



Performance Analysis of Serial vs Parallel Web API Calls to a Niche Web Service

Hywel Jones - Software Engineering Degree

1. With the proliferation of web services and thus APIs, being able to call these efficiently is of paramount importance. This project is to design, analyse and implement a serial and parallel data extraction service that calls a Niche API and then compare the different result sets based on implementation, hardware and dataset size.

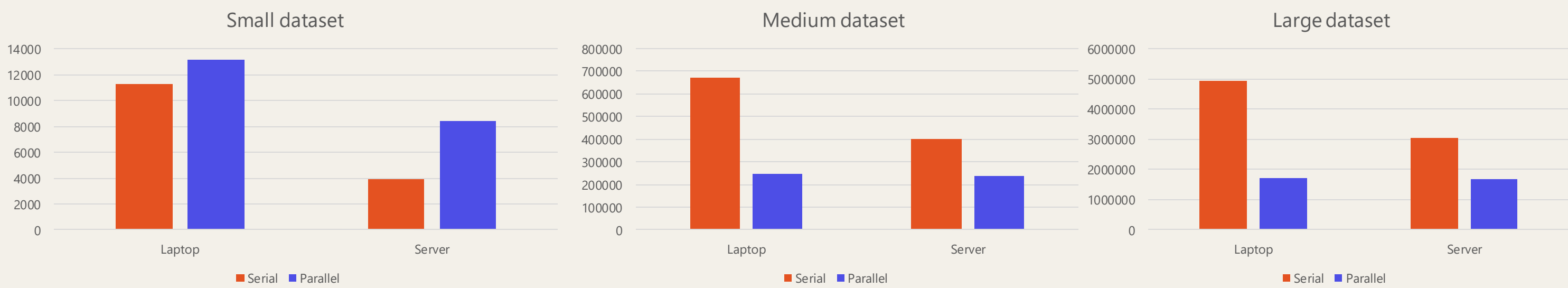
2. The first task was to establish if the problem was one that could be resolved in parallel. This meant breaking up the problem into its component parts, our total time "T_{tot}" is made up of the serial and parallel parts. The parallel part "T_{Par}" is the time for each record, each of which is independent and can be ran in parallel.

