GEOSPATIAL DATA EXCHANGE USING BINARY DATA SERIALIZATION APPROACHES

P. Mooney [1,*], M. Minghini [2]

[1] Department of Computer Science, Maynooth University, Co. Kildare, Ireland - peter.mooney@mu.ie
[2] European Commission, Joint Research Centre (JRC), Ispra, Italy - marco.minghini@ec.europa.eu

Commission IV, WG IV/4

KEY WORDS: Binary data serialization, interoperability, geospatial, GeoJSON, Protocol Buffers, Apache Avro.

ABSTRACT:

In this paper we investigate the benefits of binary data serialization as a means of storing and sharing large amounts of geospatial data in an interoperable way. De-facto text-based exchange encodings typically exposed by modern Application Programming Interfaces (APIs), including eXtensible Markup Language (XML) and JavaScript Object Notation (JSON), are generally inefficient for an increasingly higher number of applications due to their inflated volumes of data, low speed and the high computational cost for parsing and processing. In this work we consider comparisons of JSON/Geospatial JSON (GeoJSON) and two popular binary data encodings (Protocol Buffers and Apache Avro) for storing and sharing geospatial data. Using a number of experiments, we illustrate the advantages and disadvantages of both approaches for common workflows that make use of geospatial data encodings such as GeoPackage and GeoJSON. The paper contributes a number of practical recommendations around the potential for binary data serialization for interoperable (geospatial) data storage and sharing in the future.

1. INTRODUCTION AND MOTIVATION

Data-driven innovation has seen recent advances enabled by a dynamic technological context characterised by the continuous influx of data, miniaturization and massive deployment of sensing technology, data-driven algorithms, and the Internet of Things (IoT) (Granell et al., 2022). Data-driven innovation is considered a key part of several policy actions worldwide. The recently published European strategy for data (European Commission, 2020) envisions Europe's digital future in the data economy through the establishment of dedicated data spaces. These would allow the efficient flow of data between actors and sectors so that data-driven innovation is translated into concrete benefits for the economy and society. To exploit Europe's potential in our data-driven society, technologies currently used for the management, exchange and consumption of data, including geospatial data, must be evaluated in terms of their suitability to efficiently scale and adapt to streams of larger data and datasets. The increasing number of users accessing data services through mobile devices is putting pressure on service providers to make larger volumes of data available efficiently to these particular users. For many years, encodings such as JavaScript Object Notation (JSON), Geospatial JSON (GeoJSON), Comma-Separated Values (CSV) and eXtensible Markup Language (XML) have been considered as the *de facto* interoperable standards for data serialisation. The majority of Application Programming Interfaces (APIs) available today facilitate data sharing and exchange using these encodings (Vaccari et al., 2020). Whilst such encodings, mainly JSON and XML, have an almost universal support and many other advantages (e.g. they are human and machine-readable as well as open and standards-based), they also have many limitations. These limitations are particularly pronounced when the volume of data is large, resulting in reduced computational performance when exchanging or managing large data volumes.

As an alternative, binary data serialization approaches also allow for the interoperable exchange of large volumes of data (Vanura and Kriz, 2018). Binary data serialization is a process to transform data structures of an object into a binary stream that can be transmitted (and/or stored) and reconstructed later. Popular distributors of geospatial data have also begun making use of binary data encodings, with one of the most prominent examples being OpenStreetMap (OSM) (Mooney and Minghini, 2017), where extracts of the database are provided in the Protocol Buffer (PBF) format. While anecdotal evidence indicates that binary serialization approaches are more efficient in terms of computational costs, processing times, etc., there are additional overheads to consider with these approaches including special software tools, additional configurations, schema definitions, etc. (Viotti and Kinderkhedia, 2022). Additionally, there have been few, if any, investigations of binary data serialization approaches specifically for geospatial data.

The objective of this paper is to investigate the suitability and benefits of binary data serialization to store and share large amounts of data in an interoperable way as an alternative to traditional data exchange using XML or JSON encodings. Advantages and disadvantages of binary data serialization are evaluated for three commonly encountered Geographic Information Systems (GIS) workflow scenarios: i) geolocated point data retrieved from an API; ii) geolocated point data from a very large static GeoPackage dataset; and iii) a large geographic polygon dataset. For each of the three scenarios, we describe the methodology, implementation and analysis of an experiment comparing JSON and GeoJSON with two very popular binary data encodings, namely Google Protocol Buffers and Apache Avro.

The remainder of the paper is structured as follows. In Section 2 we outline a summary of the most relevant related work on binary data serialization for both geospatial and non-geospatial applications. In Section 3 we provide an overview of the experimental analysis for our three common GIS workflow scen-

---

* Corresponding author

arios. Results and subsequent discussions are provided in Subsections 3.1, 3.2 and 3.3. The paper closes with Section 4 where we offer some overall conclusions and lessons learned and propose some next steps for future work. Reproducibility information for this work is provided in Subsection 3.5.

