Parallel Computation

Shuhua Liang

March 19, 2013

In addition to combining the Unix shell, SQLite and R to speed up computations on large datasets, parallel computation also reduces computing time. With the set of airline delay data, we are interested in sampling 100,000 observations from each year and apply the random forest method. sampling directly from R is uneasy because loading the 22 .csv files to R is inefficient.

To obtain a sample of 100,000 observations from each year, we need the observation count for each year (each .csv file), and we could call the shell in R to obtain this. Then we would use R to simulate the indices of the samples from each set of observation count. Once this is done, we could sort the sampled indices, and then read in the lines of the original .csv files to R through a connection and select the sampled lines. These lines will be written into new .csv files that will be used later.

In terms of parallel computation, we can create local clusters and have the clusters split up the work when writing 22 new files. With a computer with four cores, we can create four local clusters, and then we can apply the sampling function to the files in parallel.

The idea of working in parallel speeds up our process by four times as we would repeat the process for each files. If we apply the sampling process for the 1987.csv file, it would take up 6.866 user time, 0.873 system time, and total of 9.774 elapsed time. However, if we were to apply parallel computation, for all 22 files, the user time would still reduce to 0.005 and system time to 0.006.

Once we obtained the new files, we can load them into R for further processes. However, to minimize the time and and work on loading the individual files, we can create a database in SQLite and then load the flat data into R.

Next, we can fit a random forest to the sample in parallel. Here, we can create eight clusters and fit ntree separate trees to the sample. In our case, we would fit

15 ntree separate trees. Because we have a large amount of data, and the memory is limited R, we would first need to reduce our dataset. On a 32 bit machine, we are limited to 32 categories in each variable; therefore, for variables "Origin" and "Dest" with more than 32 airport needs to be reorganized. For simplicity, we would eliminate the Dest variable, and select data from the 32 largest airports in the Origin variable.

The previous step solves the limitation in categories; however, there is a memory limit of no more than 4GB on a 32-bit OS machine, we would need to trim the data even more in order to fit the random forests. One way to do so is to pick every 15th line of data from the sample as a new dataset.

Fitting the random forests outputs 15 calls of the model with arrival delay as the response variable and all other variable as explanatory variables. The outputs show that at each fit, the function tried using 6 variables at each split, and approximately 70% of the variables are explained. Also, by default, ntree = 500 for each fit. Next, we can predict the performance of each individual trees in parallel, and then we can combine the predictions into one vote.

Bagging aggregates the fits across different bootstrap samples, while random forests selects random subsets of the predictor variables at each node in each tree. By randomizing the variables, the random forest method boosts the performance and usefulness of the variables at each split; therefore, the random forest method is more informative. If we were to specify a random subset of predictors for fitting the entire tree, this would definitely affect the amount of variables being explained and the mean of squared residuals will decrease as the random variables gets more informative. Also, this can be done with parallel computations. Note: the sample in our case is not a random sample because we generated 100,000 samples from each year, and not all years have the same total amount of data. In other words, not all data lines has the same chance of being selected.

Overall, parallel computation is a tool that is helpful when we have a large amount of data because it turns on multiple machines to operate multiple jobs at the same time. Therefore, we can reduce much processing time. In the future, this tool can be applied when we have large amount of data or a function that needs repeated computations such as the Markov Chain and Metropolis Hastings.