Parallel Computing with R using GridRPC

Junji NAKANO and Ei-ji NAKAMA

The Institute of Statistical Mathematics, Japan and COM-ONE Ltd., Japan

at SC10 on 13-19 November 2010, New Orleans, LA, USA

Introduction

R is a free software environment for statistical computing and graphics. We hope to use computing resources located on local and remote networks simultaneously by R easily and efficiently. For this aim, we make it possible to use GridRPC middleware by Ninf-G in R.

How we usually use remote resources

- We log in to a front-end of the remote system by using `ssh`
- Different remote systems require different operations even for executing the same job
- Systems have difficulty to access data located outside of firewall

We can improve them by using GridRPC.

GridRPC

- GridRPC is middleware that provides a model for access to remote libraries and parallel programming for tasks on a grid. Ninf-G and Netsolve are famous GridRPC middleware. Other GridRPC middleware includes GridSolve, DIET, and OmniRPC.
- We use Ninf-G to realize GridRPC functions in R.
- Ninf-G is a reference implementation of GridRPC system using the Globus Toolkit. Ninf-G provides GridRPC APIs which are discussed for the standardization at the Grid Remote Procedure Call Working Group of the Global Grid Forum.
- Some implementations of GridRPC (including Ninf-G) can work through `ssh` without any Grid middleware.

Overview of RGridRPC

RGridRPC is an implementation to use embedded R and submits jobs to stubs. One process starts from the generation of a handle and ends by the the destruction of it. The figure below shows how GridRPC APIs are used.

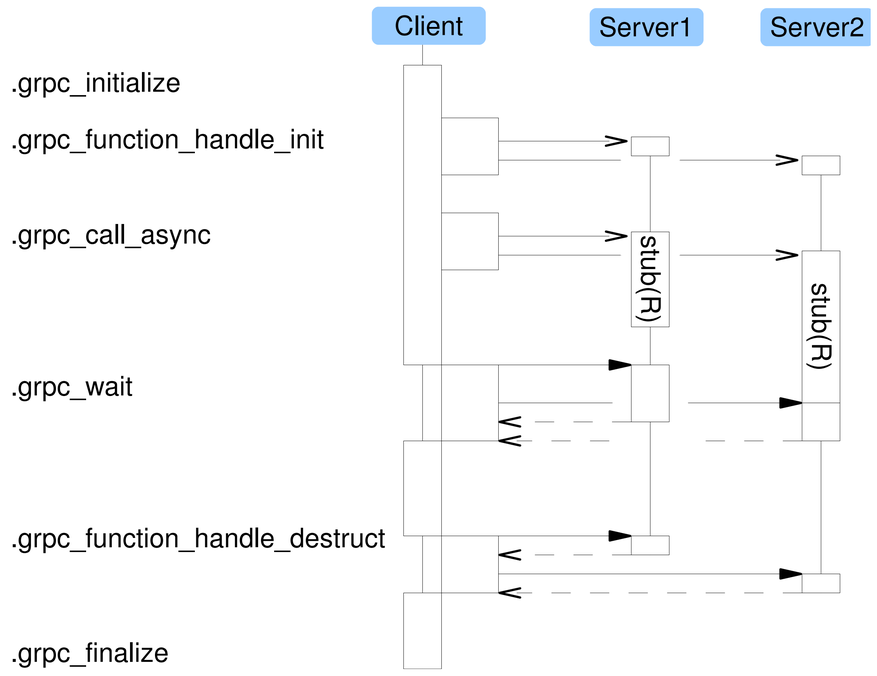

Fig 1: RGridRPC overview

RGridRPC primitive functions

- Client initialization and finalization functions
  - `.grpc_initialize(config_file)`
  - `.grpc_finalize()`
- Handle functions
  - `.grpc_function_handle_init(hostname)`
  - `.grpc_function_handle_default()`
  - `.grpc_function_handle_destruct(handle)`
- Session synchronous function
  - `.grpc_call(handle,fun,...)`
- Session asynchronous functions
  - `.grpc_call_async(handle,fun,...)`
  - `.grpc_probe(session)`
  - `.grpc_wait(session)`

RGridRPC snow-like functions

"snow" is a package for parallel computing with R.

- Client initialization and finalization functions
  - `GRPCmake(hostname)`
  - `GRPCstop(handle)`
- Synchronous functions
  - `GRPCevalq(handle,expr)`
  - `GRPCexport(handle,names)`
  - `GRPCcall(handle,fun,...)`
  - `GRPCclusterApply(handle,x,fun,...)`
- Asynchronous functions
  - `GRPCcallAsync(handle,fun,...)`
  - `GRPCprobe(section)`
  - `GRPCwait(section)`

Installation procedures

RGridRPC can be easily installed by users.

```
# download
$ wget http://prs.ism.ac.jp/RGridRPC/RGridRPC_0.10-302.tar.gz
# create R_LIBS_USER directory
$ R -q -e 'dir.create(Sys.getenv("R_LIBS_USER"),rec=T)'
# install
$ R CMD INSTALL RGridRPC_0.10-302.tar.gz
```

When we use Grid system, we install Ninf-G for each calculation node and set NG_DIR environment variable properly and install RGridRPC.

```
# Using Grid
$ NG\_DIR=/opt/ng R CMD INSTALL RGridRPC_0.10-302.tar.gz
```

