**Task-Based Software Engineering**

**Report**

**Rhys Jones**

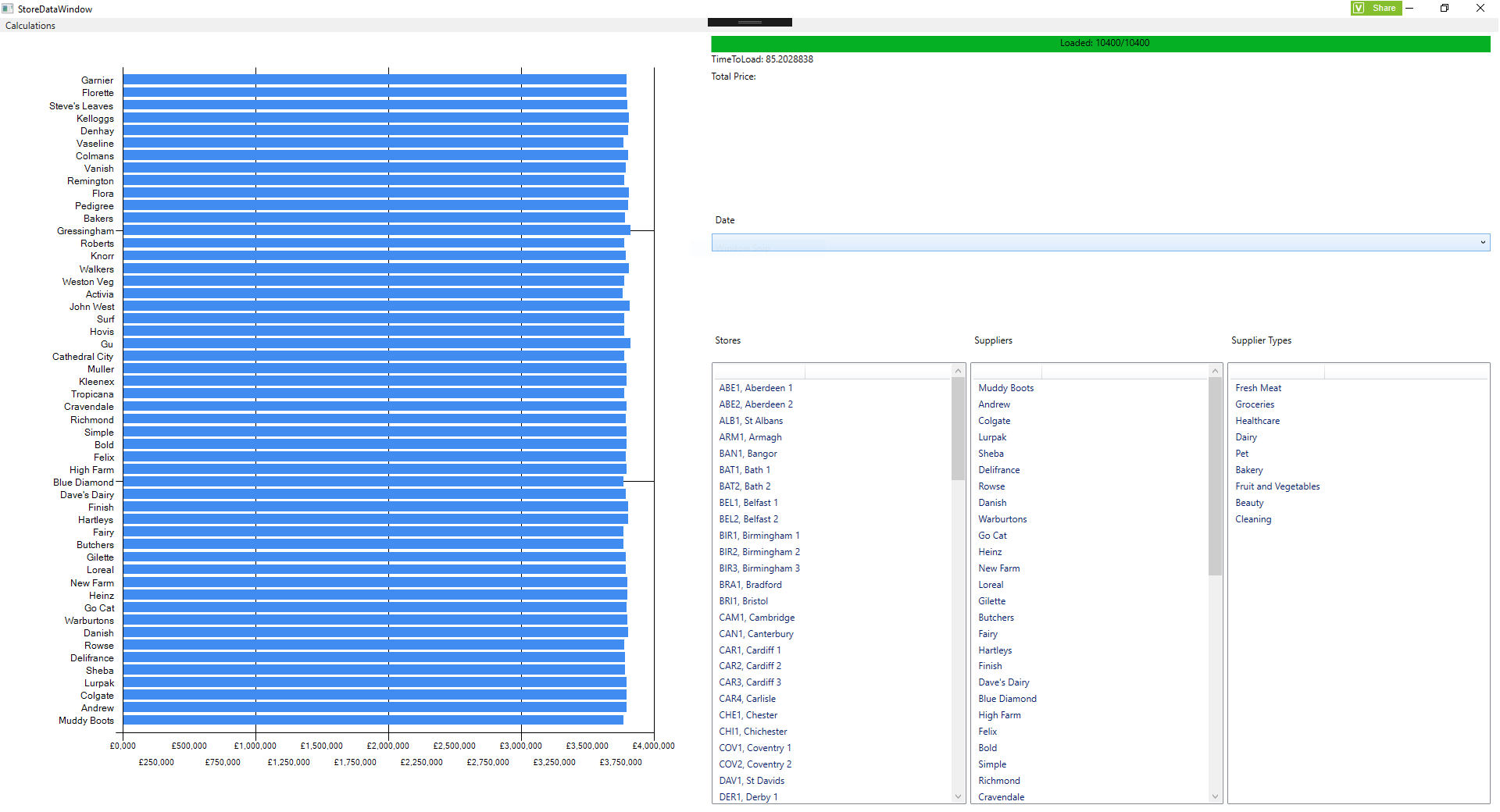
COSE50584

Staffordshire University, College Road, Stoke-on-Trent ST4 2DE

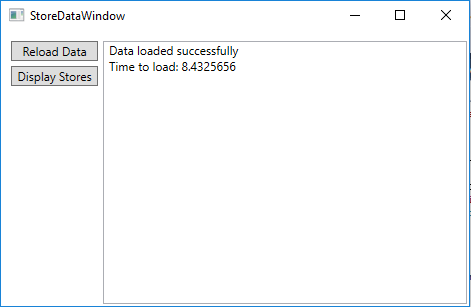
[Rhyswj94@gmail.com](mailto:Rhyswj94@gmail.com)

For this assignment I was tasked with engineering software to calculate the total cost of orders for a number of different stores over 2 years, using the most efficient approach possible, while testing both parallel and sequential approaches.

