`

Optimizing Binary Serialization with an Independent Data Definition Format

David Carrera Castillo
MSc. Computer Science
Autonomous University of Guadalajara

Jonathan Rosales
Research Professor, Computer Science Department
Autonomous University of Guadalajara

Gustavo A. Torres Blanco
Senior Engineer
Mobica Solutions

ABSTRACT

The current technologies used for message communication over a network, will begin to be inefficient as the diversity of the hardware and the need to transfer more information in less time increases, as mobile applications and the bandwidth consumption is important; therefore, the methods of sending data in the network is an important area to analyze if we want to optimize the time of data transfer and size of the information. The object serialization is a key element to optimize when looking to reduce transfer time, network saturation, processing of data sent and storage of information. In this paper we propose an algorithm to optimize binary serialization based on the current formats like MessagePack, Protocol Buffers and JSON. To show the efficiency, test cases were executed which show an optimization of 25% and 50% in file size and serialization time respectively.

Keywords

Binary Serialization, data format, optimization, algorithm, web service.

1. INTRODUCTION

The services allow us an optimized way of communication through the network, sending of data in different formats between client-server. Service oriented applications are those that use most of this technology. The exchange of information through web services is specified on the SOAP or XML-RPC protocols and these are based on the XML data model for sending messages [1].

The messages sent between web services and clients that consume these services can be in different formats and types and for different environments: mobile clients, desktop applications, applications in the cloud, web applications and clients with different types of hardware.

As Amorim mentions in [2], companies that look for a correct migration of their systems to the cloud, must start migrating their systems to service oriented applications. These service-oriented architectures (SOA) present challenges such as the constant exchange of messages to perform tasks, the number of these messages can increase to millions even for a single application [3].

Thus, we see that building applications focused on the new technologies, as well as migrating existing ones, brings us back to an initial point in the problematic that is: the transfer of data. The existing technology used for message communication may no longer be efficient due to the diversity of the current hardware; therefore, the method of sending data in the network becomes an important area to optimize in terms of data transfer time and size of information.

The object serialization process becomes a key element to optimize when looking to reduce data transfer time, network saturation, processing of sent data and storage of information. That's a key element, where the implementation of an optimized binary object serialization format has many advantages over JSON or XML formats if we focus on information processing time and storage capacity.

In this paper, we present a proposal to optimize the binary serialization, designing an algorithm that will allow creating a serialized file with the pure data separated from the data definition and building the object (deserialize) dynamically with the definition created at the beginning of the process. Some binary formats such as MessagePack and Protocol Buffers are taken as a base for our proposal as equal as JSON. Those binary formats are currently in use and have managed to optimize this process.

The rest of the paper is organized as follows: Section II presents a summary of the problem, research and advances regarding binary serialization. In Section III we present our optimization algorithm, then, in Section IV we show the results of the implementation of our proposed algorithm. Finally, in Section V we conclude our work and mention our future work.

2. RELATED WORK

There is much research that deals with the issue of serialization, whether in plain text or binary format, comparisons, as well as their performance.

In [4], the author makes a measurement of data transfer times of XML formats in calls and responses from web services through HTTP. Concluding that the overhead that exists in the transfer is not significant, however if the data is compressed, the performance of the serialized message transfer would be much better. This is one of the objectives of this optimization proposal

Another comparison is made in [5], where the author compares the time it takes to serialize / deserialize objects using serializers in text (JSON) and binary formats (Avro, Protocol Buffers, Thrif), where all the test results show that the binary serialization got smaller times compared to JSON.

`

That is why binary serialization is taken as a process to optimize with this algorithm.

In [6], the author is responsible for comparing the performance of the objects serialization in web services applications, using text-based formats such as XML, JSON and a couple of binary formats. From those formats, Google Protocol Buffers had better performance. The author concludes that when a human-understandable format is not a priority, then the binary format is the best option. In our proposal, a human-understandable format is not the priority for the optimization algorithm.

In [7], different libraries are analyzed for a quantitative as well as a qualitative measurement, in which the author reaches the same conclusion which is our starting point in this paper, which is that text-based formats are not optimal compared to the binary formats when it comes to disk space. However, the author mentions that if one tries to make the model understandable (readable), JSON and XML formats will always be preferred.

In [8], the authors evaluate the binary serialization in different platforms, concluding that all of those platforms have the capacity to process binary serializations.

