Study of apply family functions of R

Amitsingh Pardeshi

September 28, 2017 , Assignment 1

1 Repository

Please find the link to the git repository- Project Link

2 mapply

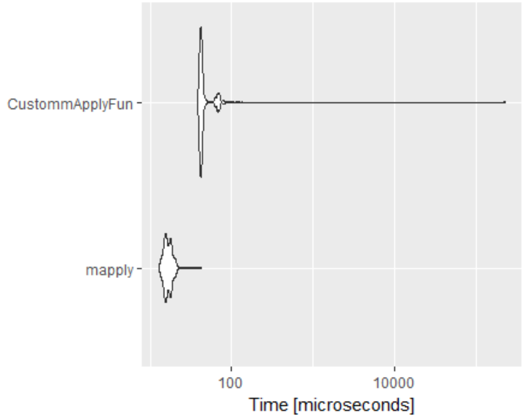
- mapply is a multivariate version of sapply.
- mapply() applies a Function to Multiple List or multiple Vector Arguments
- eg. mapply(sum,1:4,1:4), it takes both the input vectors and added them elementwise to create resultant vector
- Note - the length of both the input vector should be same. if not, then the length of larger vector should be multiple of length of smaller one. if the length of both the input vector does not satisfies above conditions then mapply throws an error message.

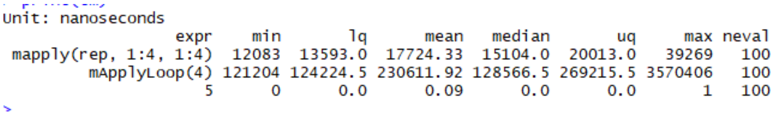
2.1 Code Snippet

```
#calculate sum using mapply
mapply(sum,1:2 ,1:4)

# cusotm functions takes two vectors and simulate the mapply for sum function
mApplySum <- function(x,y){

    #calculates the length of input vectors
    lengthX <- length(x)
    lengthY <- length(y)
    result <- c()
    if(lengthX > lengthY){
    # checks if the length of vectors are
    #multiples of each other else throw error
    # and stop the execution
```

```
   if(lengthX %% lengthY  != 0){
        stop(”In mApplySum(x,y):
longer argument not a multiple of length of shorter”)
        }
   }else if(lengthX < lengthY){
   #checks if the length of vectors are
   #multiples of each other else throw error
   #and stop the execution

        if(lengthY %% lengthX  != 0){
         stop(”In mApplySum(x,y):
longer argument not a multiple of length of shorter”)
        }
   }

   if(lengthX > lengthY){
        for(i in 1: lengthX){
         result[i] <- if(i<= lengthY) sum(x[i],y[i]) else sum(x[i],y[i/lengthY])
        }
   }else if(lengthX < lengthY){
        for(i in 1: lengthY){
         result[i] <- if(i<= lengthX) sum(y[i],x[i]) else sum(y[i],x[i/lengthX])
        }
   }else{
        for(i in 1: lengthX){
         result[i] <- sum(x[i],y[i])
        }
   }
   print(result)
}

#benchmark sum of two vectors using mapply and custom mApplySum functions

sumResult <- microbenchmark(mapply= mapply(sum, 1:4, 1:4),
 CustommApplyFun =  mApplySum(1:4, 1:4),
 times = 1000L)

##print(sumResult)

# plot the results in the graph
autoplot(sumResult)
```

2.2 Generated Output





```
Unit: nanoseconds
                 expr    min       lq      mean   median       uq     max neval
 mapply(rep, 1:4, 1:4)  12083  13593.0  17724.33  15104.0  20013.0   39269   100
         mApplyLoop(4) 121204 124224.5 230611.92 128566.5 269215.5 3570406   100
                     5      0      0.0      0.09      0.0      0.0       1   100
>
```

Figure 1: m apply plot and benchmark results

2.3 Observations

As per the microbenchmarking results above, we can say that mapply library function performs better than its custom implementation.

