Parallel Computing with R using GridRPC

Junji NAKANO [†] Ei-ji NAKAMA[‡]

[†] The Institute of Statistical Mathematics, Japan

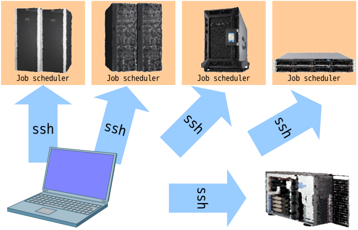
[‡] COM-ONE Ltd., Japan

The R User Conference 2010
July 20-23,
National Institute of Standards and Technology (NIST),
Gaithersburg, Maryland, USA

Our aim

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 protocol in R.

Usual use of remote resources



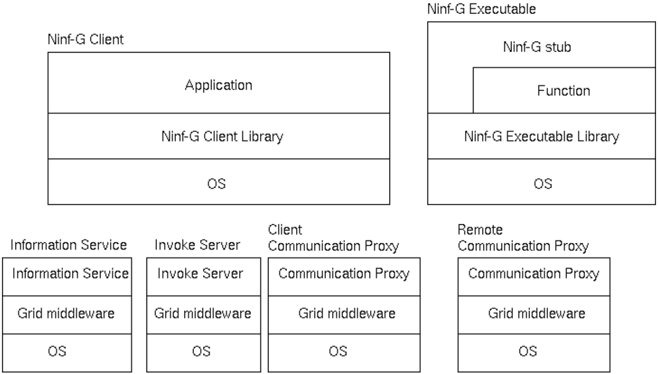
- We log in to a front-end of the remote system by using `ssh`
- Different remote systems require different operations even for executing the same job
- Systems have difficulty to access data located outside of firewall
- We can improve them by using GridRPC

GridRPC

- GridRPC is middleware that provides a model for access to remote libraries and parallel programming for tasks on a grid. Typical GridRPC middleware includes Ninf-G and Netsolve. The 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 Ninf-G

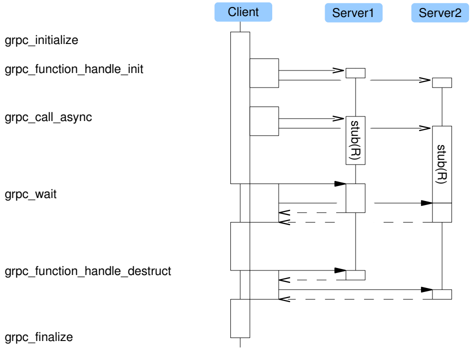
Ninf-G is a set of library functions that provide an RPC capability in a Grid environment, based on the GridRPC API specifications. Several processes shown below work together.



See `http://ninf.apgrid.org/`.

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. GridRPC APIs are used like the following figure.



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)

Examples of RGridRPC primitive functions

```
> library(RGridRPC)
> .grpc_initialize()
[1] TRUE
> c1<-.grpc_function_handle_default()
> f<-function(){Sys.sleep(1);paste(Sys.info()["nodename"],Sys.getpid(),Sys.time())}
> f()
[1] "triton 13228 2010-07-05 12:34:27"
> .grpc_call(c1, f)
[1] "r1400a 26504 2010-07-05 12:34:30"
> s1<-.grpc_call_async(c1, f)
> rc<-.grpc_probe(s1)
> while (rc$result) { cat(rc$message,fill=T); Sys.sleep(1) ; rc<-.grpc_probe(s1) }
Call has not completed
Call has not completed
> cat(rc$message,fill=T)
No error
> grpc_wait(s1)
[1] "r1400a 26504 2010-07-05 12:34:31"
> .grpc_R_finalize(c1)                  # server finalize
[1] TRUE
> .grpc_function_handle_destruct(c1)
[1] TRUE
> .grpc_finalize()
[1] TRUE
```

RGridRPC snow-like functions

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

Examples of RGridRPC snow-like functions (1)

```
> library(RGridRPC)
> prt<-function(l){unlist(lapply(l,paste,collapse=":"))}
> cpus<-get_num_cpus()
> cl<-GRPCmake(rep("localhost",cpus))
> unlist(GRPCcall(cl,Sys.getpid))
[1] 14956 14962
> A<-matrix(rnorm(1e3^2),1e3,1e3)
> B<-t(A)
> GRPCexport(cl,c("A"))
> prt(GRPCcall(cl,ls))
[1] "A" "A"
> sl<-GRPCcallAsync(cl,function(x){`%*%`(A,x)},B)
> prt(GRPCprobe(sl))
[1] "12:Call has not completed" "12:Call has not completed"
> str(GRPCwait(sl))
List of 2
 $ : num [1:1000, 1:1000] 983.48 -43.7 -9.81 -30.66 -58.44 ...
 $ : num [1:1000, 1:1000] 983.48 -43.7 -9.81 -30.66 -58.44 ...
> unlist(GRPCstop(cl))
[1] TRUE TRUE
```

