2017

FIT3140 Assignment 4

REPORT

DEKEL PILLI & JAMES PICKERING

Table of Contents

The data formats

Our work consisted of two separate spikes, as described below.

JSON

JSON, or Javascript Object Notation, is a potentially human-readable data format. The structure and syntax of its data is similar to that of hash maps in Python (and other languages), with potential for nested 'hash maps':

```
{
        fieldname: content,
        fieldname2: {
                subfieldname: content,
                subfieldname2: content
                },
        fieldname3: content
}
```

Despite being inefficient in terms of the storage space it requires, its human readability makes it a popular choice for data storage, including Google's Firebase.

MessagePack

MessagePack uses JSON for its storage structure, but then encodes the JSON data to achieve smaller file sizes. The downside to this approach is the need to encode any data before showing it to users. So, while transferring and storing information becomes easier, displaying it becomes a more computationally complicated task.

Protocol Buffer

Protocol Buffer is a data serialisation method developed by Google. While, like JSON, it is possible to store objects within objects, Protocol Buffer's syntax for doing this is very different:

```
message Object {
  string content = 1;
   int32 contentTwo = 2;
}
//A set of Objects
message Set {
   repeated Object objects = 1;
}
```

Unlike JSON, Protocol Buffer uses immutable types, reducing the flexibility of data entry. This means that the desired data structure needs to be defined before any data is entered. The downside of this is, of course, the additional setup time and reduced flexibility, but it enforces a consistent data

structure. Additionally, once data has been entered, it is serialised before it is written, meaning the data itself is not human readable until it is decoded.

Measurement process

To benchmark the efficiency of these protocols, we generated small (10,000 entries), medium (500,000), and large (1,000,000) version of each of these. While the following methods are intended to be repeatable, results may vary depending on variables such as system specifications, the operating system, and whether or not the machine used is virtual.

Once these sets of data have been generated for each of the three data formats, the server can be started. The server reads and writes the data and reports the time taken to complete this task. Once this is done for all nine files, the user can load *http://localhost:8080* to trigger the transferring of these files via sockets. Once each file is received, the time taken to receive that file is written to the browser's console. A more detailed explanation of how to use this program is available in the project's README.

Results

Due to technical difficulties with memory management, we were unable to collect information about how long it took the large files (with 1 million entries) to be received, although we do have information about their file sizes. As such, **the time information for large items shown below is an estimate** (based on the times for the small and medium files of that type).

Also, due to a bug that we were unable to fix, the file sizes reported by our program are double the actual size of the files generated. The file sizes details below are the file sizes as reported by Gnome Ubuntu 17.04, which considers each Megabyte to be 1,000,000 bytes, rather than 1,048,576 bytes.

Unsurprisingly, JSON produces the largest file sizes for all three size categories, as it is not compressed or serialised. Similarly, its transfer times are also the largest.

Despite having the smallest files, Protocol Buffer files were not the fastest to transfer. For the medium file sizes, Protocol Buffer was transferred at only 141.2MB/s, whereas MessagePack was transferred at almost double the speed, at 267.3MB/s. For comparison, JSON files were transferred at 157.7MB/s.

Additionally, the compressed sizes of these files is also noteworthy. While the large JSON file is 116% of the size of MessagePack files, and 131% of the size of Protocol Buffer files, the differences between the compressed file sizes were smaller. JSON's compressed file was only 84MB (105% of the size MessagePack's, 80MB, and almost identical (100.5%) to Protocol Buffer's compressed file, which was 83.6MB). This is not surprising, as the serialisation that MessagePack and Protocol Buffer files undergo is in itself a form of compression, and a file's size cannot be reduced infinitely.

Spikes

Our work was originally separated into three parts, to be performed in the following order:

1. Creating a tool that generates data for each of the data formats and file size categories.
    - The code for this stage is in *generateSyntheticData.py, objects.proto,* and *objects_pb2.py*.
2. Creating the client and server, that will communicate between each other and transfer the files generated in part 1.
    - The code for this stage is in the *Data Reader* folder.
3. Write this report.

Due to the aforementioned issues with part 2, the report was started before the results could be generated.

