EPJ Web of Conferences **295**, 07031 (2024) https://doi.org/10.1051/epjconf/202429507031
CHEP 2023

Modular toolsets for integrating HPC clusters in experiment control systems

Anar Manafov[1,*], *Alexey* Rybalchenko[1,**], *Dennis* Klein[1,***], and *Mohammad* Al-Turany[1,****]

[1]GSI Helmholtz Centre for Heavy Ion Research

Abstract. New particle/nuclear physics experiments require a massive amount of computing power that is only achieved by using high performance clusters directly connected to the data acquisition systems and integrated into the online systems of the experiments. However, integrating an HPC cluster into the online system of an experiment means: Managing and synchronizing thousands of processes that handle the huge throughput.
In this work, modular components that can be used to build and integrate such a HPC cluster in the experiment control systems (ECS) will be introduced.
The Online Device Control library (ODC) [1] in combination with the Dynamic Deployment System (DDS) [2, 3] and FairMQ [4] message queuing library offers a sustainable solution for integrating HPC cluster controls into an ECS.
DDS as part of the ALFA framework [5] is a toolset that automates and significantly simplifies a dynamic deployment of user-defined processes and their dependencies on any resource management system (RMS) using a given process graph (topology).
ODC, in this architecture, is the tool to control and communicate with a topology of FairMQ processes using DDS. ODC is designed to act as a broker between a high level experiment control system and a low level task management system e.g.: DDS.
In this work the architecture of both DDS and ODC will be discussed, as well as the design decisions taken based on the experience gained of using these tools in production by the ALICE experiment at CERN to deploy and control thousands of processes (tasks) on the Event Processing Nodes cluster (EPN) during Run3 as a part of the ALICE O2 software ecosystem [6].

1 Introduction

Modern particle and nuclear physics experiments stand at the intersection of unprecedented advancements and exponential demands. With an increasing need for computational prowess, research now leans heavily on integrating high performance clusters (HPC) directly into the data acquisition systems, necessitating their integration with the online systems of these experiments. These integrations, while crucial, come with their own set of challenges - foremost

*e-mail: A.Manafov@gsi.de
**e-mail: A.Rybalchenko@gsi.de
***e-mail: D.Klein@gsi.de
****e-mail: M.Al-turany@gsi.de

among them is the daunting task of managing and synchronizing thousands of processes that must seamlessly handle an enormous data throughput. This paper delves into a modular solution that manages this integration.

The ALFA framework meets these demands by providing the Online Device Control (ODC), working in tandem with the Dynamic Deployment System (DDS). ODC provides DDS session management, as well as control and configuration facilities for FairMQ processes (also called *devices*), leveraging DDS functionality. DDS is instrumental in automating the deployment of FairMQ devices on any given resource management system (RMS) using a detailed process graph or topology. ODC architecture is designed to serve as an intermediary, bridging the high-level experiment control system with the low-level FairMQ devices management.

This paper will describe the architectures of both DDS and ODC, diving into their structural and functional nuances. Moreover, it will reflect upon critical design choices, informed by experiences from their deployment in real-world settings. A testament to their efficacy is their deployment by the ALICE experiment at CERN. Here, they have been pivotal in orchestrating thousands of processes on the Event Processing Nodes cluster (EPN) during Run3, seamlessly integrating as a part of the expansive ALICE O2 software ecosystem. Through this discourse, we aim to provide a comprehensive understanding of these tools and their contribution to the realm of particle and nuclear physics experiments.

2 Core Concepts of ODC

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, while also allowing to see the state of every device. A fixed set of commands is available to ECS for launching deployments of FairMQ topologies, control device states and configure device properties. Figure 1 gives a high level overview of ODC and how it is positioned in an experiment setup, taking the broker role between ECS and FairMQ devices.