2. RELATED WORK

Binary data serialization is not a new topic in literature, since it has been used extensively within scientific communities such as meteorology and astronomy for decades (Wang, 2014). Binary data serialization was a popular topic in the early 2000s, where works such as Carpenter et al. (2000), Chiu et al. (2005) and Hericko et al. (2003) are strong examples. Serialization approaches have been popular for as long as software developers, engineers, scientists and so on have tried to efficiently exchange large amounts of data between systems or applications. Much of the early work concentrated on Java-based serialization (Philippsen et al., 2000; Maeda, 2012). To narrow down our review of related work we considered research published after the initial releases of Protocol Buffers (Protobuf) and Apache Avro, with Google releasing Protocol Buffers in 2008 while Apache Avro offering its initial release in 2009. More recent literature considering these and other similar serialization frameworks is more appropriate to this work. Srivastava et al. (2020) do not consider binary formats specifically in their work, but make the very interesting claim that the lack of portable scientific dataset formats and universal standards for scientific data exchange force scientists to rely on formats such as CSV for dataset exchange and archival, despite the risks and incompatibilities that can occur with such choices.

At the time of writing, we were unable to find any specific literature source directly related to binary data serialization applied to geospatial data. Sumaray and Makki (2012b) tested a number of data serialization formats and considered their advantages and disadvantages. They showed that XML was "largely inferior to other serialization formats" having a larger size and a slower processing speed. The authors found the performance differences between their chosen binary formats to be negligible. However, the adaptability of binary data formats is the major concern as the client or receiver of the data must have the corresponding binary schema files in order to successfully parse the serialized datasets. The work by Maeda (2012) performs a similar set of experiments with the author indicating that there is "no best solution" in terms of binary serialization approach, with the conclusion that each binary approach is "good within the context for which it was developed". The author also concludes that the size of binary serialized data is much smaller than XML or JSON-based serialization and recommends Apache Avro and Protocol Buffers for "easy interoperability and dynamic languages". Vanura and Kriz (2018) reiterate the difficulty in making decisions around which is the best approach for binary formats in regards to replacing existing non-binary approaches. The results of their work show that Apache Avro and Protocol Buffers achieve the best result but require a schema definition. The worst results are generally achieved by XML libraries. More specifically, the authors found that there are significant performance differences among languages and libraries, and it is not possible to determine the best format across platforms. Their work showed Java and Protocol Buffers to be the most efficient overall solution. For other formats outside of JSON and XML, the results vary greatly depending on the language and particular library. In another research work, Sumaray and Makki (2012a) compare different data serialization formats (XML, JSON, Apache Thrift and Protocol Buffers) for differences in speed, data size, and usability. They conclude that XML should be avoided unless necessary as JSON provides a superior alternative. Their results also suggest that binary encodings should be preferred when serializing data for storage purposes due to their superior speed and size. Only negligible differences were found between Apache Thrift and Protocol Buffers but the authors favoured Apache Thrift because (at the time of their writing) it could be compiled into more languages. In summary, Protocol Buffers and Apache Avro are two of the most popular language independent binary data serialization approaches used today (Vohra, 2016; Popić et al., 2016; Proos and Carlsson, 2020). Both approaches offer rich data structures using schemas, are supported by a large number of the most popular programming languages and are generally easy to understand for most software developers. Both Protocol Buffers and Apache Avro support interoperable approaches to data serialization. Both are evaluated through the experiments presented in Section 3.

Other binary data serialization formats are also used. In the work of Krijnen and Beetz (2017) binary serialization using Hierarchical Data Format (HDF) is applied to point cloud datasets, and a compression ratio of up to 67.7% is obtained for building models with integrated point clouds, compared to the raw source data. The work is different to our work in that we are not considering point clouds. HDF, similar to a file system in itself, has been used for more than 20 years in different engineering and scientific communities to cope with large amounts of data including e.g. physics, astronomy and medical datasets. Martínez-Prieto et al. (2012), followed by Gimenez et al. (2017), remark that Semantic Web applications using Resource Description Framework (RDF) have their potential "seriously underexploited due to the large space they take up, the powerful resources required to process them, and the large consumption time". The binary RDF representation named Header, Dictionary, Triples (HDT) described in their paper achieves compression and can, in practical terms, require up to 15 times less space than traditional RDF formats. The authors argue that scalability issues underlying to semantic web processes justify the need for a binary RDF format like HDT.

3. EXPERIMENTAL ANALYSIS

In this section we discuss the experimental work to support our investigations into the suitability and benefits of binary data serialization to store and share large amounts of data in an interoperable way as an alternative to traditional data exchange using XML or JSON encodings. For this, we have designed, developed and implemented three experiments for the evaluation of Protocol Buffers and Apache Avro encodings for binary data serialization approaches for geospatial datasets. The software code required to reproduce these experimental results is available and Subsection 3.5 includes further information around the reproduction or replication of these experiments. Figure 1 shows a flowchart diagram illustrating the processing steps for all of our three experiments, which are summarised as follows:

- As a necessary pre-processing step, serialize the input datasets to a GeoJSON file;

- Using the Protocol Buffers schema, serialize the GeoJSON file to a PBF file, while the Apache Avro schema is used to serialize the GeoJSON file to an Apache Avro file;

- Using the same approach as the previous step, deserialize both the Protocol Buffers file and the Apache Avro file back to the original GeoJSON;

- Finally, analyse the timing and other results from the binary data serialization processing.

