snow: Simple Network of Workstations

Luke Tierney

Department of Statistics & Actuarial Science
University of Iowa

September 13, 2007

Motivation and Background

- Objective: shared memory parallel computing using R.
- Several tools are available, including
  - raw socket (`socketConnection`, `serialize`, `unserialize`)
  - `rpvm` package
  - `Rmpi` package
- Also available: parallel random number generators, including
  - `rsprng` package
  - `rlecuyer` package
  - `rstreams` package
- PVM and MPI are very powerful but also complex.
- Want higher level facilities that
  - make it easy to do simple scatter-compute-gather computations
  - can transparently use different communication back ends
  - simplify handling of random number generation

The snow Package

- snow is a parallel computing package for R
- snow is motivated by the CoW package from Scientific Python.
- snow uses a master/worker model:
  - The user starts an ordinary R session as the master process.
  - This session creates (or connects to) a set of worker processes.
  - Jobs are sent to the worker processes and results are returned.
- The underlying message passing can be based on
  - raw sockets (no additional packages/software needed)
  - PVM (uses rpvm and PVM)
  - MPI (uses Rmpi and LAM-MPI; other MPIs may also work)
- Which communication mechanism is used only matters at startup.

Starting a snow Cluster

- Start up PVM or LAM-MPI
- Start up R on the master node and load the snow package (if necessary).
- Create a cluster of 10 worker processes with

```
cl <- makeCluster(10)
```

- Find out where the processes are running:

```
> do.call("rbind", clusterCall(cl, function(cl)
                                  Sys.info()["nodename"]))
      nodename
 [1,] "node02.beowulf.stat.uiowa.edu"
 [2,] "node03.beowulf.stat.uiowa.edu"
 [3,] "node04.beowulf.stat.uiowa.edu"
 [4,] "node05.beowulf.stat.uiowa.edu"
 [5,] "node06.beowulf.stat.uiowa.edu"
 [6,] "node07.beowulf.stat.uiowa.edu"
 [7,] "node08.beowulf.stat.uiowa.edu"
 [8,] "node09.beowulf.stat.uiowa.edu"
 [9,] "node10.beowulf.stat.uiowa.edu"
[10,] "node00.beowulf.stat.uiowa.edu"
```

Stopping a snow Cluster

- To stop the worker processes use

  ```
  stopCluster(cl)
  ```

  then shut down PVM or LAM-MPI
- Some back ends may allow another `makeCluster` after a `stopCluster`, others may not.
- If you forget to call `stopCluster` before exiting R
  - For PVM, `halt` the PVM.
  - For LAM-MPI, use `lamhalt` or, if that fails, `lamwipe`.
  - For sockets, workers should just stop; if not, you need to clean up by hand.
  - If things did not end cleanly be sure to check for stray `R`, `pvmd`, or `lamd` processes on the nodes you used.

Cluster Level Functions

- Calling a function on all nodes:
  ```
  clusterCall(cl, exp, 1)
  ```
- Evaluating an expression on all nodes:
  ```
  clusterEvalQ(cl, library(boot))
  ```
- Apply a function to a list, one element per node
  ```
  clusterApply(cl, 1:3, get("+"), 3)
  ```
  It is an error if there are more elements in the list than workers in the cluster.
- A load balanced version:
  ```
  clusterApplyLB(cl, 1:20, get("+"), 3)
  ```
  There is no restriction on the length of the list.
- Assign values of specified global variables on master on each worker:
  ```
  clusterExport(cl, c("x", "y"))
  ```

Higher Level Functions

- A parallel version of `lapply` can be defined as

```
parLapply <- function(cl, x, fun, ...)
    docall(c, clusterApply(cl, splitList(x, length(cl)),
                           lapply, fun, ...))
```

- `splitList` splits the list argument approximately evenly across the cluster.
- An example using `qtukey` and a cluster of size 10:

```
> x<-1:100/101
> system.time(qtukey(x, 2, df=2))
 user  system elapsed
3.661   0.000   3.662
> system.time(unlist(parLapply(cl, x, qtukey, 2, df=2)))
 user  system elapsed
0.007   0.000   0.436
```

More Higher Level Functions

- Parallel `sapply`

```
> parSapply(cl, 1:15, get("+"), 2)
 [1]  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
```

- Parallel `apply`

```
> parApply(cl, matrix(1:10, ncol=2), 2, sum)
[1] 15 40
```

- `parCapply` and `parRapply`:

```
> A<-matrix(c(1,2,3,4,5,6),nrow=2)
> A
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> parCapply(cl, A, sum)
[1]  3  7 11
> parRapply(cl, A, sum)
[1]  9 12
```