Examples of RGridRPC snow-like functions (2-1)

```
> # http://www.stat.uiowa.edu/~luke/R/cluster/cluster.html
> library(RGridRPC)
>
> library(boot)
> data(nuclear)
> nuke <- nuclear[,c(1,2,5,7,8,10,11)]
> nuke.lm <- glm(log(cost)~date+log(cap)+ne+ ct+log(cum.n)+pt, data=nuke)
> nuke.diag <- glm.diag(nuke.lm)
> nuke.res <- nuke.diag$res*nuke.diag$sd
> nuke.res <- nuke.res-mean(nuke.res)
> nuke.data <- data.frame(nuke,resid=nuke.res,fit=fitted(nuke.lm))
> new.data <- data.frame(cost=1, date=73.00, cap=886, ne=0,ct=0, cum.n=11, pt=1)
> new.fit <- predict(nuke.lm, new.data)
> nuke.fun <- function(dat, inds, i.pred, fit.pred, x.pred) {
+      assign(".inds", inds, envir=.GlobalEnv)
+      lm.b <- glm(fit+resid[.inds] ~date+log(cap)+ne+ct+
+                  log(cum.n)+pt, data=dat)
+      pred.b <- predict(lm.b,x.pred)
+      remove(".inds", envir=.GlobalEnv)
+      c(coef(lm.b), pred.b-(fit.pred+dat$resid[i.pred]))
+ }
```

Examples of RGridRPC snow-like functions (2-2)

```
> N<-500
> cpus<-get_num_cpus()
> system.time(nuke.boot <- boot(nuke.data, nuke.fun, R=N*cpus, m=1,
+                              fit.pred=new.fit, x.pred=new.data))
   user  system elapsed
185.051 616.522  66.795
>
> cl<-GRPCmake(rep("localhost",cpus))
> GRPCevalq(cl, library(boot))
[[1]]
[1] "boot"      "stats"     "graphics"  "grDevices" "utils"     "datasets"
  ...
[[12]]
[1] "boot"      "stats"     "graphics"  "grDevices" "utils"     "datasets"
>
> system.time(cl.nuke.boot <- GRPCcall(cl,boot,nuke.data, nuke.fun, R=N, m=1,
+                                     fit.pred=new.fit, x.pred=new.data))
   user  system elapsed
  0.008   0.004   7.189
>
> GRPCstop(cl)
[[1]]
[1] TRUE
  ...
[[12]]
[1] TRUE
```

RGridRPC installation by users

Download

```
$ wget http://prs.ism.ac.jp/RGridRPC/RGridRPC_0.10-197.tar.gz
```

Client and server

```
$ R -q -e 'dir.create(Sys.getenv("R_LIBS_USER"),rec=T)'
$ R CMD INSTALL RGridRPC_0.10-123.tar.gz
```

Toolchain and Python are required. When we use Grid middleware (except ssh), we install Ninf-G for each system and set NG_DIR environment variable properly and install RGridRPC.

Using Grid middleware

```
$ R -q -e 'dir.create(Sys.getenv("R_LIBS_USER"),rec=T)'
$ NG_DIR=/opt/ng R CMD INSTALL RGridRPC_0.10-123.tar.gz
```

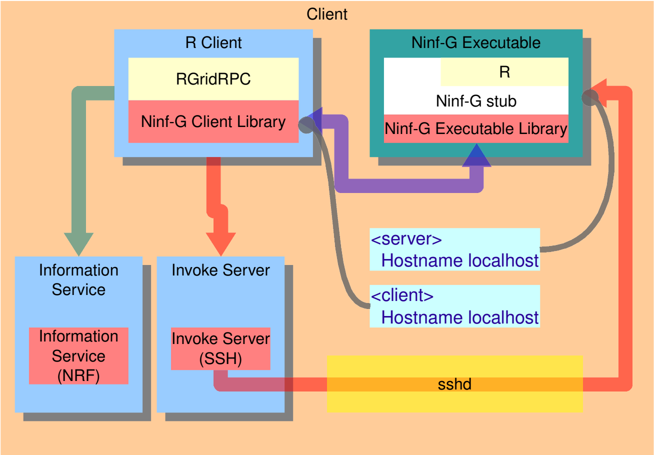
RGridRPC setup

- 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 as Information sources.

