01     1.59
"pmax"                      0.01      1.59      0.01     1.59
"apply"                     0.01      1.59      0.00     0.00
"cb$putconst"               0.01      1.59      0.00     0.00
"cmp"                       0.01      1.59      0.00     0.00
"cmpCall"                   0.01      1.59      0.00     0.00
"cmpCallArgs"               0.01      1.59      0.00     0.00
"cmpCallSymFun"             0.01      1.59      0.00     0.00
"cmpfun"                    0.01      1.59      0.00     0.00
"cmpSymbolAssign"           0.01      1.59      0.00     0.00
"compiler:::tryCmpfun"      0.01      1.59      0.00     0.00
"findCenvVar"               0.01      1.59      0.00     0.00
"findLocVar"                0.01      1.59      0.00     0.00
"format_perc"               0.01      1.59      0.00     0.00
"formatC"                   0.01      1.59      0.00     0.00
"genCode"                   0.01      1.59      0.00     0.00
"h"                         0.01      1.59      0.00     0.00
"match.arg"                 0.01      1.59      0.00     0.00
"paste0"                    0.01      1.59      0.00     0.00
"quantile.default"          0.01      1.59      0.00     0.00
"sort.list"                 0.01      1.59      0.00     0.00
"tryInline"                 0.01      1.59      0.00     0.00

$sample.interval
[1] 0.01

$sampling.time
```

```
[1] 0.63
```

Although we're still spending time in the `tapply` and `sapply`, we've cut down the total time considerably. One thing we can check is if there is an effect of the order of the loops. Currently, we are calculating the mean for all stations for each replicate. Lets switch the order so that we are calculating the means of all replicates for each station:

```
df <- read.csv("ctd.csv", stringsAsFactors = FALSE)

# Start profiling
Rprof(interval = 0.01)

# Run code
df.1 <- df[df$depth == 1, ]
temp <- df.1$temp
station <- df.1$station
st.names <- unique(station)

boot.mean <- tapply(temp, station, function(x) {
  sapply(1:1000, function(i) {
    st.temp <- sample(x, length(x), replace = TRUE)
    mean(st.temp)
  })
})
boot.mean <- do.call(rbind, boot.mean)
ci <- apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))

# Stop profiling
Rprof(NULL)

# Examine profile summary
summaryRprof()
```

```
$by.self
               self.time self.pct total.time total.pct
"sample"            0.12    34.29       0.23     65.71
"sample.int"        0.09    25.71       0.10     28.57
"mean"              0.04    11.43       0.06     17.14
"FUN"               0.03     8.57       0.35    100.00
"lapply"            0.02     5.71       0.34     97.14
"mean.default"      0.02     5.71       0.02      5.71
"!"                 0.01     2.86       0.01      2.86
"formatC"           0.01     2.86       0.01      2.86
"length"            0.01     2.86       0.01      2.86

$by.total
                     total.time total.pct self.time self.pct
"FUN"                      0.35    100.00      0.03     8.57
"block_exec"               0.35    100.00      0.00     0.00
"call_block"               0.35    100.00      0.00     0.00
"doTryCatch"               0.35    100.00      0.00     0.00
"eval"                     0.35    100.00      0.00     0.00
"evaluate_call"            0.35    100.00      0.00     0.00
"evaluate::evaluate"       0.35    100.00      0.00     0.00
"evaluate"                 0.35    100.00      0.00     0.00
```