Each of the three experiments, presented in the following Subsections 3.1, 3.2 and 3.3, contains some small experiment-specific differences and these are explained below. We executed each experiment 10 times and the mean and standard deviation of overall processing times are shown in the tables below. Each table indicates the processing steps involved in each experiment. The corresponding mean and standard deviation of processing time, in seconds, are also shown for each step. The sizes in kilobytes of the resulting serialized or deserialized files are shown. Finally, Subsection 3.4 contains a summary discussion of the important outcomes from all of the three experiments.
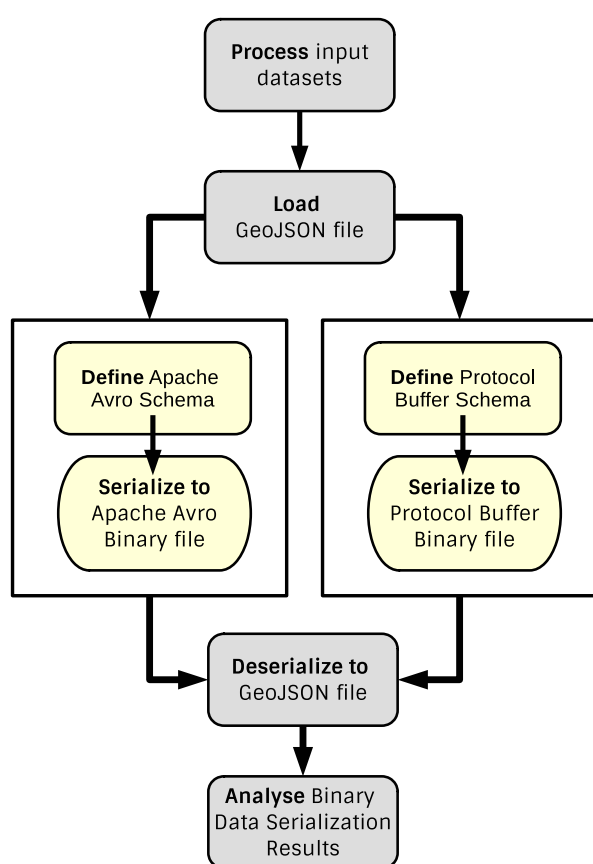

Figure 1. The main processing steps for serialization/deserialization of the input datasets.

Both Protocol Buffers and Apache Avro require the definition of schemas for structuring data. When structuring data with Protocol Buffers, one must define the schema (in a file called the `.proto` file). This schema file is processed by the Protocol Buffers compiler (`protoc`) to generate the classes needed to read and write the Protocol Buffers data. These classes can be generated for most popular languages. Messages, essentially equivalent to objects, in Protocol Buffers, can be composed of any number of fields. The `protoc` compiler (Protocol Buffers, 2022) is required to generate source code (classes) in the target implementation language, such as Python, C++ or Java. If the

original data source changes, e.g. because of the inclusion of an additional field or property, then the Protocol Buffers schema must be changed and the target source code classes must be recompiled. Binary data generated using Protocol Buffers is stored in a Protocol Buffer binary format (PBF) file with a `.pbf` extension. Apache Avro, like Protocol Buffers, is a very popular schema-based binary data serialization technique. It is also a language-neutral approach that was originally developed for serializing data within Apache Hadoop. In Apache Avro the schema is defined in an `.avsc` schema file, which uses a JSON structure for declaring the data structures. Unlike Protocol Buffers, Apache Avro does not require the use of a compiler to generate target source code classes. The schema file is processed during runtime through supported libraries in languages such as Python, C++ and so on. When data is serialized to an Apache Avro binary data file (with an `.avsc` extension) its schema is also stored with it. For both approaches, deserialization cannot happen independently of serialization. For Apache Avro deserialization, access to the original schema file is required. In the case of Protocol Buffers the compiled classes are required. In summary, data providers offering data in these binary encodings would be required to make the schema files available to consumers.

3.1 Experiment 1: Static point-based data

In Experiment 1 we consider the very common situation of using a static vector GIS file for analysis. Generally, these static vector files are available in common geospatial formats such as ESRI Shapefile or GeoPackage (GPKG). Typically, such files are manually downloaded from the Internet or copied from their source location due to their large size. The GPKG point dataset used in Experiment 1 represented the conflation of the Finnish address data from the National Land Survey of Finland and OSM. The GPKG file used was 288,760 Kb (288.8 Mb) in size and contained 1,926,298 point features. Further details about the dataset can be found in Sarretta and Minghini (2021) and in Chapter 6 of Granell et al. (2022). The summarised results of Experiment 1 using the complete GPKG dataset are presented in Table 1. Within the GitHub repository complementing this paper (see Subsection 3.5) there is a smaller GPKG dataset, containing randomly generated data, available for additional experimental analysis. Results relating to this specific dataset for Experiment 1 can be found in Granell et al. (2022). It is worth noting that the difference in sizes of the input GeoJSON file and the deserialized GeoJSON files in Table 1 are related to how the Python GeoJSON library only allows a maximum precision of 10 decimal representation, while the GeoJSON file produced by GeoPandas contains coordinates with up to 13 decimal places. Deserialization is performed using the Python GeoJSON library as this was found to be the most efficient approach.