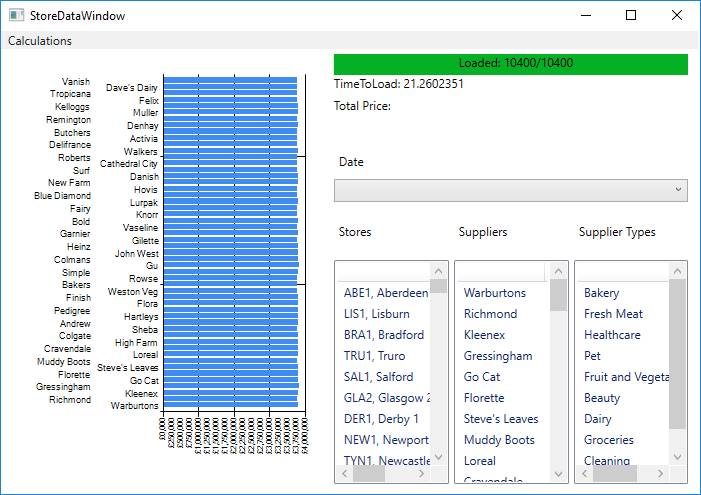
I started with sequential and loading the data, this took quite a while however, taking up to 85 seconds as you can see below. (The GUI is updated as I lost the original sequential screenshot)



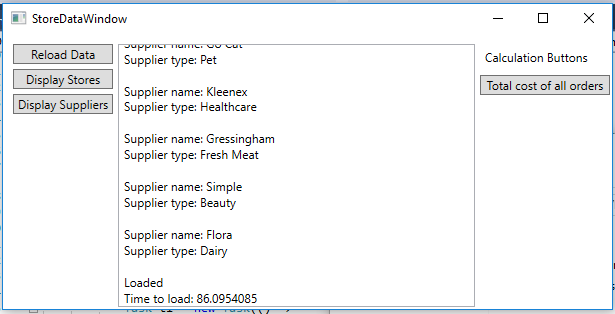
This improved considerably with the use of a parallel for each loop, going down to only 8.5 seconds;



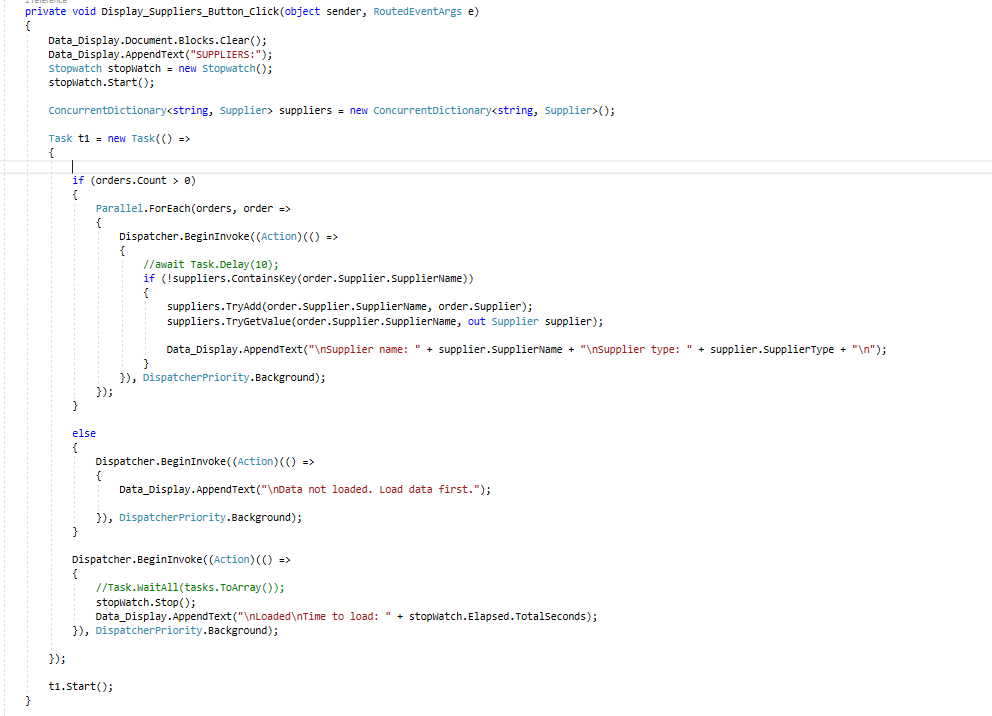
However at the conclusion of the project, I decided that after loading all of the data, I wanted to use LINQ to get specific elements of the data to populate lists for use in the calculations, this took the loading time up to 21 seconds, however the loading of the data still only takes 8 seconds, it’s just displaying the data and populating the chart that takes more time. I felt it was necessary however, due to the method I had decided to use to allow calculations to be performed.



