Distributed data processing pipelines in ALFA

Alexey Rybalchenko, Dennis Klein, Mohammad Al-Turany
GSI Helmholtzzentrum für Schwerionenforschung GmbH, Planckstraße 1, 64291 Darmstadt, Germany

E-mail: a.rybalchenko@gsi.de

Abstract.
The common ALICE-FAIR software framework ALFA offers a platform for simulation, reconstruction and analysis of particle physics experiments. FairMQ is a module of ALFA that provides building blocks for distributed data processing pipelines, composed of components communicating via message passing. FairMQ integrates and efficiently utilizes standard industry data transport technologies, while hiding the transport details behind an abstract interface. In this work we present the latest developments in FairMQ, focusing on the new and improved features of the transport layer, primarily the shared memory transport and the generic interface features. Furthermore, we present the new control and configuration facilities, that allow programmatically controlling a group of FairMQ components. Additionally, new debugging and monitoring tools are highlighted. Finally, we outline how these tools are used by the ALICE experiment.

1. Introduction

The ALICE-FAIR software framework, ALFA[1], offers a robust platform for the simulation, reconstruction, and analysis of particle physics experiments. One of the critical components of this framework is FairMQ[2], a module that provides building blocks for creating distributed data processing pipelines. This module is designed to integrate seamlessly with standard industry data transport technologies, offering a high-level, abstract interface for communication between the different components. In this work we explore the latest developments in FairMQ, highlighting new and improved features.

One of the transports provided by FairMQ is the shared memory transport[3]. This transport is designed to handle the large amounts of data generated by particle physics experiments, providing an efficient way to transfer data between processes on the same node without copy of the data buffers. The shared memory transport has been extended with support for event notifications, shared buffer ownership and externally created buffers.

Another important aspect of FairMQ is its control and configuration facilities, which allow for programmatic control over a group of FairMQ processes. This feature enables the creation of complex processing pipelines that can be easily managed and reconfigured as needed. The control facilities have evoled into a separate package called ODC (Online Device Control)[4], which uses DDS (Dynamic Deployment System)[5] under the hood. We will showcase these control facilities and their primary features.

Furthermore, we will highlight the new debugging and monitoring tools that have been introduced in FairMQ. These tools provide valuable insights into the performance and behavior of the processing pipelines, making it easier to diagnose and resolve issues.

In conclusion, we will outline how these tools are used in production by the ALICE experiment, demonstrating the practical application of FairMQ in a real-world scenario. Overall, the latest developments in FairMQ and ODC provide comprehensive and flexible building blocks for the distribution and processing of particle physics data. With its efficient transport layer, programmatic control and configuration facilities, and debugging and monitoring tools, FairMQ is a critical component of the ALICE-FAIR software framework and an essential tool for particle physics experiments.

2. Core Concepts

FairMQ follows a number of general goals, including a unified API to different data transports, hiding transport-specific details from the user, allowing for the transparent combination of different transports in one device, and transport switching via configuration without modifying user code.

FairMQ can be divided into two conceptual parts: a library component and a framework component. Library components offer APIs for data communication, exposing concepts such as *Messages* and *Channels*, that can be used as building blocks for a flexible and efficient communication. Framework components provide an opinionated way to build controllable and configurable processes with specific goals in mind.

The main framework concept of the FairMQ library is the *Device*, which is composed of a state machine, program options, and command/configuration plugins. Devices are composed into topologies, representing processing graphs. Independent topologies are isolated in *sessions*. This modular and scalable design allows for the creation of complex online processing graphs that can be easily modified and extended to meet evolving experimental requirements.

The library follows an ownership model, where a Message has single ownership by the process it is created/received in. Additionally, it is possible to allow the transport to not physically copy the underlying buffer of a message, but to share it between multiple messages if this is suitable for efficiency. However, this effectively means that a buffer copied in this way becomes read-only and modifying it afterwards is undefined behaviour.

3. Data Transport

The abstract message passing interface of FairMQ is implemented by the following three transports:

- TCP Network transport, based on ZeroMQ[6].
- RDMA (Remote Direct Memory Access) Network transport, based on OFI[7].
- Shared memory transport[3], based on ZeroMQ and boost::interprocess library[8].

The main differences between the transports are highlighted in Table 1. Transport implementation can be switched via runtime options and can also be efficiently combined within a single device or a topology as needed.

Since the introduction of the Shared Memory transport several new features have been introduced:

- Shared memory event notifications. Device can subscribe to memory region creation/destruction events in order to obtain region address, size and settings prior to data transfer. This can be used, for example, to register the memory areas for use with GPU or RDMA.
- Shared ownership for the message buffers. This is a performance optimization allowing the transport to share the underlying memory buffers, which the user can trigger with an API call. This can be used when multiple processes need to access the same buffer in parallel. Modifying shared buffers is undefined behaviour.

Table 1. FairMQ transports. Legend: '++' - supported without a data copy, '+' - supported with a data copy, '-' - not supported, 'n/a' - combination makes no sense.

	address format	ZeroMQ	shmem	OFI
intra-process	`inproc://endpoint`	++	n/a	n/a
inter-process	`ipc://endpoint`	+	++	-
	`tcp://host:port`	+	++	+
inter-node	`tcp://host:port`	+	n/a	+
	`verbs://host:port`	-	n/a	++

- Support for externally created regions. User can create (and on demand quickly reset) shared memory outside of a FairMQ session. This can be helpful when a long memory registration process is involved, which in this case can be done only once, while keeping (and resetting) the memory between sessions. No two sessions can use it in parallel however.

