18.337: Hubway and Big Data

Sean Whipple

December 17, 2012

Introduction

"Big Data" is a major area of development and research both in industry and in academics. A world with Google, Facebook, insurance companies, stock trading, etc. generates massive amount of data constantly. Data generation is nothing without a meaningful way to interpret that data and companies are constantly looking for methods of analyzing that data.

Hubway released all of the data it had pertaining the trips made in the Boston area for a visualization project. This created an opportunity for data analysis on a reasonably large data set. The data as released was approximately 1.2 GB worth of useful data. This would provide a reasonable starting point and I could test the limits of my analysis by expanding the data set.

Tools Used

To load and analyze the data I went with Julia to test the limits of its current big data capability. Because it was currently going through a major distributed array re-write at the time of implementation I was not able to use the latest version of Julia. However based upon the literature and the GitHub community all of the work that I did on the older version should port well to the new implementation of Julia when it is released. All code was developed and run on the julia.mit.edu using 8 cores generally.

Loading Big Data

The first major area step was to load the data into Julia. As stated earlier the distributed arrays were an excellent starting point for this. Because of the state Julia was in at the time of implementation distributed arrays could only be distributed across one dimension. This was not a major set back in this case because all of the

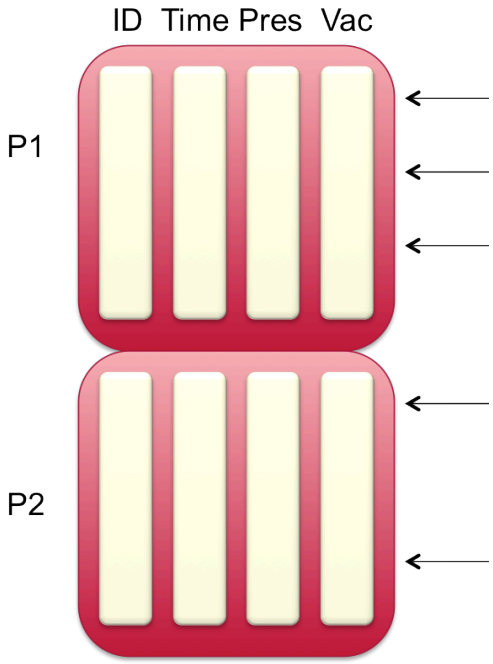
data structures I created were only large in one dimension. However larger data sets would certainly scale differently if two dimensions (or more) were necessary.

Loading the data was largely a game of trial and error. Distributed array construction was done in a manner where smaller chunks of data were gathered and stored on a local core. Once a sufficient size had been reached (largely an experiment to determine the appropriate size) it was concatenated onto the distributed array for storage in parallel. This method of data type construction requires a significant amount of copying which undoubtedly inhibits performance. It does however scale quite well and given that I wrote the code with the understanding that I would not know the file structure (particularly length) it was a reasonably successful solution.

Data Analysis

A company like Hubway is going to be very interested in some key operational metrics to help them understand how their product is being used. Number of stock outs at a station, average trip distance, etc all offer important insight into how Hubway can improve their stations around the Boston area.

The largest obstacle to the data analysis was the format the data was given in. To illustrate the example we will examine the data given for the station statuses by time. The figure below shows the distributed array structure in Julia.



The data is read in order as is given in the file (with non interesting data thrown out). The columns show station ID, time stamp of the update, number of bikes present and number of missing bikes. The arrows illustrate how a particular station ID is distributed throughout the array. Notice that they are not spaced evenly. This is intentional as this is how it appears in the data. Generally each time stamp has all 53 stations in the same order.

Normally this would lend itself quite well to analysis as one could batch reads into the distributed array and gather data in chunks. For example if one wanted to calculate the average number of bikes available or stock outs, it would be a simple task to batch read X entries (say 1000) do the analysis and grab the next batch. This could further be improved by doing that analysis in parallel, simple computations such as these are embarrassingly parallel and can be done independently.

However as stated before the data is not perfectly set up for this batch reading. Some entries are missing in the data (for unknown reasons) and batch reading cannot be applied. Randomly missing entries requires that a search be done to find the locations of the appropriate data. To apply this to parallel processing I first had

to do a search of the array to find the locations of all entries that I was to find interesting and store these locations. Once all locations had been found I would divide up the locations evenly (or as close to it as possible) across the processers and have them independently perform the analysis I would require. Results would then be combined. Batching the array reads and doing the analysis was not a major bottleneck. Searching over the entire distributed array for the locations was the major set back (this was never implemented in parallel and may be a key area for improvement). This is only a problem when one is looking for specific data (for example finding the average station capacity of station 5). If one wants to simply calculate the average distance of all the trips, batching the data in order and calculating is not an issue.

Next Steps

As it stands now the data is loaded into Julia fairly well (and I have yet to hit Julia's limits in that regard). Once the distributed array re-write allows for distributing arrays across all dimensions Julia looks like it will be able to handle the storage issue effectively (given appropriate hardware).

The data that Hubway has generated lends itself to embarrassingly parallel analysis. But the accessing of the data is problematic. Even in the station status data where there is an expected pattern of the data format, inconsistencies generate a tremendous amount of difficulty in accessing the data efficiently. A database structure that allows for efficient organizing and access of the data is the clear solution to the problem. There are currently some libraries on the GitHub community to solve this problem but I did not have sufficient time to implement given the amount of time remaining once I had found the libraries.

Acknowledgements

I would like to thank Alan Edelman, Jeff Bezanson, the Fall 2012 18.337 class and the JuliaLang GitHub community for all their support throughout the project. Overall this was an excellent learning experience both in the Julia Language but with parallel computing in general. My project was not quite as successful as I would have liked it to have been but the lessons learned will be invaluable.