Serialization Tests

Alexey Rybalchenko, GSI Darmstadt • 12.02.2016

Motivation

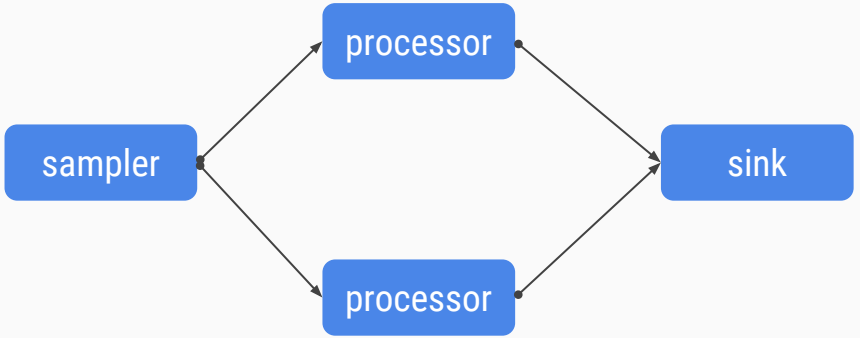
- **Extend the previous serialization tests done with FairMQ with potentially more efficient approaches.**

- **Evaluate how can the framework make best use them, either for internal use (protocol) or user side. E.g.: preallocating data buffer (filled by user), or taking the one provided by the user.**

- **Evaluate their features and performance - encoding/decoding and transferring of binary blobs.**

Examples in FairRoot (1/2)

Simple topology for the digitization->reconstruction step of FairRoot Test Detector example.



Contains data format tests for:

- boost::serialization
- Google Protocol Buffers
- ROOT TMessage
- raw binary transfer (with some manual copying to create contiguous objects)

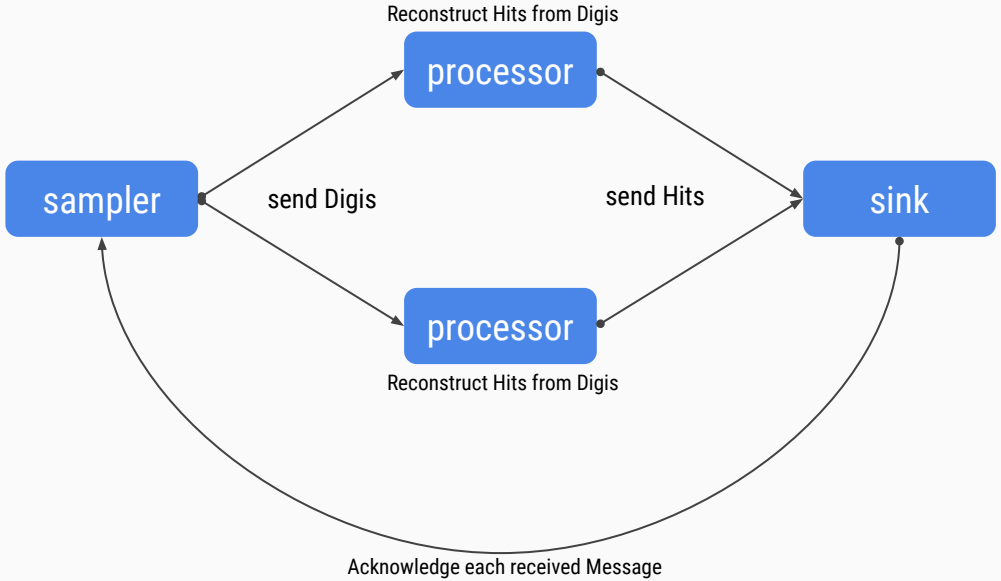
Extend it for:

- MessagePack
- Google FlatBuffers

Examples in FairRoot (2/2)

- Sampler sends arrays of Digis (created by digitization task).
- Processor uses the Digi data for (very simple) reconstruction that produces Hits. Sends arrays of Hits
- Sink receives hits and writes them to a file in Root format.
- Sink sends confirmations of received messages back to sampler for measuring the overall performance.

Messages: arrays of ~25-40 Digis or Hits.
Digi: { 3 x int, 2 x double }
Hit: { 2 x int, 8 x double }



Google FlatBuffers

Notable FlatBuffers features:

- Access to serialized data without parsing/unpacking.
- The entire buffer is allocated at the beginning.
- Random access and simple mutability (receive -> modify -> send without copying).
- Fixed schema definition (generates accessors code).
- Schema evolution.
- Nesting.
- Optional fields (do not take up space in the buffer) & Default values.
- C++, C#, Go, Java, JS, PHP, and Python, more coming (but with varying feature set).

