EPJ Web of Conferences **245**, 05021 (2020) https://doi.org/10.1051/epjconf/202024505021
CHEP 2019

ALFA: A framework for building distributed applications

Mohammad Al-Turany[1,*], *Alexey* Rybalchenko[1], *Dennis* Klein[1], *Matthias* Kretz[1], *Dmytro* Kresan[1], *Radoslaw* Karabowicz[1], *Andrey* Lebedev[1], *Anar* Manafov[1], *Thorsten* Kollegger[1], and *Florian* Uhlig[1]

[1]GSI - Helmholtzzentrum fur Schwerionenforschung GmbH (DE)

Abstract. The ALFA framework is a joint development between ALICE Online-Offline and FairRoot teams. ALFA has a distributed architecture, i.e. a collection of highly maintainable, testable, loosely coupled, independently deployable processes. ALFA allows the developer to focus on building single-function modules with well-defined interfaces and operations. The communication between the independent processes is handled by FairMQ transport layer. FairMQ offers multiple implementations of its abstract data transport interface, it integrates some popular data transport technologies like ZeroMQ and nanomsg. Furthermore it also provides shared memory and RDMA transport (based on libfabric) for high throughput, low latency applications. Moreover, FairMQ allows the single process to use multiple and different transports at the same time. FairMQ based processes can be controlled and orchestrated via different systems by implementing the corresponding plugin. However, ALFA delivers also the Dynamic Deployment System (DDS) as an independent set of utilities and interfaces, providing a dynamic distribution of different user processes on any Resource Management System (RMS) or a laptop. ALFA is already being tested and used by different experiments in different stages of data processing as it offers an easy integration of heterogeneous hardware and software.

1 Introduction

ALFA [1] is a modern C++ software framework for simulation, reconstruction and analysis of particle physics experiments. ALFA extends FairRoot [2] by providing building blocks for highly parallelised and data flow driven processing pipelines required by the next generation of experiments, such as the upgraded ALICE detector and the FAIR experiments. ALFA is a flexible, elastic system which balances reliability and ease of development with performance by using message based multi-processing. In this approach, each process assumes limited communication and reliance on other processes. Each process can be optimised independently of the other processes in the system (e.g using multi-threading or any other technique which fits at best for the particular implimentation). Applications based on such concept can scale to meet the computing and throughput demands by simply creating new instances of processing components. Moreover, the system can be extended with different hardware (accelerators) and possibly with different or new languages, without rewriting the whole system. ALFA provides a data transport layer and the tools to coordinate multiple data processing

*e-mail: m.al-turany@gsi.de

components. The algorithms of these components are normally optimised for speed and ALFA is designed to manage the high throughput of data between these algorithms.

2 Transport layer

The FairMQ library [3] is the core of data transport in ALFA. 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 streams 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[5], shared memory [6] and RDMA based transport [7] 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.

3 Process deployment and control

3.1 The Dynamic Deployment System (DDS)

DDS is a tool-set that automates and significantly simplifies the deployment of user-defined processes and their dependencies on any resource management system (RMS) using a given pre-defined process topology (Graph). It implements a single responsibility principle command line tool-set and API [8]. The system treats user tasks as a black box, e.g. it can be an executable or a script. It also provides a watchdogging and a rule-based execution of tasks. DDS implements a plug-in system to abstract the execution of the topology from the RMS. Additionally it ships an SSH and a localhost plug-in which can be used when no RMS is available. DDS does not require pre-installation or pre-configuration on the worker nodes. It deploys private facilities on demand with isolated sandboxes. The system provides a key-value property propagation engine. That engine can be used to configure tasks at runtime. DDS also provides a lightweight API for tasks to exchange messages, so-called, custom commands (see ref [8] for the details). DDS is being used for work with different user requirements. For example, online cases in which a single DDS instance has to manage many (~100k) processes. In offline scenarios each of the many DDS instances manage relatively small (~100) number of processes. Each DDS session can be a separate job which makes it a perfect tool for computing farms with RMS (Slurm, PBS etc.). Of course one can also run DDS locally on a laptop for development or debugging purposes.