$$T_{Tot} = T_{Ser} + T_{Par}$$
$$T_{Par} = T_{Record}$$

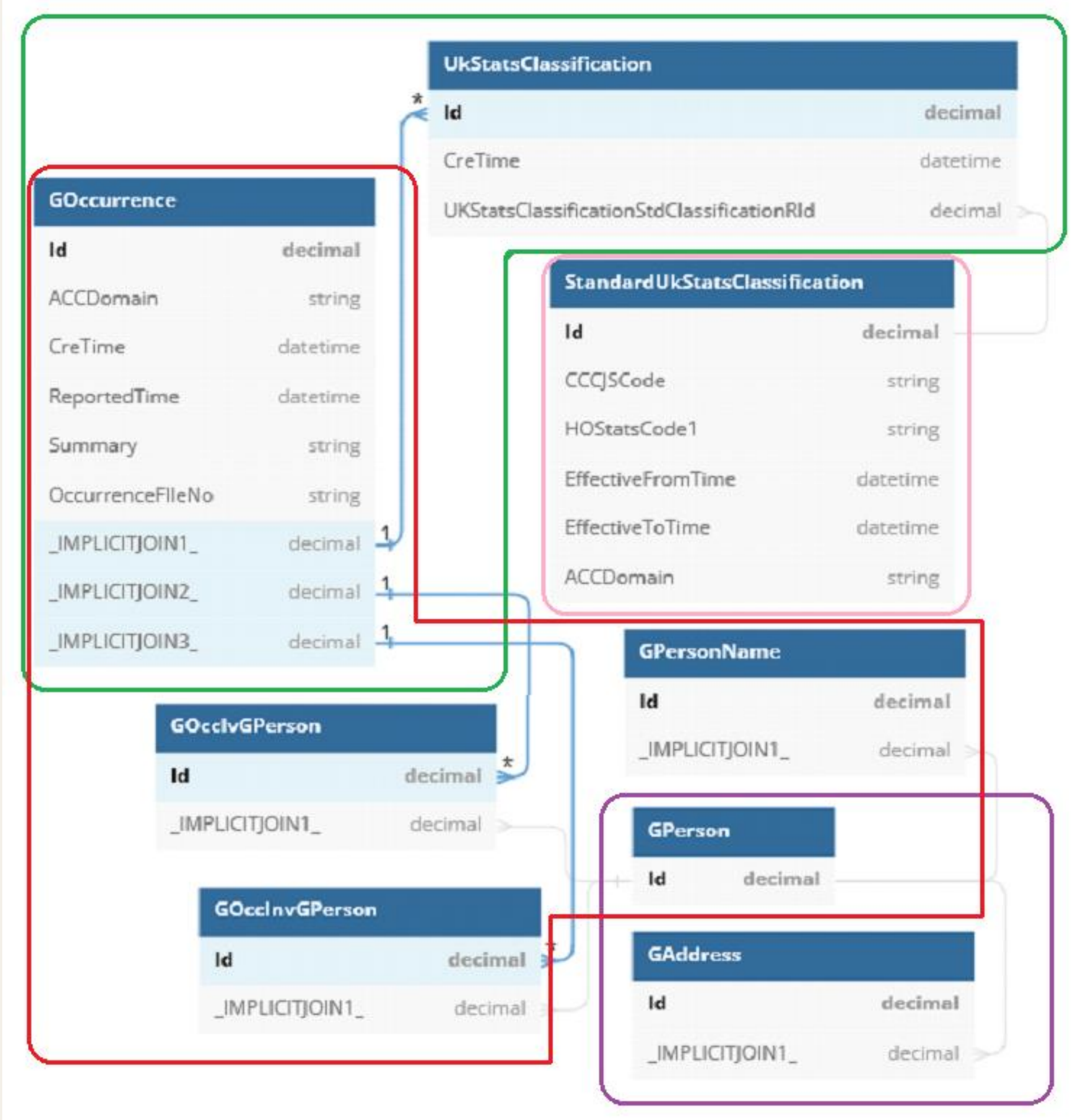
4. After completing the app SQL queries, the serial and parallel C# code can be implemented. Here is an example of the serial implementation with the comments above each foreach loop where these would be changed to use the C# 'Parallel' class.

```
List<CrimeTypeID> crimeTypeIDs = GetCrimeTypeID();
List<Occurrences> occurrences = GetOccurrences().Where(occ => crimeTypeIDs.Contains(occ.CrimeID));
//Parallelise this loop
foreach(Occurrence occ in occurrences)
{
    occ.policeNominals = GetNominals("Police", occ.Id);
    occ.nonPoliceNominals = GetNominals("NonPolice", occ.Id);
    //Parallelise this loop
    foreach(Nominal nominal in nonPoliceNominals)
    {
        nominal.addresses = GetAddresses(nominal.Id);
    }
}
```

7. Looking deeper at the results against different size data sets and on different hardware setups, we get some interesting comparisons. Below are the three data sets: 40, 3000 and 25000 records.



8. We can conclude that by using standard C# classes such as the 'Parallel' class we can effectively code a highly parallel solution that can deliver actual tangible benefits in a real-world scenario. The low performance of the parallel implementation on the small dataset is expected, it is a high percentage difference but a low nominal difference thus this might need to be considered depending on the use cases. The parallel implementation seems to be especially useful when operating on substandard or hardware that has latency as we can see a very large performance increase on the laptop during the large datasets. We can also use the 'Parallel' implementation to create parallel loops in parallel loops which can allow large datasets with large child record sets to also take advantage of the increased performance.

