Rhpc: An **R** package for High Performance Computing

Ei-ji NAKAMA [†] Junji NAKANO[‡]

[†] COM-ONE Ltd., Japan

[‡] The Institute of Statistical Mathematics, Japan

SC13
17-22 November 2013
Denver, Colorado, USA

Outline

Outline

R is a widely used free software environment for statistical computing and graphics. Recently, high-performance computing (HPC) using **R** easily and efficiently is strongly required. To realize it in a better way, we provide a new **R** package for efficient computing

Existing parallel environments of **R** for HPC

- **snow**
  The **snow** (Simple Network of Workstations) package by Tierney et al. can use PVM, MPI, NWS as well as direct networking sockets. As it is implemented mainly in **R** language, it has some inefficiency.
- **Rmpi**
  The **Rmpi** package offers access to numerous functions of MPI API, and a number of **R**-specific extensions
- **multicore**
  The **multicore** package provides a way of running parallel computations in **R** on one machine with multiple cores by using operating system functions.

Objectives of **Rhpc**

- Package **snow** is good for parallel computing by **R**, but for supercomputers, it is
  - sometimes inefficient
  - unable to handle huge data set
- We want to have a similar package as **snow** for supercomputers, which has
  - efficient use of MPI
  - ability to move huge data set
- We hope that **Rhpc** $>$ **snow** $+$ **Rmpi**

Outline

Outline of **Rhpc**

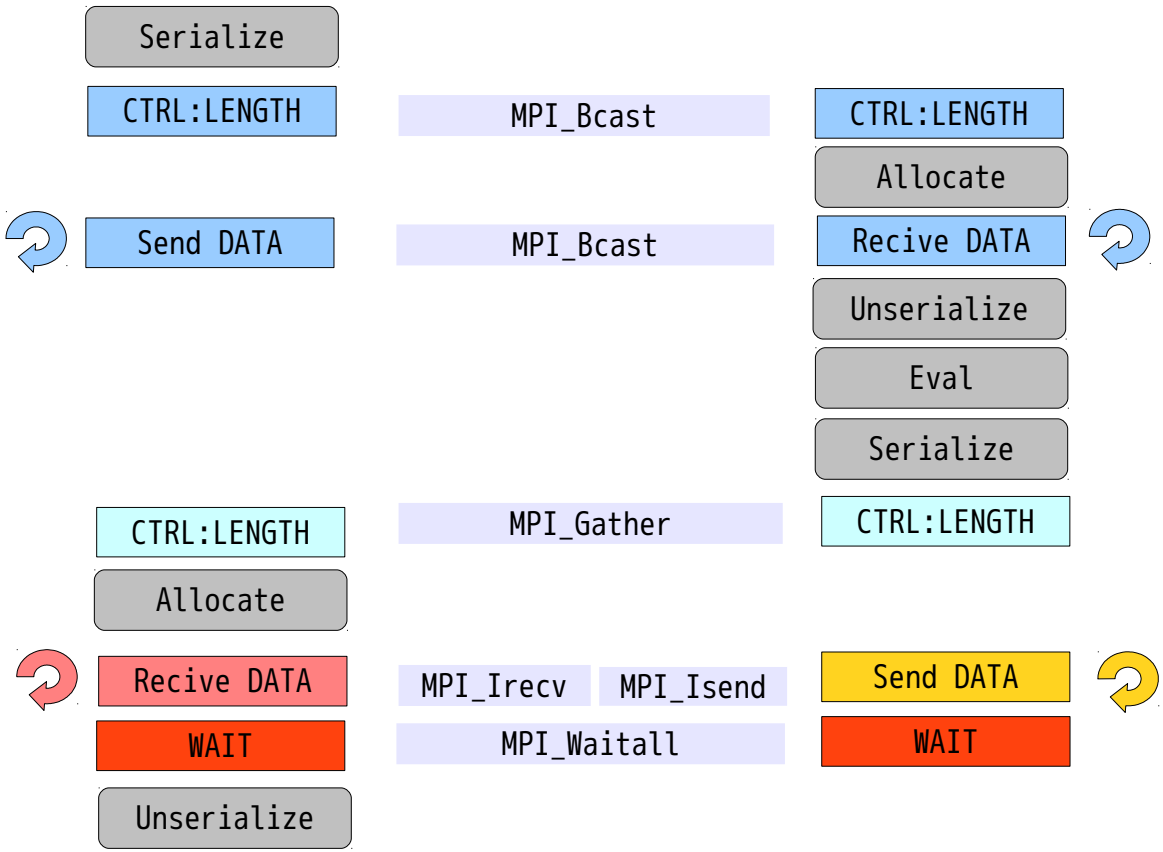
- **snow** can use three low level mechanisms for creating a virtual connection between processes: Socket, PVM (Parallel Virtual Machine) and MPI (Message Passing Interface). In **Rhpc** , we focus on MPI without using **Rmpi** and use collective communication as much as possible
- Worker process is written by Embedding **R**
- Main functions are:
  - `Rhpc_worker_call` (∼ `snow::clusterCall`)
  - `Rhpc_lapply` (∼ `snow::clusterApply`)
  - `Rhpc_lapplyLB` (∼ `snow::clusterApplyLB`)
  - `Rhpc_getHandle` (∼ `snow::makeMPICluster`)

Rhpc_worker_call (1)

Rhpc_worker_call(cl, FUN, ...)

