A Quick Guide for the pbdDEMO Package

Drew Schmidt[1], Wei-Chen Chen[2], George Ostrouchov[1,2], Pragneshkumar Patel[1]

[1]Remote Data Analysis and Visualization Center,
University of Tennessee,
Knoxville, TN, USA
[2]Computer Science and Mathematics Division,
Oak Ridge National Laboratory,
Oak Ridge, TN, USA

Contents

Acknowledgement

Schmidt, Ostrouchov, and Patel were supported in part by the project "NICS Remote Data Analysis and Visualization Center" funded by the Office of Cyberinfrastructure of the U.S. National Science Foundation under Award No. ARRA-NSF-OCI-0906324 for NICS-RDAV center. Chen and Ostrouchov were supported in part by the project "Visual Data Exploration and Analysis of Ultra-large Climate Data" funded by U.S. DOE Office of Science under Contract No. DE-AC05-00OR22725.

This work used resources of National Institute for Computational Sciences at the University of Tennessee, Knoxville, which is supported by the Office of Cyberinfrastructure of the U.S. National Science Foundation under Award No. ARRA-NSF-OCI-0906324 for NICS-RDAV center. This work also used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This work used resources of the Newton HPC Program at the University of Tennessee, Knoxville.

We also thank Brian D. Ripley, Kurt Hornik, and Uwe Ligges from the R Core Team for discussing package release issues and helping us solve portability problems on different platforms.

Warning: This document is written to explain the main functions of **pbdDEMO** (Schmidt *et al.* 2012b), version 0.1-0. Every effort will be made to ensure future versions are consistent with these instructions, but features in later versions may not be explained in this document.

Information about the functionality of this package, and any changes in future versions can be found on website: "Programming with Big Data in R" at `http://r-pbd.org/`.

1. Introduction

This vignette is to explain some pbdR (Ostrouchov *et al.* 2012) examples which are higher level applications and may be commonly found in basic Statistics. The purposes is to show how to reuse the pre-exist functions, and quickly solve problems in an efficient way. The functions built for examples may not be exactly same idea of original R (R Core Team 2012) functions, but can be adjusted in similar wa. You are very welcome to use these as templates and rewrite your own functions or packages.

1.1. Installation

One can download **pbdDEMO** from CRAN at `http://cran.r-project.org`, and the intallation can be done with the following commands

Shell Command

```
tar zxvf pbdDEMO_0.1-0.tar.gz
R CMD INSTALL pbdDEMO
```

Since **pbdEMO** depends on other **pbdR** packages, please read the corresponding vignettes if installations did not succeed. We also provide several demos for the capability of **pbdR** packages which are explained correspondingly in the next few sections.

1.2. Notation

We presume that readers already have idea about SPMD programming. If not, please read **pbdMPI**'s vignette Chen *et al.* (2012b) first. If possible, readers are encouraged to run the demo of **pbdMPI** package and go through the code step by step.

Note that we tend to use suffix `.spmd` to indicate a distributed local object which is a portion of big object. As the usual R, Supposer $\boldsymbol{X}$ is a very big matrix, then `X.spmd`'s could be R objects distributed on all processors, and usually they have the same data type. $\boldsymbol{X}$ could be a result of `cbind` or `rbind` on all `X.spmd`'s. Examples can be found in the Section 2.

We also tend to assume common variables without suffix `.spmd` in SPMD programming. For instance, `y` could be an R object on all processors. Their dimension/attributes are allowed to be differed on some processors. In SPMD case, it may be a good idea to invent a `spmd` S4 class or methods for this purpose, but most S4 methods are sufficient and fast in most computation.

While in distributed computing, in contrast, it is a better idea to invent a `ddmatrix` class as in **pbdBASE** (Schmidt *et al.* 2012a) and **pbdDMAT** (Schmidt *et al.* 2012c), then let efficient libraries handle computing and avoid tedious coding. i.e. $\boldsymbol{X}$ is a very big matrix, but `X.dmat` is in block-cyclic format where `X.dmat` can be utilized by most **pbdR** functions. Examples can be found in the Section 3.

2. Basic Statistics Examples

This section introduces four simple examples and explains a little about distributed data computing. We skip tedious derivations of all formula, please see Wikipedia at http://www.wikipedia.org for better explanations if any statistical terminology were not familiar to you. These implemented examples/functions are partly selected from the Cookbook of HPSC website (Chen and Ostrouchov 2011) at http://thirteen-01.stat.iastate.edu/snoweye/hpsc/?item=cookbook. Please see more details there.