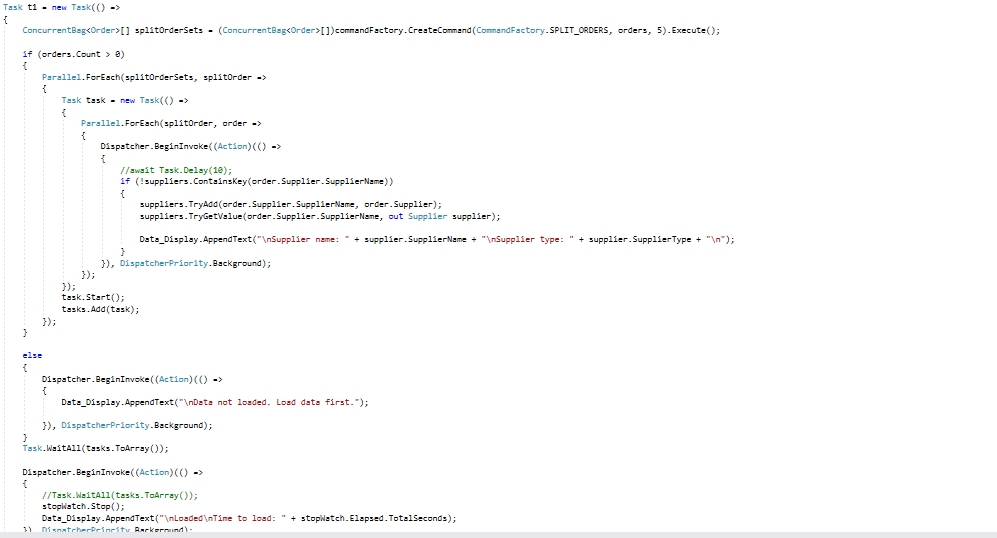
Before this however, I had a different method of displaying the suppliers, shown below.

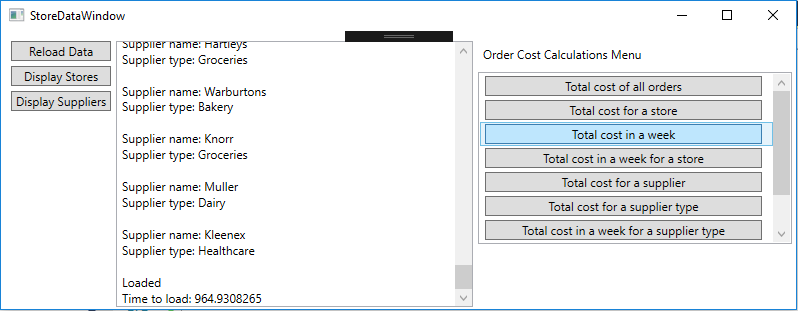


This was using the following code.

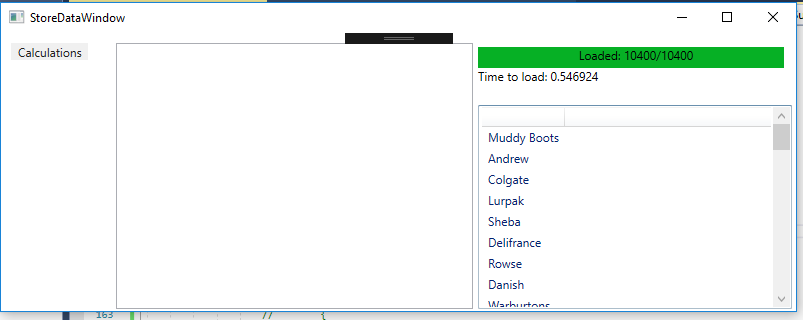


Due to the amount of time this took, I decided to change this, by splitting the data into 5 portions and then using tasks to deal with each 5 portions separately before putting it back together and displaying it.