- cl
  pointer to communicator
- FUN
  Function name or string (string expresses function name)
  Distributed by collective communication
- ... (argument)
  Distributed by collective communication

Rhpc_worker_call (2)

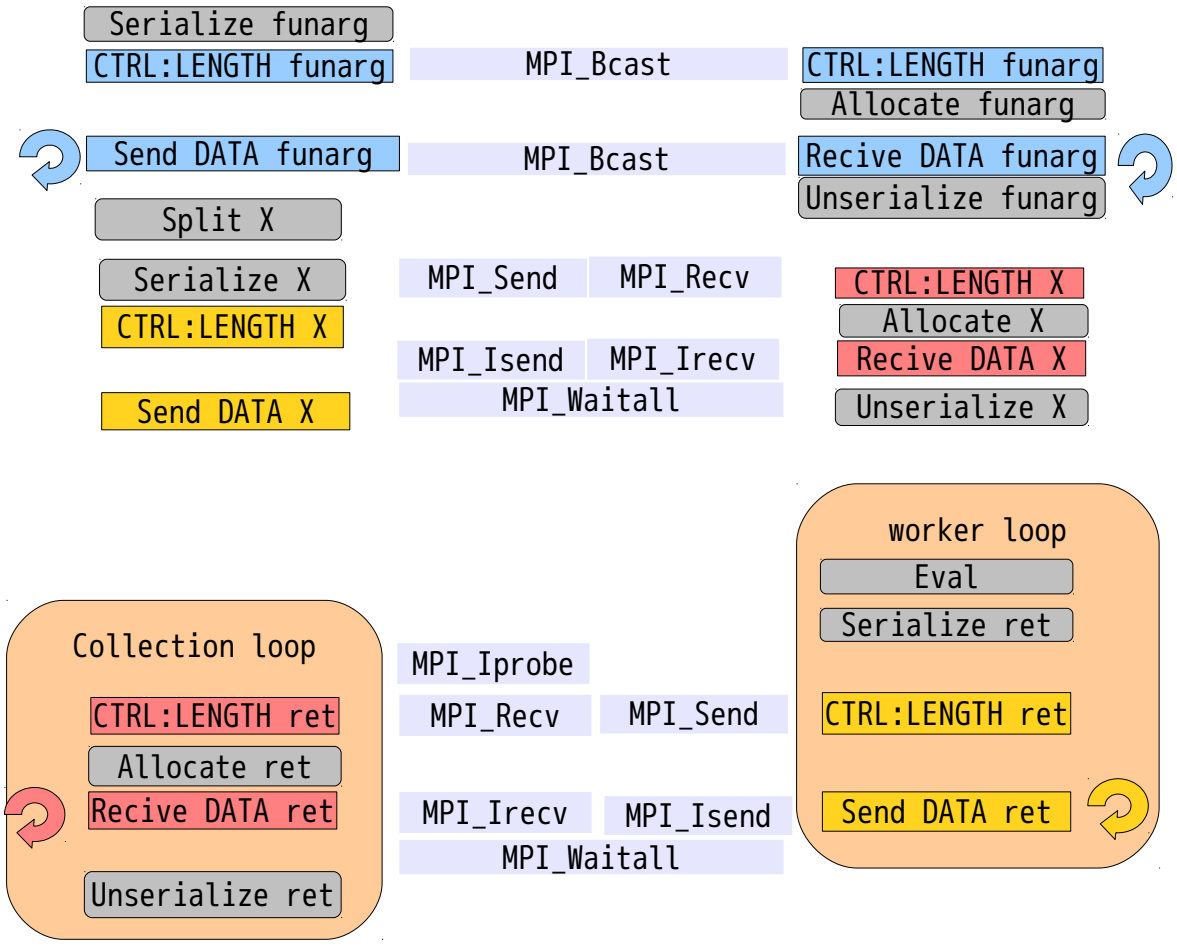


Rhpc_lapply (1)

Rhpc_lapply(cl, x, FUN, ...)

- cl
  pointer to communicator
- x
  vector or list.
  Divided into smaller vectors according to the number of workers, and distributed to workers when the function is first executed. One-sided communication is used asynchronously
- FUN
  Function name or string (string expresses function name)
  Distributed by collective communication at first, then they are not sent again
- ... (argument)
  Distributed by collective communication at first, then they are not sent again

Rhpc_lapply (2)