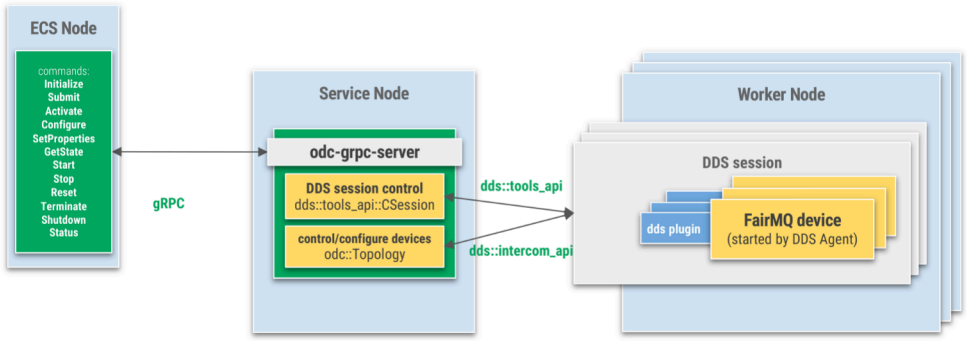

Figure 1. High level overview of ODC

Further, we describe the principles and features underpinning the ODC project.

- Parallel Session Management.
  ODC is able to handle multiple parallel sessions. This allows researchers to deploy and manage different FairMQ topologies concurrently on large computing clusters, an imperative feature in large-scale experiments where multitasking is a necessity.

- Asynchronous Control.
  ODC tools offer asynchronous access & control to parallel sessions. This asynchronous model ensures that operations in one session do not block or impede operations in another, facilitating fluid task execution.
- Command Restrictions.
  In a bid to maintain control and streamline operations, ODC restricts the execution to only one command per session at any given time. The only exceptions to this rule are the status and state commands, which can be invoked asynchronously for swift feedback.
- Real-Time Feedback with Asynchronous Status & State Commands.
  Understanding the real-time status of a session is crucial. ODC's asynchronous status & state commands are designed to provide fast feedback, especially beneficial during ongoing commands or in instances where other commands might be slow or hang.
- Session Reattachment.
  ODC is capable of reattaching to sessions that are already running prior to ODC launch. This allows more flexible command use and even combination of gRPC and CLI controllers. Additionally it enables session recovery in case of server crash or restart. This ensures continuity and reduces downtimes.
- Flexible Front-End Access.
  ODC's flexibility is further manifested in its provision of two front-end interfaces. First is the gRPC Interface. It is suitable for programmatic access, such as the integration with an Experiment Control System. Second is the Command-Line Interface For testing, debugging or running in environments where gRPC is not available. Both interfaces offer equal functionality with respect to the sessions they control.
- Task Failure Management.
  ODC introduces resilience in execution by supporting continuation even in the wake of failing tasks. Users can label certain tasks as expendable, ensuring their failure doesn't hinder the overall topology. Alternatively, by specifying a minimum number of functioning group members, the session remains unaffected unless failures exceed the defined threshold.
- Resource Management.
  ODC relies on DDS to handle resource management. The integration with Slurm [5] is seamlessly achieved by translating user-specified requirements into configurations comprehensible by DDS and its Slurm plugin. This facilitates efficient resource allocation in environments where Slurm is the resource manager. Furthermore, when running with Slurm, a more granular core-scheduling is available, allowing better use of the available hardware resources. Besides the Slurm integration, simple localhost and SSH resource managers can be used for development and test deployments.
- Dynamic Resource Allocation.
  To alleviate manual intervention and reduce human error, ODC provides dynamic calculation of resource requirements derived directly from the DDS topology. This ensures optimal resource utilization based on the real demands of the tasks.

ODC, with its array of features, plays an important role in the orchestration and management of device topologies. Its resilient nature provides a robust process control for complex particle physics experiment setups.

3 Core Concepts of DDS

DDS (the Dynamic Deployment System, Figure 2) helps to deploy and watchdog user tasks defined by topologies in the scale of hundreds of thousands of processes.