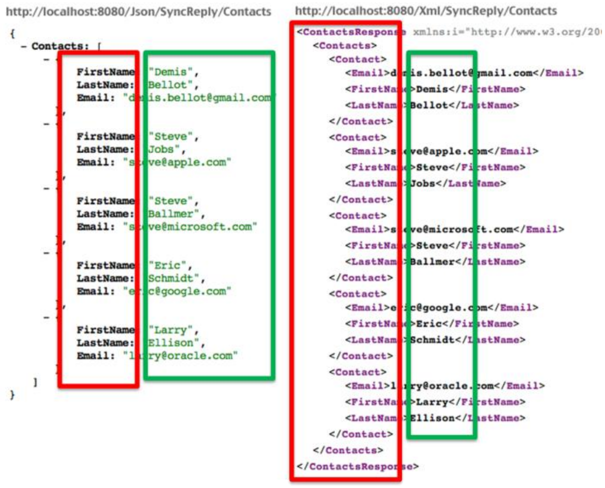
The study carried out in [9], has as a main purpose to optimize the performance of web services using serialization. An algorithm was developed that allowed reaching the objective, using message structures that are saved as templates to be used in the web service. That means a reduction in processing, with the limitation that the templates that will be used as the basis for serialization will increase over the time.

In [10], the objects serialization is presented as an indispensable component in new computer systems in which simplicity and an effective data exchange is a primary objective.

Finally, in [11], the data exchange formats: JSON, XML, MessagePack and Protocol Buffers, were analyzed to measure the data communication efficiency in the Cloud. The results show that binary formats are better when processing times are measured; they also generate smaller files, which translate into less bandwidth consumption when transferring information in the cloud. From these two serialization formats, MessagePack generates files slightly smaller than Protocol Buffers; although in processing time is Protocol Buffers the format that takes advantage.

3. DESIGN AND IMPLEMENTATION

The proposed format in this paper is an optimized combination that takes the best features of commercial and proved binary serialization formats and JSON as base for this algorithm.



http://localhost:8080/Json/SyncReply/Contacts

```
{
  - Contacts: [
     - {
          FirstName: "Demis",
          LastName: "Bellot",
          Email: "demis.bellot@gmail.com"
       },
     - {
          FirstName: "Steve",
          LastName: "Jobs",
          Email: "steve@apple.com"
       },
     - {
          FirstName: "Steve",
          LastName: "Ballmer",
          Email: "steve@microsoft.com"
       },
     - {
          FirstName: "Eric",
          LastName: "Schmidt",
          Email: "eric@google.com"
       },
     - {
          FirstName: "Larry",
          LastName: "Ellison",
          Email: "larry@oracle.com"
       }
    ]
}
```

http://localhost:8080/Xml/SyncReply/Contacts

```
<ContactsResponse xmlns:i="http://www.w3.org/20
  <Contacts>
    <Contact>
      <Email>demis.bellot@gmail.com</Email>
      <FirstName>Demis</FirstName>
      <LastName>Bellot</LastName>
    </Contact>
    <Contact>
      <Email>steve@apple.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Jobs</LastName>
    </Contact>
    <Contact>
      <Email>steve@microsoft.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Ballmer</LastName>
    </Contact>
    <Contact>
      <Email>eric@google.com</Email>
      <FirstName>Eric</FirstName>
      <LastName>Schmidt</LastName>
    </Contact>
    <Contact>
      <Email>larry@oracle.com</Email>
      <FirstName>Larry</FirstName>
      <LastName>Ellison</LastName>
    </Contact>
  </Contacts>
</ContactsResponse>
```

Fig 1: JSON and XML messages

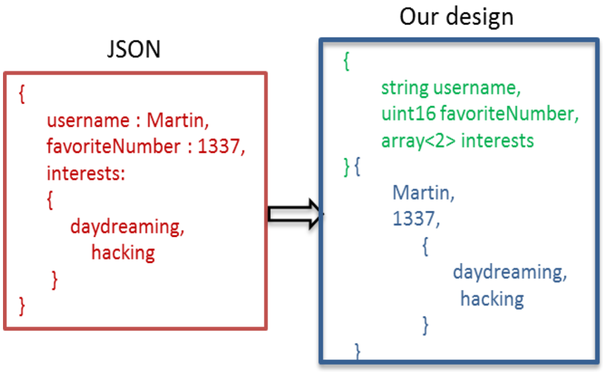
Fig. 1, shows what this design completely eliminates, making the separation of the data definition and the data itself. While JSON compared with XML, eliminated parameter name redundancy, it keeps repeating (red box) the definition of each field for each serialized element (green box); the fields: “FirstName”, “LastName”, “Email” will be repeated once for each element that contains the serialized file, this implies data redundancy that is reflected in the size of the file and transfer and processing time.