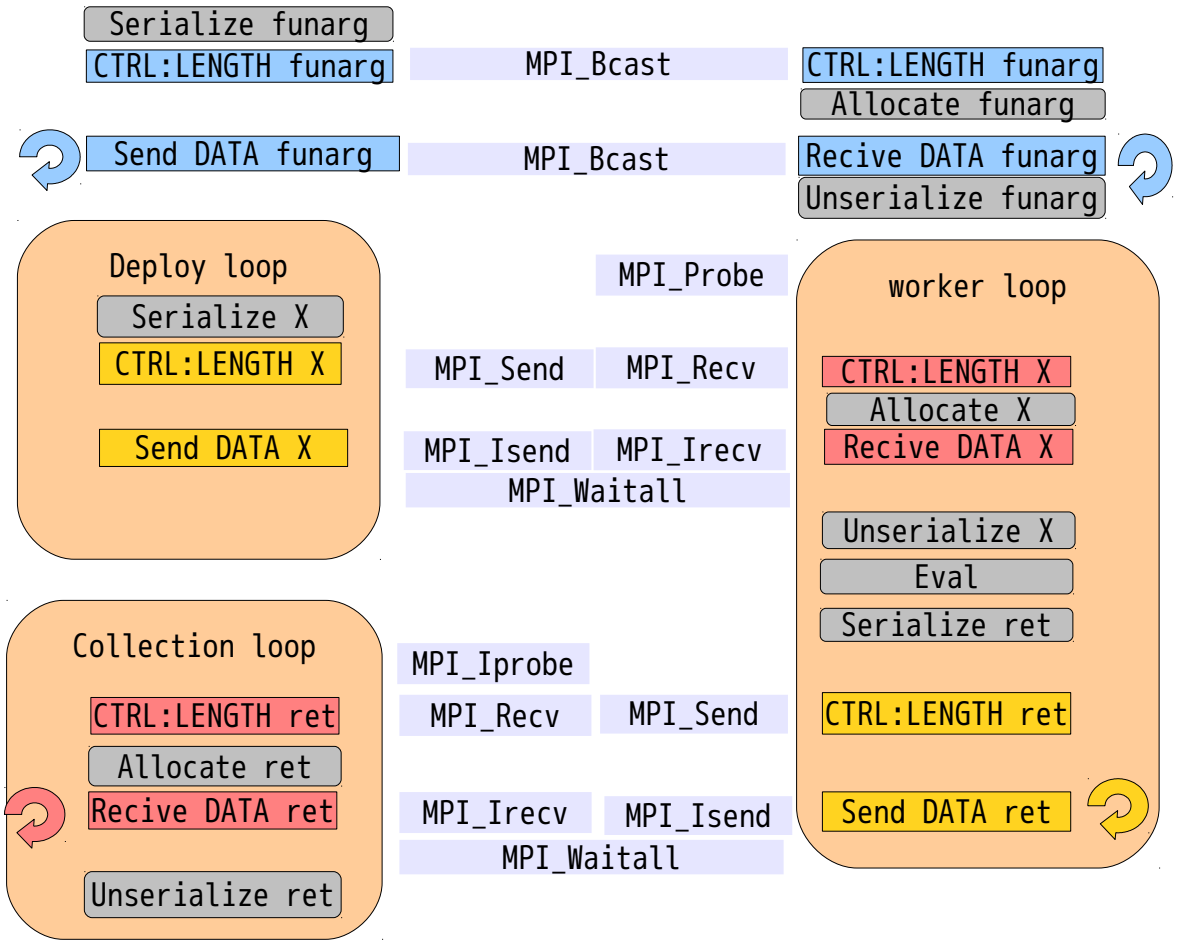


Rhpc_lapplyLB (1)

Rhpc_lapplyLB(cl, x, FUN, ...)

- cl
  pointer to communicator
- x
  vector or list.
  Distributed to workers when the function is executed. One-sided communication is used asynchronously
- FUN
  Function name or string (string expresses function name)
  Distributed by collective communication at first, then they are not sent again
- ... (argument)
  Distributed by collective communication at first, then they are not sent again

Rhpc_lapplyLB (2)



Miscellaneous functions

- Functions to initialize and finalize MPI, and get MPI communication handle
  - `Rhpc_initialize()`
  - `Rhpc_finalize()`
  - `Rhpc_getHandle([number_of_worker])`
- Functions to call workers
  - `Rhpc_Export(cl, names)`
  - `Rhpc_EvalQ(cl, expr)`
- Function to set up random number generators
  - `Rhpc_setupRNG(cl, seed)`

Optional packages (RhpcBLASctl) functions

- Functions to control the number of threads on BLAS, MKL, ACML and GotoBLAS etc.
  - `blas_get_num_procs()`
  - `blas_set_num_threads(threads)`
- Functions to control the number of threads on OpenMP
  - `omp_get_num_procs()`
  - `omp_get_max_threads()`
  - `omp_set_num_threads(threads)`

Outline

Huge data: **snow** (SOCK)

As **snow** (SOCK) utilizes pipe to serialize and unserialize data, it can handle huge data.

snow(SOCK)::clusterExport in one worker

```
> library(snow)
> cl<-makeCluster(1,type="SOCK")
> set.seed(123)
> N<-17e3
> M<-matrix(runif(N^2),N,N)
> sum(M)
[1] 144501466
> system.time(clusterExport(cl,"M"))
   user  system elapsed
  4.213   1.580   8.228
> f<-function()sum(M)
> clusterCall(cl,f)
[[1]]
[1] 144501466
```

However, socket mechanism has difficulty to control processes on many nodes by many users in supercomputer environment.

Huge data: **snow** (MPI)

At present, **Rmpi** cannot handle data more than 2GB, because the argument of `MPI_send` etc. should be `int` size.