3.2 PMI

The Process Management Interface (PMI), originally developed and distributed as part of MPICH[9], has historically been used as a means of exchanging wireup information needed for interprocess communication and deployment of processes. The modern PMI-2 demon shows significantly better scaling properties compared to its PMI-1 predecessor. However, meeting the significant orchestration challenges presented by exascale systems requires that the process to the system management stack (SMS) interface evolves to permit a tighter integration between the different components of the parallel application for existing and future SMS solutions. The Process Management Interface for Exascale (PMIx) provides:

1. Distributed key/value store for data exchange
2. Asynchronous events for coordination
3. Enable interactions with the resource manager

The PMIx plugin in FairMQ enables launching a FairMQ topology with any PMIx capable launcher, e.g. the Open Run-Time Environment (ORTE) of OpenMPI or the Slurm workload manager. This plugin is not (yet) very mature and serves as a proof of concept at the moment.

3.3 Online device controller (ODC)

The Online Device Control software [10] controls/communicates with a graph (topology) of FairMQ devices using either DDS or PMIx (under development). The FairMQ core library provides two local controllers - static (a fixed sequence of state transitions) and interactive (a read-eval-print-loop which reads keyboard commands from standard input). A local controller only knows how to steer a single FairMQ device. However, the process controller has the knowledge about the full topology of connected FairMQ devices in a cluster. Its responsibility is to facilitate the lifecycle of a distributed FairMQ-based application (executing a topology), such as:

- allocating/releasing computing resources from a resource management system,
- launching/setting up the run-time environment and the FairMQ devices,
- driving the device state machines in lock-step across the full topology,
- pushing the device configuration,
- monitoring (e.g: memory and cpu usage, data rates, ..etc) operation and handling/reporting error cases (restart crashing processes, report blocking processes, etc).

The FairMQ Software Development Kit (FairMQ-SDK) contains (as of today) a set of C++ APIs that provide essential functionality to implement a process controller.

The schematic of a DDS based control is shown in Fig. 1. The experiment control system example (ECS) implements a control client (gRPC based) to communicate with the online device controller [10].

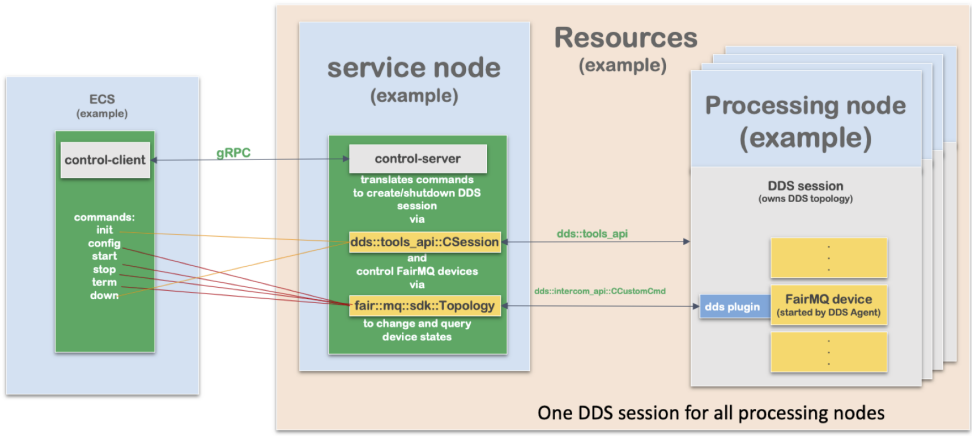

Figure 1. DDS based processing control system

Table 1. Experiment control command mapping to FairMQ states

Command	Controller actions on the Farm
INIT	Start DDS session + Submit agents using ssh plugin + Activate topology
CONFIG	InitDevice + CompleteInit + Bind + Connect + InitTask
START	Run
STOP	Stop
TERM	ResetTask + ResetDevice + End
DOWN	Shutdown DDS session

When launching a FairMQ topology via DDS the DDS plugin enables FairMQ devices to interact with DDS custom command and property subsystems. The "Property subsystem" allows the exchange of channel connection data whereas the "Custom command subsystem" allows the device to receive user commands (e.g: interactively with command line via the *fairmq-dds-command-ui*) . This system is now being tested within the integration activity in ALICE experiment and will be used for ALICE RUN3 next year.

