A tutorial on slurm system and rslurm

Javad Rahimikollu,Kayhan Batmanghelich

May 14, 2018, 16:49

Contents

1 Introduction

Simple Linux Utility for Resource Management(Slurm) is an open source job scheduling system for large and small Linux clusters. Slurm has three key functions:

- It allows access to resources (compute nodes) based on certain criteria and parametrs for users to perform computations.
- It ensbles users with a framework for starting, executing, and monitoring work on a group of assigned compute nodes.
- It manages high number of access requests to compute nodes by assigning them in a queue.

In slurm cluster systems, user first sign in to the head node and using and request access to computing nodes based on job parameters. Here is the list of some of these parameters:

```
Slurm_Options<-read.table("Slurm_Options.csv",header = TRUE, sep=",")
pander(Slurm_Options)
```

option	Description
-j	jobname
-t	Walltime requested HH:MM:SS
-p	Partition
-N	Number of Nodes
-A	Account or Group Name
–mem	Requeted Memory in GB
–ntasks-per-node	Number of cores to allocate per node

2 Submit Slurm Job

To submit the slurm job, one can create a bash script with all the paramters and use **sbatch** command to submit the job. Here is an example of a batch script. If we name this bash scipt as test.sh, we can use sbatch test.sh to submit this job to run Test.R.

```
#!/bin/bash -l
#SBATCH - N 1
#SBATCH -P DBMI
#SBATCH -A bi561ip
#SBATCH -t 00:20:00
#SBATCH -J my_job
#SBATCH --mem 10GB

Rscript ~/Test.R
```

3 Rslurm

Lets assume, we want to submit a rscript with iterative argument. Using the bash script we have to submit one job for each setting of argument which will need bash programming to enumerate all the values of argument. The rslurm package facilitates the process of distributing this type of calculation over computing nodes in the Slurm workload manager. The main function, slurm_apply, automatically splits the computation across multiple nodes and writes the submission scripts. It also includes functions to retrieve and combine the output from different nodes, as well as wrappers for common Slurm commands.

3.1 Basic Example

Lets say our task is to generate random values from gaussian distribution in three different settings and calculate the mean and standard deviation for each setting. Lets define a function called *test_func*.

```
test_func <- function(par_mu, par_sd) {
    samp <- rnorm(10^6, par_mu, par_sd)
    c(s_mu = mean(samp), s_sd = sd(samp))
}
```

To pass the different values of arguments *par_mu* and *par_sd* we store them in a dataframe.

```
pars <- data.frame(par_mu = 1:10,
                   par_sd = seq(0.1, 1, length.out = 10))
head(pars, 3)
```

```
##   par_mu par_sd
## 1      1    0.1
## 2      2    0.2
## 3      3    0.3
```

Now, we can treat the *test_func* as the rscript which will use the *pars dataframe* and submit the job for each row *pars dataframe* using *slurm_apply*.

```
library(rslurm)
sjob <- slurm_apply(test_func, pars, jobname = 'test_apply',nodes = 2, cpus_per_node = 2, submit = TRUE)
```

on the background, slurm_apply will create a submit script with the job options provided in the function argument.i.e nodes and etc. Folders with the name of *results_node_indicator.RDS* file for each node. As we can see below there are two folders with the name of *results_0.RDS* and *results_1.RDS* for *N=2* nodes.

```
list.files('_rslurm_test_apply', 'results')
```

```
## [1] "results_0.RDS" "results_1.RDS"
```

Now, we can use *get_slurm_out* to get the output for the each row of *par* dataframe on the *test_func* function.

```
res <- get_slurm_out(sjob, outtype = 'table')
head(res, 3)
```

```
##       s_mu      s_sd
## 1 1.000144 0.1000350
## 2 2.000055 0.1999433
## 3 3.000130 0.3002951
```

3.2 Machine Learning Example

Support vector machines are one of the techniques which widely used in machine learning. Lets assume that we want to find the hyperamters of RBF kernels,*sigma* and C. In this section we want to performa a grid search using rslurm.

First lets define the *par* data frame for this problem.

```
pars <- data.frame(par_cost = 0.1:1,
                   par_sd = seq(0.1, 1, length.out = 10))
head(pars, 3)
```

```
##   par_cost par_sd
## 1      0.1    0.1
## 2      0.1    0.2
## 3      0.1    0.3
```

Now lets define our main function to fit the data:

```
svm_func<-function(par_cost,par_sd){
set.seed(7);




}
```