In Fig. 2, we show the optimization that we propose, defining (green letters) only once the data type at the beginning of our serialized file and the pure data (blue letters) as the rest of the file. One of our objectives with this algorithm is to split a JSON messages and create a smaller and dynamic file.

No definitions will be repeated for N number of elements that we have, this means savings in file size and serialization time for a large list of data.

Our proposed development implies that the algorithm to serialize and deserialize reads the definition only once, build the new model or container for the objects and the subsequent data will be adjusted to that model. This separation scheme is a little bit similar to the way Protocol Buffers works, however it is not handled as an independent file of the data, nor will it be handled as a template for other data, this result in a not mandatory labels for each data.

With that logic, we will avoid trying to process a file without its schema or data definition



```
{
    username : Martin,
    favoriteNumber : 1337,
    interests:
    {
        daydreaming,
          hacking
     }
}
```

```
{
      string username,
      uint16 favoriteNumber,
      array<2> interests
}{
        Martin,
        1337,
            {
                daydreaming,
                 hacking
            }
}
```

. **Fig -2: JSON message converted to our design**

The serialization algorithm that we develop, receives as the first step, the type of data that will be serialized in order to analyze it and create the data definition of the object; that means, iterates over each of the fields in case of being a class, or identify the data type with which we are working. Then the data section is built; both sections are converted to their hexadecimal values, using a format table that we have previously defined, similar in how other binary serialization approaches work. We proceed to map each type of data with a hexadecimal value of our dictionary, and the pure data converted to its binary representation.

As shown in Fig. 3, in order to further optimize the size of the file on disk as well as the transfer time (not the serialization process time), data compression is proposed in ZIP format because it is one of the most used. Also we show the way our proposed algorithm will treat the data and data definition

The important process in our algorithm is the transformation of the data definition to the hexadecimal representation defined in our format table, which means, the labels that will identify our data and that will be used to process the data to be serialized and deserialized.

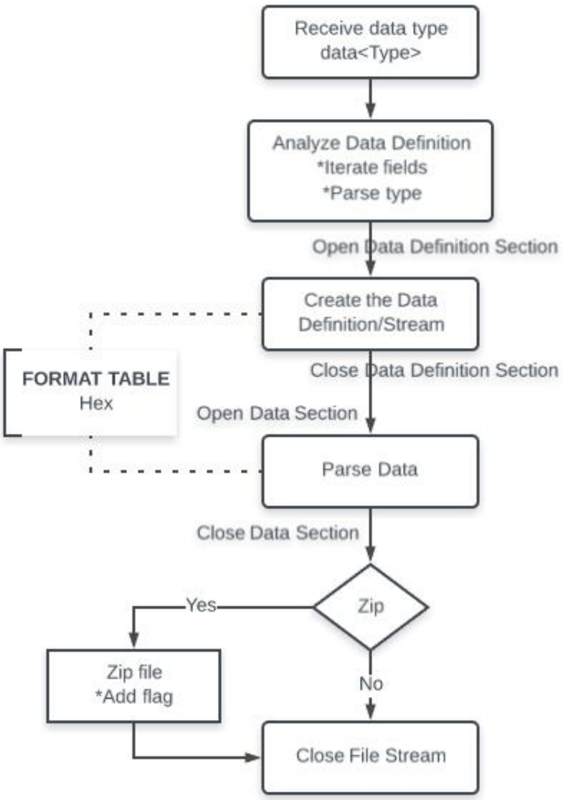


Fig -3: Our approach for Serialization

The object deserialization is basically the opposed algorithm to the serialization process, where the data structure must first be created with the definition that appears at the beginning of the file and then process the data dynamically, that will form the object to be deserialized. Dynamic process implies that if for some reason the data changes and the definition of the data is still the same, those data that don't match will be omitted in the process.

4. RESULTS

In this section we put into practice the algorithm of optimization of serialization that we explained in the previous sections; we compared the performance of our algorithm with two more formats: binary-MessagePack-and JSON.