2.1. Monte Carlo for π

The demo command is

Shell Command

```
### At the shell prompt, run the demo with 4 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 4 Rscript -e "demo(monte_carlo,'pbdDEMO',ask=F,echo=F)"
```

This is a simple Monte Carlo example for estimating π. Suppose we have N uniform samples (x_i, y_i) on $(0,1) \times (0,1)$ where $i = 1, 2, \ldots, N$, then the π can be roughly approximated by

$$\pi \approx \frac{\#\text{ of } \sqrt{x_i^2 + y_i^2} \leq 1}{N}. \tag{1}$$

The key step of the demo code is in the next

R Code

```
1 N.spmd <- 1000
2 X.spmd <- matrix(runif(N.spmd * 2), ncol = 2)
3 r.spmd <- sum(sqrt(rowSums(X.spmd^2)) <= 1)
4 ret <- allreduce(c(N.spmd, r.spmd), op = "sum")
5 PI <- 4 * ret[2] / ret[1]
6 comm.print(PI)
```

In line 1, we specify sample size in `N.spmd` for each processor, and $N = D \times$ `N.spmd` if D processors are executed. In line 2, we generate samples in `X.spmd` for every processor. In line 3, we compute how many of radios are less than or equal to 1 for each processors. In line 4, we call `allreduce` to obtain total numbers across all processors. In line 5, we use the Equation (1). Since SPMD, `ret` is common on all processors, and so is `PI`.

2.2. Sample Mean and Sample Variance

The demo command is

Shell Command

```
### At the shell prompt, run the demo with 4 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 4 Rscript -e "demo(sample_stat,'pbdDEMO',ask=F,echo=F)"
```

Suppose $\boldsymbol{x} = \{x_1, x_2, \ldots, x_N\}$ are observed samples, and N is potentially very large. We can distribute $\boldsymbol{x}$ in 4 processors, and each processor roughly takes half of data. One simple way to compute sample mean $\bar{x}$ and sample variance s_x is based on the formulas

$$\begin{aligned} \bar{x} &= \frac{1}{N}\sum_{n=1}^{N} x_n \\ &= \sum_{n=1}^{N} \frac{x_n}{N} \\ s_x &= \frac{1}{N-1}\sum_{n=1}^{N}(x_n - \bar{x})^2 \\ &= \left(\sum_{n=1}^{N} \frac{x_n^2}{N-1}\right) - \frac{N}{N-1}\bar{x}^2 \end{aligned}$$

where the second equations of $\bar{x}$ and s_x are one-pass algorithms in terms of C language which are potentially more stable and faster than the first equations especially for large N. Here, only the first and second moments are implements while the extension of one-pass algorithms to higher order moments are also feasible.

The demo `sample_stat` generates fake data on 4 processors, then utilizes `mpi.stat` function as

R Code

```
mpi.stat <- function(x.spmd){
  ### For mean(x).
  N <- allreduce(length(x.spmd), op = "sum")
  bar.x.spmd <- sum(x.spmd / N)
  bar.x <- allreduce(bar.x.spmd, op = "sum")

  ### For var(x).
  s.x.spmd <- sum(x.spmd^2 / (N - 1))
  s.x <- allreduce(s.x.spmd, op = "sum") - bar.x^2 * (N / (N - 1))

  list(mean = bar.x, s = s.x)
} # End of mpi.stat().
```

where `allreduce` in **pbdMPI** (Chen *et al.* 2012a) can be utilized in this examples to aggregate local information across all processors.

2.3. Binning

The demo command is

Shell Command

```
### At the shell prompt, run the demo with 4 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 4 Rscript -e "demo(binning,'pbdDEMO',ask=F,echo=F)"
```

Binning is a classical statistics and can quickly summarize the data structure by setting some breaks between max and min of data. This is particularly a useful tool for constructing histograms and categorical data analysis.

