EPJ Web of Conferences **214**, 05022 (2019) https://doi.org/10.1051/epjconf/201921405022
CHEP 2018

RDMA-accelerated data transport in ALFA

Dennis **Klein**[1,*], *Alexey* Rybalchenko[1,**], *Mohammad* Al-Turany[1,***], and *Thorsten* Kollegger[1,****]

[1]GSI Helmholtz Centre for Heavy Ion Research GmbH, Darmstadt, Germany

Abstract. ALFA is a modern software platform for simulation, reconstruction and analysis of particle physics experiments. The FairMQ library in ALFA provides building blocks for distributed processing pipelines in anticipation of high data rates in next-generation, trigger-less FAIR and LHC RUN3 ALICE experiments. Modern data transport technologies are integrated through FairMQ by implementing an abstract message queuing based transport interface. Current implementations are based on ZeroMQ, nanomsg and shared memory and can be selected at run-time. In order to achieve highest inter-node data throughput on high bandwidth network fabrics (e.g. Infiniband), we propose a new FairMQ transport implementation based on the `libfabric` technology.

1 Introduction

In anticipation of high data rates in next-generation, trigger-less FAIR and LHC RUN3 ALICE experiments, the FairMQ library [1] is being developed as part of the ALFA software framework [2, 3]. FairMQ supports a data-flow driven computing model by providing building blocks to construct distributed processing pipelines. Each processing stage in the pipeline receives and emits data via a stream of messages through an abstract message queuing based data transport interface.

FairMQ offers multiple implementations of its abstract data transport interface in order to integrate existing data transport technologies. Currently, implementations for ZeroMQ [4], nanomsg and shared memory [5] are available. The user of FairMQ can select a transport implementation at run-time per input/output channel. It is supported to pass messages from one channel to another even with different transports selected. This enables the user to choose the most suitable transport technology to pass data between different processing stages.

Furhermore, FairMQ adopts the concept of scalability protocols from ZeroMQ which encapsulate generic patterns in message routing, scheduling, and ordering commonly seen in distributed systems. Currently, FairMQ provides PAIR (bidirectional, blocking, one-to-one communication), PUSH/PULL (unidirectional, blocking, one-to-many communication with round-robin routing), REQUEST/REPLY (bidirectional, blocking, many-to-one communication with source routing), and PUBLISH/SUBSCRIBE (unidirectional, non-blocking, broadcasting, one-to-many communication).

*e-mail: d.klein@gsi.de
**e-mail: a.rybalchenko@gsi.de
***e-mail: m.al-turany@gsi.de
****e-mail: t.kollegger@gsi.de

Table 1. FairMQ transport matrix.

Node	Process	Address format	Transport			
			`zeromq`	`nanomsg`	`shmem`	**`ofi`** (new)
intra	intra	`inproc://endpoint`	**++**	**++**	n/a	n/a
intra	inter	`ipc://endpoint`	+	+	**++**	-
		`tcp://host:port`	+	+	**++**	+
inter	inter	`tcp://host:port`	+*	+	n/a	+**
		`verbs://host:port`	-	-	n/a	**++**

+ supported with limited bandwidth
++ supported with full bandwidth
- no support planned
* no bandwidth limit on Ethernet [6]
** development driver only, not intended for production use
n/a not available (combination not meaningful)

1.1 RDMA-accelerated data transport

FairMQ is offering efficient intra-node transports, but is lacking an efficient inter-node transport (see table 1). Currently, all FairMQ inter-node transports use the TCP/IP stack provided by the operating system. Unfortunately, even highly tuned Linux TCP/IP implementations limit the usable link capacity significantly [6, Section 6.1].

In the light of the expected high data rates, we cannot settle with a software stack, that can only utilize half or less of the available network bandwidth. So, we propose a new FairMQ transport called `ofi`, based on `libfabric` [7], ZeroMQ [4] and Boost.Asio [8]. Bulk data transfers are accelerated via RDMA transfers, which can achieve highest network throughput and lowest usage of CPU.

2 Design and Implementation

In this section we highlight the most important design decisions and implementation details of the new `ofi` transport.

2.1 Architectural View

Instead of reinventing the wheel, our `ofi` transport relies on a number of industry-proven, and open source technologies: `libfabric`, Boost.Asio, and ZeroMQ.

`libfabric` provides Remote Direct Memory Access (RDMA) services on a variety of fabrics, e.g. Omni-Path, InfiniBand, Cray GNI, Blue Gene, iWarp RDMA Ethernet, RoCE and others. `libfabric` provides inherently asynchronous, expert-level C APIs through which one can perform RDMA data transfers bypassing expensive operating system I/O routines. The current implementation of the `ofi` transport targets InfiniBand and enables the `fi_verbs` provider. This can easily be extended to other fabrics by enabling other providers.