Processing steps from Figure 1	Time (s)	Size (Kb)
Convert (GPKG → GeoJSON)	327, 11.3	614,859
Load (GeoJSON)	81, 3.2	614,859
Serialize (GeoJSON → Avro)	301, 3.4	228,102
Serialize (GeoJSON → PBF)	306, 2.9	235,821
Deserialize (PBF → GeoJSON)	378, 2.8	542,878
Deserialize (Avro → GeoJSON)	389, 3.1	546,616

Table 1. Results of Experiment 1 using the GPKG dataset: mean and standard deviation of the time required for each processing step and size of the resulting files.

3.2 Experiment 2: Dynamic point-based data

In Experiment 2 we consider another very common situation of using an openly available API to obtain geospatial data in a dynamic situation. Today, it is very common and natural for GIS systems, application software, smart devices and so on to download geospatial data directly from an API for immediate processing and analysis. For Experiment 2, we selected an OGC SensorThings API instance exposing aircraft tracking data; more details can be found in Chapter 3 of Granell et al. (2022). We used this API to download, in real-time, a JSON file containing 20,000 geographic point features. This response file in JSON format was usually around 13,120Kb (13 Mb) in size. The results of Experiment 2 using the OGC SensorThings API are presented in Table 2. There are a number of nuances to this experimental setup. In the JSON file produced by the API, the *encodingType* and *crs* properties are set to `application/vnd.geo+json` and `EPSG:4326` for every feature. We decided to ignore these two fields when serializing this JSON file to a GeoJSON file with Python GeoPandas and both of the binary encodings as we believe two fields can be automatically calculated. The API response data is JSON and the point geometry is encoded as a valid geometry object, but this is part of a JSON array of objects rather than a GeoJSON feature collection. Thus, we encoded the two coordinates of the point geometry as separate fields in both the Protocol Buffers and Apache Avro schema. When the Protocol Buffers and Apache Avro files were deserialized back to GeoJSON, we used the coordinate fields to create point geometry objects for the GeoJSON FeatureCollection.

Processing steps from Figure 1	Time (s)	Size (Kb)
JSON Response download	n/a	12,926
Load (GeoJSON)	1.23, 0.07	11,539
Serialize (GeoJSON → Avro)	0.34, 0.04	7,001
Serialize (GeoJSON → PBF)	0.32, 0.04	7,109
Deserialize (PBF → GeoJSON)	1.14, 0.07	11,515
Deserialize (Avro → GeoJSON)	1.10, 0.03	11,554

Table 2. Results of Experiment 2 using the GeoJSON file: mean and standard deviation of the time required for each processing step and size of the resulting files.

3.3 Experiment 3: Static polygon-based data

In Experiment 3 we consider a very similar workflow to Experiment 1 (see Subsection 3.1). Here we use a static GPKG file containing polygon vector features. This GPKG file represents land cover in Tuscany, Italy based on Corine Land Cover (CLC). The dataset was downloaded from the Geoportal of Tuscany Region (Regione Toscana, 2022), where it is available under the CC BY license, and subsequently clipped on the administrative boundary of Florence province, using the polygon provided by the Italian National Institute of Statistics (ISTAT) (Istituto Nazionale di Statistica, 2022). The resultant GPKG file, which is made available together with the source code (see Subsection 3.5) is 198,340 Kb (198.3 Mb) and contains 161,191 geographic polygon features. The original dataset contains 31 attributes for each polygon feature.

In Experiment 3 we made a number of decisions to allow for a more practical exploration of the effects of using a polygon dataset for discussion in this paper. Upon closer investigation, many of the 31 attributes of the GPKG file were redundant or contained the same property value for every feature. We thus

decided to reduce the number of attributes for each polygon feature. We inspected the attributes in the dataset and found many of them relating to the land use/cover classification of each polygon in 2007, 2010, 2013, 2016, and 2019. We decided to only retain the attributes $c1$, $c2$ and $c3$, corresponding to the three levels of CLC classification for 2019. The main processing steps from Figure 1 were retained with an additional pre-processing step, in which we created a second GeoJSON file containing all of the original geometries from the GPKG file, but with just 10 attributes per feature. The results of Experiment 3 using this GPKG dataset (containing 161,191 polygon features and 10 attributes) are shown in Table 3. We also created a subset dataset containing 17,000 polygon features (just over 10% of total features) from the original dataset. The subset dataset is also a GPKG file and has a file size of 20,500 Kb (20.5 Mb). This smaller subset provides the reader with an opportunity to compare the effects of binary data serialization on the two polygon datasets. We refer to this small subset (which still includes only the 10 selected attributes) as the reduced dataset in the experimental results below. The results of Experiment 3 using the reduced GPKG dataset are discussed in Table 4.