4. Device Control

A topology of FairMQ devices can be launched and controlled via the ODC (Online Device Control) component. ODC acts as a command broker between an Experiment Control System and one or many topologies of FairMQ devices. ODC shows a homogeneous topology state to the ECS. The task of process deployment and command exchange between them is implemented via the APIs of the DDS (Dynamic Deployment System) component.

ODC provides a gRPC server component and a sample gRPC client. A fixed set of commands is available to launch deployments with FairMQ topologies, control device states and configure device properties. A single server process can handle multiple DDS/FairMQ sessions in parallel and reconnect to those which are running prior to server launch.

A simple command line controller is also provided that can be used for testing without gRPC. Resource management is delegated to the resource managers supported by DDS, such as Slurm.

5. Debugging & Monitoring

Shared memory transport provides a tool to monitor and debug the status of shared memory and existing messages. The tool provides data for a given FairMQ session, such as:

- Used shared memory segments.
- Total and used size for every segment.
- Information about the creator user and process.
- Number of devices in the session.
- List of messages in a session, with their size, creator ID, memory location, creation timestamp (available in debug mode).

These debugging features are also provided via an API for integration in user software. This can be especially interesting for message object information - knowing the data types contained in the buffers can help to extract additional debugging info, e.g. from data headers.

For general information on a running topology ODC provides status and state commands, that provide detailed information on the running sessions, topologies and devices.

6. Use in the ALICE experiment

The upgraded ALICE experiment at CERN uses FairMQ in several parts of their system. Some parts use only the library components of FairMQ, such as the O2 Readout software. This is

deployed on the First Level Processors (FLPs) for detector readout. Other parts use FairMQ framework and library components as building blocks for a higher level framework, called Data Processing Layer (DPL)[9][10][11], used for global reconstruction and data aggregation. The deployment of FairMQ devices on EPNs involves approximately 70,000 devices per session spread across 200 nodes. Each node handles around 350 devices, which communicate through shared memory. The topologies are established and managed through ODC using DDS and Slurm. As the final output data rates of the upgraded experiment are anticipated to be higher, the number of nodes is expected to increase to 1500, with the same or higher number of devices per node.

7. Conclusion

In this paper, we have presented the latest developments in the FairMQ module, which is a key component of the ALICE-FAIR software framework. We have shown how FairMQ provides an efficient and flexible platform for distributed data processing pipelines, with support for shared memory transport, event notifications, shared buffer ownership, and externally created buffers.

We have also highlighted the control and configuration facilities of FairMQ, provided by the ODC package, which enable the creation of complex processing pipelines that can be easily managed and reconfigured. Additionally, we have discussed the new debugging and monitoring tools that have been introduced in FairMQ, which provide valuable insights into the performance and behavior of the processing pipelines.

Importantly, we have demonstrated the practical application of FairMQ in a real-world scenario by showcasing its use in the ALICE experiment. This illustrates the critical role that FairMQ plays in the distribution and processing of particle physics data.

In summary, the latest developments in FairMQ and ODC offer a comprehensive and flexible set of building blocks for the simulation, reconstruction, and analysis of particle physics experiments. With its high-speed transport layer, programmatic control and configuration facilities, and powerful debugging and monitoring tools, FairMQ is a crucial component of the ALICE-FAIR software framework and an essential tool for particle physics research.

References

[1] Al-Turany M et al 2015 "ALFA: The new ALICE-FAIR software framework" *J. Phys.: Conf. Ser.* **664** 072001
[2] Al-Turany M et al 2014 "Extending the FairRoot framework to allow for simulation and reconstruction of free streaming data" J. Phys.: Conf. Ser. **513** 022001
[3] Rybalchenko A et al 2019 "Shared memory transport for ALFA", *EPJ Web of Conf.* **214** 05029
[4] ODC main repository https://github.com/FairRootGroup/ODC accessed: 2023-02-28
[5] Lebedev A and Manafov A 2019 "DDS: The Dynamic Deployment System", *EPJ Web of Conf.* **214** 01011
[6] ZeroMQ website http://zeromq.org/ accessed: 2023-02-28
[7] Klein D et al 2019 "RDMA-accelerated data transport in ALFA", *EPJ Web of Conf.* **214** 05022
[8] Boost website https://www.boost.org/doc/libs/1_81_0/doc/html/interprocess.html accessed: 2023-02-28
[9] Eulisse G, Konopka P, Krzewicki M, Richter M, Rohr D and Wenzel S 2019 "Evolution of the ALICE Software Framework for Run 3", *EPJ Web of Conf.* **214** 05010
[10] Eulisse G, Alkin A, Grosse-Oetringhaus J F, Hristov P, Innocenti G M and Kabus M J 2020 "Data Analysis using ALICE Run 3 Framework", *EPJ Web of Conf.* **245** 06032
[11] Alkin A, Eulisse G, Grosse-Oetringhaus J F, Hristov P and Kabus M 2021 "ALICE Run 3 Analysis Framework", *EPJ Web of Conf.* **251** 03063