ASN1C Java Runtime Benchmarks

Ethan Metsger*

We discuss here some benchmarks for Objective Systems' ASN1C Java runtime. Examples when possible are taken from real-life applications; when such examples are not available, we provide examples using the sample programs provided with an ASN1C installation.

Disclaimer

This document is intended for informational purposes only and does not represent any guarantee of performance. The data contained in this document may not be consistent with the latest versions of ASN1C or newer hardware.

1 Introduction

This document provides benchmarking information for the ASN1C Java Runtime. These figures are provided for BER and PER unless otherwise noted. It should be noted from the outset that benchmarking any ASN.1 application is difficult and depends just as much on the complexity of the specification and chosen encoding rules as it does on the computer hardware on which the tests are performed. Performance will also vary with the code generation options selected by the customer; a reasonable set of defaults is chosen for the examples provided.

Benchmarking is further complicated by the metrics used to assess the application. Some options will optimize throughput while hurting latency or vice versa; other options may limit memory use at the expense of speed. The metric used in this report is the relative number of records per second that may be encoded or decoded.

*emetsger@obj-sys.com

2 Test System

The system used to test the software has the following relevant specifications:

- Intel Core2 Duo Processor (4300) clocked at 1.80GHz
- 3GB RAM
- Linux 10.04, kernel 2.6.32-26-generic
- Java version 1.6.0_22-b04
- ASN1C 6.3.4

To calculate the time involved in encoding and decoding, typical Java API calls were used to get the system time.

3 BER Benchmarks

The BER benchmarks are taken from custom projects developed by Objective Systems. These results use data provided by our customers to provide estimates of records processed per second as well as bytes per second. The latter metric is particularly significant as it demonstrates the affect of message size and complexity on throughput.

3.1 Ericsson R12

The Ericsson R12 specification results in approximately 360 generated files. These were collected into an API and slightly modified for a customer to permit serializing some of the data to CSV and XML. The results of some sample data are given below:

Test #	# of records	Decoding, s[1]	μ_{dec}, r/s	~μ_{dec}, b/s
1	19177	6.985	2745	293k
2	19222	6.334	3034	323k
3	19594	6.912	2835	296k
4	19603	6.930	2828	295k
5	19598	6.776	2892	302k

[1] During decode, records are converted to CSV.

3.2 Ericsson IN SCP3

The Ericsson SCP3 specification results in approximately 25 generated files. These were collected into an API and slightly modified for a customer to permit serializing some of the data to XML. The results of some sample data are given below:

Test #	# of records	Encoding, s[2]	μ_{enc}, r/s	μ_{enc}, b/s
1	8738	2.41	3625	870k
2	9806	2.30	4263	890k
3	4568	1.91	2391	1098k

3.3 TAP3 Batch Sample

For comparison purposes, we provide figures from the TAP3 Batch sample program. There are 120 generated files in the TAP3Batch sample. Results from running the sample Writer and Reader are below:

Test #	# of records	Encoding, s	Decoding, s	μ_{enc}, r/s	μ_{enc}, b/s	μ_{dec}, r/s	μ_{dec}, b/s
1	10000	6.218	49.255	1608	4070k	203	513k
2	10000	6.431	50.621	1555	3930k	198	500k
3	10000	6.342	49.361	1577	3990k	203	512k

The reference test here shows two things:

1. The JVM optimizes encoding when done in a naively repetitive way.
2. Message size will certainly affect decoding speeds.

To the first point, we note that the encoding is exceptionally fast—roughly four MB per second. This is not particularly surprising, since the data encoded are identical through the loop, and we are effectively concatenating the same encoding to itself 10,000 times. Decoding is not parallel, however, because of the nature of BER encodings (*i. e.*, back-to-front).

To the second point, we note that the performance in terms of records per second is significantly worse than in Ericsson R12. On the other hand, the number of bytes per

[2] This includes decoding from BER and serialization to XML.

second is nearly double. This is explained by the fact that the relative message size in the TAP3 specification is an order of magnitude larger, and that TAP3 is roughly half as complex (as measured by the number of generated files). The average message size in the R12 case is about 105 bytes, while the average message size in TAP3 is over two kilobytes.

