Big Data

Shilo Wilson

Mar 16, 2017

Scrub.py

**Goal**

The purpose of scrub.py is to take a file of form date {yyyymmdd:hh:mm:ss.ffffff}, price {#.##}, volume {integer} and filter out “noise”. Noise in this instance is defined a malformed, or clearly inaccurate data. It must be able to scale, and operate in parallel.

**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. This program also, in parallel writes the clean and dirty lines to separate files.

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 scrub.py, each processor, regardless of its segment size, tackles 10mb at a time.

The filter is based on a single general expression. This general expression confirms that the date matches the appropriate format, the price is above 0, and less than 10,000, and that the volume is a positive integer, with no malformation across the line. The original approach was to parse the data into the correct data types, and run functional test on the individual fields. This however, took a significant amount of time, and did not justify the cost. Very few bad data points, such as extreme dates, price, or volume make it past the current test. Additionally, due to the sheer size of the data set, the individual data points that make it through are unlikely to significantly affect the outcome of the normality test, and if enough data points exist to affect the normality test, they are likely not mere noise, but an indication of the nature of the data. While parsing data, each processor finished a 10mb block in 25 seconds, however the current method, with no logger or profiler, can complete the same task in .77 seconds.

Finally, each processor writes its block, in parallel to the appropriate output file (signal and noise) which are taken as command line arguments. The write is done using a shared file pointer, and MPI.Write\_ordered. This allows signal to remain in relative order.

**Drawbacks**

The largest drawback is that scrub.py does not test for duplicates, while there are many. This was intentional, as the time it takes to search for duplicates did not justify the cost. While searching the last 1000, perhaps may be cheap, many data points were more than 1000 out of order, and still needed to be addressed in norm.py. If this was already being addressed in norm.py, it did not make sense to expend any effort in scrub.py.