Processing steps from Figure 1	Time (s)	Size (Kb)
Convert (GPKG → GeoJSON)	128.7, 2.30	528,386
Reduce GeoJSON attributes	148.3, 2.2	320,472
Load (GeoJSON - reduced)	48.7, 0.31	320,472
Serialize (GeoJSON → Avro)	61.3, 0.18	264,555
Serialize (GeoJSON → PBF)	58.0, 0.15	264,937
Deserialize (PBF → GeoJSON)	80.4, 1.53	320,472
Deserialize (Avro → GeoJSON)	82.6, 1.93	320,472

Table 3. Results of Experiment 3 using the original GPKG file: mean and standard deviation of the time required for each processing step and size of the resulting files.

Processing steps from Figure 1	Time (s)	Size (Kb)
Convert (GPKG → GeoJSON)	24.5, 1.3	46,655
Reduce GeoJSON	17.5, 1.21	24,657
Load (GeoJSON - reduced)	5.4, 0.21	24,657
Serialize (GeoJSON → Avro)	7.2, 0.41	18,795
Serialize (GeoJSON → PBF)	6.3, 0.31	18,844
Deserialize (PBF → GeoJSON)	11.4, 0.41	24,657
Deserialize (Avro → GeoJSON)	11.6, 0.59	24,657

Table 4. Results of Experiment 3 using the reduced GPKG file: mean and standard deviation of the time required for each processing step and size of the resulting files.

3.4 Overall discussion

The experimental analyses above confirm, for these particular datasets, that the serialized binary data files, both PBF and Avro, were on average at least 20% smaller than the original non-binary data files for all experiments. The overall processing times for binary serialization of the datasets were, on average, at least 10% faster than serialization to JSON or GeoJSON encodings. Deserialization was slower than serialization in every experiment. This ranged from around 1.2 times slower in Experiment 1 to around 3.5 times slower in Experiment 2, while in both versions of Experiment 3 deserialization was around 1.3 times slower. It would require some further investigation to understand if the performance of deserialization can be improved on these results. In our experiments, we deserialized the PBF and Avro files back to GeoJSON, but this may not be necessary

for all applications. However, in order for a client device or application to use the PBF or Avro file, deserialization cannot be avoided due to the very limited query and search opportunities with these encodings. In addition to this, neither of these binary files could be easily visualised in a map-based application without, at least, some form of deserialization.

Experiment 3 brought the question of redundant attributes into focus. In Experiments 1 and 2, the number of attributes for each geographic feature was relatively small (around 8) whereas the dataset used in Experiment 3 had 31 attributes. Regardless of the serialization schemes involved, there are implications for computational performance and indeed data usability in retaining redundant attributes within the dataset when exchanging these datasets.

Neither Protocol Buffers nor Apache Avro can encode the geometry of objects in a high level way as it is represented in GeoJSON. The coordinates of a point object must be either stored in two separate variables or fields or stored in a string- or text-based representation such as Well Known Text (WKT). The situation is similar for Experiment 3, where the polygon geometries must be serialized in Protocol Buffers and Apache Avro as strings. Conveniently these strings contain WKT representations of the polygons, which support faster deserialization to GeoJSON.

We also noted an interesting situation around the precision of geographic coordinates within the source datasets and their subsequent conversions. In Experiment 3 the original dataset represents coordinates with 15 decimal places, whereas the GeoJSON produced by GeoPandas and the GeoJSON library in Python represents the same coordinates with only 6 decimal places. This is a difference of 7 characters in an ASCII representation and a seemingly small 7 bytes. For point datasets this may not make a significant difference to overall filesize as 74,898 coordinate pairs would be required to have an overall difference of 1 Mb between the two representations. However, this seemingly small difference is almost a 46% reduction and this scales substantially over any files larger than a few megabytes. This also becomes particularly noticeable in polygon files (such as the one used in Experiment 3) where each geographic object could have hundreds of coordinate pairs. Of course a discussion of precision is outside the scope of this paper, but we do suggest that data providers consider the precision at which geographic coordinates are being represented within the source datasets. As far back as 1989 authors such as Goodchild (1991) suggested that "precision should match the accuracy". Depending on the target application, the precision could be changed accordingly and this could result in drastically improved performances in terms of overall file sizes. As an example, Figure 2 shows a screenshot of an object from Experiment 3 where coordinates are represented to 15 decimal places.

