An Evaluation of Protocol Buffer

Gurpreet Kaur and Mohammad Muztaba Fuad
Department of Computer Science
Winston-Salem State University
Winston-Salem, NC 27110, USA
{gkaur108, fuadmo}@wssu.edu

***Abstract*—World is shrinking each day through the use of Internet and people are communicating better than before in this widely distributed network. There is a great need to manage this communication over various networks supporting different specifications. One of the widely used techniques for this type of data management is XML data interchange format. Google developers recently introduced Protocol Buffer as an alternative to XML claiming that it overcomes the shortcomings suffered by XML. This paper compares XML and Protocol Buffer data formats by extensive analysis of the two. The paper evaluates the claims made by Google by developing an algorithm to map an existing XML to Protocol Buffer format and drawing any conclusion on the efficiency and effectiveness of this format as compared to XML. It can be hoped that this work will contribute to the upcoming research in this field as people are looking for more robust data interchange format for the future of the Internet.**

I. INTRODUCTION

With developing Technologies like World Wide Web and the Internet, a large number of its users are getting connected either for mutual communication , information sharing or retrieval of information. There is a great need to organize, synchronize and manage the data to be transmitted from one machine to another over different networks. This process becomes more critical when it comes to the machines serving many users at the same time. Current widely used technique to serve this purpose is the use of eXtensible Markup Language or XML [1] data format which offers various features such as portability, safety, security, human-readability etc. But it suffers from certain limitations like redundancy of the metadata, time consumption, space consumption [2] etc.; making this format a bit unsuitable for large data transfer between *peta* and *exa* scale databases.

There arises the need to overcome these limitations in order to make the Internet services faster. Thus, the World's largest internet search engine, Google, introduces a technique known as Protocol Buffer [3] for representing data in their Intranet infrastructure. This standard has been open-sourced by Google in recent years. For implementing Protocol Buffer, the structure of the data is to be defined just once in a file and then the concept of classes, objects and methods are automatically generated and used to store, manipulate and retrieve the desired information. The data handling is made easier by the automatic generation of source code, parsing and serialization files in high level languages amongst Java, C++, Python. The data structure can be updated also without breaking deployed programs that are compiled against the "old" format.

This paper critically examines and evaluates the effectiveness of Protocol Buffer by implementing the unique and useful features of Protocol Buffer in real life situation and developing an algorithm to convert an XML document to a Protocol Buffer equivalent, providing a guide to those wishing to replace an existing XML document to a Protocol Buffer document.

The rest of the paper is organized as follows: Section II introduces Protocol Buffer and other alternate data interchange formats along with any related works. Section III provides details on the mapping algorithm. Section IV discusses the results of the evaluation and finally Section V concludes the paper.

II. BACKGROUND AND RELATED WORK

XML is a set of rules for encoding documents electronically. It is defined in the XML 1.0 Specification created by the W3C [1] and several other related specifications. XML data interchange format is used to mark-up the data which is to be stored in a text file, prior to its transfer over the network. Then it is merged with other web technologies to present in the form of web document to the end user. The mark-up of the data is done by using related descriptor words known as metadata tags or elements. XML is just plain text. Software that can handle plain text can also handle XML. However, XML-aware applications can handle the XML tags especially. The functional meaning of the tags depends on the nature of the application.

Protocol Buffer (PB) is a language-neutral, platform-neutral, extensible way of serializing structured data for use in communications protocols, data storage, and more. Google has open sourced this internal development tool as 'Protocol Buffer' in 2008. It is a Data Description Language (DDL) that forms a basic part of the operation of the company's vast computing cluster. Google's requirement to satisfy a large number of simultaneous requests leads to the development of Protocol Buffer. These requests have to be dealt within a short amount of time by retrieving the desired data from its distributed infrastructure.

The most common alternatives to Protocol Buffer include Java serialization and different domain specific encoding scheme for message's data along with XML. There are a couple of data interchange formats available which are similar to Protocol Buffer.

