On the Seriousness of Serialization

Abhishek Grover

Student ID: 1503611  
[abgrover@ucsc.edu](mailto:abgrover@ucsc.edu)

Mentor: Andrew Shewmaker

Instructor: Prof. Carlos Maltzahn

[carlosm@ucsc.edu](mailto:carlosm@ucsc.edu)

Abstract

* 1. **Background**

I first begin by describing the meaning of serialization. In the language of computer science, serialization is the process of converting a *data structure* in memory into a stream of bytes which you can store on a disk or send over a network[11]. The main goal of serialization is to store the state of the data structure so that it can be reconstructed when needed. This process of recreating the data structure from its serialized format is called *deserialization.*

Serializers play a very important role in distributed systems. In large distributed systems data is sent and received all the time be it during remote procedure calls, messaging, updating logs etc. The main utilization of serialization happens at application level. When applications want communicate with each other over wire or for storing data on disk, they do it in form of serializable objects/data structure. The performance of an application depends a lot on how fast it can send/receive data on the wire before it is able to use it. So data serialization format is an important factor.

* 1. **Types of Serializers**

There are two types of serialization formats: text-based and binary format[abstract]. As their names suggest text-based format is human readable while binary isn’t. The two most commonly used text-based formats are eXtensible Markup Language (XML) and JavaScript Object Notation[2.]. Binary format is relatively new. FlatBuffers, Cap’nProto, Protocol Buffers (protobuf), Simple Binary Encoding(SBE), Apache Thrift are some of the recent binary serializers. Most of these formats have been designed to address the shortcomings of the text-based serializers, although there still might be some use cases where text-based serializers will be more suited than binary serializers like when there is a need for the serialized data structure to be in a human readable format. In this paper I will be discussing the performance of various binary serializers and benchmarking them.

* + 1. **Performance criteria**

There are various aspects of serialization that need to be considered before selecting a serializer for a particular use case. The most important of them being serialized size, serialization speed and ease of use[2.]. *Data size* is important for systems where the network bandwidth (such as sensor networks) as well as device memory (mobile phones) are very limited. Small serialized size also reduces the transfer rate. *Speed of serialization/deserialization* is important for systems which are CPU intensive (low latency systems, games etc) and don’t want to spend a lot of time serializing/deserializing data before using it. Real time applications such as games, web applications such as google docs which require immediate update capability gain a lot by improvement in speed of serialization. Serialization speed is one of the reasons for the *lag* that we experience very frequently while using these applications. Lastly ease of use is important for using the serializers in development of such systems. Although it is difficult to compare usability of different serializers, one factor that we can consider is the support a serializer provides in different programming languages. Now I will discuss how various text-based serializers perform with these criteria in mind. Later on in another section I will discuss binary serializers in a lot more detail.

* + 1. **XML**

Extensible Markup Language is a markup language which defines the rules for encoding documents in a format that is both human and machine readable. The most important factor behind the design of XML is human readability and ease of use. These design goals are written in the W3C specification of XML. The structure of an XML document self explains its content[new paper]. XML has been extensively used for serializing objects/data structures and in RPCs. An example of a data structure of type *Person* is shown below.

<person>

<name>Abhishek Grover</name>

<school>UCSC</school>

</person>

XML is very often criticized for being extremely verbose. As a result of its verbosity the size of the serialized data structure is really large which in turn leads to increase in transfer rate of a data structures serialized using XML[paperone]. The deserialization step is also slower as compared to other serializers one because of large size and second because it involves a parsing step. Binary serializers implement something called *zero-copy* serialization which makes deserialization even faster. I will discuss this later.

* + 1. **JSON**

JavaScript Object Notation (JSON) is the other widely used text-based serializers. JSON is considered as an efficient alternative to XML as it serializes data in lesser size. The following is an example of a data structure serialized using JSON.

{

“name”: “Abhishek Grover”,

“school”: “UCSC”

}

As we can see the markup overhead is very less which attributes to its lesser size. Consequently it also deserializes faster. However JSON also requires a parsing step. Although compared to XML, JSON parses upto 100 times faster but it also has certain disadvantages discussion about which are beyond the scope of this report.

**2.1 Benchmarking Binary Serializers**

With the recent development of binary serializers such as FlatBuffers, Protobuf, Cap’nProto and Simple Binary Encoding, there isn’t much data about their performance in different scenarios. By carrying out this project I aim at analyzing their performance and compare them based on the criteria I discussed earlier. The data I gathered will hopefully assist people in deciding which serialization format to choose for their project.

*Environment*

The benchmarking code has been run on a 64 bit machine with Windows 10.

*Benchmarking Object*

All these serializers support java so I wrote my code in java. In order to compare the serializers, the same object should be serialized in all of them with the same state and data. The developers of SBE have created an object to benchmark their serializer and compare its performance with protobuf. I created the same object for FlatBuffers and Cap’nProto.

The benchmarking object they used was an object of class Car which contains a mixture of strings, integers and objects of other types like Engine, Performance figures etc. Below is the schema of the car object I created for FlatBuffers.