The run times presented from all experiments in Subsections 3.1, 3.2 and 3.3 should be considered in the context of the laptop machine used for this research work. These run times would naturally be faster with a higher specification machine. Therefore in reproducing this work there will likely be variation in run times due to differences in machine specification, operating system, processing load when executing the Python code, and so on. However, the file sizes for the PBF, Avro and GeoJSON files should remain constant across all implementations. This explanation captures the complexity of geospatial data exchange in practical implementations. We seek to exchange files resulting from resource intensive serialization and

```
{
"type": "FeatureCollection",
"crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },
"features": [
{ "type": "Feature", "properties": { "pk_uid": 315522, "codreg": "09", "siglareg":
"RT", "regione": "TOSCANA", "codprov": "048", "siglaprov": "FI", "codistat":
"048014", "nome": "EMPOLI", "ucs07": "122", "c1_07": "1", "c2_07": "12", "c3_07":
"122", "ucs10": "122", "c1_10": "1", "c2_10": "12", "c3_10": "122", "ucs13": "122",
"c1_13": "1", "c2_13": "12", "c3_13": "122", "ucs16": "122", "c1_16": "1", "c2_16":
"12", "c3_16": "122", "ucs19": "122", "c1_19": "1", "c2_19": "12", "c3_19": "122",
"area": 3749437.5630899998 }, "geometry": { "type": "MultiPolygon", "coordinates": [
[ [ [ 10.903503565087291, 43.682857623379178 ], [ 10.903465596900459,
43.682887216907254 ], [ 10.903484822513866, 43.682898292404055 ], [
10.903501116765398, 43.682907474636167 ], [ 10.903637750430752, 43.682983348769703
], [ 10.903918084967952, 43.68313885028936 ], [ 10.903927214479017,
43.683152923691914 ], [ 10.903915324478982, 43.683169238059804 ], [
10.903983901204082, 43.683207530706106 ], [ 10.903942098768562, 43.683246942266088
], [ 10.903921136793276, 43.683235316894127 ], [ 10.903767925762478,
43.683380575002772 ], [ 10.903867312889615, 43.683435911197655 ], [
10.903968086935922, 43.683491944475008 ], [ 10.90401418269847, 43.683517646659773 ],
[ 10.90414589995931, 43.68359099108369 ]]]]}}}
```

Figure 2. An object from Experiment 3's GeoJSON dataset, where coordinates are represented to 15 decimal places.

deserialization processes where the files have overall sizes as small as practically possible. These files may then be delivered to target client devices and machines, which may not have high technical specifications.

The results appear to confirm that binary data serialization approaches are more efficient than JSON or XML. On deeper reflection, it is difficult to indicate that binary data formats are an overwhelmingly better choice for data exchange than XML, JSON or GeoJSON. While binary data formats enjoy very good expert developer level support in major programming language implementations, this is dwarfed by the near universal levels of support for XML, JSON and GeoJSON in almost all major programming languages. Most modern programming languages can consume these formats automatically and easily provide access to the objects contained in the data file. Binary data protocols such as Protocol Buffers and Apache Avro are not self-describing protocols and when they are used, their corresponding schemas must be provided. For example, in the case of an API providing responses in binary data formats, the API publisher must also publish the schema definition (usually a `.proto` file in the case of Protocol Buffers and an `.avsc` file in the case of Apache Avro) that can be consumed using the available software tools. Without the schema definition file (in the case of these two formats) the binary encoded data files are rendered almost unusable. Hence, binary data encodings could be used where the source datasets do not frequently change their data models. Then binary data encodings could be made available to specialist clients who are capable of writing their own deserialization software code or executing pre-compiled software code. Given this situation, these approaches do not appear to be ready for widespread usage.

3.5 Reproducibility information

The full source code to support the experimental analyses performed in this paper is provided on GitHub (`https://github.com/petermooney/binaryDataFOSS4G2022`). Within this repository there are three folders or directories containing the source code, configuration and input data for all of the three experiments described in Subsections 3.1, 3.2 and 3.3. The implementation language is Python and the experiments were originally written in Python 3.8.10 on Ubuntu 20.04 (64 bit). The laptop computer was a DELL Inspiron 5567 with 16 Gb memory and Intel Core(TM) i7-7500U CPU @ 2.70G processor. To reproduce and replicate the experiments using the content of the GitHub repository, it will be necessary to install

a number of libraries within Python that are outlined in the repository `README` file.

4. CONCLUSIONS AND FUTURE WORK

In this paper we have described the methodology, implementation and analysis of a set of experiments to evaluate the suitability of binary data serialization as an alternative to data exchange in XML or JSON encodings, for three commonly encountered GIS workflows. As discussed in Subsection 3.4, there are some obvious and quantifiable performance advantages with binary data serialization approaches for the workflows considered. Both Protocol Buffers and Apache Avro support interoperable data exchange and enjoy strong programming language support. While the performance results are very positive, many overheads still remain that could impede wider adoption of binary data serialization approaches. These overheads include the need for manual or semi-automated schema updating, specialist knowledge about the binary data implementation and the fact that at the time of writing there is still a rather small worldwide user community support compared to the one for approaches such as JSON or GeoJSON. Software vendor lock-in is avoided with the binary data serialization approaches discussed in this paper. But it remains that case that while there is good overall programming language support available, specialist software development knowledge is required at the implementation stage. We believe that binary data exchange could work very well for specific types of applications. These include applications where the underlying data model rarely if ever changes, and where the recipients of the binary data are capable of developing their own customised solutions for storing, querying, visualising and managing these data.