First, in section 4.1 we describe the hardware and software environment in which we ran the tests. Then in section 4.2 we show the results we obtained in terms of size on disk after we serialized the objects and compared with the other formats. Then, in section 4.3 we describe the results in terms of time, demonstrating the optimization that we achieved with the developed algorithm.

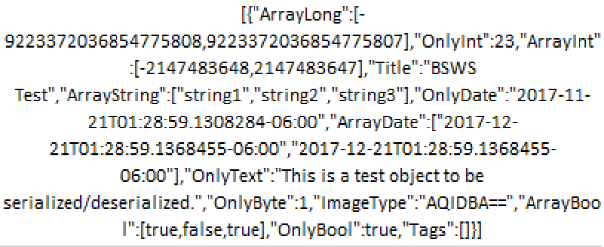

[{"ArrayLong":[-9223372036854775808,9223372036854775807],"OnlyInt":23,"ArrayInt":[-2147483648,2147483647],"Title":"BSWS Test","ArrayString":["string1","string2","string3"],"OnlyDate":"2017-11-21T01:28:59.1308284-06:00","ArrayDate":["2017-12-21T01:28:59.1368455-06:00","2017-12-21T01:28:59.1368455-06:00"],"OnlyText":"This is a test object to be serialized/deserialized.","OnlyByte":1,"ImageType":"AQIDBA==","ArrayBool":[true,false,true],"OnlyBool":true,"Tags":[]}]

Fig -4: Object serialized - representation

4.1 Test Environment

The tests ran on Windows 10 Professional, Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz, 2601 MHz, 2 Cores, 4 Logical Processors, 16 GB RAM, NTFS 1.82 TB. Running a .Net Framework 3.5 console application. Lists with N elements were serialized using the object shown in Fig. 4 as a base. The MessagePack version was 0.9.2; .Net DataContractJsonSerializer library was used for JSON serialization.

4.2 Performance: File Size

Chart-1 shows the size of the file on disk after serializing a list with 100,000 objects as shown in Fig. 4. We can notice the big difference in Kilobytes if we compare JSON against binary serialization. Comparing with MessagePack, our optimization manages to reduce the final file size approximately 25%.

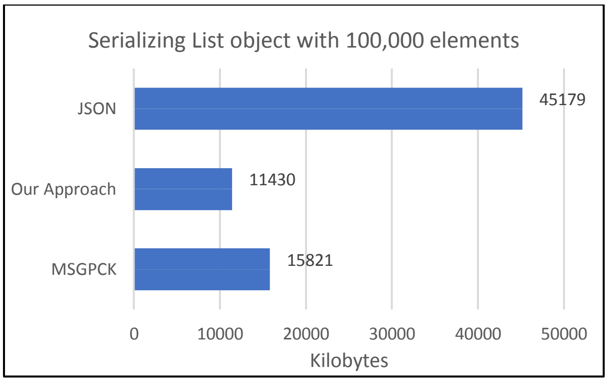


Chart -1: File size after 100,000 serialized elements

In Chart-2, we executed the same previous test, but with a list of 500,000 elements. We can appreciate a notorious difference with JSON. Our optimization was approximately 25%, so we see a constant in our algorithm regardless of the number of elements to serialize.

`

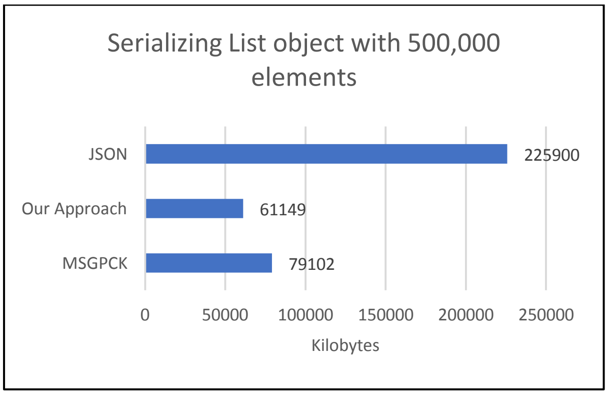


Chart -2: File size after 500,000 serialized elements

4.3 Performance: Serialization Time

In Chart-3, we see the time in milliseconds that it took to serialize a list of N elements with the object shown in Fig-4. The tests proved that it shares almost the same serialization time initially, however, as the number of elements increases; our algorithm decreases the time compared to MessagePack, the optimization algorithm reduce up to 50% the time in a list of 500,000 items.