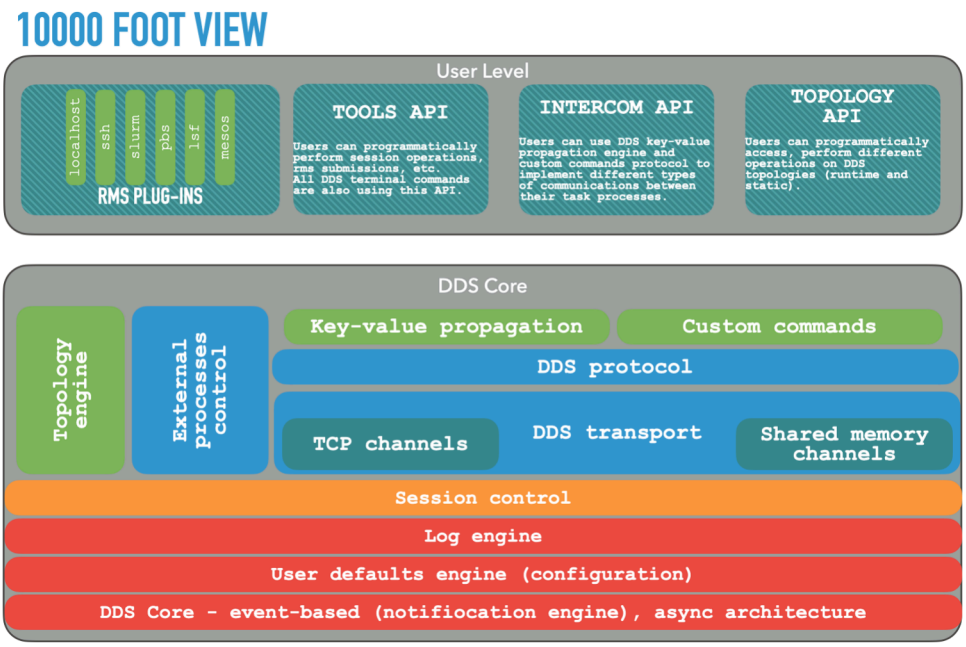

Figure 2. DDS: 10000 foot view

The deployment process is transparent and doesn't depend on the underlying RMS (see Figure 3). A user can deploy a small topology on a local computer, for instance when testing. Then take the same set of tasks in the same topology but multiplied by thousands to deploy on a computing cluster managed by Slurm, or any other RMS.

Below is a list of DDS design highlights.

- A single responsibility principle command line tool-set and API.
  DDS implements a rich C++ API, which can be used to programmatically control DDS, work with static and runtime topologies and perform different types of communications with tasks on the given topology.
  There is also a set of command line tools (CLI), all of which are implemented using DDS API. The CLI is used to work with DDS from the command line or scripts.

- Resource usage optimization.
  The internal asynchronous, event-based core engine helps to keep low resource requirements, provide a fast message throughput and responsiveness.
  The core engine of DDS implements a low level transport protocol, which is unified to use TCP and shared memory (SHM) channels. DDS automatically uses SHM channels whenever possible/suitable to gain communication performance.

- DDS treats user tasks as black boxes.
  A user task can be an executable or a script.

- Watchdogging.
  All user tasks started by DDS are constantly monitored. Once a task exists, DDS will create an event with the exit status, signal, etc. This event will be delivered to all API subscribers.

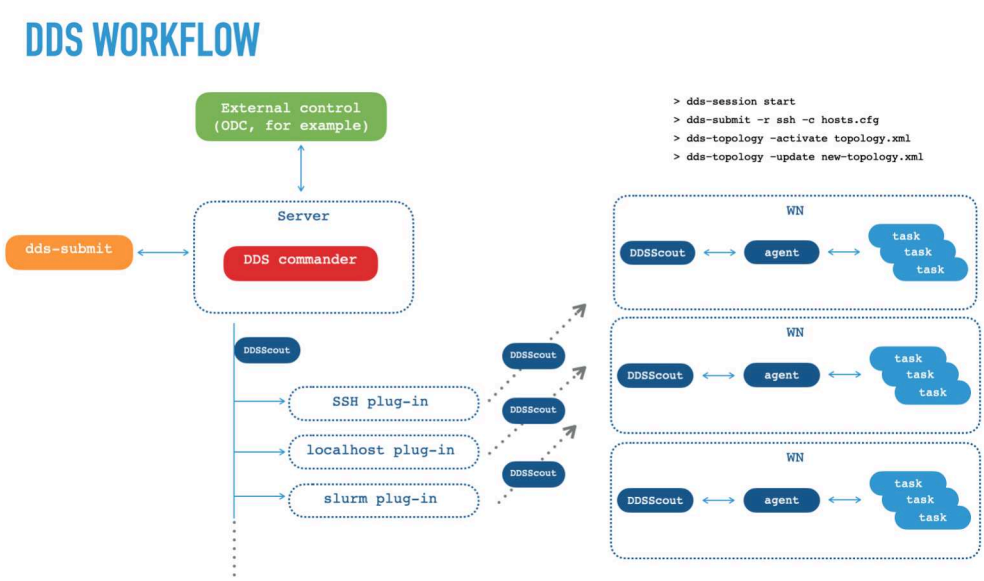

Figure 3. DDS Workflow

On DDS shutdown, all still running tasks will receive a graceful terminate signal followed by an unconditional kill for unresponsive tasks.