978-1-4244-5853-0/10/$26.00 ©2010 IEEE

Thrift [4] is very similar to Protocol Buffers in many respects but Protocol Buffers supports some additional functionality also. This standard seems to be a good source of inspiration for Protocol Buffers. It is an open source project backed by *Facebook* and accepted for incubation at Apache. It combines a software stack with a code generation engine to build services that work efficiently and seamlessly between popular programming languages by offering more or less the same data types.

ICE [5] is an object-oriented middleware that provides object-oriented Remote Procedure Call developed by ZeroC and dual-licensed under the GNU, GPL and a proprietary license. It supports C++, Java, .NET languages, Python, PHP and Ruby on most major operating systems. This standard is similar to Protocol Buffers in some respects. As compared to the Protocol Buffer, ICE does not support extensions and it is required to explicitly and manually define the serialization and deserialization of the data to be transferred. JSON [6] is also another data interchange format that shares similarities with the above mentioned formats and Protocol Buffer.

There has not been much (that we know of) work in this context but the following work is similar to the work presented in this paper. There has been a brief effort made to generate a Protocol Buffer message from an existing XML in the past and presented through an online forum [7]. Although this is a good introduction, this work suffered from limitations such as the work deals with a specific type of class and its instances, making it hard for the readers to come up with a general picture of the expected result. Also, the author neither provides a proper documentation nor any algorithm for such conversion. The above shortcomings motivated us to work on this research as Protocol Buffers is capable of emerging as a powerful and efficient alternative to XML.

III. MAPPING ALGORITHM

An XML file consists of root element, child nodes and sub-children. This structure forms a tree starting from the root and following until the children in the form of its branches. On the other hand, a Protocol Buffer interface (saved with a *.proto* file extension), consists of nested messages and its fields to manipulate the data. Field declaration is done using either one of the reserved keywords, depending on the element's content in the schema definition.

The mapping algorithm presented in this section can be used to map an existing XML document to its Protocol Buffer equivalent. There are two phases to map a XML into its Protocol Buffer equivalent message. Phase 1 is concerned with deciding the appropriate data types (from the XML) for the message fields and Phase 2 generates the structure of the Protocol Buffer interface from the XML.

A. Phase 1

Protocol Buffer supports two types of data i.e. *scalar* or *user-defined* for any specific field. Either one of them has to be used for declaring the field. *Scalar* data types includes primitive types, such as string, double, float, int32 etc. Whereas *user-defined* data type include data which are not pre-defined in the Protocol Buffer vocabulary. These data types are defined by the user depending on the message content. There should be a separate message for using this type of data type.

The element's data content is to be considered to select an appropriate data type. While considering the data content for any particular field, it happens sometimes that the same field is storing one type of data at one time and different type of data later times. The solution to this problem is to declare that data type with the highest rank among its alternatives. For example, If an element's content is *integer* during the first occurrence but a *double* in the following occurrences, then it must be assigned a *double* data type as it outranks integer data type. Table 1 list the rules used to map XML data types for generating an equivalent Protocol Buffer interface message.

TABLE 1. MAPPING XML DATA TYPES.

Data content in XML	Protocol Buffer data type
String	*string*
Integer	*int32/int64*
Only negative integer	*sint32/sint64*
Numbers with fraction	*flaot/double*
Logical/Boolean	*bool*
List of alternate values	*enum*

B. Phase 2

Once the data types for each of the XML elements are gathered in Phase 1, this phase generates the actual interface file for Protocol Buffer. The very first element in the XML which is generally known as '*root*', is mapped as the main message in the interface acting as the parent to the nested messages. The elements at the level below the '*root*' will be the children of the root. These elements are dealt twice for generating a *proto* message because:

- First, these elements should be declared as the field of the main message. If these elements exist more than one times, the modifier for the particular field should be '*repeated*' otherwise '*required*' to follow Protocol Buffer specification.
- Second, to define the type of these elements as a user defined data type. Each element should be declared as a nested message inside the main message.