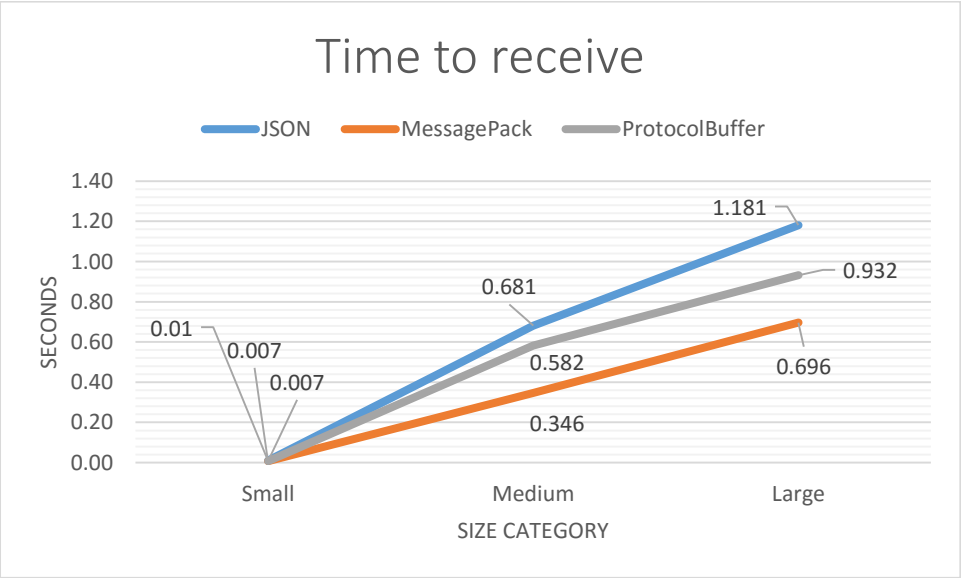
Conclusion

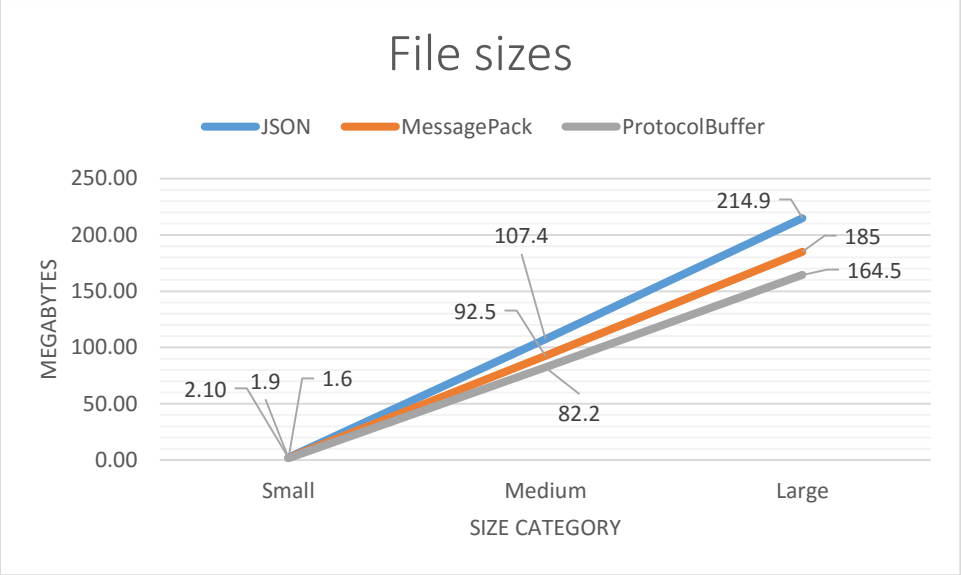
Each of these three data formats comes with its own set of pros and cons. The following are scenarios where each is best decision:

- In scenarios where human readability and ease of use are the highest priority aspects, the JSON data format is ideal. While MessagePack can be changed back into JSON, this is not as quick. A good example of one such platform is Firebase.
- In scenarios where lightweight communication and cheap storage are a high priority, but the user base are unlikely to be comfortable using a programming language, MessagePack is ideal.
- In scenarios where a strict enforcement of a pre-set data structure is important, or in scenarios where minimising local storage space is the highest priority, ProtocolBuffer is ideal, given that the user base is comfortable with the relatively difficult setup and maintenance.