The demo `binning` generates fake data on 4 processors, then utilize `mpi.bin` function as

R Code

```
mpi.bin <- function(x.spmd, breaks = pi / 3 * (-3:3)){
  bin.spmd <- table(cut(x.spmd, breaks = breaks))
  bin <- as.array(allreduce(bin.spmd, op = "sum"))
  dimnames(bin) <- dimnames(bin.spmd)
  class(bin) <- class(bin.spmd)
  bin
} # End of mpi.bin().
```

An easy implementation is to utilize `table` function to obtain local counts, then call `allreduce` to obtain global counts.

2.4. Quantile

The demo command is

Shell Command

```
### At the shell prompt, run the demo with 4 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 4 Rscript -e "demo(quantile,'pbdDEMO',ask=F,echo=F)"
```

Quantile is the other useful tool from fundamental statistics which provides data distribution for the desired value. This example can be extended to construct Q-Q plot, compute cumulative density function and nonparametric statistics, solve maximum likelihood estimators.

This is only an inefficient implementation to approximate a quantile and is not equivalent to the original `quantile` function in R. But in some sense, it should work well in large scale. The demo `quantile` generates fake data on 4 processors, then utilizes `mpi.quantile` function as

R Code

```
mpi.quantile <- function(x.spmd, prob = 0.5){
  if(sum(prob < 0 | prob > 1) > 0){
    stop("prob should be in (0, 1)")
  }

  N <- allreduce(length(x.spmd), op = "sum")
  x.max <- allreduce(max(x.spmd), op = "max")
  x.min <- allreduce(min(x.spmd), op = "min")

  f.quantile <- function(x, prob = 0.5){
    allreduce(sum(x.spmd <= x), op = "sum") / N - prob
  }

  uniroot(f.quantile, c(x.min, x.max), prob = prob[1])$root
} # End of mpi.quantile().
```

where a numerical function is solved by `uniroot` to find out the appropriate value such that cumulated probability is less than or equal to the specified quantile.

This simple example shows that the SPMD is greatly applicable on large scale data analysis and likelihood computing. Note that the `uniroot` call is working in parallel and on distributed data, i.e. other optimization functions such as `optim` and `nlm` can be utilized in the same way, since SPMD simply assumes every processors do the same work simulatinuously.

2.5. Ordinary Least Square

The demo command is

Shell Command

```
### At the shell prompt, run the demo with 4 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 4 Rscript -e "demo(ols,'pbdDEMO',ask=F,echo=F)"
```

Ordinary least square (OLS) is a fundament tool of linear models to find a solution for

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where $\boldsymbol{y}$ is $N \times 1$ observed vector, $\boldsymbol{X}$ is $N \times p$ designed matrix which is full rank and $N >> p$, $\boldsymbol{\beta}$ is the insterested parameters and unknown to be estimated, and $\boldsymbol{\epsilon}$ is errors and to be minimized. A classical solution is to

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^t\boldsymbol{X})^{-1}\boldsymbol{X}^t\boldsymbol{y}$$

This example can be also generalized to weighted least square (WLS), and linear mixed effect models (LME).

The implementation is straight forward as

R Code

```
mpi.ols <- function(y.spmd, X.spmd){
  if(length(y.spmd) != nrow(X.spmd)){
    stop("length(y.spmd) != nrow(X.spmd)")
  }

  t.X.spmd <- t(X.spmd)
  A <- allreduce(t.X.spmd %*% X.spmd, op = "sum")
  B <- allreduce(t.X.spmd %*% y.spmd, op = "sum")

  solve(matrix(A, ncol = ncol(X.spmd))) %*% B
} # End of mpi.ols().
```

Note that this is a silly implementation for demonstrations and explain fundamental idea of OLS. This is only efficient for small N and small p. For larger scale, we suggest to simply convert `y.spmd` and `X.spmd` into block-cyclic format as in the Section **??** and to utilize **pbdBASE** and **pbdDMAT** for all matrix computation via **pbdSLAP** (Chen *et al.* 2012c).

3. Data Input Examples

In order to utilize pbd packages efficiently, a good input method is necessary from high performance point of view since I/O can be a large portion of whole computing. Especially in distributed computing, reading data from file, database, or even memory can take much longer than serial computing. We introduce some priminary methods here to handle basic data input.

For this section, we presume readers already understand some basic idea of block-cyclic matrix. If not, we strongly recommend readers to scan **pbdSLAP**, **pbdBASE**, and **pbdDMAT** vignettes (Chen *et al.* 2012d; Schmidt *et al.* 2012d,e).