We may conclude this section, then, by noting that the complexity of the specification and message size greatly influence the encoding and decoding speed. It is important to look at a variety of metrics to properly evaluate the performance of the runtime library.

4 PER Benchmarks

The PER benchmarks are taken from sample programs distributed with ASN1C.

PER presents complexity that is not present in BER, being a bit-packed encoding. However, PER messages are usually significantly smaller than their BER counterparts. This tradeoff often results in improved performance.

4.1 NBAP

This test uses a modified NBAP ASN.1 file and so presents less complexity than we might find in the regular 3GPP specifications. There are 82 generated files in this sample. Results from the test run are below:

Test #	# of records	Encoding, s	Decoding, s	μ_{enc}, r/s	μ_{enc}, b/s	μ_{dec}, r/s	μ_{dec}, b/s
1	10000	1.092	1.248	9158	339k	8013	296k
2	10000	0.932	1.252	10729	397k	7987	296k
3	10000	0.940	1.236	10638	394k	8091	299k

Here we see the relative performance in the number of records increase substantially, but the overall throughput remains relatively modest (in keeping with the Ericsson R12 sample program). This is hardly surprising, since PER encodings are more processor-intensive.

All NBAP encodings are octet-aligned; tests were run using table constraint encoding and decoding. Each message was 37 bytes long.

4.2 Employee

The employee sample program was chosen for its simplicity (seven files) and to show the relative differences between aligned and unaligned encodings. Following are encoding results:

Test #	# of records	Aligned, s	Unaligned, s	μ_a, r/s	μ_a, b/s	μ_u, r/s	μ_u, b/s
1	10000	2.804	2.772	3566	336k	3607	303k
2	10000	2.920	2.714	3424	321k	3686	309k
3	10000	2.824	2.775	3541	332k	3603	303k

Using unaligned encoding costs a little in terms of throughput, as the additional complexity required to encode the unaligned variant decreases output. Overall encoding time is consistently a little faster, however, reflecting the difference in overall output size (in this case, from 94 bytes in the aligned variant to 84 bytes in the unaligned variant). These messages were over twice the size of the NBAP messages, which seems to mitigate whatever performance increase we obtain by a simpler specification.

Following are decoding results:

Test #	# of records	Aligned, s	Unaligned, s	μ_a, r/s	μ_a, b/s	μ_u, r/s	μ_u, b/s
1	10000	2.632	2.596	3799	357k	3852	324k
2	10000	2.648	2.556	3776	355k	3912	329k
3	10000	2.592	2.540	3858	363k	3937	331k

These results are consistent with encoding. The throughput decreases in decoding as we switch from aligned to unaligned, but the overall time spent in decoding is also a little shorter. As noted in the encoding case, the unaligned encoding is about 10.5% smaller than the aligned encoding; this size reduction leads on average to a 8.5% decrease in throughput and a 2% decrease in decoding time (which isn't really significant).

These results will naturally vary from application to application; in some cases, the unaligned variant may decrease the message size more, and in some cases less. The selection is usually made based on the constraints of the devices that use those messages. So for example, the Radio Resource Control (RRC) schemas by convention are encoded with UPER, while NBAP and other elements of the 3GPP specification universe are usually encoded in aligned PER.

5 Conclusion

As with any benchmarking analysis, there is room for some criticism in this one. It serves as a beginning, however, for showing the relative difficulty in assessing the efficiencies of ASN.1 applications. Performance is largely dependent on features such as:

- encoding rules selected[3]
- specification complexity
- message size

Selecting metrics can also be a difficult problem. Looking at the number of records encoded and decoded per second can be deceiving and should be tempered with a comparison to the relative throughput.

In general, we may conclude from the results that BER is more sensitive to the complexity of the specification than is PER, whose bit-aligned philosophy correlates with greater demand on the processor. PER encodings are by nature significantly smaller than BER, however, which can help to mitigate the increased processor load. In cases where an *a priori* decision can be made about what encoding to use, other variables than performance should be considered (for example, target device constraints).

[3] And their variants. BER performance will vary based on whether indefinite-length or definite-length messaging is used; PER performance will vary based on the variant.