https://doi.org/10.1051/epjconf/201921401033

# Towards the ALICE Online-Offline ($O^2$) control system

*Teo* Mrnjavac[1,*] and *Vasco* Chibante Barroso[1,**]

[1]CERN, Geneva, Switzerland

**Abstract.** The ALICE Experiment at CERN LHC (Large Hadron Collider) is under preparation for a major upgrade that is scheduled to be deployed during Long Shutdown 2 in 2019-2020 and that includes new computing systems, called $O^2$ (Online-Offline). To ensure the efficient operation of the upgraded experiment along with its newly designed computing system, a reliable, high performance and automated control system will be developed with the goal of managing the lifetime of all the $O^2$ processes, and of handling the various phases of the data taking activity by interacting with the detectors, the trigger system and the LHC. The ALICE $O^2$ control system will be a distributed system based on state of the art cluster management and microservices which have recently emerged in the distributed computing ecosystem. Such technologies weren't available during the design and development of the original LHC computing systems, and their use will allow the ALICE collaboration to benefit from a vibrant and innovating open source community. This paper illustrates the $O^2$ control system architecture. It evaluates several solutions that were considered during an initial prototyping phase and provides a rationale for the choices made. It also provides an in-depth overview of the components, features and design elements of the actual system.

## 1 Introduction

### 1.1 The $O^2$ computing system

The ALICE Experiment [1] is on track for a major upgrade [2], scheduled to be deployed during LHC's Long Shutdown 2 (2019-2020), in time for LHC Run 3. In order to keep up with the increased data rate coming out of the detectors, a new computing system called $O^2$ [3] will be deployed.

The $O^2$ computing system will consist of 100,000s of processes deployed over roughly 2000 nodes performing readout, processing and storage. The system will read out 27 Tb/s of raw data and record 800 Gb/s of reconstructed data.

The $O^2$ computing system will run on two main typologies of computing nodes: FLPs (First Level Processors) and EPNs (Event Processing Nodes). Each FLP will be fitted with CRU (Common Readout Unit) [4] or C-RORC (Common Readout Receiver Card) [5] hardware, depending on the detector. These readout cards are capable of two way communication with detector front end electronics.

*e-mail: teo.m@cern.ch
**e-mail: vmcb@cern.ch

The O$^2$ computing system will be capable of two kinds of data-driven workflows: synchronous operation, intended to be *synchronous* with detector readout, and asynchronous operation, which will take place at any time regardless of detector/beam conditions. Each node is expected to run dozens of processes of different kinds, including long running services, WLCG-like (Worldwide LHC Computing Grid) environments for asynchronous processing, and data-driven process workflows. Synchronous workflows operate on data coming from detector data links, so they must run in the O$^2$ facility at LHC Point 2. Asynchronous workflows do not have this constraint, so they can run at any time on WLCG nodes, or on O$^2$ facility resources when they are not needed for synchronous operation.

The O$^2$ project has chosen FairMQ [6] as the common message passing and data transport framework for its data-driven processes. It has been developed in the context of FairRoot [7, 8], a simulation, reconstruction and analysis framework for particle physics experiments. FairMQ provides the basic building blocks to implement complex data processing workflows, including a message queue, a configuration mechanism, a state machine, and a plugin system.

### 1.2 Target operational improvements in a control system solution for O$^2$

A new control system will be developed with the goal of managing the lifetime of all the O$^2$ processes, and of handling the various phases of the data taking activity by interfacing with the detectors, the trigger system and the LHC.

The goals and requirements of the ALICE O$^2$ control system (O$^2$ control) are derived from experience in running the current computing system, and they are motivated by a desire for greater reliability, performance, maintainability, and operational flexibility.

Target operational improvements include

1. no workflow redeployment when including or excluding a detector from data taking,
2. recovery from process and server crashes,
3. process reconfiguration without mandatory restart,
4. and EPN scaling during data taking (e.g. as luminosity decreases towards the end of a LHC fill).

The O$^2$ project includes a redesign of user interfaces, in favor of next-generation web-based GUIs with SSO (single sign-on) and a revamped design. O$^2$ control will come with a number of command line and graphical user interfaces, including shifter oriented GUIs for the O$^2$ equivalent of the current ECS (Experiment Control System) [9].