The XML elements with the data content should be replaced as the fields of the nested message (which is the parent of these elements) in the Protocol Buffer message. The mapping algorithm can be summarized as follows:

1. Create the main message for the root of the XML document.
2. Recursively do:
   a. Create a field (of user defined data type) for a child node in the main message.
   b. Create a sub message for that user type data field
   c. If the child node is a leaf node (i.e. has data values)

i. Create a field in the message with the scalar data type found in Phase 1.
ii. There must be a value assigned for each field which is equals to the number of local fields declared inside the message from 0 and each field has a unique number assigned. This is to satisfy the Protocol Buffer specification.
iii. Exit from recursive call

d. Put proper closing braces for the messages.

Once the interface file is generate from an XML by the above algorithm, the Protocol Buffer compiler is run on that file to create the automatically generated classes, providing an easy way to work with the actual data fields defined in the message interface. It is achieved by using the nested classes and methods declared in the main protocol interface class. It is assumed for the algorithm that the XML always has a root element. However,

- If there is no root element, a dummy root will be added. This will not change the semantics of the XML and the corresponding Protocol Buffer equivalent.
- If there is more than one root existing for a given XML, a super root will be added as before without sacrificing the structural consistency.

IV. EVALUATION

While analyzing and evaluating Protocol Buffer, this research work ended with some important findings which have to be carefully considered to use this data format.

- The existing XML structure (which is to be mapped) should be unique. There should not be more than one element with the same name. This is to satisfy Protocol Buffer's specification on naming message fields.
- The name of the interface file can not be same as an existing element in the XML. This specification has to be fulfilled to have a unique set of auto generated classes (by the Protocol Buffer compiler) for each field in the XML.
- XML serves in a better way to apply this mapping algorithm to an existing XML to generate its Protocol Buffer equivalent rather than using a Document Type Definition (DTD) because DTD does not specify the elaborated data type for an element.

We performed an analysis on different XML file sizes to generate their equivalent Protocol Buffer interface using the newly developed mapping algorithm. Figure 1 shows the distribution of time in mapping an existing XML to its Protocol Buffer equivalent on a cent percent time scale. Whole process of converting a XML to its Protocol Buffer equivalent is divided in the following four stages:

- Parse an XML file to generate a tree structure in the computer's memory using a DOM.
- The total time taken by Phase 1 to determine the appropriate data types.
- The total time taken by Phase 2 to generate the Protocol Buffer interface.
- The total time taken by the Protocol Buffer compiler to generate classes to facilitate data handling.

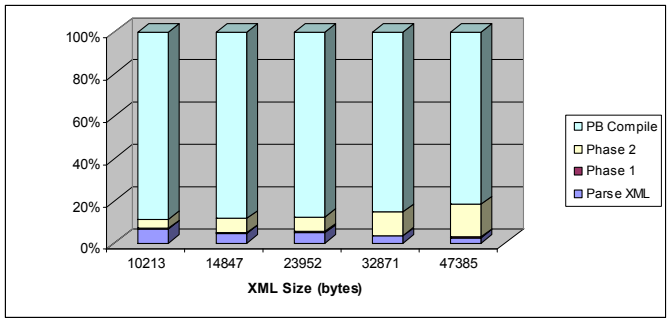

Figure 1: Time for the overall conversion process.

It can be easily concluded from Figure 1 that more than half of the time is consumed in the compilation of the Protocol Buffer interface file to generate the automatic classes and methods for manipulating the Protocol Buffer messages. Other than that, the mapping algorithm is proportional to the XML file size. The total time taken by Phase 1 is negligible in most of the cases while the Phase 2 is directly related to the XML size. On the contrary the time taken by the Protocol Buffer compiler can be neither predicted nor reduced as it can not be controlled by the user, and moreover it depends on the multiprogramming environment and the hardware specifications of the machine.