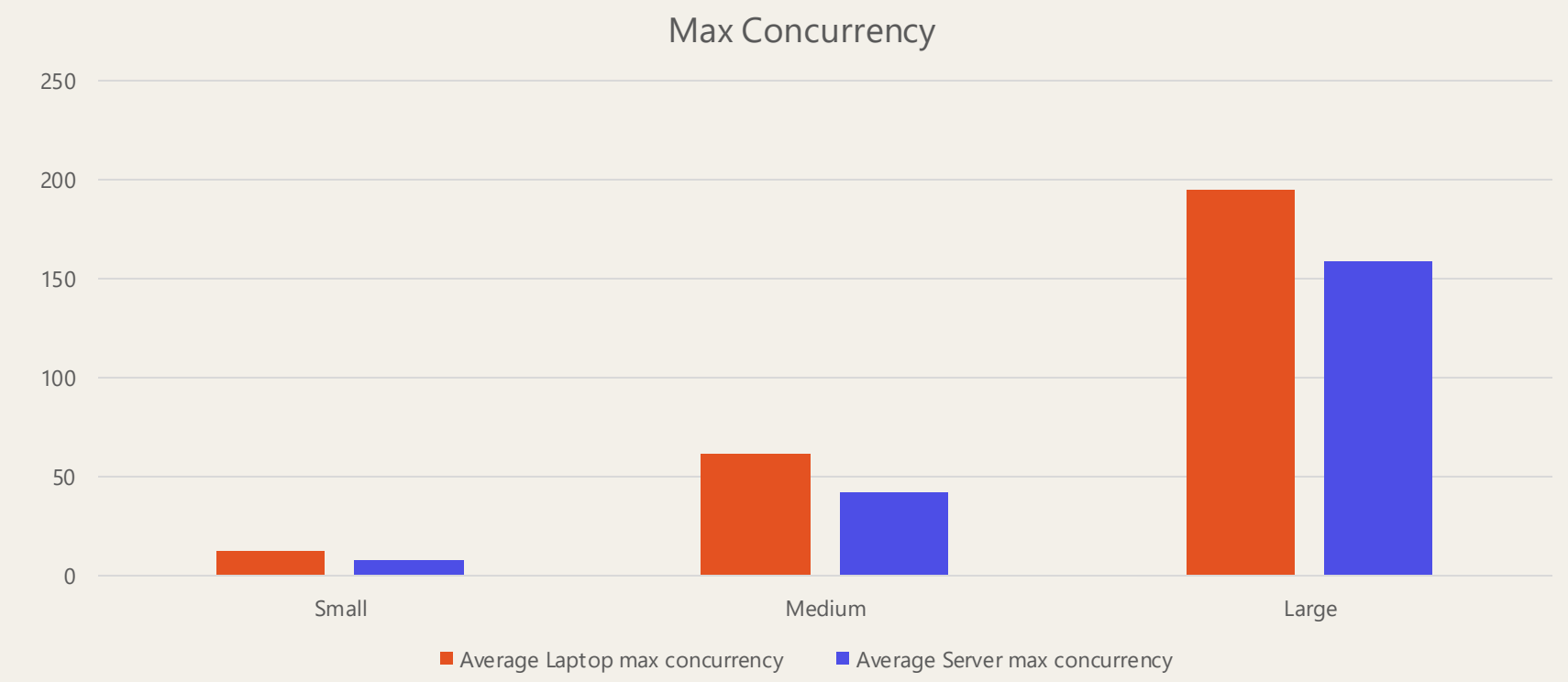


3. Another one of the challenges was to create the app SQL queries to send to the Niche API. In a similar way to a subway map we can visualise the breakdown of the entities, the entity queries are grouped by colour. The queries thus being in order of:

- Query for crime types [pink].
- Staging the occurrence IDs by crime type [green].
- Gathering the nominal structures, victims and police nominals, along with nominal names [red].
- Gathering the addresses on the nominals, addresses include physical addresses, mobile numbers and emails [purple].

These queries, to extract data for a single record, can be parallelised along with the parallel querying of the record set itself.

5. Our main result sets are the run times for the different hardware by set size., we can see these on the right here. On these graphs the red bars represent the serial implementation time to completion whilst the blue is the parallel implementation time to completion. We can see plainly that the serial increase dramatically as the set size increases. We can also see that the parallel implementations save significantly more time the larger the result sets are. Another thing to note is the servers faster run times of the serial implementations yet similar completion times when running the parallel implementation.



6. We can also see the maximum level of concurrency, graph above, that we have reached on each data set and by hardware. Interestingly the concurrency is consistently higher on the laptop, suggesting that the underlying parallel implementation increases concurrency if the running threads are taking longer but further research would be needed to clarify this behaviour.