Finally, the O$^2$ project is an opportunity to take advantage of modern developments in computing, thus O$^2$ control will be built with the best practices of a microservices distributed application paradigm, and harnessing the features of modern cluster resource management systems.

## 2 Requirements of a control solution for O$^2$

The primary duty of a control system for O$^2$ is to launch, configure and control a set of data-driven processes and workflows inside a computer cluster. Specifically, the control system solution is in charge of

1. managing the lifetime of thousands of processes in the O$^2$ facility,
2. minimizing the waste of beam time by reusing processes and avoiding time-consuming process restart operations,
3. interfacing with the LHC, the trigger system, the DCS (Detector Control System) [10] and other systems, ideally through common APIs,
4. and ensuring fair and efficient resource multiplexing between synchronous and asynchronous tasks.

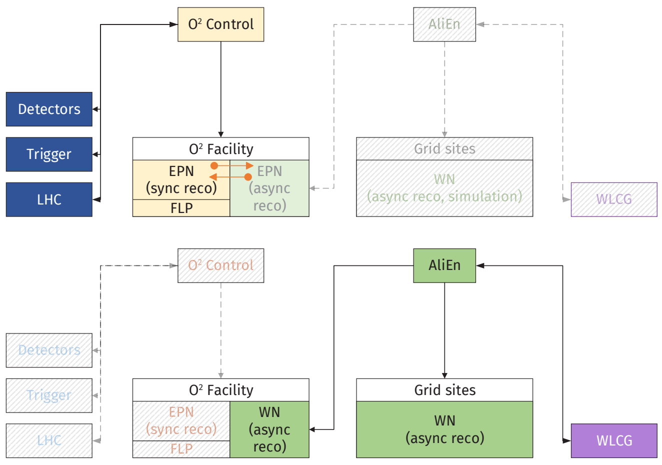

**Figure 1.** $O^2$ control will be able to designate the resources of a compute node for synchronous or asynchronous processing. If a node is assigned to synchronous processing (top image), $O^2$ control stays in charge and keeps fine-grained control over resource allocation. When $O^2$ control assigns a node to asynchronous operation (bottom image), it will bootstrap a pilot job to set up a WLCG-like asynchronous execution environment.

### 2.1 Synchronous and asynchronous workflows

The primary task of $O^2$ control is to handle the details of synchronous workflows, as this kind of workflow is time-critical and directly affected by experiment operations. Asynchronous workflows will be executed in Grid-like environments, both on the WLCG and inside the $O^2$ facility when resources are available, on a best-effort basis. At times when the $O^2$ facility has free resources (i.e., compute resources not used for synchronous operation), $O^2$ control will have the responsibility of bootstrapping AliEn [11] pilot jobs, which set up asynchronous processing environments for tasks like asynchronous reconstruction, analysis, and simulation (see Fig. 1). $O^2$ control will be able to reclaim resources assigned to asynchronous operation if synchronous processing workflows require them.

In order to satisfy such use cases, the $O^2$ control system is a distributed system in charge of the $O^2$ facility, with full knowledge and control over its resources. It implements a reliable and distributed state machine mechanism to represent the aggregated state of the constituent $O^2$ processes of a data-driven workflow. Furthermore, it allows reconfiguration and reuse of running $O^2$ processes as often as possible to avoid process restarts, it allows simultaneous operation of multiple asynchronous and synchronous workflows, with easy reallocation of resources among workflows. Finally, it reacts promptly to inputs, handling events from the user, the LHC, the trigger system, the DCS, and the cluster itself with a high degree of autonomy.

## 3 Resource management in the $O^2$ facility

We implement $O^2$ control as a distributed application, using Apache Mesos [12, 13] as toolkit. This custom solution integrates a task scheduler component, a purpose-built distributed state machine system, a general purpose process configuration mechanism, and a control plugin and library compatible with any data-driven $O^2$ process.

### 3.1 An overview of Apache Mesos

Apache Mesos is a cluster resource management system. It greatly streamlines distributed application development by providing a unified distributed execution environment. Mesos facilitates the management of $O^2$ components, resources and tasks inside the $O^2$ facility, effectively enabling the developer to program against the datacenter (i.e., the O$^2$ facility at LHC Point 2) as if it was a single pool of resources.