- A topology-based execution of tasks.
  Users can define their tasks, task properties, task/resource requirements via a simple DDS topology schema. DDS can then deploy the required set of tasks and their dependencies to given hardware resources. During runtime execution users can update topologies by removing or adding tasks or their properties.

- A plug-in system to abstract from RMS.
  DDS can use different job submission front-ends for tasks deployment. In order to cover various RMS, a plug-in architecture has been implemented, which gives external developers a possibility to create RMS plug-ins. The architecture guarantees isolated and safe execution of plug-ins.
  The following RMS plug-ins are currently implemented in DDS: localhost, SSH, Slurm, PBS [8], LSF [9], and Apache Mesos [10].

- It doesn't require pre-installation and pre-configuration on worker nodes. DDS deploys a lightweight agent scripts with binary payload attachments. The script checks the environment and starts agents, which then manage user tasks.
  Users can run multiple agents per host with multiple tasks per agent.
  The current, 3rd gen. DDS deployment architecture, can use only one agent for hundreds of user tasks.

- Private facilities on demand with isolated sandboxes.
  DDS works in sessions. Each user session is isolated. Users can have multiple sessions at time.

- A key-value propagation and custom commands.
  This feature allows user's tasks to exchange and synchronize the configuration (key-value)

dynamically at runtime. For example, startup synchronization of the multiprocessing reconstruction requires tasks to exchange theirs connection strings so that they are able to connect to each other.

4 ALICE Experiment Use Case

The upgraded ALICE experiment at CERN uses FairMQ in several parts of their system. The deployment of FairMQ devices on Event Processing Nodes of ALICE involves around 130,000 devices per session spread across several hundreds of nodes. Each node handles around 400 devices, which communicate through shared memory.
DDS is used to deploy and manage runtime topologies on a Slurm cluster. ODC manages the states of the underlying FairMQ devices.
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.

5 Conclusion

The evolving landscape of particle and nuclear physics demands sophisticated tools that can manage vast computational loads. The scale of these tasks — often involving synchronization and management of thousands of processes — requires robust solutions, which the combined capabilities of the ODC, DDS and FairMQ deliver.

ODC offers a bridge between high-level experiment control systems and the more granular process controls inherent in FairMQ, positioning itself as the mediator in this complex orchestration.

DDS allows for scalable and efficient deployment of device topologies, abstracting from the underlying Resource Management System. The modular nature of DDS, coupled with its plug-in architecture, provides flexibility across various deployment environments. Its capacity to manage user tasks as black boxes ensures that DDS remains versatile and applicable across different setups and needs.

Further validation of these tools' robustness and applicability is evident in their successful deployment in the ALICE experiment at CERN. Navigating the intricacies of deploying around 130,000 devices per session across hundreds of nodes is a testament to the tools' capabilities and underscores their significance in contemporary physics experiments.

References

[1] FairRootGroup, "ODC git repository", Last accessed 14th of November 2022: "https://github.com/FairRootGroup/ODC"
[2] FairRootGroup, "DDS home site", Last accessed 14th of November 2022: "http://dds.gsi.de"
[3] FairRootGroup, "DDS source code repository", Last accessed 14th of November 2022: "https://github.com/FairRootGroup/DDS"
[4] FairMQ, "FairMQ git repository", Last accessed 14th of November 2022: "https://github.com/FairRootGroup/FairMQ"
[5] "https://indico.gsi.de/event/2715/contributions/11355/attachments/8580/10508/ALFA_Fias.pdf"
[6] ALICE Technical Design Report (2nd of June 2015), Last accessed 14th of November: "https://cds.cern.ch/record/2011297/files/ALICE-TDR-019.pdf"

[7] Slurm - "https://slurm.schedmd.com/"
[8] PBS. Portable Batch System - "http://www.pbspro.org/"
[9] LSF. Platform Load Sharing Facility - "https://en.wikipedia.org/wiki/IBM_Spectrum_LSF"
[10] Apache Mesos - "http://mesos.apache.org/"