Big Data

Shilo Wilson

Mar 16, 2017

norm.py

**Goal**

The purpose of norm.py is to take a file of form date {yyyymmdd:hh:mm:ss.ffffff}, price {#.##}, volume {integer} which has been scrubbed, and test for normality..

**Execution**

This program begins, after initializing MPI by reading in parallel. I took this approach, as I did not think it necessary to read the data with a single processor, and scatter it, as at very large sizes, the program will be limited by the IO time.

The read was initially intended to be an MPI.Iread, which is non-blocking and allows all processors to read simultaneously. This however did not work, as my the buffer would often read in null values, and print straight to noise, not only skipping its assigned section, but making the noise file very large with null values. I believe, had I tested Iread on Penzais (if it didn’t crash) the parallel IO system would have corrected that. On large files, just splitting the file into segments may not be enough. Each processor should also tackle their assignment in blocks. In norm.py, each processor, regardless of its segment size, tackles 10mb at a time.

The first step of the normality test is to sort the block by the dates, this is done rapidly, as it’s a local block of only 10mb, then to parse the dates and the price. I attempted to convert dates to an integer form, as its faster, however this was not successful. Once that is complete, I loop through each record, in all instances recording the current price and date to use as previous values in the next iteration. I compute log returns, as this allows a linear relationship between time and return, instead of a continuous relationship. This was necessary as the time period was not a fixed interval. In this respect each log return is “standardized” with the respect to time. Once log return is computed, I use list comprehension to generate a list of each return squared, cubed, and to the fourth power. I then aggregate the sums of these in a variable, that persist through each block. After each 10mb block is processed, these variables contain the sums of number of observations, log returns, log returns squared, log returns cubed, and log returns to the fourth. It may be faster to map these function, however profiling shows most time is spent in date parsing. These values are then reduced to process 0, which uses the values to compute the mean, standard deviation, skewness, and kurtosis. A t-test is then conducted on skewness and kurtosis with a .05 alpha. If both test fail to reject the null, the user is notified the data may be normal, else the user is notified the data is not normal. In all instances, the aforementioned statistics are reported Duplicates are dealt with a try/except block. A duplicate tuple will cause an error of dividing by 0 as the timedelta is 0. This value is simply ignored, with the benefit of still gathering data from the value as if it were a single record, as it computes a return for the previous and next iteration.

**Drawbacks**

Based on my experimentation, up to a 20gb file (generated by simply duplicating the original file and adding ridges to the time), the norm.py file does scale with size. However, the scalability is not realistic, as it is generally slow. Tested on a 4 processor system, it took about 15 seconds per 10mb block, per process, causing it to take about two and a half hours, by comparison, scrub took about 15 minutes. That means, that in order to scale, more processors must be used. This is likely a result of inexperienced or inefficient programming, especially when it comes to parseing dates. Throughout the project, in both programs, it seems I had to resort to brute force methods to complete the assigned task.