A Case Study on Alternate Representations of Data Structures in XML

Daniel Gruhl
IBM Almaden Research Center
San Jose, California USA
dgruhl@almaden.ibm.com

Daniel Meredith
IBM Almaden Research Center
San Jose, California USA
dnm@almaden.ibm.com

Jan Pieper
IBM Almaden Research Center
San Jose, California USA
jhpieper@almaden.ibm.com

ABSTRACT

XML provides a universal and portable format for document and data exchange. While the syntax and specification of XML makes documents both human readable and machine parsable, it is often at the expense of efficiency when representing simple data structures.

We investigate the "costs" associated with XML serialization from several resource perspectives: storage, transport, processing and human readability. These experiments are done within the context of a large text-centric service oriented architecture – IBM's WebFountain project.

We find that for several applications, human readable formats outperform binary equivalents, especially in the area of data size, and that the costs of processing encoded binary data often exceeds that of processing terse human readable formats.

Categories and Subject Descriptors

E.4 [**Data Structures**]: Arrays; E.4 [**Coding and Information Theory**]: Data compaction and compression; E.5 [**Files**]: Organization/Structure

General Terms

PERFORMANCE, EXPERIMENTATION

Keywords

XML, Data Structures, Serialization, Compression, WebFountain

1. INTRODUCTION

With the increasing popularity of XML, it is natural to consider the best representation of common data structures within XML. There is a rich history of optimal data representations being non-obvious – Knuth[7] has a particularly nice example regarding optimal expression of a list of prime numbers as the difference between elements, as opposed to the numbers themselves.

For us, the search for optimal representations grew out of our work on IBM's WebFountain[5], a project to ingest, analyze, and store corpora of unstructured text, in excess of 5 billion documents, such as the World Wide Web. Text is normalized to aid in data processing and stored in a standard XML format for later ingestion by other components of the platform.

WebFountain leverages XML in document storage, remote procedure calls, and document transport utilizing a fast XML transport mechanism[1] and as such, requires the representation of large simple data structures. Of special interest is data which can be thought of as a vector (e.g., a tokenization).

The remainder of the paper will examine prior efforts in related areas, outline four representations for comparison, examine the results of these comparisons and draw conclusions.

2. RELATED WORK

Considerable investigation has occurred in the area of optimal representations for whole XML documents. Binary encodings, such as Binary XML[12], wbXML[11] and XTalk[1] reduce the size and parsing complexity. Specialized compression algorithms such as XMill[8], XMLPPM[3] and Millau[10] strive to achieve higher compression rates than traditional alternatives (e.g. gzip[4]). All these methods require matching specialized software to serialize and parse. However, specific data structure representation in XML such as we explore in this paper are less widely discussed, with space delimited vectors (e.g., as defined in XMI[9]), the W3C SOAP[2] representation for RPC messages, and embedded binary object via Base64[6] encoding being three obvious examples.

3. PROBLEM STATEMENT

There are many types of recurrent metadata that might be represented in a document, but for the purpose of this discussion we will consider storing tokenized text and the discovered entities therein. Suppose a document contained the sentence:

```
George W. Bush met with Tony Blair.
```

The first step in processing such a sentence might be the tokenization into the following tokens (in this case words):

```
George␣W.␣Bush␣met␣with␣Tony␣Blair.
```

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DocEng2005 2005 Bristol, United Kingdom
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$5.00.

Later, an entity spotter takes these tokens and notes that two known "entities" (in this case people) occur in the sentences and produce a piece of metadata for each:

```
Entity found: Bush Location: 0 Span: 3
Entity found: Blair Location: 5 Span: 2
```

This is the kind of data we seek to represent in a manner that is efficient from a processing, network transfer, storage and human interpretation standpoint. We anticipate that with machine generated metadata, there will be dozens (if not hundreds) of such annotations associated with any given document, thus an efficient representation is critical.

Consider the following data structure: an ordered set of tuples consisting of a term (identifier string), the location (position in a stream) of that term, and the span (number of tokens in the stream). These so-called "SpanLoc Vectors"(*SLVs*) are used in WebFountain to store a variety of data, including tokenized documents, existing metadata and auto-generated annotations. These annotations need to be stored for use in subsequent algorithms since not all analytics run in the same process space or even temporal space. Thus, the task is to find the most efficient representation for such a data structure.

