R. Bivand. Using the R statistical data analysis language on GRASS 5.0 GIS data base files. *Computers & Geosciences*, 26:1043–1052, 2000. 9

R. Bivand and A. Gebhardt. Implementing functions for spatial statistical analysis using the R language. *Journal of Geographical Systems*, 3(2):307–317, 2000. 9

V. Gómez-Rubio and A. López-Quílez. RArcInfo: using GIS data with R. *Computers & Geosciences*, 31: 1000–1006, 2005. 9

E. Pebesma. Multivariable geostatistics in S: the gstat package. *Computers & Geosciences*, 30:683–691, 2004. 9

P. Ribeiro and P. Diggle. geoR: A package for geostatistical analysis. *R-News*, 1(2), 2001. URL: `www.r-project.org`, ISSN: 1609-3631. 9

B. Ripley. Spatial statistics in R. *R-News*, 1(2), 2001. URL: `www.r-project.org`, ISSN: 1609-3631. 9

B. Ripley. *Spatial Statistics*. Wiley, New York, 1981. 11

B. Rowlingson, A. Baddeley, R. Turner, and P. Diggle. Rasp: A package for spatial statistics. In A. Z. K. Hornik, F. Leisch, editor, *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003), March 20–22, Vienna, Austria.*, 2003. ISSN 1609-395X. 12

Edzer Pebesma
`e.pebesma@geog.uu.nl`

Roger Bivand
Norwegian School of Economics and Business Administration
`Roger.Bivand@nhh.no`

Running Long R Jobs with Condor DAG

by Xianhong Xie

For statistical computing, R and S-Plus have been around for quite some time. As our computational needs increase, the running time of our programs can get longer and longer. A simulation that runs for more than 12 hours is not uncommon these days. The situation gets worse with the need to run the program many times. If one has the access to a cluster of machines, he or she will definitely want to submit the programs to different machines, the management of the submitted jobs could prove to be quite challenging. Luckily, we have batch systems like Condor (`http://www.cs.wisc.edu/condor`), PBS (`http://pbs.mrj.com/service.html`), etc. available.

What is Condor?

Condor is a batch processing system for clusters of machines. It works not only on dedicated servers for running jobs, but also on non-dedicated machines, such as PCs. Some other popular batch processing systems include PBS, which has similar features as Condor. The results presented in this article should translate to PBS without much difficulties. But Condor provides some extra feature called checkpointing, which I will discuss later.

As a batch system, Condor can be divided into two parts: job management and resource management. The first part serves the users who have some jobs to run. And the second part serves the machines that have some computing power to offer. Condor matches the jobs and machines through the ClassAd mechanism which, as indicated by its name, works like the classified ads in the newspapers.

The machine in the Condor system could be a computer sitting on somebody else's desk. It serves not only the Condor jobs but also the primary user of the computer. Whenever someone begins using the keyboard or mouse, the Condor job running on the computer will get preempted or suspended depending on how the policy is set. For the first case, the memory image of the evicted job can be saved to some servers so that the job can resume from where it left off once a machine is available. Or the image is just discarded, and the job has to start all over again after a machine is available. The saving/resuming mechanism is called checkpointing, which is only available with the Condor Standard universe jobs. To be able to use this feature, one must have access to the object code files for the program, and relink the object code against the Condor library. There are some restrictions on operations allowed in the program. Currently these preclude creating a Standard universe version of R.

Another popular Condor universe is the Vanilla universe, under which most batch ready jobs should be able to run. One doesn't need to have access to the object code files, nor is relinking involved. The restrictions on what one can do in one's programs are much less too. Anything that runs under the Standard universe should be able to run under the Vanilla universe, but one loses the nice feature of checkpointing.

Condor supports parallel computing applications

with 2 universes called PVM and MPI. In addition, it has one universe for handling jobs with dependencies, one universe that extends Condor to grid computing, and one universe specifically for Java applications.

Condor in Action

The typical steps for running a Condor job are: preparing the programs to make them batch ready, creating Condor submit file(s), submitting the jobs to Condor, checking the results. An example is given in the following.

Suppose we have to run some simulations; each of them accepts one parameter `i` which controls how many iterations needs to be done. To distribute these simulations to different machines and to make the management of jobs easier, one could consider using Condor. The Condor submit file (named `Rjobs.submit`) for the jobs contains

```
Universe        = vanilla
Executable      = /path/to/R
Getenv          = true
Arguments       = --vanilla
Input           = foo.R
Error           = foo.err
Log             = foo.log

Environment     = i=20
Output          = foo_1.out
Queue

Environment     = i=25
Output          = foo_2.out
Queue

Environment     = i=30
Output          = foo_3.out
Queue
```

Note in the submit file, there are 2 occurrences of `vanilla`. The first one is the Condor universe under which the R jobs run. The second occurrence tells R not to load the initialization files and not to save the '.RData' file, among other things. We let all the jobs shared the same log file and error file to reduce the number of files generated by Condor. To pass different inputs to the R code, we used one environmental variables `i`, which can be retrieved in R with `i <- as.numeric(Sys.getenv("i"))`. To submit the jobs to Condor, one can do

```
% condor_submit Rjobs.submit
Submitting job(s)...
Logging submit event(s)...
3 job(s) submitted to cluster 6.
```

The output shows 3 jobs were submitted to one cluster, the submit events being logged in the given log file. To check the Condor queue, one can do

```
% condor_q

-- Submitter: gaia.stat.wisc.edu : <128.105.5.24:34459> : ...
 ID      OWNER    SUBMITTED    RUN_TIME ST PRI SIZE CMD
   6.0   xie     5/17 15:27  0+00:00:00 I  0   0.0  R --vanilla
   6.1   xie     5/17 15:27  0+00:00:00 I  0   0.0  R --vanilla
   6.2   xie     5/17 15:27  0+00:00:00 I  0   0.0  R --vanilla

3 jobs; 3 idle, 0 running, 0 held
```

The jobs are all idle at first. When they are finished, the Condor queue will become empty and one can check the results of the jobs.