Latest release: 1.3 (2016-02-03)

Schema example:

```
namespace TestDetectorFlat;

table Digi {
  x:int;
  y:int;
  z:int;
  timestamp:double;
  timestampError:double;
}

table DigiPayload {
  digis:[Digi];
  buffer:[ubyte];
}

root_type DigiPayload;
```

MessagePack

Notable MessagePack features:

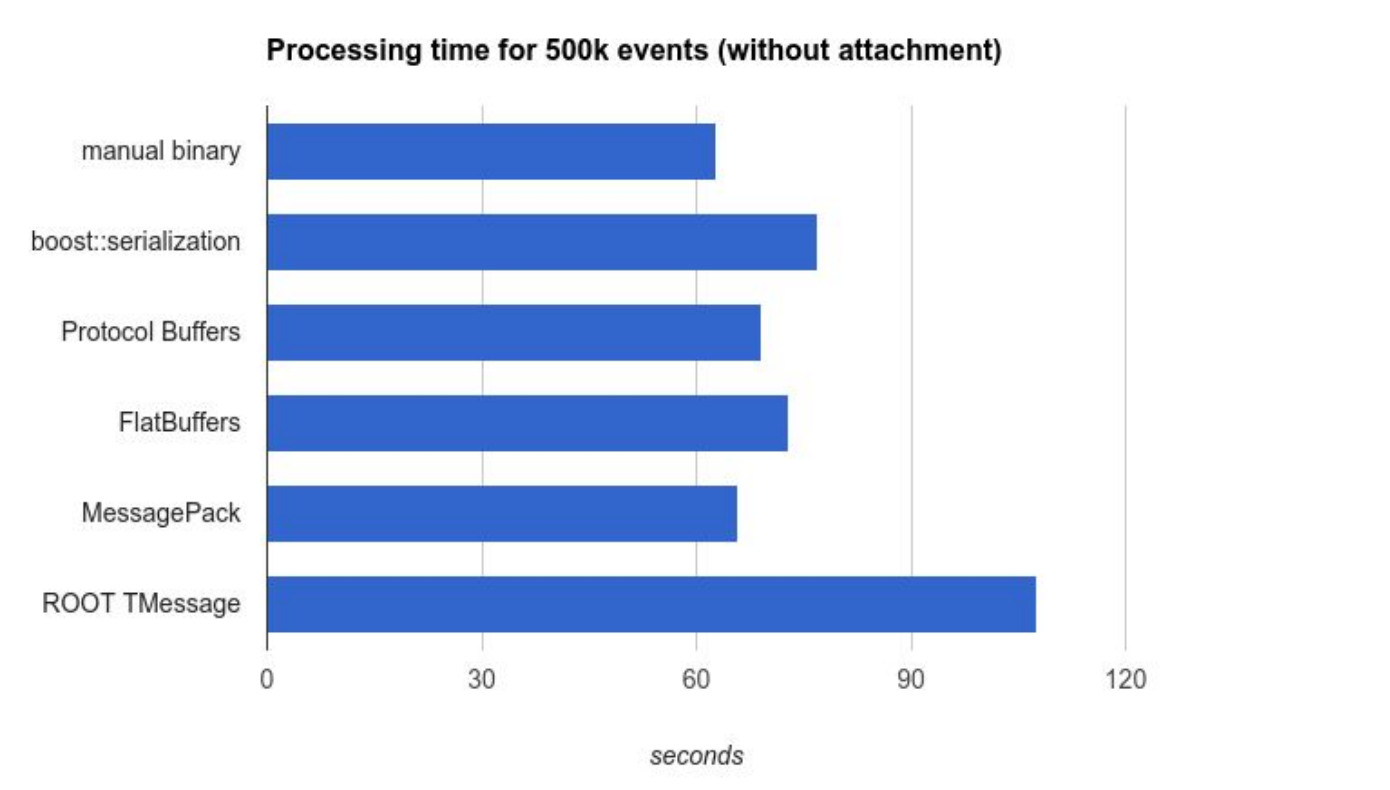
- The C++ interface supports referencing buffer in addition to the simple copying buffer. This produces a scatter buffer in a zero-copy way. Msgpack examples and tests then copy the data into a simple buffer.
- Support for streaming.
- Can serialize user types (requires adding special macros to the classes).
- No schema definition necessary.
- Supports over 50 languages (but with varying feature set).