4. APPROACHES

While one could imagine more data elements per annotation than the three mentioned, with little loss of generality we will consider tuples of an identifier, a location and a span as representative of this class of problem. We will explore two "true" XML representations, a binary encoding and our proposed terse human readable format.

4.1 True XML representations

Using an *element-only* representation in XML the vector would appear as below:

```
<slv>
  <token>
    <term>Entity:Bush</token>
    <loc>0</loc>
    <span>3</span>
  </token>
  <token>
    <term>Entity:Blair</token>
    <loc>5</loc>
    <span>2</span>
  </token>
</slv>
```

This format is referred to as *XML (element)*. While the readability and intent of the structure is clear, a document represented in this format grows large. Alternate forms of XML representation are also available and by using attributes we can reduce the previous content:

```
<slv>
  <token loc="0" span="3">Entity:Bush</token>
  <token loc="5" span="2">Entity:Blair</token>
</slv>
```

With this attribute based format, referred to as *XML (attrib)*, the data size decreases by about 40%. This is a substantial savings for a project like WebFountain, where we attempt to store over 5 billion webpages for text analysis. For a corpus of this size, each byte of overhead per document translates to 5GB of disk space.

4.2 Binary encoded leaves

Another option is to directly serialize the data structure in a common binary format. Represented as binary the SLV is about 80% smaller than *XML (element)*. However the structure cannot be represented easily in XML due to the constraint that binary data is not allowed in documents and generally available tools are not able to parse such a document. The binary format of the SLV uses 4 bytes for the number of elements in the vector, BER encoding for the location offset, span, and term length (8, 4 and 4 bytes respectively), and `UTF-8` encoding for the term itself:

```
[#of elements] {loc, span, {term length, term}}+
```

To counter these problems the binary serialization of the SLV needs to be Base64[6] encoded prior to insertion within a document to ensure proper transport of the data. The resulting Base64 encoded data structure is around 30% larger than the binary data structure, and as you can see from the example, is far from human readable.

```
<slv>/////gEAAwtFbnRpdHk6QnVzaAUCDEV
udGlOeTpCbGFpcg==</slv>
```

While this representation, referred to as *Binary + Base64*, has an advantage in storage over the full XML representations above, it presents numerous issues due to the lack of human readability; namely the difficulty in development of technologies which leverage the format.

4.3 Terse Human Readable Format

The advantages of human readable formats were some of the drivers behind the decision to use XML as our universal storage format. The binary SLV serialization of data is completely orthogonal to that decision and the intent of XML, but the storage benefit is clear. As a middle ground we propose a terse human readable format, referred to as *Terse*, which maintains the intent of XML and the data structure, but which has a smaller size, and thus better transport, processing and storage properties. Using the terse form:

```
<slv>location:span␣"term"␣...</slv>
```

The example vector is represented as the following:

```
<slv>0:3␣"Entity:Bush"␣5:2␣"Entity:Blair"</slv>
```

To further reduce the overhead of the terse representation, we can encode the value of a token location as the difference between the previous token location in the vector and the current token location.

It is also possible to reduce the number of locations and spans written to the stream by assuming that certain values in the vector have implicit values if they are not present in the stream. For example we could assume the location (or delta) is one plus the previous location if it is not present, or assume the span of the token is one if the span is not present. Our terse format utilizes delta based locations that are always present and an implicit span of 1 if the colon and span value are omitted.

5. EXPERIMENTS

We performed various experiments on the four approaches using a random 30k document sample from the 5B document WebFountain corpus (including multi-byte language examples). The average XML document size was 33 KB, which

Approach	Serialized (KB) Raw	GZip	Serialize (ms/doc)	Deserialize (ms/doc)
XML (element)	14.6	.99	.67	3.0
XML (attrib)	9.1	.88	.52	2.5
Binary+Base64	3.9	1.5	.35	.38
Terse	2.4	.70	.54	.25

Table 1: Average size and time by approach

included on average, 11 vectors per page at 2.9 KB per vector and 200 tokens per vector. These experiments were run on a dual 3.06 GHz Pentium workstation with 4GB of RAM running Gentoo Linux kernel 2.6.11. All implementations were compiled with IBM Java 1.4.2 and run within the IBM 1.4.2 JVM. The JDOM XML package was used for XML document manipulation in conjunction with the underlying Apache Crimson parser.