Why Use Condor DAG?

Due to the restrictions of Condor, there is no Condor Standard universe version of R. This means R can only be run under the Vanilla universe of Condor. In the scenario described before, long-running Vanilla jobs could be evicted many times without making any progress. But when we divide the big jobs into smaller ones (say 3 four-hour jobs instead of one 12-hour job), there is a much better chance that all the smaller jobs will finish. Since the smaller jobs are parts of the big job, there are usually some dependencies between them. The Condor Directed Acyclic Graph (DAG) system was created to handle Condor jobs with dependencies.

Brief Description of Condor DAG

In a Condor DAG, each node represents a Condor job, which is specified by a Condor submit file. The whole graph is specified by a DAG file, where the lists of the nodes, relationships between the nodes, macro variables for each node, and other things are described. When a DAG job is submitted to Condor, Condor deploys the parent jobs in the DAG first; a child job is not deployed until all its parents have been finished. Condor does this for all the jobs in the DAG file.

An Example of R and Condor DAG

Here is a fake example in which a big R job is divided into 3 parts. We suppose they can only be run sequentially. And we want to replicate the sequence 100 times. The Condor DAG file will look like this

```
Job  A1  Rjob_step1.submit
Job  B1  Rjob_step2.submit
Job  C1  Rjob_step3.submit
Job  A2  Rjob_step1.submit
Job  B2  Rjob_step2.submit
Job  C2  Rjob_step3.submit
...
Parent  A1  Child  B1
Parent  B1  Child  C1
```

```
...
Vars  A1  i="1"
Vars  B1  i="1"
Vars  C1  i="1"
...
```

Notice in the DAG file, we let all the R jobs for the same step share one submit file. And we pass a macro variable `i` to each job to identify which replicate the job belongs to. The submit file for step one of the replicate looks like this

```
Universe     = vanilla
Executable   = /path/to/R
Arguments    = --vanilla
Input        = foo_test_step1.R
Getenv       = true
Environment  = i=$(i)
Error        = foo_test.err
Log          = foo_test.log
Notification = never
Queue
```

In the submit file, we used the macro substitution facility `$(var)` provided by Condor. The environment variable `i` can be retrieved in R as before. This variable is used in generating the names of the files for input and output within the R code. Step 2 of one replicate will read the data generated by step 1 of the same replicate. So is the case with step 3 and step 2 of the same replicate. We set `Notification` to be `never` in the submit file to disable excessive notification from Condor, because we have 100 replicates.

To run the DAG job we just created, we can use the following command

```
condor_submit_dag foo_test.dag
```

say, where 'foo_test.dag' is the name of the DAG file.

To check how the DAG jobs are doing, we use the command `condor_q -dag`. If one wants to remove all the DAG jobs for some reason, he or she can do the following `condor_rm` *dagmanid*, where *dagmanid* is the id of the DAG manager job that controls all the DAG jobs. Condor starts one such job for each DAG job submitted via `condor_submit_dag`.

Generating DAG File with R Script

Notice the DAG file given before has a regular structure. To save the repetitive typing, a little R scripting is in order here. The R code snippet for generating the DAG file is given in the following.

```
con <- file("foo_test.dag", "wt")
for (i in 1:100) {
  cat("Job\tA", i, "\tRjob_step1.submit\n",
      sep="", file=con)
  cat("Job\tB", i, "\tRjob_step2.submit\n",
      sep="", file=con)
  cat("Job\tC", i, "\tRjob_step3.submit\n",
      sep="", file=con)
}
...
close(con)
```

Other Ways of Parallel Computing

In the article, we discussed a way of dividing big R jobs into smaller pieces and distributing independent jobs to different machines. This in principle is parallel computing. Some other popular mechanisms for doing this are PVM and MPI. There are R packages called Rmpi and rpvm, which provide interfaces for MPI and PVM respectively in R. It is unwise to argue which way is better, Condor DAG or Rmpi/rpvm. Our way of doing parallel computing should be considered as an alternative to the existing methods. Condor has 2 universes for PVM jobs and MPI jobs too. But they require the code be written in C or C++.

Conclusion

Condor is a powerful tool for batch processing. DAG is a very nice feature of Condor (especially DAG's potential for parallel computing). To run the statistical computing jobs in R that take very long to finish, we can divide the R jobs into smaller ones and make use of the DAG capability of Condor extensively. The combination of Condor DAG and R makes the managing of R jobs easier. And we can get more long running R jobs done under Condor in a reasonable amount of time.

Thanks

The author would like to thank Dr. Douglas Bates for suggesting that I write this article.

Xianhong Xie
University of Wisconsin-Madison, USA
xie@stat.wisc.edu