`snow(MPI)::clusterExport` in one worker

```
> library(snow)
> cl<-makeCluster(1,type="MPI")
> set.seed(123)
> N<-17e3
> M<-matrix(runif(N^2),N,N)
> sum(M)
[1] 144501466
> system.time(clusterExport(cl,"M"))
Error in mpi.send(x = serialize(obj, NULL), type = 4, dest = dest, tag = tag,  :
  long vectors not supported yet: memory.c:3100
Calls: system.time ... sendData.MPInode -> mpi.send.Robj -> mpi.send -> .Call
```

Huge data: **Rhpc**

Rhpc can handle huge data, because it divides huge data into appropriate size and uses MPI many times.

Rhpc::Rhpc_Export in one worker

```
> library(Rhpc)
> Rhpc_initialize()
> cl<-Rhpc_getHandle()
> set.seed(123)
> N<-17e3
> M<-matrix(runif(N^2),N,N)
> sum(M)
[1] 144501466
> system.time(Rhpc_Export(cl,"M"))
   user  system elapsed
  9.241   1.700  10.972
> f<-function()sum(M)
> Rhpc_worker_call(cl,f)
[[1]]
[1] 144501466
> Rhpc_finalize()
```

Many workers: **snow** (MPI)

As `clusterCall` of **snow** starts workers sequentially, it becomes slow when the number of workers increases.

`snow(MPI)::clusterExport` in 63 workers

```
> library(Rmpi)
> library(snow)
> cl<-makeMPIcluster()
> set.seed(123)
> N<-4e3
> length(cl)
[1] 63
> M<-matrix(runif(N^2),N,N)
> system.time(clusterExport(cl,"M"))
   user  system elapsed
 26.715  10.903  37.761
> f<-function()sum(M)
> all.equal(rep(sum(M),length(cl)),unlist(clusterCall(cl,f)))
[1] TRUE
> stopCluster(cl)
```