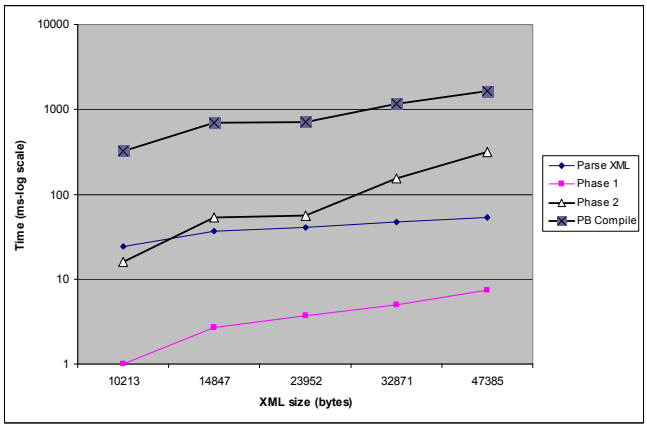

Figure 2: The Trend of the mapping algorithm with different XML Sizes.

The trend analysis shown in the Figure 2 indicates the linear nature of mapping algorithm on its algorithmic analysis. XML file size influences the time taken by the various phases involved in the mapping algorithm. More time (*in ms*) is consumed in mapping a bigger XML as compared to a relatively smaller XML file size. It is evident from Figure 2 that the time taken by the Protocol Buffer compiler is going to be a substantial overhead for the whole conversion process. However, since the compiler has to be executed only once and Protocol Buffer allows users to use existing Protocol Buffer interfaces with ease, this overhead can be minimized over the lifespan of an application.

V. CONCLUSIONS

This paper presents an algorithm to map an existing XML document to its Protocol Buffer equivalent. The goal of the mapping algorithm is to aid those who wish to implement this new standard in the near future and convert their XML data to Protocol Buffer format. Various conclusions have been drawn after researching the XML and Protocol Buffer data formats in various contexts. The practical analysis to query each of these data formats gave us a pretty good way to compare both of them and to determine how one data format is more effective than the other in different contexts.

Memory consumption: XML and Protocol Buffer interface file consumes almost the same number of bytes in the memory. An important note to make here is that XML has the data content in it however the Protocol Buffer interface does not provide such details. Unfortunately the Protocol Buffer does not outweigh the XML standard in this regard.

Binary support: Since Protocol Buffer saves data in binary format, there is no overhead for parsing as in XML which leads to fast and easy data transfer over the web.

Implementation time: Once the Protocol Buffer interface is defined, the rest of the implementation process is straight forward.

Human effort: Protocol Buffer requires similar human effort for data manipulation as XML.

Human readability: XML wins in this regard because it is completely human readable involving the use of simple text files but at the cost of sacrificing memory space and processing time. Although the Protocol Buffer provides limited human readable files, it is mostly concerned with binaries, encoding details, parsing and serialization files and wire type formats etc. which are not presented in a simple language.

Prerequisite knowledge: Protocol Buffer is an object oriented data format basically related to working with classes and methods. Thus, it requires a good understanding of any supported programming language (amongst Java, C++ or Python) for the effective implementation of Protocol Buffer. In contrast, XML does not demand such a requirement as XML consists of working with text files, storing the elements and the corresponding data in an organized tree structure.

From the above interpretations, the claims made by Google regarding the efficiency and effectiveness of Protocol Buffer are proved to some extent. However, extensive analysis and experimentation with diverse XML data will provide us with more insight into Protocol Buffer.

REFERENCES

[1] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau, "Extensible markup language (XML) 1.0", Fifth Edition, November, 2008.

[2] R. Fernandes and M. Raghavachari, "Inflatable XML processing", Proceedings of the ACM/IFIP/USENIX 2005 international Conference on Middleware, G. Alonso, Ed. Springer-Verlag, NY, pp. 144-163.

[3] Google Inc., Protocol Buffer, http://code.google.com/apis/protocolbuffers/docs/overview.html.

[4] M. Slee, A. Agarwal and M. Kwiatkowski, "Thrift: scalable cross-language services implementation", Whitepaper, Facebook, 156 University Ave, Palo Alto, CA.

[5] M. Henning, "Choosing middleware: why performance and scalability do (and do not) matter", Whitepaper, ZeroC Inc.

[6] JavaScript Object Notation, www.json.org.

[7] Small Protocol Buffer test, http://www.ogre3d.org/forums/viewtopic.php?f=16&t=47408.