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

Max Yang*, Thomas Stitt† (Advisor)

***Georgia Institute of Technology, †Lawrence Livermore National Laboratory**

Cluster-to-user data transfers present challenges with cross-platform endianness (byte-order) compatibility and handling a variety of numeric types, and may occur over suboptimal network links. Two serialization libraries, Protocol Buffers and Conduit, were selected for their ability to handle endianness and their cross-language support, and their performance in both size and speed was measured. It was found that the throughput of Protocol Buffers was significantly more than that of Conduit while exhibiting less protocol overhead. Adding a compression stage after serialization dramatically reduced the size of messages on certain types of data, but had some impact on throughput.

Introduction

Considerations in serialization and transmission of numeric data include:

- *Architecture differences*: LLNL clusters use both big-endian IBM POWER and little-endian x86 processors, so handling of byte-order differences may be necessary
- *Network performance*: Remotely-connected users may encounter adverse network performance, so smaller messages may be desirable
- *Multiple numeric array types*: Data can be any one of a variety of data types, such as floating-points and integers of varying byte-width

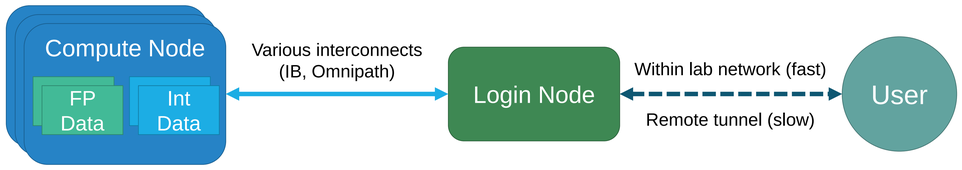
We consider a variety of serialization protocols based on how they solved these challenges, and in the end selected two for evaluation:

- Protocol Buffers, an open-source binary wire serialization library developed by Google
- Conduit, an open-source JSON-based serialization library developed by LLNL

Both libraries support cross-platform byte-order handling and have bindings for C++ and Python. However, while Conduit uses a code-first model to define messages, Protobuf requires a separate message definition file.

Furthermore, we evaluate the impact of adding Zlib-based compression after serialization.

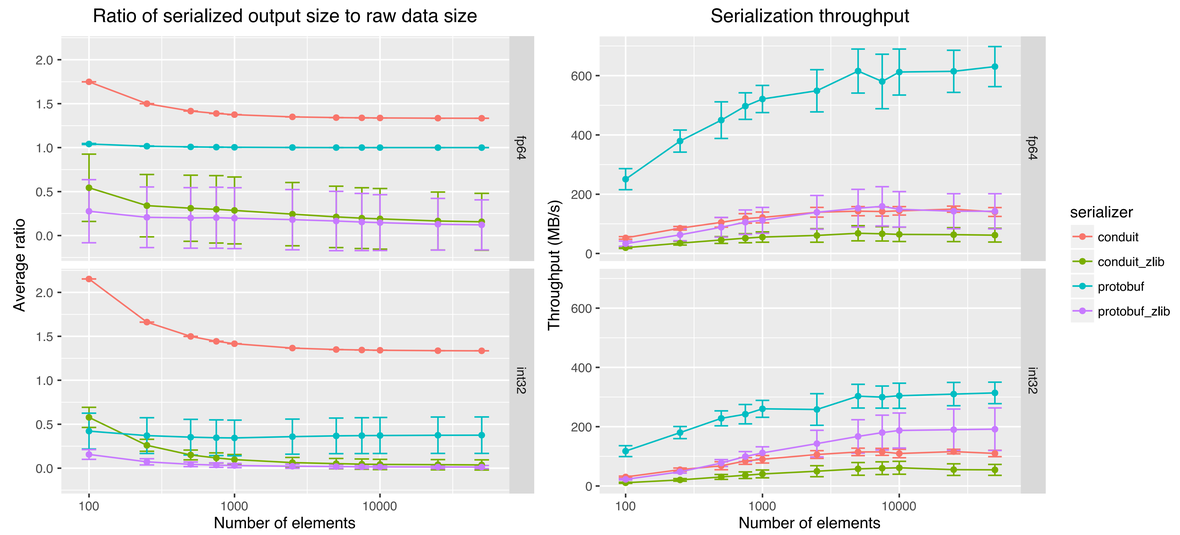
Simulation Topology



Methods

- Tests conducted on Macbook Pro (Intel Core i7-4980HQ @ 2.8GHz, 16GB DDR3 RAM)
- Tested Protobuf, Conduit, Protobuf+Zlib, Conduit+Zlib
- Measure serialization runtimes with POSIX's CLOCK_MONOTONIC_RAW (no NTP adjustments)
- Test on variety of datasets:
  - 32-bit integers or 64-bit floating-points
  - Repeated: all zeroes, all 1M, all pi
  - Cycles: one to ten repeating, each number 10 times starting from one, sparse-n
  - Real-world data: time, time step, solver iterations

Results



Discussion

Protobuf achieves significantly higher throughput and less serialization overhead than Conduit. In particular, Protobuf can achieve serialized/original data ratios better than 1:1 for integer data, due to its variable-length integer encoding; however, this results in Protobuf having slower throughput for integers as compared to floating-point numbers.

The use of Zlib reduced serialized output sizes as expected, at a considerable cost to throughput.

Conclusion

- Protobuf optimal in cases where performance is critical
- Compression can help users connecting over slow network links, but performance impact should be considered

Potential Future Research

- Different compression algorithms/levels
- Applicability to intra-cluster communication

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

LLNL-POST-735837