Journal of Physics: Conference Series **664** (2015) 072001 doi:10.1088/1742-6596/664/7/072001

ALFA: The new ALICE-FAIR software framework

M. Al-Turany[1,2], P. Buncic[2], P. Hristov[2], T. Kollegger[1], C. Kouzinopoulos[2], A. Lebedev[1], V. Lindenstruth[1,3], A. Manafov[1], M. Richter[2,4], A. Rybalchenko[1], P. Vande Vyvre[2], N. Winckler[1]

[1]GSI Helmholtzzentrum fr Schwerionenforschung GmbH, Planckstrae 1, 64291 Darmstadt
[2]CERN, European Laboratory for Particle Physics. 1211 Geneve 23, Switzerland
[3]FIAS, Frankfurt Institute for Advanced Studies, Ruth-Moufang-Strae 1, 60438 Frankfurt, Germany
[4]Department of Physics, University of Oslo, Norway

E-mail: Mohammad.al-turany@cern.ch

Abstract. The commonalities between the ALICE and FAIR experiments and their computing requirements led to the development of large parts of a common software framework in an experiment independent way. The FairRoot project has already shown the feasibility of such an approach for the FAIR experiments and extending it beyond FAIR to experiments at other facilities[1, 2]. The ALFA framework is a joint development between ALICE Online-Offline (O^2) and FairRoot teams. ALFA is designed as a flexible, elastic system, which balances reliability and ease of development with performance using multi-processing and multi-threading. A message- based approach has been adopted; such an approach will support the use of the software on different hardware platforms, including heterogeneous systems. Each process in ALFA assumes limited communication and reliance on other processes. Such a design will add horizontal scaling (multiple processes) to vertical scaling provided by multiple threads to meet computing and throughput demands. ALFA does not dictate any application protocols. Potentially, any content-based processor or any source can change the application protocol. The framework supports different serialization standards for data exchange between different hardware and software languages.

1. Introduction

ALFA is the new ALICE-FAIR concurrency framework for high quality parallel data processing and reconstruction on heterogeneous computing systems. It provides a data transport layer and the capability to coordinate multiple data processing components. ALFA is a flexible, elastic system which balances reliability and ease of development with performance by using multi-processing in addition to multi-threading. With multi-processing, each process assumes limited communication and reliance on other processes. Such applications are much easier to scale horizontally (i.e: start new instances on the same node or a newly added node) to meet computing and throughput demands than applications that exclusively rely on multiple threads which can only scale vertically (to scale we need to create more threads to the already existing threads in the same process). Moreover, such a system can be extended with different hardware (accelerators) and possibly with different or new languages, without rewriting the whole system.

2. Data transport layer

The data transport layer is the part of the software which ensures the reliable arrival of messages and provides error checking mechanisms and data flow controls. The data transport layer in ALFA provides a number of components that can be connected to each other in order to construct a processing topology. They all share a common base class called *device*. Devices are grouped in three categories:

- Source: Devices without inputs are categorised as sources. A sampler is used to feed the pipeline (Task topology) with data from files.
- Message-based Processor: Devices that operate on messages without interpreting their content.
- Content-based Processor: This is the place where the message content is accessed and the user algorithms process the data.

2.1. FairMQ: Base for data transport layer in ALFA

FairMQ [3] is the module of the ALFA framework which allow the user to run reconstruction and analysis tasks based on processing components interacting via messages. This can be applied on a continuous stream of data (online), as well as on separate data samples (offline). It provides a set of components to build processing pipelines, called *Topologies*, in which the user can integrate custom code in form of tasks, that are to be executed on the streamed data. The system provides a number of ready to use components for generating, processing, routing and forwarding (to different systems) the data stream and also provides the means for the users to create new components with custom functionality. The system can transport data between processes and/or nodes via the network, inter-process and/or inter-thread communication. The transport functionality is accessible via an abstract transport interface (currently two implementations are tested - ZeroMQ [5] and nanomsg [6]). The interface can be implemented by other means to provide support for future emerging technologies.

FairMQ provides functionality for creating and configuring the communication channels (FairMQSockets) and for transmitting data over them via messages (FairMQMessages). The transport interface implementations can create messages of any size and guarantee that either these are delivered in full or not at all, so that the user does not have to take care of partitioning the data. It is also possible to create multi-part messages, composed of any number of logically independent parts. This allows the developer to add and remove custom data to an already existing message, without modifying content of the other parts. Leaving the original message untouched can avoid expensive de-/serialization process. From the point of view of the application, the message parts are sent and received separately, but on the transport level they are transferred as a single entity, which keeps the guarantees of delivering either all or nothing. The framework provides extensive support for handling the data in a zero-copy fashion. The zero-copy handling of the data by different components happens without actual copying of the contents between different memory locations, but rather by accessing the same data via a pointer or a reference. The control over a FairMQMessage is taken over by the framework after the sending operation is executed. After the framework has finished the sending (asynchronously), it will take care of the deallocation. Should the data be sent more than once, a $COPY$ method is provided by the framework. This method does not actually copy the content, but rather creates an additional reference, preventing it from deallocation until all "copies" are no longer needed.

