Remote R Evaluation (rreval) Manual

March 2012, Version 1.0

Barnet Wagman bw@norbl.com

rreval is a means for using R on a remote system from within a local R session. Any R expression can be evaluated on the remote server. All non-graphical results are returned to the local R session: this includes the results of remote evaluations and (nearly) all textual output, including errors and warnings.

Communication is via ssh port forwarding, so the system is reasonably secure. *rreval* supports uploading and downloading R objects and scp file tranfers.

Expressions are evaluated by an R session on a remote system running the rreval server. When a local R session connects to a server, the local client has exclusive use of the remote R session until it disconnects; i.e. an R server handles only one client at a time.

This manual describes how to configure and use *rreval* as a standalone package. *rreval* is also used by the *cloudRmpi* package. If you are using it in that context, please refer to the *cloudRmpi* manual, rather than this document. Creating the RSA key pair is different and configuring *rreval* for use with *cloudRmpi* is simpler than setting it up for standalone use.

Contents

Requirements

rreval has two requirements besides the contents of the R package:

1. Java (>= 1.6), on the client and server systems. The Java interpreter must be in the execution path. To test whether you have an accessible copy of Java, at a command line type

   ```
   java -version
   ```

   If Java is installed, you should get something like

   ```
   java version "1.6.0_26"
   Java(TM) SE Runtime Environment (build 1.6.0_26-b03)
   Java HotSpot(TM) 64-Bit Server VM (build 20.1-b02, mixed mode)
   ```

(Why Java? See the *Architecture* section below.)

2. ssh daemon on the server side only. (An ssh client is not needed).

Installation

The *rreval* package needs to be installed into R on both the client and server, using the standard installation method (`R CMD INSTALL ...` from the shell or `install.packages(...)` in an R session).

SSH configuration

To keep things reasonably secure, the rreval client and server communicate via ssh port forwarding. The client logs into the server system using an RSA key pair (rather than a password).

Server configuration

The server must be running an ssh daemon and it must be configured to allow RSA key pair login. This is usually the default on Linux systems. To enable RSA key pair login, the ssh daemon configuration file `/etc/ssh/sshd_config` should contain (uncommented) the lines

```
RSAAuthentication yes
PubkeyAuthentication yes
AuthorizedKeysFile    %h/.ssh/authorized_keys
```

On some systems, you may also need to add

```
StrictModes no
```

RSA Key Pair

For RSA key pair login, you'll need an RSA key pair on the client and the public key registered on the server.

On the client side:

If you don't already have one, create an RSA key pair. On Unix system, you can do this with

```
ssh-keygen -t rsa -f <output_filename> -P ''
```

This command creates two files: `<output_filename>` which contains the key pair and `<output_filename>.pub` which contains the public key. You will need to supply the full pathname of `<output_filename>` as an argument to the `rre.startClient()` function when you connect to the server (see *Using rreval* below).

On the server side:

Copy the *public* key file to the server. Append it to the `authorized_keys` file in the

`.ssh` directory of the account where you'll be running the rreval server. E.g. for a user 'some-user' and an RSA public key file `an_rsa_file.pub`

```
cat an_rsa_file.pub >> ~some-user/.ssh/authorized_keys
```

Note that ssh is very finicky about file permissions. The `.ssh` directory and `authorized_keys` file must be accessible to the user only (modes 0700 and 0600 respectively on Unix systems).

Launching the rreval server

The rreval server must be running before you start the client. On the server, launch the rreval server from the command line with

```
echo 'library(rreval); rreServer(workingDir="some-dir")' | R --no-
save --no-restore --slave
```

or from within an R session with

```
library(rreval);
rreServer(workingDir="some-dir")
```

Using rreval

On the client

```
library(rreval)
```

To connect to a running server

```
rre.startClient(hostName="the-host-name",userName="the-user-
name",pemFile="/full/path/to/rsa/file")
```

where the host and user names correspond to the system and user account where the rreval server is running. *pemFile* is the full path to the RSA key pair file (see above). Note that this must be the path to the key pair file, *not* the public key file.