Latest release: 1.4 (2016-01-22)

Performance without attachment

example topology (see prev. slides): (De-)serializing & transferring data, running trivial processing task on the Processor

Messages:
arrays of ~25-40
Digis or Hits.
Digi: { 3 x int, 2 x double }
Hit: { 2 x int, 8 x double }



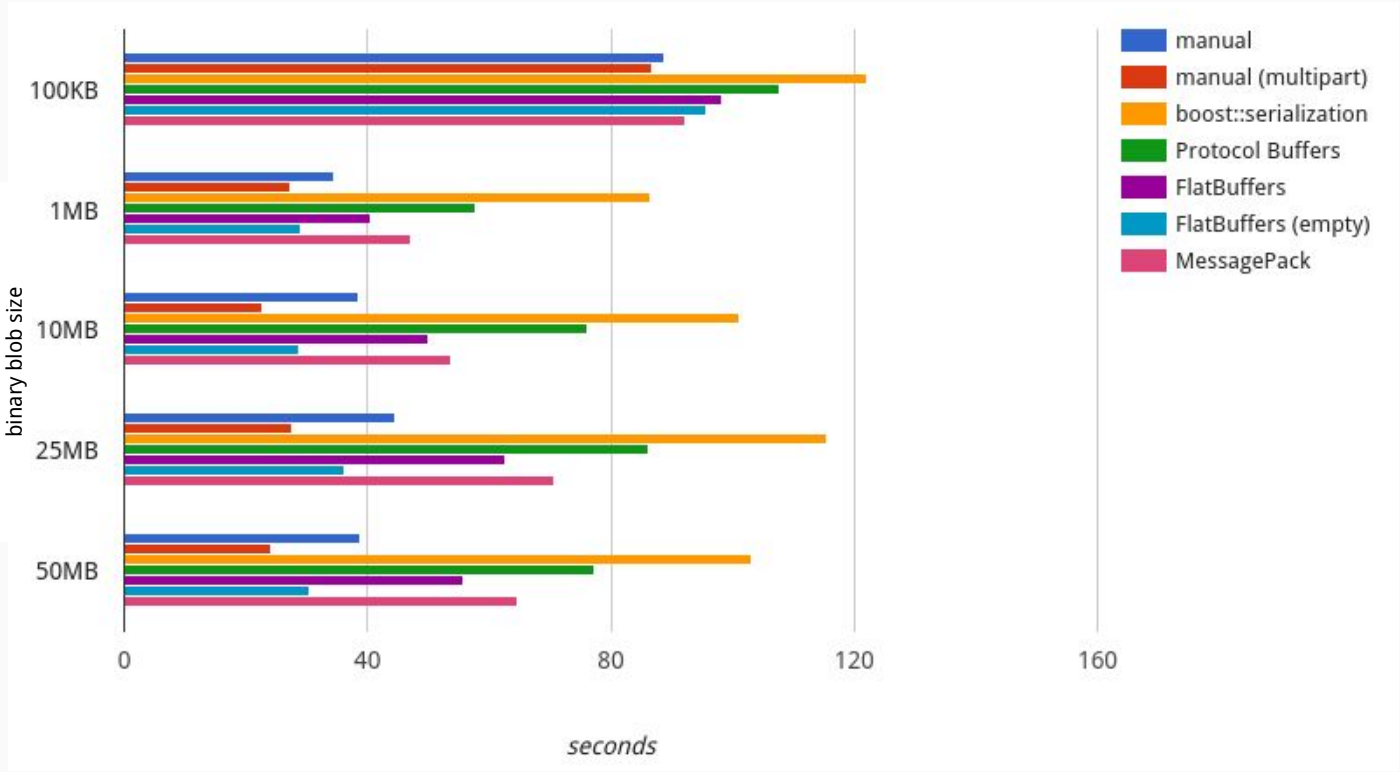
Hardware: Intel Xeon E5-1607 v3 (4 cores), 8GB RAM

Performance with binary blob attachment

Messages:
serialized arrays of
~25-40 Digis or Hits.
Digi: { 3 x int, 2 x double }
Hit: { 2 x int, 8 x double }
+
a binary blob attachment of a specified size