OpenStreetMap (OSM) provides a particularly impactful example of where dissemination of geospatial data in binary encodings can be very successful. Faced with the severe limitations of XML and ESRI Shapefile formats for distributing very large amounts of OSM data, some alternative data exchange arrangement was required by the OSM community. Today, one can download all OSM data extracts (planet, country, region) in the PBF format (GeoFabrik, 2022). OSM-XML format and ESRI Shapefiles are still available for most extracts. As the OSM data model is reasonably static, the OSM community have developed their own software solutions around this data model. As a result, many excellent implementations are available in languages such as Python, Java, C++, R and so on. We believe that this type of arrangement is viable and scalable. Major stakeholders and distributors of geospatial data should investigate this approach to data provision. The provision of both binary and non-binary data encodings for large scale data exchange has the potential to make these binary formats more visible and eventually gain popularity and adoption among a wider audience.

Binary data serialization is not a panacea for all data compression and exchange problems. Improvements in performance for a specific aspect of a process may reduce performance in another. For example, Gimenez et al. (2017) show using HDF that very large RDF datasets can be serialized and also queried without prior serialization. The binary approach using HDT reduces data volume and increases retrieval velocity. However, this performance achievement comes at the cost of the expensive RDF-to-HDT serialization in terms of both computational resources and time. Understanding the data structures for serialization is an important consideration before implementing a binary data-driven solution. Aihkisalo and Paaso (2011) find in their work on web service object optimisation that binary formats are better suited to heavy binary and text payloads, while XML and JSON are more suited to lengthy array-type structures.

There are a number of potential avenues for future work, including: automated semantic interoperability for binary data serialization using linked geodata, opportunities for more integrated software tool support for binary data processing, and further computational experimentation on different types of datasets and services that could benefit from binary data serialization. There is also the considerable challenge of making binary data approaches as user-friendly as the non-binary data alternatives. For binary data serialization, schema definitions will always be required and at present maintenance of these schemata is mostly a manual process. Binary data files, such as those described and generated in Section 3, require additional special code generation in order to query or search them in the same way that almost any JSON/XML-aware tool can offer. To provide basic query and search functionality, data model specific code must be developed. Making binary data serialization approaches more user-friendly could be one of the most impactful but difficult considerations for future work.

ACKNOWLEDGEMENTS

DISCLAIMER

The views expressed are purely those of the author and may not in any circumstances be regarded as stating an official position of the European Commission.

REFERENCES

Aihkisalo, T., Paaso, T., 2011. A performance comparison of web service object marshalling and unmarshalling solutions. *2011 IEEE World Congress on Services*, 122–129. `https://doi.org/10.1109/SERVICES.2011.61`.

Carpenter, B., Fox, G., Ko, S. H., Lim, S., 2000. Object serialization for marshaling data in a Java interface to MPI. *Concurrency: Practice and Experience*, 12(7), 539–553. `https://doi.org/10.1145/304065.304099`.

Chiu, K., Devadithya, T., Lu, W., Slominski, A., 2005. A binary xml for scientific applications. *First International Conference on e-Science and Grid Computing (e-Science'05)*, IEEE, 8–343. `https://doi.org/10.1109/e-science.2005.1`.

European Commission, 2020. Communication from the Commission to the European Parliament, the Council, the European Economic and Social Committee and the Committee of the Regions: A European strategy for data. COM(2020) 66 final. `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52020DC0066` (accessed on 26 June 2022).

GeoFabrik, 2022. OpenStreetMap Data Extracts. `https://download.geofabrik.de` (accessed on 26 June 2022).

Gimenez, J. M., Fernandez, J. D., Martinez, M. A., 2017. A MapReduce-based Approach to Scale Big Semantic Data Compression with HDT. *IEEE Latin America Transactions*, 15(7), 1270–1277. `https://doi.org/10.1109/tla.2017.7959346`.

Goodchild, M. F., 1991. Geographic information systems. *Progress in Human Geography*, 15(2), 194-200. `https://doi.org/10.1177/030913259101500205`.

Granell, C., Mooney, P., Jirka, S., Rieke, M., Ostermann, F., van Den Broecke, J., Sarretta, A., Verhulst, S., Dencik, L., Oost, H., Micheli, M., Minghini, M., Kotsev, A., Schade, S., 2022. Emerging approaches for data-driven innovation in Europe: Sandbox experiments on the governance of data and technology. EUR30969 EN, Publications Office of the European Union, Luxembourg. `https://dx.doi.org/10.2760/511755`.

Hericko, M., Juric, M. B., Rozman, I., Beloglavec, S., Zivkovic, A., 2003. Object Serialization Analysis and Comparison in Java and .NET. *SIGPLAN Not.*, 38(8), 44–54. `https://doi.org/10.1145/944579.944589`.