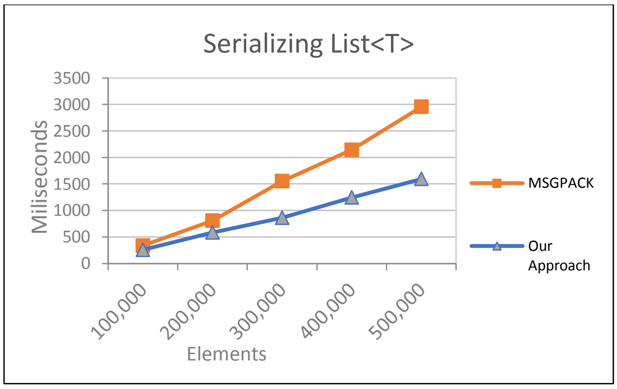


Chart -3: Serializing process time

5. CONCLUSIONS AND FUTURE WORK

With our algorithm presented in this paper, we looked for optimizing binary serialization process for the transfer of large amounts of data over the network. This optimization is based on an algorithm that separates the data from its definition, avoiding repeating labels like JSON message does. The serialized labels in binary format are the key to create the model and then transform the serialized data dynamically to the data definition/model previously created.

With the performance tests that were executed, we have been able to demonstrate the efficiency of our algorithm, by optimizing up to 25% the size of the files on disk and up to 50% of the serialization time of large amount of data.

Future work includes preparing this serialization process to be interoperable, that means implements our algorithm to run over different platforms not only .NET. Also we will work on the integration of parallelism to this algorithm to be able to read and process a large list of data.

Due serialization is an important process in the applications that exchange messages, we will identify and study what external processes or under what conditions in hardware of software our performance could be affected.

6. REFERENCES

[1] J. M. Tekli, E. Damiani, R. Chbeir y G. Gianini, “SOAP Processing Performance and Enhancement”, IEEE Transactions on Services Computing, vol. 5, nº 3, 2012.

[2] G. Amorim, “The Importance of SOA to Cloud Computing”, Service Technology Magazine, vol. 1, nº 87, 2014.

[3] Z. Mahmood, “Service Oriented Architecture: Potential Benefits and Challenges”, 11th WSEAS International Conference on COMPUTERS, Agios Nikolaos, Crete, Greece, 2007.

[4] A. B. Dauda, Z. Saber, F. Alotaibi, M. A. Mustapha and M. T. Abdullah, "Effect of serialized messaging on Web services performance", 2017 International Conference on Computing Networking and Informatics (ICCNI), Lagos, 2017, pp. 1-5.doi: 10.1109/ICCNI.2017.8123774.

[5] A. Nagy and B. Kovari, "Analyzing .NET serialization components", 2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI), Timisoara, 2016, pp. 425-430. doi: 10.1109/SACI.2016.7507414.

[6] T. Aihkisalo and T. Paaso, "A Performance Comparison of Web Service Object Marshalling and Unmarshalling Solutions", 2011 IEEE World Congress on Services, Washington, DC, 2011, pp. 122-129. doi: 10.1109/SERVICES.2011.61.

[7] K. Maeda, “Performance Evaluation of Object Serialization Libraries in XML, JSON and Binary Formats”, Digital Information and Communication Technology and its Applications (DICTAP) 2012 Second International Conference, 2012.

[8] C. J. M. Tauro, N. Ganesan, S. Mishra y A. Bhagwat, “Object Serialization: A Study of Techniques of Implementing Binary Serialization in C++, Java and .NET”, International Journal of Computer Applications, vol. 45, nº 6, 2012.

[9] N. Abu-Ghazaleh y M. J. Lewis, “Differential Deserialization for Optimized SOAP Performance”, Proceedings of the 2005 ACM/IEEE conference on Supercomputing, 2005.

[10] S. &. R. Chawla, “Object Serialization Formats and Techniques a Review”, Global Journal of Computer Science and Technology Software & Data Engineering, vol. 13, nº 6, 2013.

[11] V. C. Emeakaroha, P. Healy, K. Fatema y J. P. Morrison, “Analysis of Data Interchange Formats for Interoperable and Efficient Data Communication in Clouds”, IEEE/ACM 6th International Conference on Utility and Cloud Computing, 2013.