3 rapply

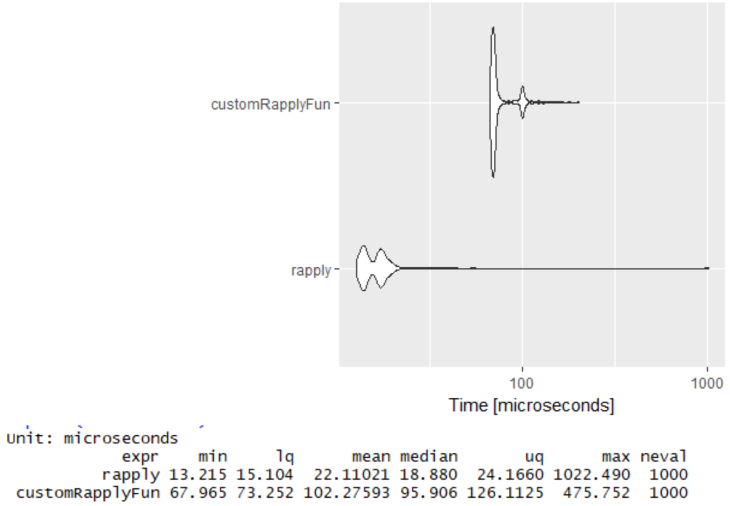
- rapply stands for recursive apply.
- It is used to apply a function to all elements of a list recursively.

3.1 Code Snippet

```
# import libraries
library(microbenchmark)
library(ggplot2)

# creating list containing three sublists
data1 <- list(10,20,30)
data2 <- list(100,200,300)
data3 <- list(1000,2000,3000)
data <- list(data1,data2,data3)
```

```
# custom function which simulates the rapply behaviour
#for doubling the each elements of the list and
# printing result vector

rApplyLoop <- function(x){
  result <- c()
  for(i in 1: length(x)){
    x1 <- x[[i]]
    for(j in 1:length(x1)){
      x2 <- x1[[j]]
      if(is.numeric(x2)){
        result <- c(result, x2*2)
      }

    }
  }
  print(result)
}
# rapply for doubling each elements of the input list recursively
rapply(data,function(x) x*2,class=c(”numeric”))
# custome function to achieve the same
rApplyLoop(data)

#benchmark sum of two vectors using mapply and custom mApplySum functions
doubleResult <- microbenchmark(rapply = rapply(data,function(x) x*2,class=c(”num
 customRapplyFun = rApplyLoop(data),
 times = 1000L)

##print(doubleResult)

# plot the results in the graph
autoplot(doubleResult)
```

3.2 Generated Output



```
Unit: microseconds
            expr    min     lq      mean median       uq      max neval
          rapply 13.215 15.104  22.11021 18.880  24.1660 1022.490  1000
 customRapplyFun 67.965 73.252 102.27593 95.906 126.1125  475.752  1000
```

Figure 2: r apply plot and benchmark results

3.3 Observations

As per the microbenchmarking results below, we can say that rapply library function performs better than its custom implementation.

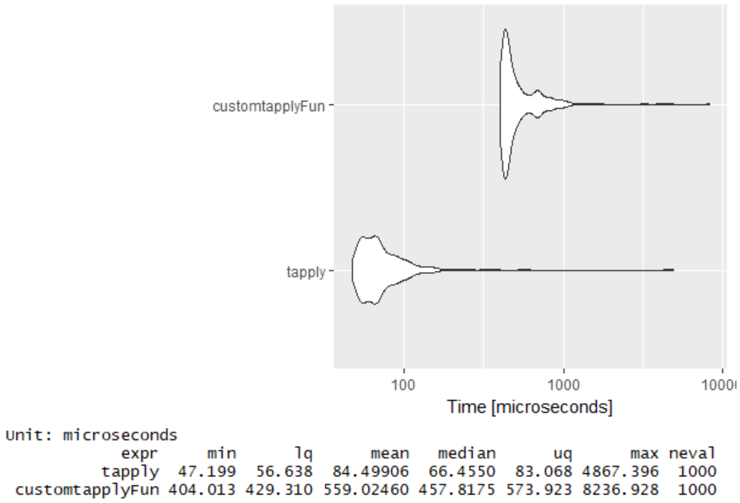
4 tapply

- tapply is used to apply the function on the dataset using group by of unique combinations of the factors.
- This tapply function is used in cases where dataset needs to be broken into groups
- And then apply functions within each group.