3. Payload protocol

ALFA does not dictate any application protocols. Potentially, any content-based processor or any source can change the application protocol. Therefore, only a generic message class is provided that works with any arbitrary and contiguous chunk of memory. A pointer must

be passed to the memory buffer, the size in bytes and, if required, a function pointer to the destructor, which will be called once the message object is discarded.

The framework supports different serialisation standards that allow for data exchange between different hardware and software languages. Moreover, some of these include a built-in schema evolution which naturally simplifies the development of the software and guarantees the backward compatibility of the payloads. The following are supported by the framework:

- Boost serialization
  This method depends only on ANSI C++ facilities[7]. Moreover, it exploits features of C++ such as RTTI (Run-Time Type Information), templates or multiple inheritance. It also provides independent versioning for each class definition. This means that when a class definition changes, older files can still be imported to the new version of the class. Another useful feature is the save and restore of deep pointers.
- Protocol buffers
  Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data [8] . The structure of the data is defined once and used to generate code to read and write data easily to and from a variety of data streams, using a variety of languages: Java, C++ or Python.
- ROOT
  The ROOT Streamer can decompose ROOT objects into data members and write them to a buffer [9]. This buffer can be written to a socket for sending over the network or to a file.
- User defined
  In case it is decided not to use any of the above methods, binary structures or arrays can still be written or sent to a buffer. Although this method does not include any overhead for size of the data, issues can occur and will need to be managed. These include: schema evolution, different hardware, different languages.

4. The Dynamic Deployment System

The Dynamic Deployment System (DDS) is an independent set of utilities and interfaces, providing a dynamic distribution of different user processes for any given topology on any Resource Management System (RMS). The DDS uses a plug-in system in order to deploy different job submission front-ends. The first and the main plug-in of the system is a Secure Shell (SSH) that can be used to dynamically transform a set of machines into user worker nodes. The DDS functions are the following:

- Deploy a task or set of tasks
- Use any RMS (Slurm, Grid Engine, ...etc)
- Execute nodes securely (watchdog)
- Support different topologies and task dependencies
- Support a central log engine

During 2014, the core modules of the DDS were developed and the first stable prototype was released. This has been tested on the ALICE HLT development cluster using 40 computing nodes with 32 processes per node. The SSH plugin for DDS has been used to successfully distribute and manage 1281 ALICE O^2 user tasks (640 First Level Processor (FLPs) and 640 Event Processing units (EPN)). The FLP processes here are emulating the FLP nodes which will collect the data wheres the EPN emulates the second step of data processing: assigning each cluster to a track ([10])

The DDS was able to propagate the allocated ports for each process to the dependent processes and set the required topology for the test. Throughout the test on this cluster, one DDS commander server propagated more than 1.5 million properties in less than 5 seconds.

A test with more nodes and cores is ongoing at the test cluster of the new GSI cluster (Kronos). A rack with 50 dual-Xeon machines (Intel Xeon E5-2660 with 10 physical cores and 20 threads each) is being used for the test. On this cluster there are more than 10000 ALFA devices (FLPs, EPNs, and one Sampler). During this second test, DDS has propagated about 77 millions key-value properties. The start-up time of the whole deployment by the DDS was 207 seconds for the propagation of all required properties and devices, the bind/connect and enter into RUN state. This test was an important milestone for the DDS, demonstrating a scalability adequate for the O^2 system. The target is to scale up to 100 k devices and reduce the start-up time for the full deployment down to 10-50 seconds (depending on the number of properties the system needs to propagate).

5. Performance measurement

Two different systems were used for the performance measurement of data transport layer in ALFA. The performance tools delivered by ZeroMQ were also used to investigate any penalties introduced by FairMQ package. The goal of these measurement is to test the usability of the framework on different and existing system, so no effort was made to optimise or tune the network on the existing systems.

5.1. Ethernet-based prototype

This system consists of 8 dual-Xeon machines, 4 connected with 40 Gb Ethernet while the other 4 are connected with 10 Gb Ethernet. The throughput was measured as function of message size (see Fig.1). For the ALICE RUN3 a message part size of 10 MB is expected, for this size, a rate of about 37.6 Gbs was achieved using 4 core CPUs for sending data between two of the machines (point to point). This test demonstrates that the overhead introduced by the FairMQ and ZeroMQ is marginal with a bandwidth equivalent to 94% of the theoretical one and that the technology scales well above the performance required by the FLPs on their output network link. More details are in Reference [11].