A (Too) Simple Parallel Matrix Multiply

```
parMM <- function (cl, A, B) {
    if (nrow(A) >= ncol(B))
        docall(rbind, clusterApply(cl, splitRows(A, length(cl)),
                              function(a, B)  a %*% B, B))
    else
        docall(cbind, clusterApply(cl, splitCols(B, length(cl)),
                              function(b, A)  A %*% b, A))
}
```

- Using `parMM` does not pay for small matrices:

```
> A<-matrix(rnorm(10000),100)
> system.time(A %*% A)
 user  system elapsed
0.002   0.000   0.002
> system.time(parMM(cl,A , A))
 user  system elapsed
0.048   0.008   0.072
```

A (Too) Simple Parallel Matrix Multiply

- Using `parMM` pays (a little) for larger matrices:

```
> A<-matrix(rnorm(4000000),2000)
> system.time(A %*% A)
   user  system elapsed
 35.306   0.030  35.343
> system.time(parMM(cl,A , A))
   user  system elapsed
 15.125   3.498  29.469
```

- For this algorithm less parallelism is better:

```
> system.time(parMM(cl[1:4],A , A))
   user  system elapsed
  6.802   1.614  22.521
```

- There are much better algorithms.

Some Details

- Functions and arguments are converted to sequences of bytes and back using *serialization*.
- This is the same mechanism used for saving R workspaces.
- The user level interface is provided by `serialize` and `unserialize`.
- Non-top-level environments of functions are transmitted as copies.
- Some consequences:
  - Lexical scope can be used to bind constants needed by a function.
  - Care is needed to avoid unintended transfers of large objects.
  - Since copies are sent, assignments on workers remain local.
- Top-level environments are resolved to top-level environments on the workers:
  - `.GlobalEnv`
  - name space environments
  - environments of loaded package or the base package.

Parallel Random Numbers

- Random number generation needs some help:

```
 > clusterCall(cl, runif, 3)
[[1]]
[1] 0.2293371 0.2965413 0.2588331
[[2]]
[1] 0.2293371 0.2965413 0.2588331
....
[[10]]
[1] 0.2293371 0.2965413 0.2588331
```

- Identical streams are very likely but not guaranteed.
- If you want identical streams you can set a common seed.
- If you want “independent” streams you need something else.
- Using random seeds may work.
- A better alternative is to use a parallel generator package.

Using the rlecuyer Package

- Several parallel generators are available for R.
- These use R's facility to replace the core uniform generator.
- The rlecuyer package provides an interface to the streams library of L'Ecuyer, Simard, Chen, and Kelton.
- The function clusterSetupRNG assigns separate random number streams to each worker:

```
> clusterSetupRNG(cl)
> clusterCall(cl, runif, 3)
[[1]]
[1] 0.1270111 0.3185276 0.3091860
[[2]]
[1] 0.7595819 0.9783106 0.6851358
...
[[10]]
[1] 0.2925952 0.3593174 0.2368010
```

- Specifying a seed makes the streams reproducible.

Starting snow under PVM

There are three ways to start up PVM:

- Start the pvm console and add some nodes:

  ```
   [luke@node00 ~]$ pvm
  pvm> add node01 node02 node03
  add node01 node02 node03
  ...
  pvm>
  ```

- Start the pvm console with

  ```
   [luke@node00 ~]$ pvm pvmhosts
  ```

  where pvmhosts looks like

  ```
  node00
  node01
  ...
  node21
  ```

- Use xpvm, which needs a .xpvm_hosts file.

Starting snow under PVM

- The .xpvm_hosts file looks like

  ```
  node00
  &node01
  &node02
  ...
  &node21
  ```

  nodes marked with & are initially inactive.
- Click on the nodes you want to add to the virtual machine.
- Do not put xpvm in background — things get confused.
- xpvm provides useful visualizations of the computation.

Running `snow` under LAM-MPI

There are three ways to run `snow` under LAM-MPI:

- Using process spawning:
  - Start LAM-MPI with `lamboot`.
  - Start R and load the `snow` package.
  - Create an MPI cluster with

    ```
    cl<-makeCluster(type="MPI",3)
    ```
- Using `mpirun`
  - Start LAM-MPI with `lamboot`.
  - start R using a special shell script with

    ```
    mpirun -np 11 RPMISNOW
    ```
  - Get a reference to the running cluster with

    ```
    cl<-getMPIcluster()
    ```
  - Soon either of these will also work:

    ```
    cl<-makeCluster()
    cl<-makeCluster(10)
    ```
- Using `xmpi` and `RMPISNOW`.

Running snow under LAM-MPI

To use xmpi:

- Start LAM-MPI with lamboot.
- Start up xmpi from a terminal.
- Choose **Build&Run** from the **Application** menu.
- Choose the nodes to use.
- Enter RMPISNOW in the **Prog:** field.
- Press the **Run** button.
- The master R session will be running in the terminal where you started xmpi.
- Use getMPIcluster to get a reference to the running cluster.
- xmpi provides similar visualizations to the ones provided by xpvm.