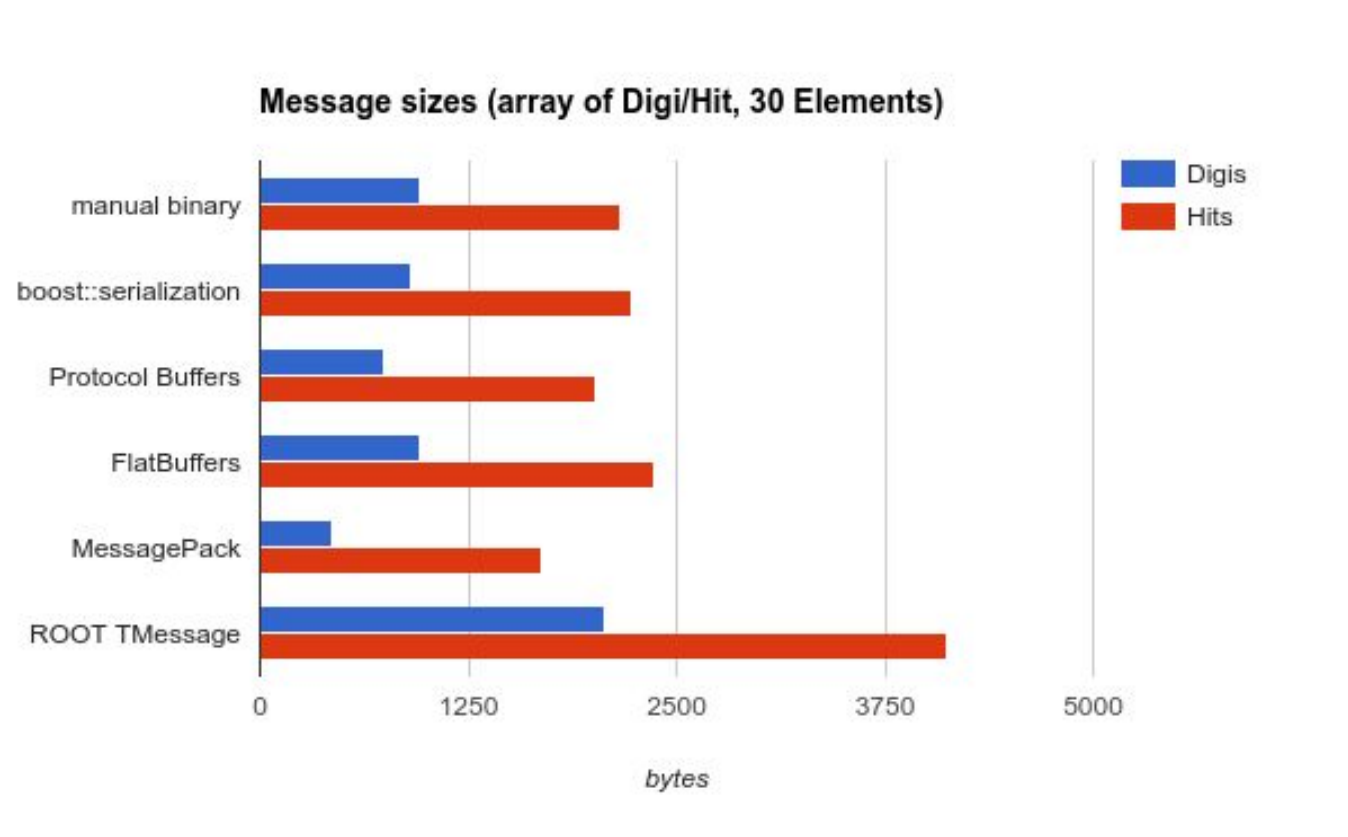
Number of messages:
50 GB / blob size

multipart available in ZeroMQ transport. nanomsg has very limited scatter buffers as alternative.



Hardware: Intel Xeon E5-1607 v3 (4 cores), 8GB RAM

Message sizes



Hardware: Intel Xeon E5-1607 v3 (4 cores), 8GB RAM

Summary

	schema	random access	mutability	“zero-copy”	encoding user defined types	...	links
boost	no	no	no	no	yes, with or without modification of the classes		www.boost.org/doc/libs/release/libs/serialization/
msgpack	no	no	no	yes, producing scatter buffer	yes, with small modification of the classes		msgpack.org
protobuf	yes, very flexible	no	yes	no	only manually		developers.google.com/protocol-buffers
flatbuffers	yes, very flexible	yes	yes, limited	skipping encode/decode step	only manually		google.github.io/flatbuffers/index.html

https://github.com/rbx/FairRoot/tree/bigger-buffer/examples/advanced/Tutorial3/MQ