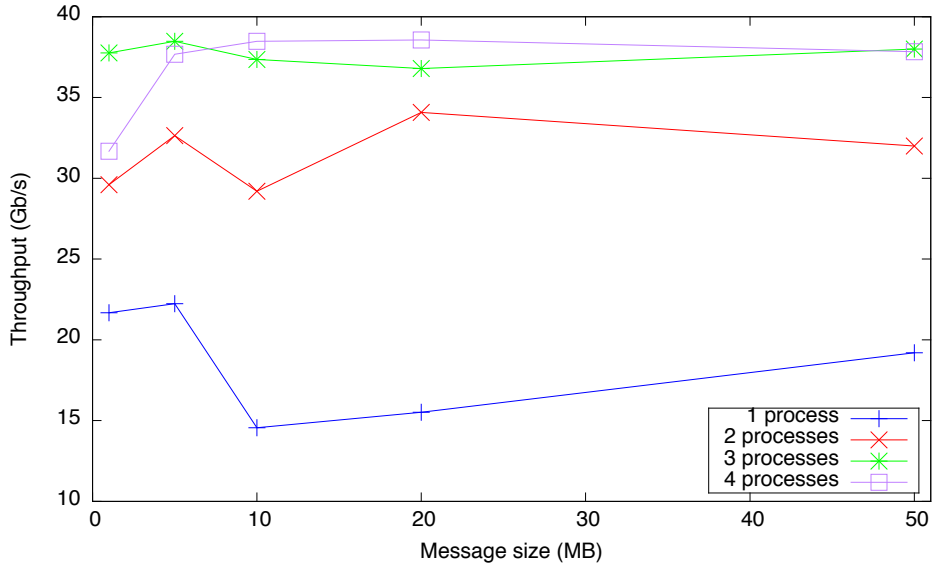

Figure 1. Throughput between 2 machines connected via 40 Gb/s Ethernet.

5.2. InfiniBand-based prototype

The second system is composed of a 40 Gb IB using 4 dual-Xeon machines (Intel Xeon E5520 with 4 physical cores and 8 threads each) all running the same software but with the IPoIB protocol. Three processes were used to send data from one machine and 4 processes on each of the other machines received data. A message size of 10 MB was used. An average rate of 2.5 GBs was reached without any optimisation of the kernel parameters. This test confirms that the marginal overhead introduced by the FairMQ and ZeroMQ software with a measured performance equivalent to the one measured with benchmarking programs (see [10]). The test also demonstrates the portability of the FairMQ software to different network technologies (Ethernet and IB) which provides the independence about the underlying network technology.

6. Conclusion

The ALFA framework meet the high throughput requirements for the upcoming upgrade of the ALICE experiment and FAIR. The scalability of the system has been demonstrated on a computing cluster equipped with QDR and FDQ InfiniBand Host Channel Adapters (DDS). The system maintained maximum throughput when scaled to a large number of computing nodes and multiple processes per node. Furthermore, a traffic shaping has been implemented on the application level on the FLP nodes to reduce network contention when several FLP nodes transfer data to the same EPN node [11]. The ALFA framework offers a concept for the communication with asynchronous messaging system and a tool set that can be used for data processing scenarios with high throughput requirements, such as the data transport for the upgrade of the ALICE detector at CERN and the upcoming FAIR accelerator facility in Darmstadt, Germany .

References

[1] M. Al-Turany et al. 2012, The FairRoot framework *J. Phys.: Conf. Ser. 396 022001*
[2] FairRoot: http://fairroot.gsi.de.
[3] M. Al-Turany et al. 2015, Extending the FairRoot framework to allow for simulation and reconstruction of free streaming data *J. Phys.: Conf. Ser. 513 022001*
[4] ZeroMQ Code connected: http://www.zeromq.org/
[5] P. Hintjens, 2013 *ZeroMQ Messaging for Many Applications* (O'Reilly Media)
[6] nanomsg: http://nanomsg.org/
[7] BOOST C++ Libraries: http://www.boost.org.
[8] Google protocol buffers: https://developers.google.com/protocol-buffers/
[9] R. Brun and F. Rademakers. 1997, Root - an object oriented data analysis framework *Nuclear Instruments and Methods in Physics Research A , 389, 81-86*
[10] P. Buncic et al. The ALICE Collaboration. Technical Design Report for the Upgrade of the Online-Offline Computing System. ALICE-TDR-019, 2015. CERN-LHCC-2015-006.
[11] A.Rybalchenko , 2015 *Efficient Time Frame Building for Online Data Reconstruction with the ALICE Detector System at CERN* , Master Thesis. Darmstadt University of Applied Sciences Darmstadt, Germany