Apache Mesos comes with two main components: masters and agents. In a Mesos-enabled cluster there is a Mesos agent running on every node: its purpose is to collect information on the resources available on that node, and to handle task deployment. A Mesos-enabled cluster must also have at least one Mesos master. In this context, a Mesos-aware distributed application is called a *framework*. When developing a framework, the developer must build a *scheduler* process (which subscribes to the Mesos master), as well as one or more *executors*. The Mesos master acts as an authoritative source of knowledge on cluster resources, and periodically sends resource offers to the schedulers of the frameworks running on the cluster, which can then use these resources to run tasks.

In order to run a task, a Mesos agent runs the selected executor component of the framework that accepted the resources provided by this agent, and the executor can then run a process or perform any other operation as required by the scheduler.

### 3.2 The role of Apache Mesos in the O$^2$ facility

Apache Mesos has become a household name in the industry, and it has been used in deployments of 10,000s of nodes. It is an open source project, hosted by the Apache Software Foundation. Commercial support is available.

For O$^2$ control, benefits of using Mesos include

1. the knowledge of what runs where,
2. resource management, which facilitates various deployment steps including port assignment, node selection, configuration, and others,
3. transport facilities for O$^2$-specific control messages,
4. task status tracking (e.g. an event is raised if a task dies unexpectedly),
5. and advanced features such as node attributes, resource overprovisioning, checkpointing, and others.

The drawback of having Apache Mesos as an additional component in the stack is compensated by its benefits. We also argue that implementing a computing system at the scale of O$^2$ with modern techniques would in any case involve a resource management system component or mechanism.

It is important to note that Apache Mesos is not a control system. The requirements, and thus the design of O$^2$ control include much beyond Mesos. Apache Mesos is not a service discovery system, though its role as an authoritative source of knowledge on cluster resources makes it suitable for integration with such a system. Mesos is also not a distributed state machine, and in fact it makes no assumption on the stateful nature of the tasks it helps deploy. Furthermore, while Apache Mesos can run containerized tasks, it is not a container orchestration platform. Finally, Mesos is not a plug-and-play platform-as-a-service (PaaS) system for cloud native applications (one such system is DC/OS, and it is built on top of Mesos).

## 4 Design overview of O$^2$ control

### 4.1 O$^2$ control components

Our proposed solution for the problem of O$^2$ synchronous control is under development. The current implementation of O$^2$ control can be found on GitHub [14], and it consists of

1. the O$^2$ control core (which includes the Apache Mesos-facing scheduler component),

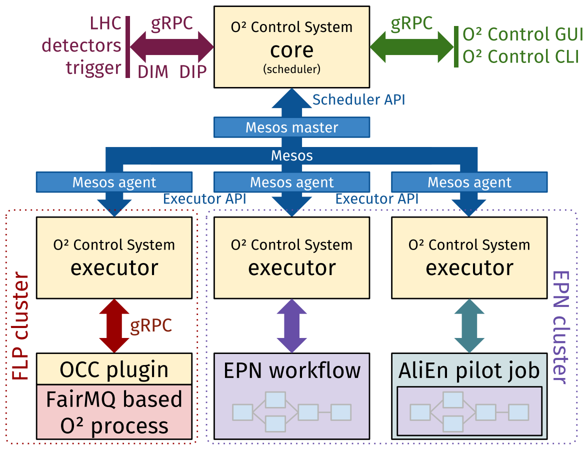

**Figure 2.** O$^2$ control architecture. All control communication between core and executor instances is piggybacked on Mesos messages. RPC-pattern interaction between the user interfaces and the O$^2$ control core, and between the executor and the controlled O$^2$ process is implemented with gRPC. The O$^2$ control and configuration plugin hides the complexities of handling gRPC connections and driving the state machine. The AliEn pilot job is not to be considered a controlled process, as O$^2$ control is only in charge of bootstrapping it.

2. the O$^2$ control executor,
3. the O$^2$ control and configuration plugin for FairMQ devices (`FairMQPlugin_OCC`),
4. the O$^2$ control and configuration library (`libOcc`),
5. the O$^2$ control and configuration command line utility (`coconut`),
6. a deployment utility for O$^2$ development and testing (`fpctl`),
7. and the web-based O$^2$ control GUI.