Boost.Asio was chosen, because it provides a mature and high-performance asynchronous C++ programming model, specifically designed for network programming. We developed Boost.Asio C++ bindings around a subset of the `libfabric` C APIs to ease usage and increase maintainability of our codebase. Furthermore, we integrated `libfabric` wait objects with Boost.Asio's portable abstraction of the native I/O event loop of the operating system. This enables composition with wait objects from other libraries without the risk of oversubscribing resources; as the execution context (e.g. a thread pool) is supplied by the user. The `libfabric` Boost.Asio C++ bindings are developed as a standalone open source project called `asiofi`[9] to maintain an un-biased and generic context. We believe this is beneficiary

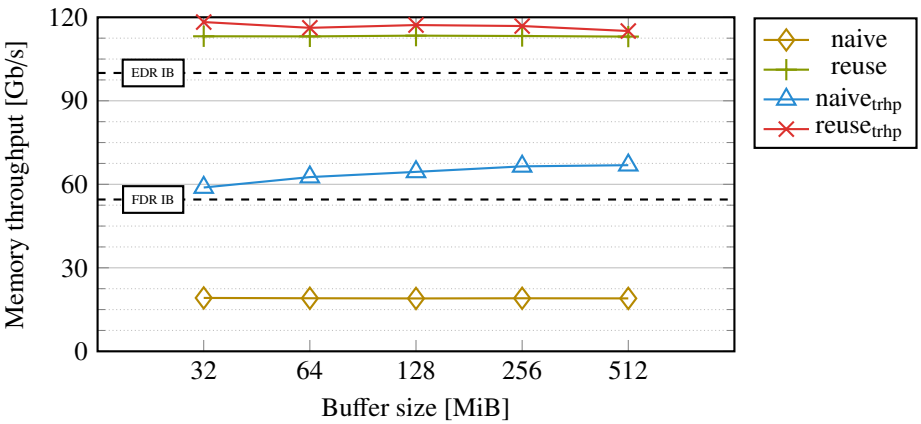
to the quality of `asiofi` and we hope, that a standalone `asiofi` will be more useful to other C++ projects that need such functionality.

Over a ZeroMQ based control band we setup the RDMA transfer with the remote endpoint (e.g. allocate an appropriate receive buffer). The RDMA transfer itself is performed on the `asiofi` based data band. ZeroMQ was a natural choice for the control band, because it is used already heavily in FairMQ.

2.2 Memory Management

Memory management is the key to maximise throughput in combination with RDMA engines. The page fault handling of the virtual memory system is a costly operation that can limit memory throughput significantly. A page fault occurs, when a freshly allocated virtual memory page is accessed the first time. The page fault handler allocates a free physical memory page and creates the appropriate entry in the page tables. Figure 1 shows the results of micro benchmarking memory throughput with different memory management strategies. While increasing the page size already helps significantly, best memory throughput was achieved by reusing already allocated buffers.

Figure 1. Memory Throughput with different memory management strategies. The naive benchmark allocates a buffer, writes to each byte, then deallocates the buffer. The reuse benchmark reuses the same buffer and effectively has to pay the cost of the page fault handlers only once. $_{\text{trhp}}$ means that transparent hugepages of 2 MiB were used (instead of 4kiB). Measurements were performed on a Intel Core i5-6200U 2.8GHz CPU with 16GB Micron PC4-2133 DDR4 SDRAM main memory.



Based on the previous observations we replaced the global allocator with a local pool allocator, that only occasionally falls back to the global allocator to grow its memory pool. On deallocation the buffer is returned to the memory pool. It stays allocated (from the point of view of the operating system) and is available for reuse.