Many workers: **Rhpc**

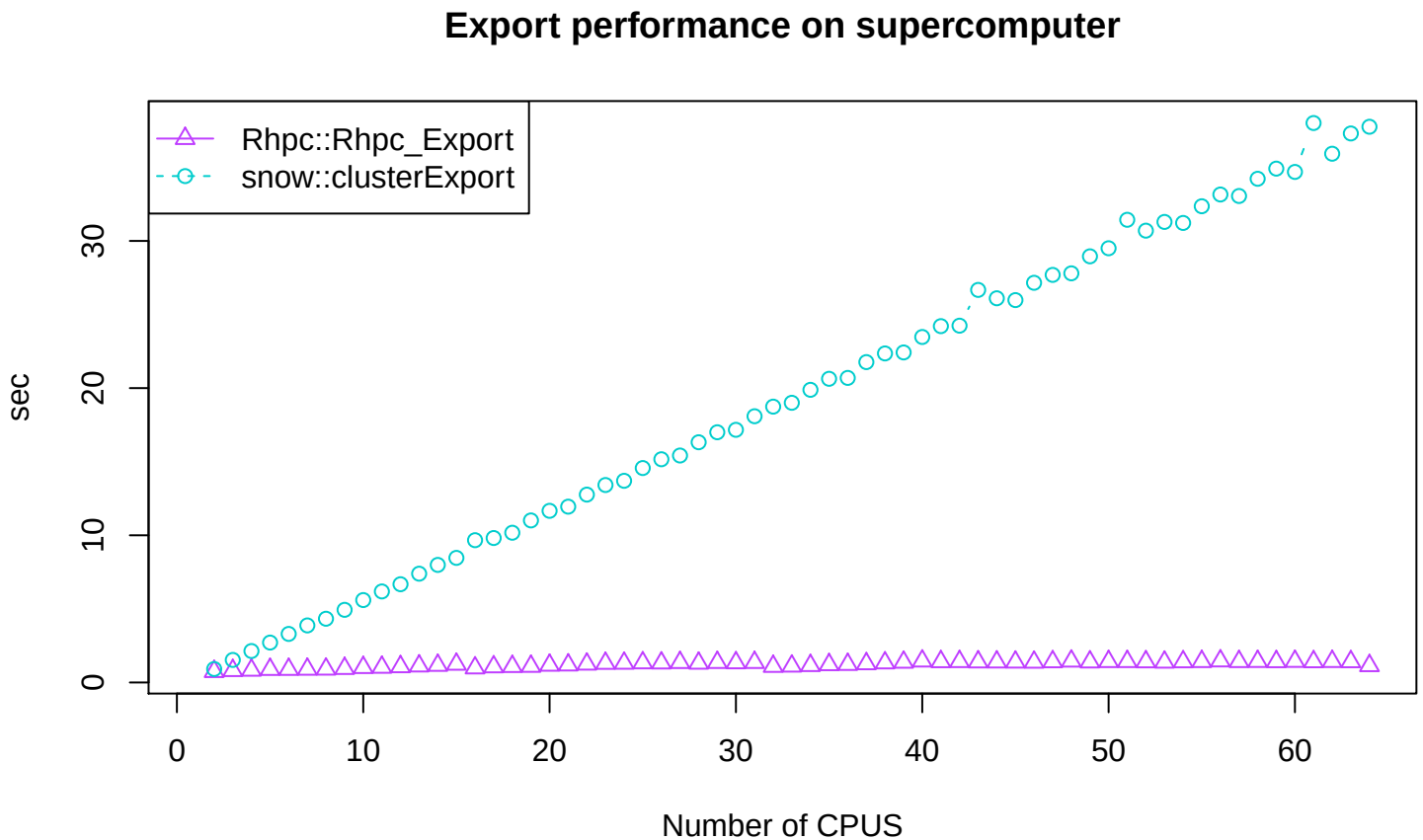
Rhpc::Rhpc_Export

As **Rhpc** uses collective communication by `MPI_Bcast`, data transportation to workers is still fast even when the number of workers increases.

`Rhpc::Rhpc_Export` in 63 workers

```
> library(Rhpc)
Loading required package: rlecuyer
> Rhpc_initialize()
> cl<-Rhpc_getHandle()
Detected communication size 64
> set.seed(123)
> N<-4e3
> Rhpc_numberOfWorker(cl)
[1] 63
> M<-matrix(runif(N^2),N,N)
> system.time(Rhpc_Export(cl,"M"))
   user  system elapsed
  1.012   0.116   1.139
> f<-function()sum(M)
> all.equal(rep(sum(M),Rhpc_numberOfWorker(cl)),unlist(Rhpc_worker_call(cl,f)))
[1] TRUE
> Rhpc_finalize()
```

Many workers: Export by **Rhpc** and **snow** (MPI)



Many workers2: **snow** (MPI)

As the main parts of **snow** and **Rmpi** are written in **R** language, they are rather slow.

snow(MPI)::clusterApply in 63 workers

```
> library(Rmpi)
> library(snow)
> cl<-makeMPIcluster()
> system.time(ans<-clusterApply(cl,1:10000,sqrt))
   user  system elapsed
  1.423   0.005   1.429
> all.equal(sqrt(1:10000),unlist(ans))
[1] TRUE
> stopCluster(cl)
```

Many workers2: **snow** (MPI)

As the load balancing function is written in **R** language, it becomes slow according to the number of parallel workers.

snow(MPI)::clusterApplyLB in 63 workers

```
> library(Rmpi)
> library(snow)
> cl<-makeMPIcluster()
> system.time(ans<-clusterApplyLB(cl,1:10000,sqrt))
   user  system elapsed
  4.395   0.003   4.413
> all.equal(sqrt(1:10000),unlist(ans))
[1] TRUE
> stopCluster(cl)
```

Many workers2: **Rhpc**

Rhpc::Rhpc_lapply

As the main part of **Rhpc** is written in C language, it is efficient.

Rhpc::Rhpc_lapply in 63 workers

```
> library(Rhpc)
Loading required package: rlecuyer
> Rhpc_initialize()
> cl<-Rhpc_getHandle()
Detected communication size 64
> system.time(ans<-Rhpc_lapply(cl, 1:10000, sqrt))
   user  system elapsed
  0.045   0.001   0.046
> all.equal(sqrt(1:10000),unlist(ans))
[1] TRUE
> Rhpc_finalize()
```