A prototype for this controller was tested using a test cluster of 27 nodes each of 48 logical cores (total of 1296 cores). In this test DDS-ssh plugin was used (no external resource management system [8]). DDS version 3.1, Boost 1.71.0 and the latest FairMQ where used. Table 1 shows the experiment control command mapping to the processing farm actions initiated. The timing results of this test are shown in Figure 2.

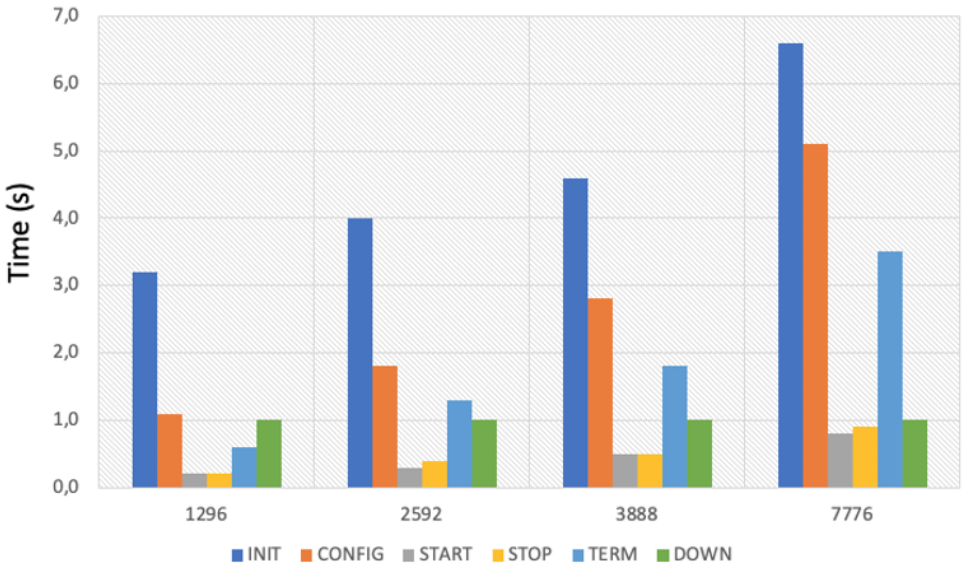

Figure 2. Time in seconds to execute the actions with the number of processes.

4 Conclusion

The ALFA framework meets the high throughput requirements for FAIR as well as for the upcoming upgrade of the ALICE experiment. The scalability of the system has been demonstrated on a computing cluster equipped with Ethernet 100 GB, EDR and HDR InfiniBand

Host Channel Adapters. The system maintained maximum throughput when scaled to a large number of computing nodes and multiple processes per node. The newly developed tools within the framework helped building a modular control system for the ALICE experiment for the next run (RUN3).

References

[1] M. Al-Turany et al., Journal of Physics: Conference Series, Volume 664, 072001 (2015)
[2] M. Al-Turany et al., Journal of Physics: Conference Series, Volume 396, 022001 (2012)
[3] FairMQ, https://github.com/FairRootGroup/FairMQ, visited 13.3.2020
[4] ZeroMQ, http://zeromq.org/, visited 13.03.2020
[5] nanomsg, https://nanomsg.org, visited 13.03.2020
[6] A. Rybalchenko et al, EPJ Web Conf., Volume 214, 05029 (2019)
[7] D. Klein. et al, EPJ Web Conf., Volume 214, 05022 (2019)
[8] A. Lebedev, et. al., EPJ Web of Conferences 214, 01011 (2019)
[9] Mathematics and Argonne National Laboratory Computer Science Division. 2006. MPICH-2, implementation of MPI 2 standard. http://www-unix.mcs.anl.gov/mpi/mpich2/. (2006). [Online; accessed 26-Apr-2017].
[10] ODC, https://github.com/FairRootGroup/ODC, visited 13.3.2020