Images

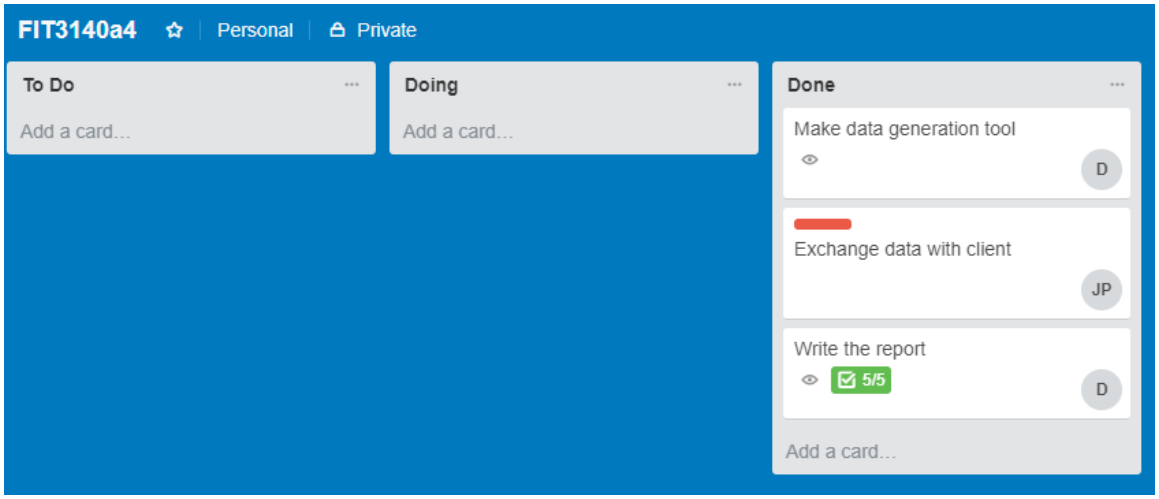
Results

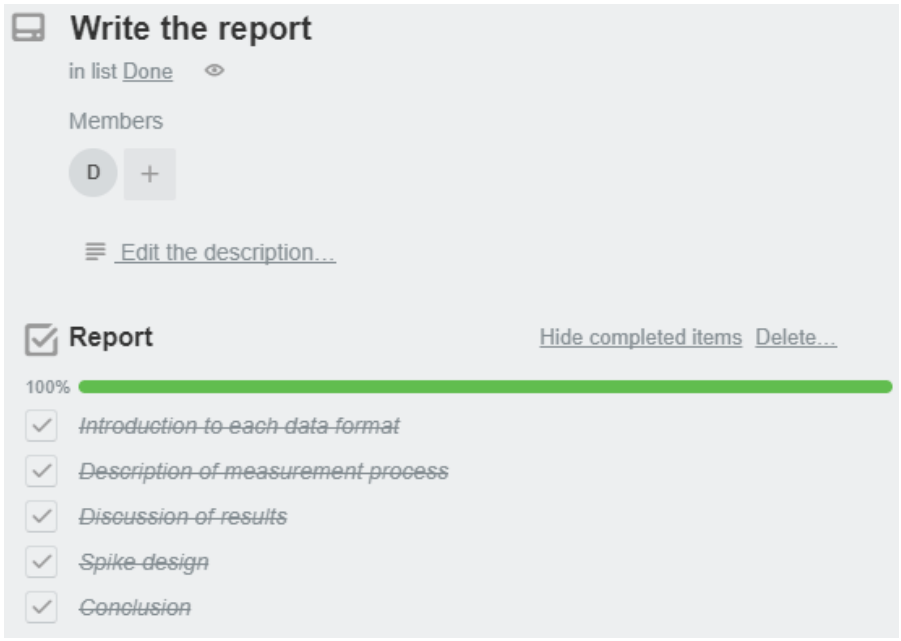




These graphs can also be found in *results.xslx.*

Trello





These are screenshots of our Trello board, which can be found here. Please email us if you want an invitation.

Harvest (time tracking)

Name ▲	Clients	Hours
Data generation tool	Nawfal	3.16
Exchange data with client	Nawfal	13.70
Exchange data with local storage	Nawfal	1.49
Report	Nawfal	3.71
Total		22.06

Name ▲	Hours
Dekel Pilli	9.37
James Pickering	12.69
Total	22.06

These are screenshots of our Harvest app, which can be found here. Please email us if you want an invitation, but note that the free trial period for this app will expire on 13/10/2017.