4.1 Code Snippet

```
# import libraries
library(microbenchmark)
library(ggplot2)

# creating dataframe of baseball teams
baseball.example <-
  data.frame(team = gl(5, 5,labels = paste(”Team”, LETTERS[1:5])),
            player = sample(letters, 25),
            batting.average = runif(25, .200, .400))

# custom function to simulate tapply
tApplyLoop <- function(x){
  if(!is.data.frame(x)){
    stop(”x is expected to be datframe”)
  }
  out <- split( x , f = baseball.example$team)
  for(i in 1:length(out)){
    out2 <- out[[i]]
    print(max(out2[[3]]))
  }
}


# tapply for finding max batting avg group by team
tapply(baseball.example$batting.average, baseball.example$team,max)

# custome function to achieve the same
tApplyLoop(baseball.example)

#benchmark sum of two vectors using mapply and custom mApplySum functions

tApplyResult <- microbenchmark(tapply=tapply(baseball.example$batting.average,
 baseball.example$team,max),customtapplyFun = tApplyLoop(baseball.example),
 times = 1000L)

print(tApplyResult)

# plot the results in the graph
autoplot(tApplyResult)
```

4.2 Generated Output



```
Unit: microseconds
            expr     min      lq      mean    median      uq      max neval
          tapply  47.199  56.638  84.49906  66.4550  83.068 4867.396  1000
 customtapplyFun 404.013 429.310 559.02460 457.8175 573.923 8236.928  1000
```

Figure 3: t apply plot and benchmark results

4.3 Observations

As per the microbenchmarking results below, we can say that tapply library function performs better than its custom implementation.

5 eapply

- eapply applies FUN to the named values from an environment and returns the results as a list.

6 vapply

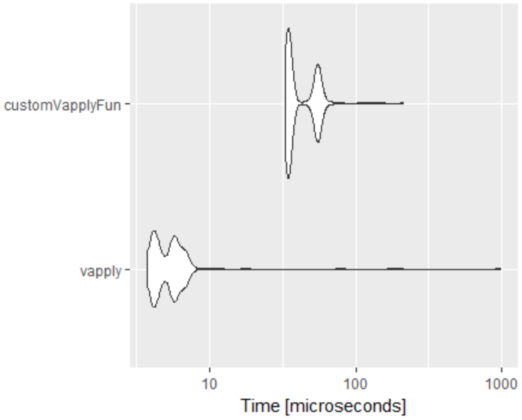
- vapply is similar to sapply having below two advantages.
- execution time is little better than sapply.
- improves consistency by providing limited return type checks. As it helps catch errors before they happen.

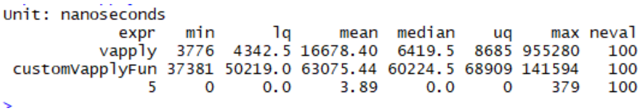
6.1 Code Snippet

```
# import libraries
library(microbenchmark)
library(ggplot2)
```

```
# creating list containing three sublists
data <- list(10,20,30)

vApplyLoop <- function(x){
  v <- double(0)
  for(i in 1: length(x)){
    v <- c(v, x[[i]]*2)
  }
  print(v)
}


# vapply for doubling each elements of the input list
#and expects double of length 1 as output

vapply(data,function(x) x*2,FUN.VALUE = double(1))

# custom function to achieve the same
vApplyLoop(data)

#benchmark sum of two vectors using mapply and custom mApplySum functions

vApplyResult <- microbenchmark(vapply=vapply(data,function(x) x*2,FUN.VALUE = do
 customVapplyFun = vApplyLoop(data),
 times = 1000L)

print(vApplyResult)

# plot the results in the graph
autoplot(vApplyResult)
```

6.2 Generated Output





```
Unit: nanoseconds
            expr   min      lq     mean  median    uq    max neval
          vapply  3776  4342.5 16678.40  6419.5  8685 955280   100
 customVapplyFun 37381 50219.0 63075.44 60224.5 68909 141594   100
               5     0     0.0     3.89     0.0     0    379   100
>
```

Figure 4: v apply plot and benchmark results

6.3 Observations

As per the microbenchmarking results below, we can say that vapply library function performs better than its custom implementation.

7 References

R documentation