```
"handle"                    0.35    100.00      0.00     0.00
"in_dir"                    0.35    100.00      0.00     0.00
"knitr::knit"               0.35    100.00      0.00     0.00
"process_file"              0.35    100.00      0.00     0.00
"process_group.block"       0.35    100.00      0.00     0.00
"process_group"             0.35    100.00      0.00     0.00
"rmarkdown::render"         0.35    100.00      0.00     0.00
"timing_fn"                 0.35    100.00      0.00     0.00
"try"                       0.35    100.00      0.00     0.00
"tryCatch"                  0.35    100.00      0.00     0.00
"tryCatchList"              0.35    100.00      0.00     0.00
"tryCatchOne"               0.35    100.00      0.00     0.00
"withCallingHandlers"       0.35    100.00      0.00     0.00
"withVisible"               0.35    100.00      0.00     0.00
"lapply"                    0.34     97.14      0.02     5.71
"sapply"                    0.34     97.14      0.00     0.00
"tapply"                    0.34     97.14      0.00     0.00
"sample"                    0.23     65.71      0.12    34.29
"sample.int"                0.10     28.57      0.09    25.71
"mean"                      0.06     17.14      0.04    11.43
"mean.default"              0.02      5.71      0.02     5.71
"!"                         0.01      2.86      0.01     2.86
"formatC"                   0.01      2.86      0.01     2.86
"length"                    0.01      2.86      0.01     2.86
"apply"                     0.01      2.86      0.00     0.00
"format_perc"               0.01      2.86      0.00     0.00
"paste0"                    0.01      2.86      0.00     0.00
"quantile.default"          0.01      2.86      0.00     0.00

$sample.interval
[1] 0.01

$sampling.time
[1] 0.35
```

This is slightly faster because we are not doing the tapply 1000 times, which takes some time. It doesn't look like we can do much more. However, there is a more efficient way of looping, although it is not necessarily as compact. Instead of using the interior `sapply` constructs, we can pre-allocate a result vector, and use a `for` loop to fill it:

```
df <- read.csv("ctd.csv", stringsAsFactors = FALSE)

# Start profiling
Rprof(interval = 0.01)

# Run code
df.1 <- df[df$depth == 1, ]
temp <- df.1$temp
station <- df.1$station
st.names <- unique(station)

# an empty result vector
boot.vec <- vector(length = 1000)

boot.mean <- tapply(temp, station, function(x) {
```

```
  for(i in 1:length(boot.vec)) {
    st.temp <- sample(x, length(x), replace = TRUE)
    boot.vec[i] <- mean(st.temp)
  }
  boot.vec
})
boot.mean <- do.call(rbind, boot.mean)
ci <- apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))

# Stop profiling
Rprof(NULL)

# Examine profile summary
summaryRprof()
```

```
$by.self
               self.time self.pct total.time total.pct
"sample.int"         0.15    46.88       0.17     53.12
"sample"             0.06    18.75       0.23     71.88
"mean"               0.04    12.50       0.06     18.75
"length"             0.02     6.25       0.02      6.25
"FUN"                0.01     3.12       0.32    100.00
"mean.default"       0.01     3.12       0.02      6.25
"getInlineInfo"      0.01     3.12       0.01      3.12
"is.numeric"         0.01     3.12       0.01      3.12
"which"              0.01     3.12       0.01      3.12

$by.total
                      total.time total.pct self.time self.pct
"FUN"                       0.32    100.00      0.01     3.12
"block_exec"                0.32    100.00      0.00     0.00
"call_block"                0.32    100.00      0.00     0.00
"doTryCatch"                0.32    100.00      0.00     0.00
"eval"                      0.32    100.00      0.00     0.00
"evaluate_call"             0.32    100.00      0.00     0.00
"evaluate::evaluate"        0.32    100.00      0.00     0.00
"evaluate"                  0.32    100.00      0.00     0.00
"handle"                    0.32    100.00      0.00     0.00
"in_dir"                    0.32    100.00      0.00     0.00
"knitr::knit"               0.32    100.00      0.00     0.00
"process_file"              0.32    100.00      0.00     0.00
"process_group.block"       0.32    100.00      0.00     0.00
"process_group"             0.32    100.00      0.00     0.00
"rmarkdown::render"         0.32    100.00      0.00     0.00
"timing_fn"                 0.32    100.00      0.00     0.00
"try"                       0.32    100.00      0.00     0.00
"tryCatch"                  0.32    100.00      0.00     0.00
"tryCatchList"              0.32    100.00      0.00     0.00
"tryCatchOne"               0.32    100.00      0.00     0.00
"withCallingHandlers"       0.32    100.00      0.00     0.00
"withVisible"               0.32    100.00      0.00     0.00
"lapply"                    0.31     96.88      0.00     0.00
"tapply"                    0.31     96.88      0.00     0.00
"sample"                    0.23     71.88      0.06    18.75
```

