Using the **`snowfall`** library in R

Phil Spector (`spector@stat.berkeley.edu`)
Statistical Computing Facility
Department of Statistics
University of California, Berkeley

1 Introduction

The `snowfall` library of R provides a simplified interface to the `snow` (Simple Network of Workstations) library, allowing parallelizable jobs to be distributed across a number of machines.

2 Basic Commands

The basic commands to set up a cluster, distribute information, and collect results are as follows:

- `sfInit` - The `sfInit` function sets up the machines which will be used in the parallel calculation. By listing machines multiple times, multiple CPUs on those machines will be used. In addition, the `type=` argument tells which type of messaging system will be used. To use snow's internal mechanism, `type='SOCK'` can be used. The `parallel=` argument allows selecting parallel operation (`TRUE`) or sequential operation (`FALSE`). When using `snowfall` on the SCF computers, all the hosts specified must be of the same architecture.

- `sfExport` - The `sfExport` function distributes copies of any data or functions which will be needed on the remote machines in the cluster. It takes an unlimited number of arguments, each representing a character string with the name of an object which needs to be distributed.

- `sfLibrary` The `sfLibrary` function is called once for each library which will be needed in the distributed computation. The name of the library required is provided without quotes.

- `sfClusterSetupRNG` - The `sfClusterSetupRNG` function sets up the random number generator to insure that each distributed machine gets a unique stream of random numbers. An optional `seed=` argument allows specification of a seed for reproducible results.

- `sfLapply, sfSapply, sfApply` - These functions provide parallelized versions of the functions `lapply`, `sapply`, and `apply`, respectively.
- `sfStop` - The `sfStop` function should be called with no arguments when the distribued computations are finished, to close the connections with the other machines.

The basic strategy of using `snowfall` is to choose the machines for the cluster using `sfInit`, pass required data and libraries to the cluster using `sfExport` and `sfLibrary`, initialize the random number generator with `sfClusterSetupRNG`, and then use one of the "`sf-apply`" functions to do the work.

3 An Example

This example uses simulation for a power calculation. The test being studied is a paired t-test, where observations have a correlation of 0.6. The goal is to determine the power for sample sizes from 10 to 80, using 1000 simulations for each sample size.

First, here's the program without any use of a cluster:

```
library(mvtnorm)
makedat = function(n,mean,var,delta){
   rmvnorm(n,c(mean,mean+delta),matrix(c(var,.6*var,.6*var,var),ncol=2))
}

runsim = function(nsim,n,mean,var,delta){
   nsig = 0
   for(i in 1:nsim){
      dat = makedat(n,mean,var,delta)
      if(t.test(dat[,1],dat[,2],paired=TRUE)$p.value < 0.05)nsig = nsig + 1
      }
   nsig / nsim
}

res <- sapply(seq(10,80,by=1),function(n)runsim(1000,n,.3,.3^2,.1))
```

The elapsed time to run this job on a single CPU is around 170 seconds.

Next, here's the program modified to use `snowfall`:

```
library(snowfall)
sfInit(socketHosts=rep(c('bilbo','roo','wanjina','beren','treebeard'),2),
       cpus=10,type='SOCK',parallel=TRUE)
```

```
makedat = function(n,mean,var,delta){
   rmvnorm(n,c(mean,mean+delta),matrix(c(var,.6*var,.6*var,var),ncol=2))
}

runsim = function(nsim,n,mean,var,delta){
   nsig = 0
   for(i in 1:nsim){
      dat = makedat(n,mean,var,delta)
      if(t.test(dat[,1],dat[,2],paired=TRUE)$p.value < 0.05)nsig = nsig + 1
      }
   nsig / nsim
}

sfExport('makedat','runsim')
sfLibrary(mvtnorm)
sfClusterSetupRNG(seed=7221)

wrapper = function(n){
  runsim(1000,n,.3,.3^2,.1)
}

res = sfSapply(seq(10,80,by=1),wrapper)
sfStop()
```

The execution time using 10 CPUs across the network is around 45 seconds.

4 Using `snowfall` with `pvm`

Instead of using `snow`'s internal message passing, the `snowfall` library allows using external messaging systems as well, for example `pvm`. (On the SCF system, `pvm` is only available on the Linux computers.) To use `pvm` with snowfall, a `pvm` process must be started at the command line (and halted when it's no longer needed). This is done using the `pvm` command. For example, to add the same hosts as used previously to a `pvm` cluster, the `pvm` command is executed at the UNIX prompt, and the `add` command is used to add the desired hosts:

```
% pvm
pvm> add bilbo roo wanjina beren treebeard
add bilbo roo wanjina beren treebeard
5 successful
                    HOST     DTID
                   bilbo    80000
                     roo    c0000
                 wanjina   100000
```

```
                         beren    140000
                     treebeard    180000
pvm>
```

Note that `pvm` will still accept commands, so the window in which the command was initiated should be left undisturbed until the `halt` command is issued after the cluster is no longer needed.

The only change needed to use `pvm` instead of the internal method is in the call to `sfInit`. For `pvm`, a call like the following should be used:

```
sfInit(cpus=10,type='PVM',parallel=TRUE)
```

Note that there is no need to specify the machine names, once `pvm` is started. Specifying more cpus than machines will cause `pvm` to use multiple cpus on some or all of the machines specified. While multiple architectures are supported by `pvm`, it appears that the `snowfall` library requires that all the machines in the `pvm` cluster are of the same architecture. After the `pvm` cluster is no longer needed, the `halt` command should be entered in the shell running `pvm` to terminate the process. (This will also terminate the R session which was running `snowfall`).

Using `pvm`, the example program used previously had an elapsed time of around 33 seconds.