Many workers2: Rhpc

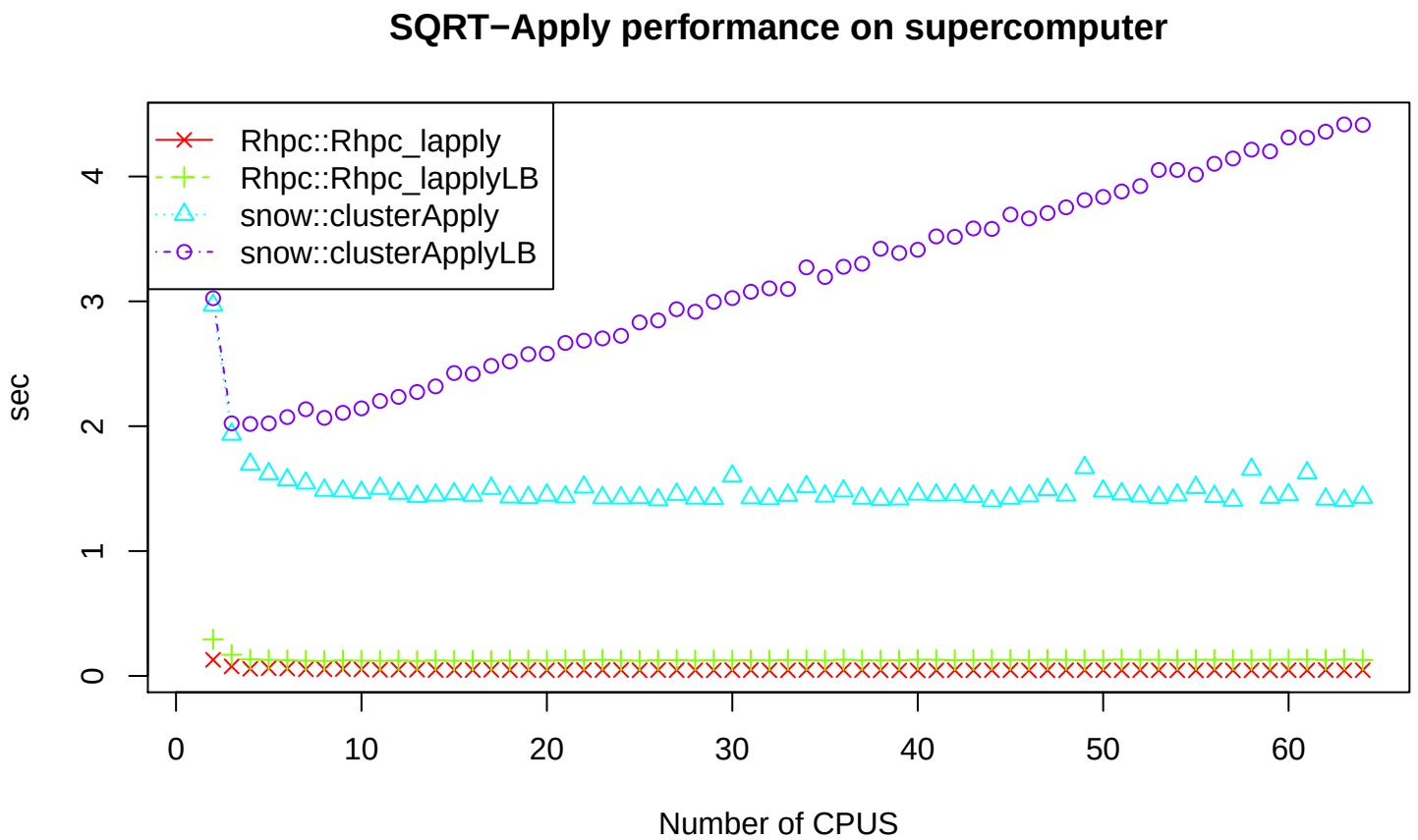
Rhpc::Rhpc_lapplyLB

Rhpc_lapplyLB sends the argument x to an available worker. So it produces a little delay.

Rhpc::Rhpc_lapplyLB in 63 workers

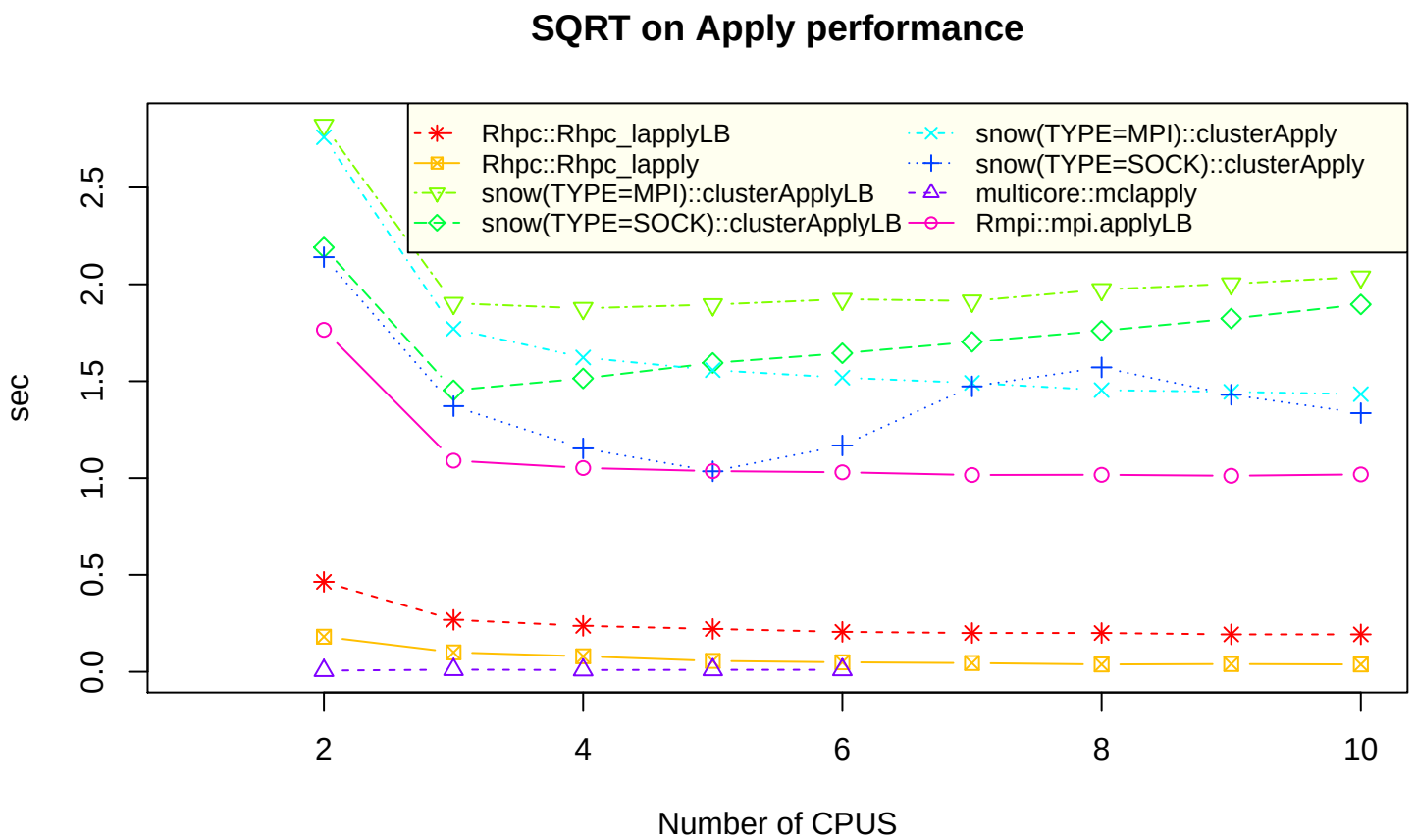
```
> library(Rhpc)
Loading required package: rlecuyer
> Rhpc_initialize()
> cl<-Rhpc_getHandle()
Detected communication size 64
> system.time(ans<-Rhpc_lapplyLB(cl, 1:10000, sqrt))
   user  system elapsed
  0.125   0.001   0.127
> all.equal(sqrt(1:10000),unlist(ans))
[1] TRUE
> Rhpc_finalize()
```