client.conf : localhost only

```
<CLIENT>
   hostname                localhost
</CLIENT>
<SERVER>
   hostname                localhost
   invoke_server           SSH
   environment             NG_DIR=${R_LIBS_USER}/RGridRPC
   environment             OMP_NUM_THREADS=1
</SERVER>
<INFORMATION_SOURCE>
   type                    NRF
   tag                     nrf
   source                  RGridRPC.localhost.nrf
</INFORMATION_SOURCE>
```

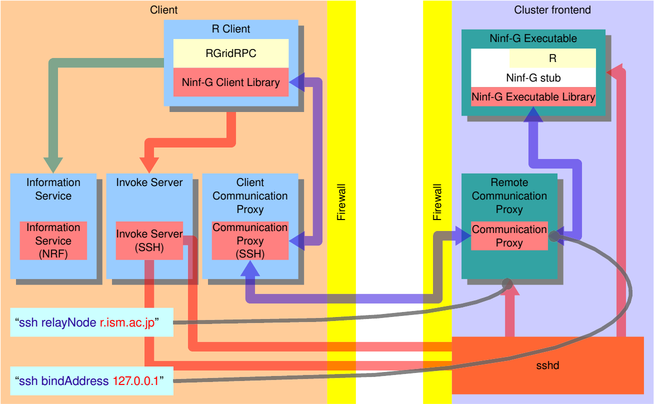
Information flow : localhost only



client.conf : using a remote server directly

```
<CLIENT_COMMUNICATION_PROXY>
  type                      SSH
</CLIENT_COMMUNICATION_PROXY>
<SERVER>
  hostname                  r.ism.ac.jp
  invoke_server             SSH
  environment               NG_DIR=${R_LIBS_USER}/RGridRPC
  environment               OMP_NUM_THREADS=1
  communication_proxy       SSH
  communication_proxy_option ''ssh_relayNode   r.ism.ac.jp''
  communication_proxy_option ''ssh_bindAddress 127.0.0.1''
</SERVER>
<INFORMATION_SOURCE>
  type                      NRF
  tag                       nrf
  source                    RGridRPC.r.ism.ac.jp.nrf
</INFORMATION_SOURCE>
```

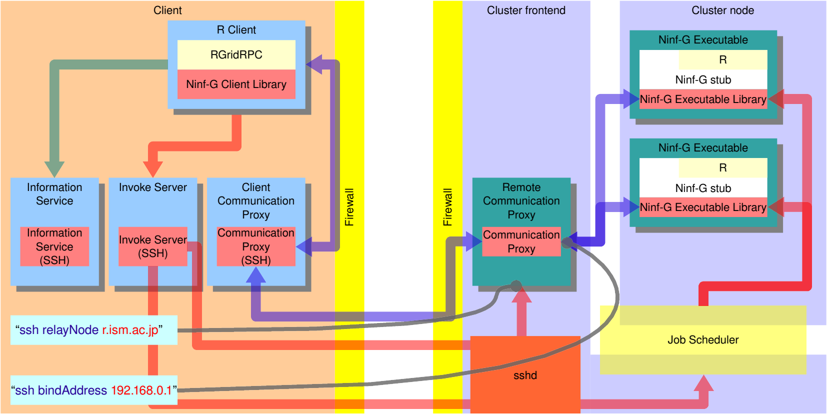
Information flow : using a remote server directly



client.conf : using a remote cluster

```
<CLIENT_COMMUNICATION_PROXY>
  type                        SSH
</CLIENT_COMMUNICATION_PROXY>
<SERVER>
  hostname                    r.ism.ac.jp
  invoke_server               SSH
  environment                 NG_DIR=${R_LIBS_USER}/RGridRPC
  environment                 OMP_NUM_THREADS=1
  jobmanager                  jobmanager-pbs
  communication_proxy         SSH
  communication_proxy_option ''ssh_relayNode   r.ism.ac.jp''
  communication_proxy_option ''ssh_bindAddress 192.168.0.1''
</SERVER>
<INFORMATION_SOURCE>
  type                        NRF
  tag                         nrf
  source                      RGridRPC.r.ism.ac.jp.nrf
</INFORMATION_SOURCE>
```

Information flow : using a remote cluster



RGridRPC eazy setup

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

makeninfgconf

```
makeninfgconf(hostname=c(
                "pbscluster.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))
```

Concluding remarks

- Advantages of RGridRPC
  - R can use many different cluster servers.
  - R can use resources outside of firewall.
- Disadvantages of present RGridRPC
  - If the limit of job scheduler is tight and some scattered jobs are waiting in the queue, the execution does not stop at all.
  - We cannot handle big objects because of serialization
    Serialize →base64 →RAW(max 2Gbyte)
  - The present implementation depends on Ninf-G.
    - Available Job Schedulers are limited to PBS,Torque and SGE.
    - Other Grid RPC middleware is not available.