Investigating performance of serialization methods for networked data transfer in HPC applications

Max Yang
Georgia Institute of Technology
Atlanta, Georgia
myang300@gatech.edu

Thomas Stitt (Advisor)
Lawrence Livermore National Laboratory
Livermore, California
stitt4@llnl.gov

ACM Reference Format:
Max Yang and Thomas Stitt (Advisor). 2017. Investigating performance of serialization methods for networked data transfer in HPC applications. In *Proceedings of ACM/IEEE Supercomputing Conference, Denver, Colorado, USA, November 2017 (SC17),* 2 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Cluster-to-user data transfers present challenges with cross-platform byte-order compatibility and handling a variety of numeric types, and may occur over suboptimal network links. Two serialization libraries, Protocol Buffers and Conduit, are proposed to handle these obstacles transparently. Furthermore, a post-serialization Zlib-based compression stage is considered as a method to reduce the size of transferred data for users connected to a compute cluster over a low-bandwidth network link.

1.1 Serializer Overview

Protocol Buffers is an open-source binary wire serialization library developed by Google [1]. Messages are defined ahead of time in a `.proto` file, which is used to generate boilerplate serialization and deserialization code. Notably, Protobuf can encode integers in variable-length format.

Conduit is an open-source serialization library developed by Lawrence Livermore National Laboratory [2]. Serialized messages are JSON-based, and raw data is encoded as base64 data.

Both libraries handle byte-order changes across platforms of different endianness, and have bindings for C++ and Python.

2 TESTING PROCEDURE

Numeric arrays were constructed with both 32-bit integer and 64-bit floating-point numbers, broadly categorized into arrays of a single number, cyclic arrays, and real-world data. The real-world data consisted of simulation time-step data.

Each array was serialized 5000 times, with the elapsed time of each serialization run measured with CLOCK_MONOTONIC_RAW, a hardware clock not subject to NTP adjustments. The test system was a mid-2015 Macbook Pro, equipped with an Intel Core i7-4980HQ processor and 16GB of DDR3 RAM.

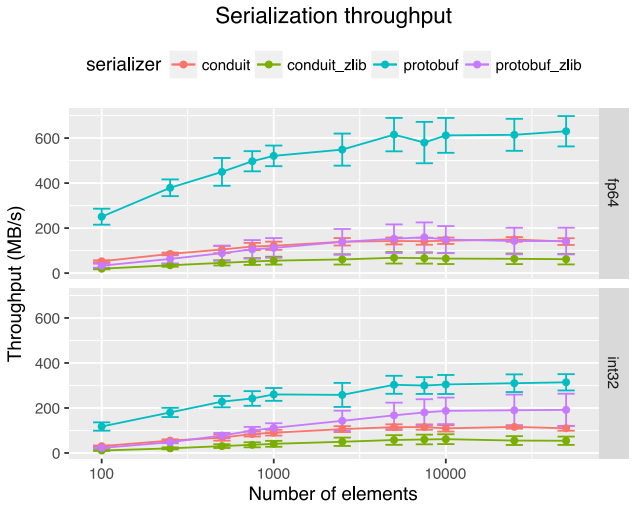

Figure 1: Serialization throughput for varying numbers of elements

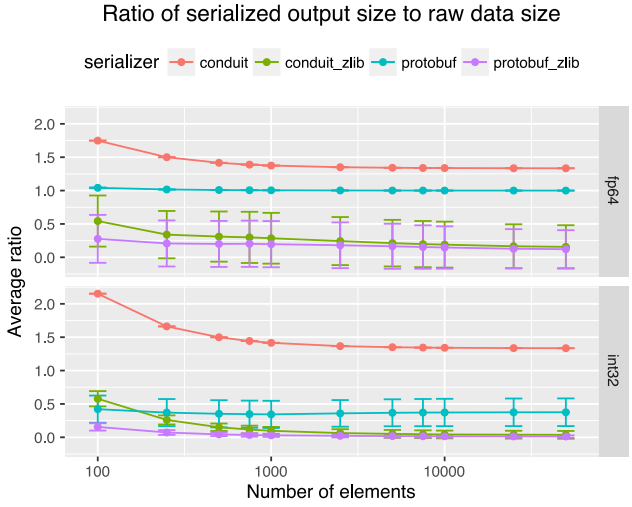

Figure 2: Encoding efficiency ratio for varying numbers of elements

SC17, November 2017, Denver, Colorado, USA
2017. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[LLNL-ABS-735821]

3 RESULTS

Throughput stabilizes for numeric arrays greater than 20KB. As seen in Figure 1, Protocol Buffers without compression has significantly higher serialization throughput than Conduit without compression, with Protobuf reaching 600MB/s for floating-point data with a size greater than 20KB. However, Protobuf is slower for integers, reaching around 300MB/s for integer data greater than 20KB. Conduit throughput is around 100MB/s for integer data, and around 140MB/s for floating-point data.

With regard to the size overhead of the serialization format, it is again observed in Figure 2 that Protobuf outperforms Conduit, with Protobuf achieving close to a 1:1 ratio for floating point numeric arrays. For integer numeric arrays, Protobuf can even produce output smaller than the original data. On the other hand, Conduit produces output around 1.33x the size of the original data.

Adding DEFLATE compression using Zlib dramatically shrinks the size of some types of messages. For cycles and constant arrays, ratios can be smaller than 1% of original data size. On real-world data, compression is somewhat less dramatic, but can still reach 10% of original data size for some data sets. Integer data does appear to have better compressibility compared to floating-point data. Compression does come at a cost of significant overhead, with throughput for compressed Protobuf dropping to 190MB/s and 140MB/s for integer and floating-point data, respectively; throughput of compressed Conduit also sees a significant drop to 55-60MB/s.

4 CONCLUSION

Protobuf and Conduit both provide a fast method of serializing scientific data for transport across heterogeneous platforms, with low protocol overhead. Protobuf offers superior performance in both size and speed, but requires predefining message types, which may increase development complexity, and also prevents it from parsing arbitrary message formats. Conduit, on the other hand, is self-describing, and also provides a programmatic model for constructing messages. Adding compression can reduce message sizes, at the cost of throughput; this may be advantageous in certain circumstances, such as when a slow connection between the cluster and the user exists. Overall, these results lead us to conclude that these serialization frameworks are both appropriate methods for networked communication in HPC applications.

ACKNOWLEDGMENTS

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

REFERENCES

[1] Google Developers. 2017. Protocol Buffers. (2017). Retrieved July 24, 2017 from https://developers.google.com/protocol-buffers/
[2] Lawrence Livermore National Lab. 2017. Conduit 0.2.1 documentation. (2017). Retrieved July 24, 2017 from https://software.llnl.gov/conduit/