```
"sample.int"                0.17     53.12      0.15    46.88
"mean"                      0.06     18.75      0.04    12.50
"length"                    0.02      6.25      0.02     6.25
"mean.default"              0.02      6.25      0.01     3.12
"getInlineInfo"             0.01      3.12      0.01     3.12
"is.numeric"                0.01      3.12      0.01     3.12
"which"                     0.01      3.12      0.01     3.12
"apply"                     0.01      3.12      0.00     0.00
"cmp"                       0.01      3.12      0.00     0.00
"cmpCall"                   0.01      3.12      0.00     0.00
"cmpForBody"                0.01      3.12      0.00     0.00
"cmpfun"                    0.01      3.12      0.00     0.00
"cmpSymbolAssign"           0.01      3.12      0.00     0.00
"compiler:::tryCmpfun"      0.01      3.12      0.00     0.00
"genCode"                   0.01      3.12      0.00     0.00
"h"                         0.01      3.12      0.00     0.00
"quantile.default"          0.01      3.12      0.00     0.00
"tryInline"                 0.01      3.12      0.00     0.00

$sample.interval
[1] 0.01

$sampling.time
[1] 0.32
```

Well, that was faster overall. We can now extend the concept and try the same thing for the tapply loop. This time, we have to create a matrix to hold the results.

```r
df <- read.csv("ctd.csv", stringsAsFactors = FALSE)

# Start profiling
Rprof(interval = 0.01)

# Run code
df.1 <- df[df$depth == 1, ]
temp <- df.1$temp
station <- df.1$station
st.names <- unique(station)

# an empty result vector
boot.mean <- matrix(nrow = length(st.names), ncol = 1000)
rownames(boot.mean) <- st.names

for(st in st.names) {
  x <- temp[station == st]
  num.temp <- length(x)
  for(i in 1:ncol(boot.mean)) {
    st.temp <- sample(x, num.temp, replace = TRUE)
    boot.mean[st, i] <- mean(st.temp)
  }
}
ci <- apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))

# Stop profiling
Rprof(NULL)
```

```r
# Examine profile summary
summaryRprof()
```

```
$by.self
               self.time self.pct total.time total.pct
"mean"              0.10    27.03       0.13     35.14
"sample.int"        0.09    24.32       0.09     24.32
"eval"              0.07    18.92       0.37    100.00
"sample"            0.06    16.22       0.15     40.54
"mean.default"      0.03     8.11       0.03      8.11
"grep"              0.01     2.70       0.01      2.70
"sort.int"          0.01     2.70       0.01      2.70