Once you're connected, you can evaluate expressions. E.g.

```
re(1+1)
```

will return the value 2. An expression like

```
re(a <- seq(1,100)^2)
```

will create an object named 'a' in the remote R session *and* return the value of the expression.

To retrieve an expression from the remote session

```
re(a)
```

or to assign it locally

```
aa <- re(a)
```

To upload an object

```
z <- sin(pi)
upo(z)
```

You can evaluate any expression in the remote session, including things like

```
re(library(npRmpi))
re(ls())
```

It is perfectly acceptable to upload functions, e.g.

```
fn <- function(x) { quantile(x,seq(0,1,0.1)) }
upo(fn)
```

rreval will handle errors gracefully, e.g.

```
re(lss())
```

yields the error message

```
<simpleError in eval(expr, envir, enclos): could not find function
"lss">
```

Note that all expression evaluations and assignments in the remote R session are performed in `.GlobalEnv` (the top level of the scope hierarchy). Of course you can explicitly specify other environments (e.g. `re(assign(x="a",value=1234,envir=some.other.envir))`).

When you are finished using the remote session, disconnect with

```
rre.closeClient()
```

The server handles only one client at time. This will free it up to handle another client. *Warning:* any objects that you created in the remote session will still exist after you disconnect, unless you explicitly delete then before you disconnect.

Handling large objects

The remote evaluation function `re()` always returns the results of evaluations, even if an assignment is made in the remote session. E.g. `re(a <- seq(1,10^6))` will return the sequence as well as assign it in the remote session. For large object, this can take a long time, so by default `re()` only returns the result of an evaluation if its serialized size is less than the value of `re()`'s `maxBytesReturnable` argument, which can be set to any value in [0,Inf].

To move large objects, *rreval* supports scp file transfers. E.g. for some funtion `fn()`

```
re(z <- fn(),maxBytesReturnable=0)
re(save(z,file="z_file.save"))
scpDownload("z_file.save");
load("z_file.save")
```

will get object `z` into the local R session.

rreval moves objects between the client and server in serialized form (using `serialize(ascii=TRUE,...)`). `serialize()` has a maximum object size of $2^{31} - 1$

bytes. R objects can exceed this size. Furthermore, the serialized version of an object is larger than the object it represents. The rreval functions `scpUpload()` and `scpDownload()` are used of circumvent this limitation. Note that for a large object, scp tranfers are faster than transfer mechanism used by `re()` and `upo()`.

Using multiple servers

It is possible to have open connections to more than one rreval server in a given local session. E.g. if there are rreval servers running on `user1@host1` and `user2@host2` you can connect to both:

```
rre.startClient(hostName="host1",userName="user1",pemFile="/path/to/pem")
rre.startClient(hostName="host2",userName="user2",pemFile="/path/to/pem")
```

With multiple connections open, you need to specify the (hostName,userName) pair in evaluation, object upload and scp functions. E.g.

```
re(a <- 2^4,hostName="host1",userName="user1")

upo(z,hostName="host2",userName="user2")
```

Architecture

Communication between the client and server is performed by a pair of java apps, `rreval.RReClientApp` and `rreval.RReServerApp`. The local R session sends a command to the `rreval.RReClientApp`. After performing error checks, the command is sent to `rreval.RReServerApp` which runs on the remote system. It in turn passes the command to the rreval server. The results of evaluation are returned by this path in reverse.

The two java apps communicate via ssh port forwarding, so communication between them should be secure. Note that communications between an R session and a java app are *en clair*. These are local socket communications so security should not an issue.

Since R does not support java directly, the use of java apps may seem a somewhat odd choice. While *rreval* is a self-contained package, it was primarily developed to support the *cloudRmpi* package, which is a means of doing R parallel processing on a network of Amazon ec2 instances. Managing an ec2 network is best done in java (Amazon does not provide a C or C++ api), so for consistency java was used in *rreval* as well.