RGridRPC setup

- RGridRPC reads the file `client.conf` in the current directry as a configuration file
- Two-way connections are required for RGridRPC.
  - Client should be specified by a client hostname from server side in `client.conf`
  - Or Proxy should be specified by a Proxy IP address from server side in `client.conf`
- An execution module of stub requires NG_DIR environment variable to know the top directory of Ninf-G
- RGridRPC uses NRF (Ninf-G Remote Information File) as Information sources

We provide an R function `makeninfgconf` to generate `client.conf` and `RGridRPC.servername.nrf` file. Bind address of each cluster needs to be specified manually.

```
makeninfgconf(
  hostname=c(
    "pbsclister.ism.ac.jp",
    "remotesv.ism.ac.jp",
    "localhost"),
  pkgpath=c(
    "/home/eiji/R/i486-pc-linux-gnu-library/2.11/RGridRPC/",
    "/home/eiji/R/x86_64-pc-linux-gnu-library/2.11/RGridRPC/",
    "/home/eiji/R/powerpc-unknown-linux-gnu-library/2.11/RGridRPC/"),
  ngdir=c(
    "/home/eiji/R/i486-pc-linux-gnu-library/2.11/RGridRPC/",
    "/home/eiji/R/x86_64-pc-linux-gnu-library/2.11/RGridRPC/",
    "/opt/ng"),
  invoke_server=c("SSH", "SSH", "SSH"),
  jobmanager=c("jobmanager-pbs", NA, NA))
```

Examples

Example for (non-parallel) bootstrapping by pvclust:

```
$ R CMD BATCH pvclust1.R
```

```
> library(pvclust)
> library(MASS)
> data(Boston)
> nboot<-6000
> set.seed(1)
> system.time(boston <- pvclust(Boston, nboot=nboot))
Bootstrap (r = 0.5)... Done.
Bootstrap (r = 0.6)... Done.
Bootstrap (r = 0.7)... Done.
Bootstrap (r = 0.8)... Done.
Bootstrap (r = 0.9)... Done.
Bootstrap (r = 1.0)... Done.
Bootstrap (r = 1.1)... Done.
Bootstrap (r = 1.2)... Done.
Bootstrap (r = 1.3)... Done.
Bootstrap (r = 1.4)... Done.
   user system elapsed
497.263 0.200 497.480
> postscript("pvclust1.eps", width=9, height=7, family="Helvetica",
+ horizontal=FALSE, onefile=FALSE, paper="special")
> plot(boston)
> dev.off()
null device
          1
```

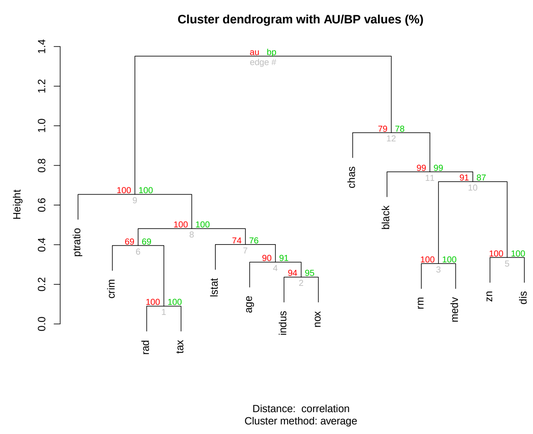

Fig 2: pvclust by single task

Example for parall bootstrapping by pvclust with RGridRPC:

```
$ eval `ssh-agent`
Agent pid
$ ssh-add .ssh/id_rsa
Enter passphrase for .ssh/id_rsa:
Identity added: .ssh/id_rsa (.ssh/id_rsa)
$ R CMD BATCH pvclust2.R
```

```
> library(pvclust)
> library(MASS)
> data(Boston)
>
> library(RGridRPC)
> library(snow)
> # Replace a snow function by an RGridRPC function
> parLapply<-function (cl, x, fun, ...){ docall(c,
+ GRPCclusterApply(cl, splitList(x, length(cl)), lapply, fun, ...))
+ }
> # Make 30 workers on 3 systems
> cl<-GRPCmake(c(rep("prs.ism.ac.jp",2),
+ rep("prcs.ism.ac.jp",12),
+ rep("prds.ism.ac.jp",16)))
> nboot <- 6000
> # Set seeds
> dummy<-GRPCclusterApply(cl,1:length(cl),set.seed)
> system.time(boston.pv <- parPvclust(cl, Boston, nboot=nboot))
Multiscale bootstrap... Done.
   user system elapsed
  0.172 0.036 21.432
> GRPCstop(cl)
> postscript("pvclust2.eps", width=9, height=7, family="Helvetica",
+ horizontal=FALSE, onefile=FALSE, paper="special")
> plot(boston.pv)
> dev.off()
null device
          1
```

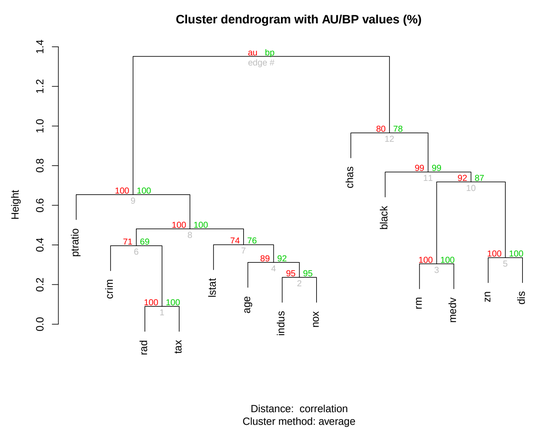

Fig 3: pvclust with RGridRPC