3.1. Load Balance and Unload Balance

The demo command is

Shell Command

```
### At the shell prompt, run the demo with 4 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 4 Rscript -e "demo(balance,'pbdDEMO',ask=F,echo=F)"
```

In this example, suppose we have an unbalanced input `X.spmd`, then we call `balance.info` on it to save some information for later uses. Then, we call `load.balance` to obtain balanced matrix `new.X.spmd`. Then, we call `unload.balance` to obtain original matrix `org.X.spmd` which should be exactly the same as `X.spmd`. We list the key steps in the next.

R Code

```
bal.info <- balance.info(X.spmd)
new.X.spmd <- load.balance(X.spmd, bal.info)
org.X.spmd <- unload.balance(new.X.spmd, bal.info)
```

The details are depicted in the Figure 3.1 where we distributed a matrix in two ways: `X.spmd` is unbalanced and is usually filtered or selected from other matrics, `new.X.spmd` is a balanced version of `X.spmd`.

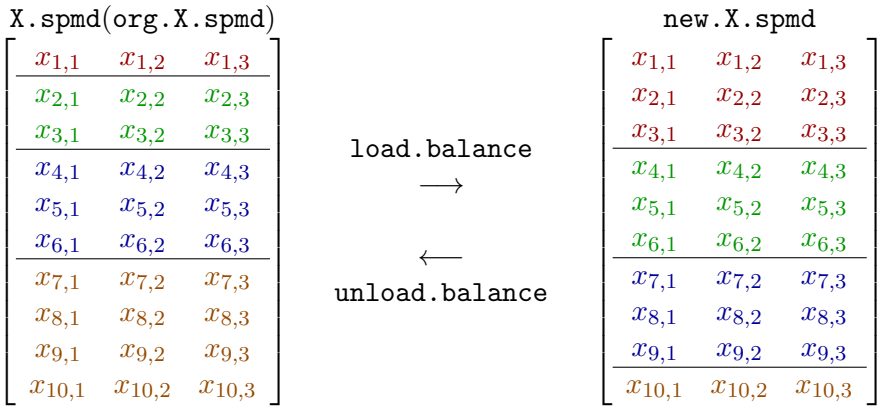

Figure 1: $\boldsymbol{X}$ is distributed in `X.spmd`(`org.X.spmd`) and `new.X.spmd`. Both are distributed by row in 4 processors where colors represents processor 0, 1, 2, and 3.

The useful function `balance.info` will returns information, says `bal.info`, how to load balance the given data matrix `X.spmd`. The return of `balance.info` is a list containing two `data.frame`'s (`send` and `recv` and two `vector`'s (`N.allspmd` and `new.N.allspmd`).

The `send` records the original rank and the belonging rank of "unbalanced" data matrix. The `load.balance` uses this table to `isend` data. If any belonging rank is not the original rank, then the corresponding entire rows will be sent to where they should belong.

The `recv` also records the original rank and the belonging rank of "balanced" data matrix. The `load.balance` uses this table to `recv` data. If any original rank is not the current rank, then the corresponding entire rows will be received from where they original rank.

The `N.allspmd` and `new.N.allspmd` both have length equals to `comm.rank(comm)` containing all numbers of rows of `X.spmd` before and after balanced. This is for double checking and avoiding 0-row matrix issue.

For `unload.balance`, the process is just a little bit tricky by reversing `bal.info` and passing it to `load.balance`.

3.2. Conver SPMD and DMAT

The demo command is

Shell Command

```
### At the shell prompt, run the demo with 4 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 4 Rscript -e "demo(spmd_dmat,'pbdDEMO',ask=F,echo=F)"
```

As usual serial R, it is intuitively to read data in parallel for each processor. We assume the input method is in SPMD to yield data in `X.spmd`, and convert it into `ddmatrix` format in `X.dmat`, then convert again to a R object in `X` which is common on all processors, as in the next. The Figure 3.2 illustates `X.spmd` and `X.dmat`.

R Code

```
X.dmat <- spmd2dmat(X.spmd)
X <- as.matrix(X.dmat)
new.X.spmd <- dmat2spmd(X.dmat)
```

Note that we also provide a reversed function `dmat2spmd` for `spmd2dmat`.