Istituto Nazionale di Statistica, 2022. Boundaries of Administrative Units for Statistical Purposes as of 1st January 2022. `https://www.istat.it/it/archivio/222527` (accessed on 26 June 2022).

Krijnen, T., Beetz, J., 2017. An IFC schema extension and binary serialization format to efficiently integrate point cloud data into building models. *Advanced Engineering Informatics*, 33, 473-490. `https://doi.org/10.1016/j.aei.2017.03.008`.

Maeda, K., 2012. Performance evaluation of object serialization libraries in xml, json and binary formats. *2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*, IEEE, 177–182. `https://doi.org/10.1109/dictap.2012.6215346`.

Martínez-Prieto, M. A., Arias Gallego, M., Fernández, J. D., 2012. Exchange and consumption of huge RDF data. *Extended Semantic Web Conference*, Springer, 437–452. `https://doi.org/10.1007/978-3-642-30284-8_36`.

Mooney, P., Minghini, M., 2017. A review of OpenStreetMap data. G. Foody, L. See, S. Fritz, P. Mooney, A.-M. Olteanu-Raimond, C. C. Fonte, V. Antoniou (eds), *Mapping and the Citizen Sensor*, Ubiquity Press, 37–59. `https://doi.org/10.5334/bbf.c`.

Philippsen, M., Haumacher, B., Nester, C., 2000. More efficient serialization and RMI for Java. *Concurrency: Practice and Experience*, 12(7), 495-518. `https://doi.org/10.1002/1096-9128(200005)12:7<495::AID-CPE496>3.0.CO;2-W`.

Popić, S., Pezer, D., Mrazovac, B., Teslić, N., 2016. Performance evaluation of using Protocol Buffers in the Internet of Things communication. *2016 International Conference on Smart Systems and Technologies (SST)*, IEEE, 261–265. `https://doi.org/10.1109/sst.2016.7765670`.

Proos, D. P., Carlsson, N., 2020. Performance comparison of messaging protocols and serialization formats for digital twins in IoV. *2020 IFIP networking conference (networking)*, IEEE, 10–18.

Protocol Buffers, 2022. Protocol Compiler Installation. `https://github.com/protocolbuffers/protobuf#protocol-compiler-installation` (accessed on 26 June 2022).

Regione Toscana, 2022. Land Use and Cover years 2007-2019. `http://www502.regione.toscana.it/geonetwork/srv/api/records/r_toscan:0d4d6640-9a1c-47a4-9a5d-a85cdb36927c` (accessed on 26 June 2022).

Sarretta, A., Minghini, M., 2021. Towards the integration of authoritative and OpenStreetMap geospatial datasets in support of the European strategy for data. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLVI-4/W2-2021, 159–166. `https://doi.org/10.5194/isprs-archives-XLVI-4-W2-2021-159-2021`.

Srivastava, D. J., Vosegaard, T., Massiot, D., Grandinetti, P. J., 2020. Core Scientific Dataset Model: A lightweight and portable model and file format for multi-dimensional scientific data. *Plos one*, 15(1), e0225953. `https://doi.org/10.1371/journal.pone.0225953`.

Sumaray, A., Makki, S. K., 2012a. A Comparison of Data Serialization Formats for Optimal Efficiency on a Mobile Platform. ICUIMC '12, Association for Computing Machinery, New York, NY, USA. `https://doi.org/10.1145/2184751.2184810`.

Sumaray, A., Makki, S. K., 2012b. A comparison of data serialization formats for optimal efficiency on a mobile platform. *Proceedings of the 6th international conference on ubiquitous information management and communication*, 1–6. `https://doi.org/10.1145/2184751.2184810`.

Vaccari, L., Posada, M., Boyd, M., Gattwinkel, D., Mavridis, D., Smith, R., Santoro, M., Nativi, S., Medjaoui, M., Reusa, I., Switzer, S., Friis-Christensen, A., 2020. Application Programming Interfaces in Governments: Why, what and how. EUR30227 EN, Publications Office of the European Union, Luxembourg. `https://doi.org/10.2760/58129`.

Vanura, J., Kriz, P., 2018. Perfomance evaluation of java, javascript and php serialization libraries for xml, json and binary formats. *International Conference on Services Computing*, Springer, 166–175. `https://doi.org/10.1007/978-3-319-94376-3_11`.

Viotti, J. C., Kinderkhedia, M., 2022. A Benchmark of JSON-compatible Binary Serialization Specifications. *arXiv preprint arXiv:2201.03051*.

Vohra, D., 2016. Apache avro. *Practical Hadoop Ecosystem*, Springer, 303–323. `https://doi.org/10.1007/978-1-4842-2199-0`.

Wang, Y. Q., 2014. MeteoInfo: GIS software for meteorological data visualization and analysis. *Meteorological Applications*, 21(2), 360–368. `https://doi.org/10.1002/met.1345`.