$by.total
                       total.time total.pct self.time self.pct
"eval"                       0.37    100.00      0.07    18.92
"block_exec"                 0.37    100.00      0.00     0.00
"call_block"                 0.37    100.00      0.00     0.00
"doTryCatch"                 0.37    100.00      0.00     0.00
"evaluate_call"              0.37    100.00      0.00     0.00
"evaluate::evaluate"         0.37    100.00      0.00     0.00
"evaluate"                   0.37    100.00      0.00     0.00
"handle"                     0.37    100.00      0.00     0.00
"in_dir"                     0.37    100.00      0.00     0.00
"knitr::knit"                0.37    100.00      0.00     0.00
"process_file"               0.37    100.00      0.00     0.00
"process_group.block"        0.37    100.00      0.00     0.00
"process_group"              0.37    100.00      0.00     0.00
"rmarkdown::render"          0.37    100.00      0.00     0.00
"timing_fn"                  0.37    100.00      0.00     0.00
"try"                        0.37    100.00      0.00     0.00
"tryCatch"                   0.37    100.00      0.00     0.00
"tryCatchList"               0.37    100.00      0.00     0.00
"tryCatchOne"                0.37    100.00      0.00     0.00
"withCallingHandlers"        0.37    100.00      0.00     0.00
"withVisible"                0.37    100.00      0.00     0.00
"sample"                     0.15     40.54      0.06    16.22
"mean"                       0.13     35.14      0.10    27.03
"sample.int"                 0.09     24.32      0.09    24.32
"mean.default"               0.03      8.11      0.03     8.11
"grep"                       0.01      2.70      0.01     2.70
"sort.int"                   0.01      2.70      0.01     2.70
"apply"                      0.01      2.70      0.00     0.00
"cb$putconst"                0.01      2.70      0.00     0.00
"cmp"                        0.01      2.70      0.00     0.00
"cmpCall"                    0.01      2.70      0.00     0.00
"cmpCallArgs"                0.01      2.70      0.00     0.00
"cmpCallSymFun"              0.01      2.70      0.00     0.00
"cmpForBody"                 0.01      2.70      0.00     0.00
"cmpSym"                     0.01      2.70      0.00     0.00
"cmpSymbolAssign"            0.01      2.70      0.00     0.00
"compile"                    0.01      2.70      0.00     0.00
"compiler:::tryCompile"      0.01      2.70      0.00     0.00
```

```
"FUN"                       0.01      2.70      0.00      0.00
"genCode"                   0.01      2.70      0.00      0.00
"h"                         0.01      2.70      0.00      0.00
"is.ddsym"                  0.01      2.70      0.00      0.00
"quantile.default"          0.01      2.70      0.00      0.00
"sort.default"              0.01      2.70      0.00      0.00
"sort"                      0.01      2.70      0.00      0.00
"tryInline"                 0.01      2.70      0.00      0.00

$sample.interval
[1] 0.01

$sampling.time
[1] 0.37
```

That pre-allocation speeds us up even more. The only other things we can do is use the function `sample.int` directly rather than through `sample`, and use the internal function `.Internal(mean())` rather than the generic `mean`:

```
df <- read.csv("ctd.csv", stringsAsFactors = FALSE)

# Start profiling
Rprof(interval = 0.01)

# Run code
df.1 <- df[df$depth == 1, ]
temp <- df.1$temp
station <- df.1$station
st.names <- unique(station)

# an empty result vector
boot.mean <- matrix(nrow = length(st.names), ncol = 1000)
rownames(boot.mean) <- st.names

for(st in st.names) {
  x <- temp[station == st]
  num.temp <- length(x)
  for(i in 1:ncol(boot.mean)) {
    j <- sample.int(1:num.temp, num.temp, replace = TRUE)
    boot.mean[st, i] <- .Internal(mean(x[j]))
  }
}
ci <- apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))

# Stop profiling
Rprof(NULL)

# Examine profile summary
summaryRprof()
```

```
$by.self
             self.time self.pct total.time total.pct
"sample.int"      0.15    65.22       0.15     65.22
"eval"            0.06    26.09       0.22     95.65
"formatC"         0.01     4.35       0.01      4.35
```