Here, the `spmd2dmat` function does the following:

1. Check numbers of columns of `X.spmd`, all processors should be the same.
2. Row balance the SPMD matrix, since number of rows of `X.spmd` may vary in different processors.
3. Call `new` on the balanced matrix to yield a `ddmatrix` object, says `X.dmat`, in block context 2 (`ICTXT = 2`).
4. Convert the `X.dmat` to other block contexts (default`ICTXT = 0`) via `base.reblock`.

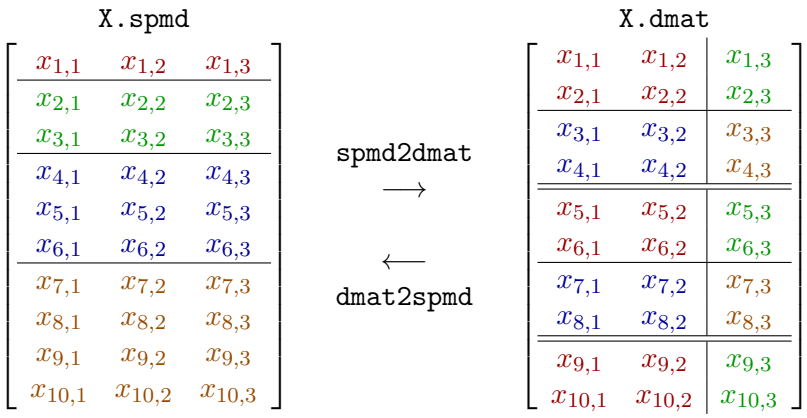

Figure 2: $\boldsymbol{X}$ is distributed in `X.spmd` and `X.dmat`. Both are distributed in 4 processors where colors represents processor 0, 1, 2, and 3. Note that `X.dmat` is in block-cyclic format of 2×2 grid with 2×2 block dimension.

Note that the first step is done by the `load.balance` function (Section 3.1) which is particular useful to balance SPMD matrix and yield performance even the matrix is not in block-cyclic format.

3.3. The csv Files

The best way is to read in data directly in block-cyclic format.

References

Chen WC, Ostrouchov G (2011). "HPSC – High Performance Statistical Computing for Data Intensive Research." URL http://thirteen-01.stat.iastate.edu/snoweye/hpsc/.

Chen WC, Ostrouchov G, Schmidt D, Patel P, Yu H (2012a). "pbdMPI: Programming with Big Data – Interface to MPI." R Package, URL http://cran.r-project.org/package=pbdMPI.

Chen WC, Ostrouchov G, Schmidt D, Patel P, Yu H (2012b). "A Quick Guide for the pbdMPI package." R Vignette, URL http://cran.r-project.org/package=pbdMPI.

Chen WC, Schmidt D, Ostrouchov G, Patel P (2012c). "pbdSLAP: Programming with Big Data – Scalable Linear Algebra Packages." R Package, URL http://cran.r-project.org/package=pbdSLAP.

Chen WC, Schmidt D, Ostrouchov G, Patel P (2012d). "A Quick Guide for the pbdSLAP package." R Vignette, URL http://cran.r-project.org/package=pbdSLAP.

Ostrouchov G, Chen WC, Schmidt D, Patel P (2012). "Programming with Big Data in R." URL http://r-pbd.org/.

R Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.r-project.org/.

Schmidt D, Chen WC, Ostrouchov G, Patel P (2012a). "pbdBASE: Programming with Big Data – Core pbd Classes and Methods." R Package, URL http://cran.r-project.org/package=pbdBASE.

Schmidt D, Chen WC, Ostrouchov G, Patel P (2012b). "pbdDEMO: Programming with Big Data – Demonstrations of pbd Packages." R Package, URL http://cran.r-project.org/package=pbdDEMO.

Schmidt D, Chen WC, Ostrouchov G, Patel P (2012c). "pbdDMAT: Programming with Big Data – Distributed Matrix Algebra Computation." R Package, URL http://cran.r-project.org/package=pbdDMAT.

Schmidt D, Chen WC, Ostrouchov G, Patel P (2012d). "A Quick Guide for the pbdBASE package." R Vignette, URL http://cran.r-project.org/package=pbdBASE.

Schmidt D, Chen WC, Ostrouchov G, Patel P (2012e). "A Quick Guide for the pbdDMAT package." R Vignette, URL http://cran.r-project.org/package=pbdDMAT.