The size of a serialized data structure impacts three core areas of the system: storage, processing, and transport. In practice compression is useful within the area of storage. While compression could be utilized for transport the need for compression aware clients, as well as the generally smaller size of transported documents, suggests that other "in transport" compression approaches are more appropriate. The other consideration is the data structure serialization/deserialization time (including the Base64 encoding/decoding). The results are summarized in Table 1.

The *XML (element)* representation is over 6 times the size of a terse human readable representation (see Table 1). Even when compressed using a standard text compression (`gzip` level 3) the difference is still over 40%. However, the most interesting observation is how poorly the binary representation compresses. This is explained by the fact that Base64 maps every byte to multiple possible representations. Unfortunately the encode and compress steps cannot be inverted, since compress is a document level concept and encode is an element level one.

In the raw format (e.g., what gets sent over the network), *Terse* is still only around 60% the size of *Binary+Base64*. The performance of the true XML representations suffer from the relatively slow speed of the document model and parser (see Figure 1). However, the *Binary+Base64* and *Terse* format deserialize about an order of magnitude faster. The additional overhead of Base64 hampers the deserialization performance of the binary format, although the format has an advantage in serialization.

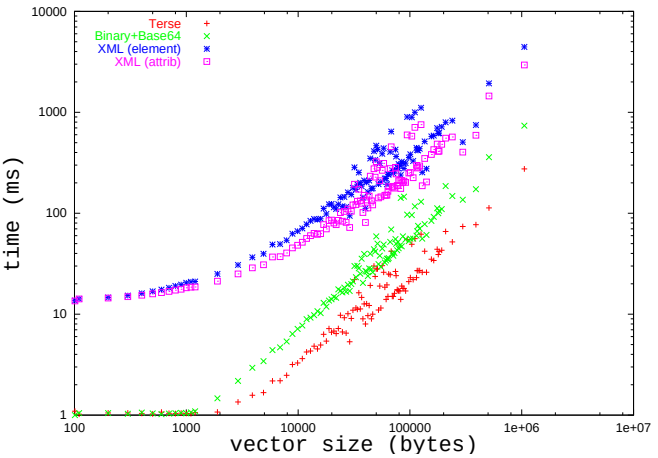

Figure 1: Deserialize Time Comparison

6. CONCLUSION

And so, as in Knuth, we find the best representation is not necessarily the obvious one. All too often we hear of implementations of "XML systems" that abandon simplicity and human readability to become little more than brute force "binary payload in XML document" approaches. We hope we have shown that by exploring textual alternatives it is possible to preserve the clarity and obvious intent that is the hallmark of XML, while actually gaining in performance and compactness when compared to binary approaches.

7. REFERENCES

[1] R. Agrawal, R. Bayardo, D. Gruhl, and S. Papadimitriou. Vinci: A service-oriented architecture for rapid development of web applications. In *Proceedings of the Tenth International World Wide Web Conference (WWW2001)*, pages 355–365, Hong Kong, China, 2001.

[2] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winder. Simple Object Access Protocol. http://www.w3.org/TR/SOAP/, May 2000.

[3] J. Cheney. Compressing xml with multiplexed hierarchical ppm models. In *DCC '01: Proceedings of the Data Compression Conference (DCC '01)*, page 163, Washington, DC, USA, 2001. IEEE Computer Society.

[4] P. Deutsch. Gzip file format specification version 4.3. RFC 1952, 1996.

[5] D. Gruhl, L. Chavet, D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, and J. Zien. How to build a webfountain: An architecture for very large-scale text analytics. *IBM Systems Journal*, 43(1):64–77, 2004.

[6] S. Josefsson. The base16, base32, and base64 data encodings. RFC 3548, 2003.

[7] D. Knuth. *The Art Of Computer Programming: Sorting and Searching*. Addison Wesley, 1973.

[8] H. Liefke and D. Suciu. Xmill: an efficient compressor for xml data. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 153–164, New York, NY, USA, 2000. ACM Press.

[9] OMG. Xml metadata interchange (xmi). http://www.omg.org/technology/documents/formal/xmi.htm, 2002.

[10] N. Sundaresan and R. Moussa. Algorithms and programming models for efficient representation of xml for internet applications. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 366–375, New York, NY, USA, 2001. ACM Press.

[11] W3C. Wap binary xml content format. http://www.w3.org/TR/wbxml/, 1999.

[12] W3C. Xml binary characterization. http://www.w3.org/TR/xbc-characterization/, 2005.