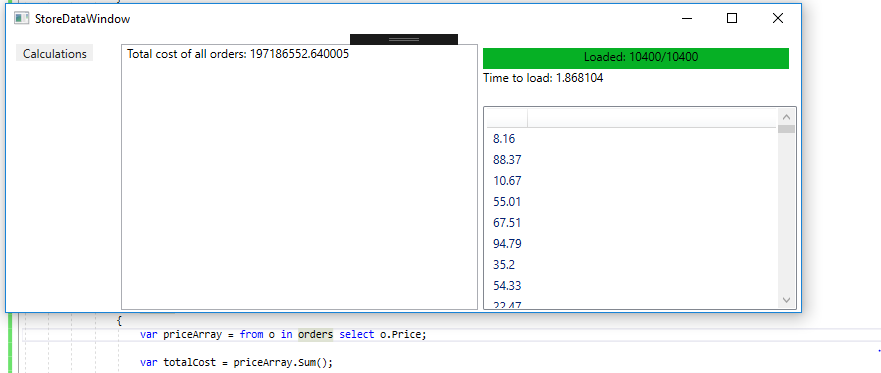


This increased the time by a large amount, making this method of separating data completely unusable for the application. I then tried LINQ for this, and saw a vast improvement, from 86 seconds all the way down to 0.5 seconds.

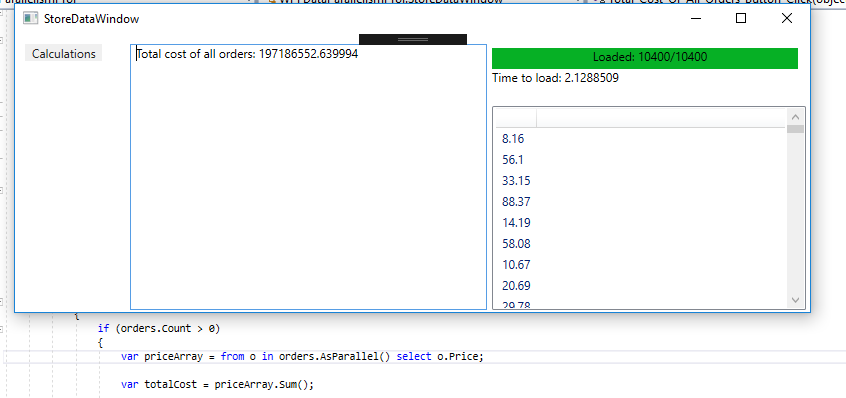




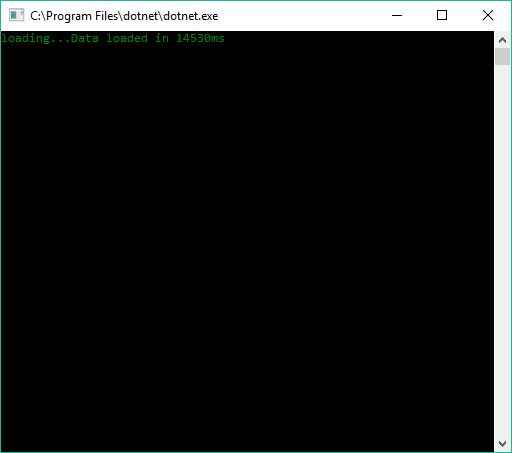
After the extremely successful results of the LINQ compared to other methods, I then used it for calculating the total of all of the orders, once again seeing extremely good results.



Having seen the results of LINQ compared to other methods, especially the vast difference between them, I decided that it was highly unlikely any other method could be faster, so I decided to compare this to PLINQ, giving the following results.



These results were consistent for all of the calculations, and as you can see, PLINQ was a little slower than LINQ, therefore I decided to use LINQ for the rest of my C#, and F# implementation, as the results were also consistent within F#. Given the speed of the LINQ calculations within F# I only noted the speed at which the data loaded, finding it to be moderately slower than the C# application before it was displaying all of the data. The C# was loading in 8 seconds, whereas the F# is loading in 14 seconds, as seen below.



F# is a succinct functional language, which in terms of efficiency, was somewhat disappointing, given it’s considered it’s main feature. This could however simply be that my logic was more efficient within C# than it is in F#. I found the language to have little documentation, making the use of some methods and objects a little difficult, for example, when I am loading data, I used a lock when adding to a hashed set of orders, within C# lock is well documented and easy to use, whilst within F# there is little to no documentation and it didn’t work the way I had intended. Meaning I had to find other solutions from the software community.

I found the language to be much different to anything I am used to, I found it distinctly opaque and I find it difficult to imagine I would use it again.

**Architecture**

The Software implementation used a simple command factory, with a singleton file handler to hold and keep the file location chosen at the beginning of the applications life. After programming the application, i’m not certain that there was a need for multiple handlers, however it allows the program to be open to further development and keep logic adequately seperated.