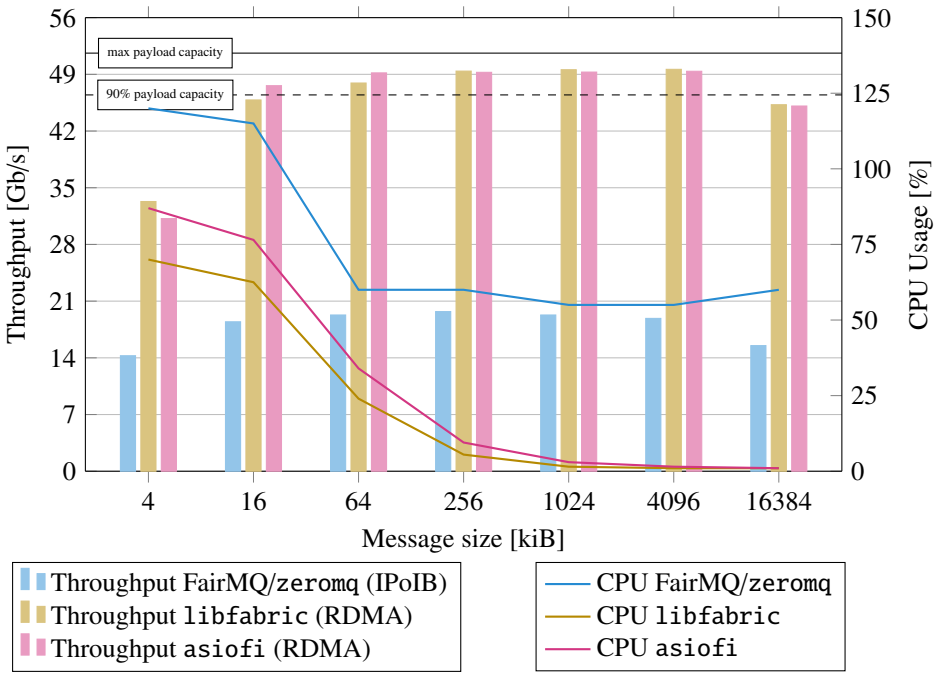
While global allocators on modern Linux distributions (e.g. `malloc`) also include optimizations to workaround the cost of page faults, we found that those are effective for very small buffer sizes only.

3 Results

Figure 2 shows a first performance evaluation of the `asiofi` library developed and used in the new FairMQ `ofi` transport on FDR Infiniband (54.54 Gb/s). Over 90% of the theoretical link

capacity for application payload throughput is achieved (FDR InfiniBand uses a 64/66 bit line encoding and 26-98 Bytes for the InfiniBand wire protocol per MTU (maximum transmission unit) which we do not include in the measurements). Furthermore, a significantly reduced CPU usage compared to the IPoIB based transport is visible. The CPU overhead introduced by `asiofi` compared to the native `libfabric` is due to the way the benchmark programs handle the completion of the asynchronous RDMA operations. In `asiofi` a distinct callback per operation is called, while the `libfabric` API allows to acknowledge the completion of multiple operations in a single step. In any real world application, such cheap completion handling is not relevant, because one usually wants to process the received data buffer further. The minor differences in throughput between native `libfabric` and `asiofi` have not been investigated in detail. The fact that `asiofi` is even faster in two cases suggests that the differences lie within the measurement error on this given hardware.

Figure 2. Throughput and CPU usage benchmark. One-to-one connection, FDR InfiniBand link (54.54 Gb/s). CPU usage is the arithmetic mean of CPU usage (Linux semantics, 100% means one CPU core) of the two communicating processes. Measurements were performed on two dual socket Intel Xeon E5-2660 v3 @2.60GHz CPU boards, with 128 GB DDR4-2133 reg ECC RAM and a Mellanox ConnectX-3 network adapter each.



4 Conclusion

A new FairMQ transport called `ofi` that is based on `libfabric`, Boost.Asio, and ZeroMQ is in development. It complements the existing suite of FairMQ transports with an RDMA-accelerated and therefore a memory and CPU efficient inter-node transport. With this new transport, the ALFA framework will provide optimized networking support for virtually all relevant high-performance fabrics in our community.

C++ Boost.Asio bindings for a subset of the `libfabric` C APIs were developed as a standalone open source project called `asiofi` [9]. It is used to implement the data band in

a FairMQ `ofi` transport connection. A first performance evaluation of `asiofi` meets the expected throughput and CPU consumption on FDR Infiniband.

References

[1] FairMQ, `https://github.com/FairRootGroup/FairMQ`, visited 4.3.2019

[2] M. Al-Turany et al., Journal of Physics **Conference Series 664**, 2015, "ALFA: The new ALICE-FAIR software framework"

[3] FairRoot, `https://github.com/FairRootGroup/FairRoot`, visited 4.3.2019

[4] ZeroMQ, `http://zeromq.org/`, visited 4.3.2019

[5] A. Rybalchenko et al., CHEP 2018, "Shared Memory Transport for ALFA"

[6] A. Wegrzynek and ALICE Collaboration 2017, Journal of Physics **Conference Series 898 032026**, 2017, "Using ALFA for high throughput, distributed data transmission in the ALICE O^2 system"

[7] OpenFabrics Interfaces `libfabric`, `https://ofiwg.github.io/libfabric/`, visited 4.3.2019

[8] Boost, `https://www.boost.org`, visited 4.3.2019

[9] asiofi, `https://github.com/FairRootGroup/asiofi`, visited 4.3.2019