Many workers2: SQRT by **Rhpc** and **snow** (MPI)



Many parallel apply functions

At present, **Rhpc** is a little slower than **multicore**.



Example: **TWIX** package

- **TWIX** is a binary-split decision tree algorithm for classification and data mining developed by Sergej Potapov, Martin Theus and Simon Urbanek.
- It has **snow** and **multicore** codes. Small changes are required for **Rhpc**.

```
-       clusterEvalQ(cluster, library(TWIX))
+       if(typeof(cluster)=="externalptr"){
+               Rhpc_EvalQ(cluster, library(TWIX))
+              BG <- Rhpc_lapply(cluster,1:nbagg,
+                              cluster_bagg, data=m, t.data=NULL, devmin=Devmin,
+                              Minsplit=minsplit, Minbucket=minbucket, topn=topN,
+                              Topn.method=topn.method, Level=level, Method=method,
+                              Tol=tol, Splitf=splitf, Ns=ns, repl=replace, combi=comb)
+              if(!is.null(comb)){
+                      add.BG <- lapply(BG,function(x) x$A)
+                      BG <- lapply(BG,function(x) x$B)
+              }
+       }else{
+               clusterEvalQ(cluster, library(TWIX))
```

Read data

We use "Wine Quality Data Set"
http://archive.ics.uci.edu/ml/datasets/Wine+Quality

```
> library(TWIX)
> URL <-  "http://archive.ics.uci.edu/ml/machine-learning-databases/"
> r<-read.csv(paste(URL,"wine-quality/winequality-red.csv",sep=""),sep=";")
> w<-read.csv(paste(URL,"wine-quality/winequality-white.csv",sep=""),sep=";")
> x<-rbind(r,w)
> x$quality=factor(x$quality)
> set.seed(123)
> i <- sample(nrow(x),nrow(x)/4)
> ic <- setdiff(1:nrow(x),i)
> training <- x[ic,]
> test <- x[i,]
> dim(x)
[1] 6497   12
```

Single process

```
> system.time(TWIX(quality~., data=training[,1:12],
+                  topN=c(12,12),method="local"))
n = 1685
Deviance gain and TIC of the best TWIX-tree:      645.3553 0.7482044
Deviance gain and TIC of the greedy tree(Nr.1116):  849.0834 0.7297353
   user  system elapsed
 50.779   0.064  50.979
```

Rhpc

```
> library(Rhpc)
> Rhpc_initialize()
> cl<-Rhpc_getHandle()
Detected communication size 6
> system.time(TWIX(quality~. ,data=training[,1:12],
+                  topN=c(12,12), method="local",
+                  cluster=cl))
Loading required package: rlecuyer
n = 1685
Deviance gain and TIC of the best TWIX-tree:      645.3553 0.7482044
Deviance gain and TIC of the greedy tree(Nr.1116):  849.0834 0.7297353
   user  system elapsed
 10.704   5.636  16.384
> Rhpc_finalize()
```

snow(SOCK)

```
> library(snow)
> cl<-makeSOCKcluster(rep("localhost",5))
> system.time(TWIX(quality~., data=training[,1:12],
+                  topN=c(12,12), method="local",
+                  cluster=cl))
n = 1685
Deviance gain and TIC of the best TWIX-tree:      645.3553 0.7482044
Deviance gain and TIC of the greedy tree(Nr.1116):  849.0834 0.7297353
   user  system elapsed
 12.408   0.312  17.769
> stopCluster(cl)
```