```
"match.call"      0.01     4.35       0.01      4.35

$by.total
                     total.time total.pct self.time self.pct
"block_exec"               0.23    100.00      0.00     0.00
"call_block"               0.23    100.00      0.00     0.00
"evaluate_call"            0.23    100.00      0.00     0.00
"evaluate::evaluate"       0.23    100.00      0.00     0.00
"evaluate"                 0.23    100.00      0.00     0.00
"in_dir"                   0.23    100.00      0.00     0.00
"knitr::knit"              0.23    100.00      0.00     0.00
"process_file"             0.23    100.00      0.00     0.00
"process_group.block"      0.23    100.00      0.00     0.00
"process_group"            0.23    100.00      0.00     0.00
"rmarkdown::render"        0.23    100.00      0.00     0.00
"withCallingHandlers"      0.23    100.00      0.00     0.00
"eval"                     0.22     95.65      0.06    26.09
"doTryCatch"               0.22     95.65      0.00     0.00
"handle"                   0.22     95.65      0.00     0.00
"timing_fn"                0.22     95.65      0.00     0.00
"try"                      0.22     95.65      0.00     0.00
"tryCatch"                 0.22     95.65      0.00     0.00
"tryCatchList"             0.22     95.65      0.00     0.00
"tryCatchOne"              0.22     95.65      0.00     0.00
"withVisible"              0.22     95.65      0.00     0.00
"sample.int"               0.15     65.22      0.15    65.22
"formatC"                  0.01      4.35      0.01     4.35
"match.call"               0.01      4.35      0.01     4.35
"apply"                    0.01      4.35      0.00     0.00
"format_perc"              0.01      4.35      0.00     0.00
"FUN"                      0.01      4.35      0.00     0.00
"paste0"                   0.01      4.35      0.00     0.00
"quantile.default"         0.01      4.35      0.00     0.00
"set_hooks"                0.01      4.35      0.00     0.00
"stopifnot"                0.01      4.35      0.00     0.00

$sample.interval
[1] 0.01

$sampling.time
[1] 0.23
```

Using `.Internal` functions can be tricky because the code base of internal functions can change over time. Also, CRAN will not permit packages using `.Internal` to be submitted.

Benchmarking

We've seen incremental achievements in our code, but it would be good to know how much better one version of code is than another. There are a couple of ways to do this. The first is to use `system.time` to record the CPU time required for a set of expressions to execute. For our comparison, let's make three functions that represent our code at different stages and see how long each actually takes:

```
# the first one
bootMean.1 <- function(fname, nrep) {
```

```
  boot.mean <- sapply(1:nrep, function(i) {
    df <- read.csv(fname, stringsAsFactors = FALSE)
    sapply(unique(df$station), function(st) {
      i <- which(df$station == st & df$depth == 1)
      i <- sample(i, length(i), replace = TRUE)
      mean(df$temp[i])
    })
  })
  apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))
}


# halfway through optimizing, using two sapply functions
bootMean.2 <- function(fname, nrep) {
  df <- read.csv(fname, stringsAsFactors = FALSE)
  boot.mean <- sapply(1:nrep, function(i) {
    sapply(unique(df$station), function(st) {
      i <- which(df$station == st & df$depth == 1)
      i <- sample(i, length(i), replace = TRUE)
      mean(df$temp[i])
    })
  })
  apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))
}

# fully optimized, using for-loops
bootMean.3 <- function(fname, nrep) {
  df <- read.csv(fname, stringsAsFactors = FALSE)
  df <- df[df$depth == 1, ]
  temp <- df$temp
  station <- df$station
  st.names <- unique(station)

  boot.mean <- matrix(nrow = length(st.names), ncol = nrep)
  rownames(boot.mean) <- st.names

  for(st in st.names) {
    x <- temp[station == st]
    num.temp <- length(x)
    for(i in 1:ncol(boot.mean)) {
      j <- sample.int(1:num.temp, num.temp, replace = TRUE)
      boot.mean[st, i] <- .Internal(mean(x[j]))
    }
  }
  apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))
}

# CPU time for each function for 10 replicates
system.time(bootMean.1("ctd.csv", 10))
```

```
   user  system elapsed
  6.456   0.184   6.660
```

```
system.time(bootMean.2("ctd.csv", 10))
```

```
   user  system elapsed
  1.030   0.096   1.127
```

```
system.time(bootMean.3("ctd.csv", 10))
```

```
   user  system elapsed
  0.557   0.007   0.565
```

It looks like we were able to make it about 10 times faster. Another useful way to do this is to use the `microbenchmark` package, which will execute each expression a number of times and give a distribution of the run times.

```
library(microbenchmark)
microbenchmark(
  bootMean.1 = bootMean.1("ctd.csv", 10),
  bootMean.2 = bootMean.2("ctd.csv", 10),
  bootMean.3 = bootMean.3("ctd.csv", 10),
  times = 10
)
```

```
Unit: milliseconds
       expr       min        lq      mean    median        uq       max
 bootMean.1 6025.3251 6511.3441 6563.9109 6614.6252 6715.7502 6939.5958
 bootMean.2 1013.1163 1025.1814 1072.2907 1067.0896 1115.6412 1128.7739
 bootMean.3  535.1911  551.2707  578.7207  573.5427  615.2071  631.6029
 neval
    10
    10
    10
```

Parallel computing

Once we have code that is optimized as much as possible in R, if we need it to execute faster, we can explore distributing the processing among several CPUs if we have access to a computer that has multiple cores. This is only good if we have processes that are independent, such that if one is running on one core, it does not need to transfer information to a process on another core. This is good for simulations, permutations, bootstrapping, and other similar loop-based operations. There is a bit of overhead in setting up the monitoring of the parallel processes, which takes some time, so make sure that this will be made up for in the time it takes to run processes in parallel rather than serially.

We'll use the `tapply` version of our example and structure the code to spread the replicates for a station among 2 cores. We'll use the `parallel` package.

```
library(parallel)

bootMean.tapply <- function(fname, nrep) {
  df <- read.csv(fname, stringsAsFactors = FALSE)
  df <- df[df$depth == 1, ]
  temp <- df$temp
  station <- df$station
  st.names <- unique(station)

  boot.mean <- tapply(temp, station, function(x) {
    sapply(1:nrep, function(i) {
```

```
      st.temp <- sample(x, length(x), replace = TRUE)
      mean(st.temp)
    })
  })
  boot.mean <- do.call(rbind, boot.mean)
  apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))
}

bootMean.par <- function(fname, nrep) {
  df <- read.csv(fname, stringsAsFactors = FALSE)
  df <- df[df$depth == 1, ]
  temp <- df$temp
  station <- df$station
  st.names <- unique(station)

  # make 2 clusters
  cl <- makeCluster(2)

  boot.mean <- tapply(temp, station, function(x) {
    clusterExport(cl, "x")
    parSapply(cl, 1:nrep, function(i) {
      st.temp <- sample(x, length(x), replace = TRUE)
      mean(st.temp)
    })
  })
  stopCluster(cl)
  boot.mean <- do.call(rbind, boot.mean)
  apply(boot.mean, 1, quantile, probs = c(0.025, 0.975))
}
```

Here's the result for 100 replicates:

```
system.time(bootMean.tapply("ctd.csv", 100))
```

```
   user  system elapsed
  0.635   0.010   0.649
```

```
system.time(bootMean.par("ctd.csv", 100))
```

```
   user  system elapsed
  0.767   0.052   1.299
```

Here's the result for 1000 replicates:

```
system.time(bootMean.tapply("ctd.csv", 1000))
```

```
   user  system elapsed
  0.984   0.022   1.010
```

```
system.time(bootMean.par("ctd.csv", 1000))
```

```
   user  system elapsed
  0.789   0.057   1.471
```

Here's the result for 10,000 replicates:

```
system.time(bootMean.tapply("ctd.csv", 10000))
```

```
   user  system elapsed
  4.748   0.115   4.880
```

```
system.time(bootMean.par("ctd.csv", 10000))
```

```
   user  system elapsed
  1.016   0.076   3.498
```

You can see that at 100 replicates, the parallel version takes more time than the non-parallel version. However, as the number of replicates increases, the parallel version gets much faster. You can see that it would be very useful to parallelize code if each replicate was even moderately time-intensive.