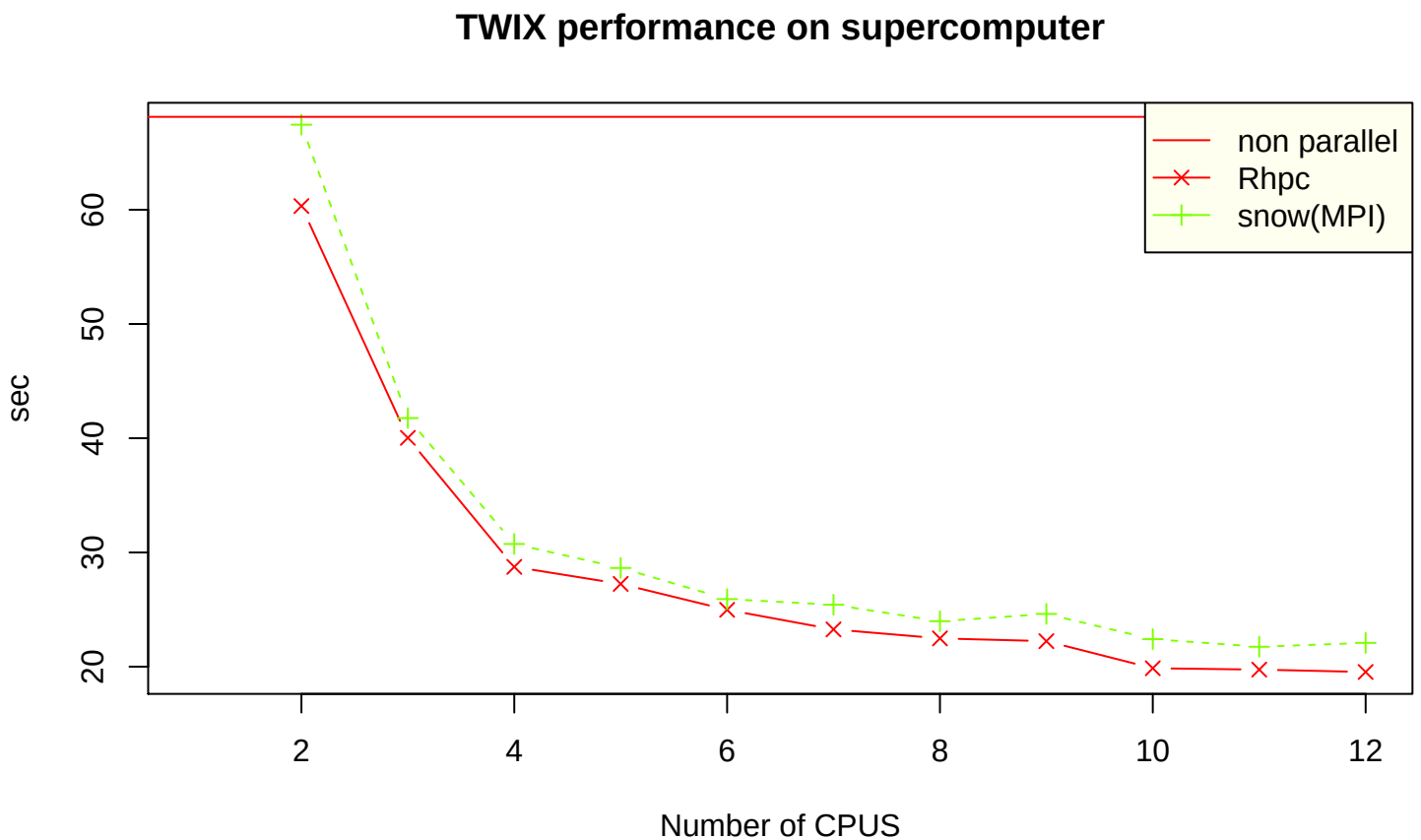
snow(MPI)

```
> library(snow)
> cl<-getMPIcluster()
> system.time(TWIX(quality~., data=training[,1:12],
+                  topN=c(12,12), method="local",
+                  cluster=cl))
n = 1685
Deviance gain and TIC of the best TWIX-tree:      645.3553 0.7482044
Deviance gain and TIC of the greedy tree(Nr.1116):  849.0834 0.7297353
   user  system elapsed
 13.992   6.172  23.038
```

multicore

```
> library(parallel)
> options("mc.cores"=5)
> system.time(TWIX(quality~. ,data=training[,1:12],
+                  topN=c(12,12), method="local",
+                  multicore=T))
n = 1685
Deviance gain and TIC of the best TWIX-tree:      645.3553 0.7482044
Deviance gain and TIC of the greedy tree(Nr.1116):  849.0834 0.7297353
   user  system elapsed
 53.046   1.228  20.968
```

TWIX performance



Pararrel BLAS control (optional package **RhpcBLASctl**)

Number of threads for parallel BLAS can be controlled to reserve cores for **Rhpc**.

pararrel BLAS control

```
library(RhpcBLASctl)
> A<-matrix(runif(3e3*3e3),3e3,3e3)
> blas_get_num_procs()
[1] 1
> system.time(A%*%A)
   user  system elapsed
  3.992   0.012   4.003
> blas_set_num_threads(4)
> system.time(A%*%A)
   user  system elapsed
  4.332   0.048   1.234
```

Outline

Concluding remarks

- **R** has been used on supercomputers especially for big data analysis and/or large scale simulations.
- Good parallel computing technology is required for **R** on supercomputers.
- **Rhpc** is an attempt for this purpose. **Rhpc** depends heavily on MPI implementation.
- **Rhpc** is still under development. It will be available from our site soon and from CRAN in future.