The O$^2$ control core accepts requests from the O$^2$ control GUI or from `coconut`. These requests are then processed, and they result in Mesos API calls, handlers for Mesos API events, or O$^2$-specific control messages (using Mesos API calls and handlers for transport).

Furthermore, O$^2$ control interfaces via a configuration wrapper library with Consul [15], a key-value store which acts as the system's configuration repository. The design also includes interfacing with information sources from the LHC, the trigger system, and the DCS.

Most components of O$^2$ control are written in Go, a statically typed general purpose programming language in the tradition of C, which is particularly suitable for distributed systems development because of its advanced synchronization and threading facilities. The O$^2$ control and configuration plugin for FairMQ devices is developed in C++14, and it works with any FairMQ-based process. A non-plugin library equivalent of the latter is also provided, for O$^2$ processes which do not support the FairMQ plugin system.

## 4.2 Inter-process communication in O$^2$ control

The common idiom of inter-process communication in O$^2$ control is gRPC [16], an open source, cross-language RPC (remote procedure call) system backed by Google. It is widely used in the microservices community. gRPC comes with a code generator which provides client and server stub code based on a common descriptor file. Once the developer describes the client-server interface in this file, the generator can output stubs for C++, Go or any other supported language, enabling seamless interaction between processes in a heterogeneous cross-language distributed system. Depending on the language, for the developer this gRPC-mediated remote interaction mimics local function calls.

In O$^2$ control, gRPC is used for communication between the core and a user interface, and for communication between the executor and the OCC plugin (see Fig. 2). We also expect to use gRPC to interface with other systems such as the trigger system and the DCS.

### 4.3 O$^2$ control concepts

The basic unit of scheduling in O$^2$ control is a *task*. A task generally corresponds to a process, specifically a process that can receive and respond to OCC-compatible control messages. An O$^2$ control workflow is ultimately made of tasks.

Tasks are the leaves in a tree of *roles*. A role is a runtime subdivision of the complete system, it represents a kind of operation along with its resources. Each task implements one or more roles. Roles allow binding tasks or groups of tasks to specific host attributes, detectors and configuration values. Each role represents either a single task, or a group of child roles. If tasks are leaves, roles are all the other nodes in the control tree of an environment. Thus, for example, one can have a task "`readout-bin`", which is a child of role "`readout-tpc-026`". The latter role, in turn, is a child of role "`readout-tpc`", which is a child of role "`readout`", which is a child of top-level role "`physics-1`".

In comparison with the ECS partitions used in Run 2, we aim to provide novel, more flexible, and more easily deployable abstractions. In memory, a tree of O$^2$ roles, along with their tasks and their configuration is a *workflow*. A workflow aggregates the collective state of its constituent O$^2$ roles. A running workflow, along with associated detectors and other hardware and software resources associated with experiment operation constitutes an *environment*. When an environment is in state `RUNNING`, it implements an *activity*, such as a data taking run.

### 4.4 The environment state machine

Environments have states, constrained by a state machine. In memory, each role as a node in the control tree also has a state, which aggregates the states of its child roles (or of its single child task). Thus, the state machines of each individual process are directed by the top-level state machine of the environment.

The environment state machine is the entry point for all process control operations. Some examples of control requests at this level include creating a new environment by loading a workflow template from the configuration repository (which also instantiates the new environment's state machine), requesting a state transition for an environment, or modifying a subtree in an environment's workflow.

### 4.5 Workflow configuration

The O$^2$ Configuration repository contains a list of O$^2$ task templates. A task template is a configuration item that describes the command that launches the relevant process, plus some command-specific, FairMQ-specific, or Mesos-specific configuration parameters.

The O$^2$ Configuration repository also contains workflow template structures, which, when instantiated, result in a workflow associated with an environment. These template structures can be represented via common hierarchical human-readable data formats such as JSON or YAML. This representation is further enriched with a template syntax, which allows expressing iterators, variables, and internal references.

The workflow template format is not primarily intended as a data interchange format: it is rather the human-readable representation of a curated list of templates, which are maintained in the O$^2$ control configuration repository. Importing and exporting such data in a meaningful way will be handled by the `coconut` tool, which will also provide support for topologies generated by other workflow description mechanisms, such as the O$^2$ DPL (Data Processing Layer) [17].

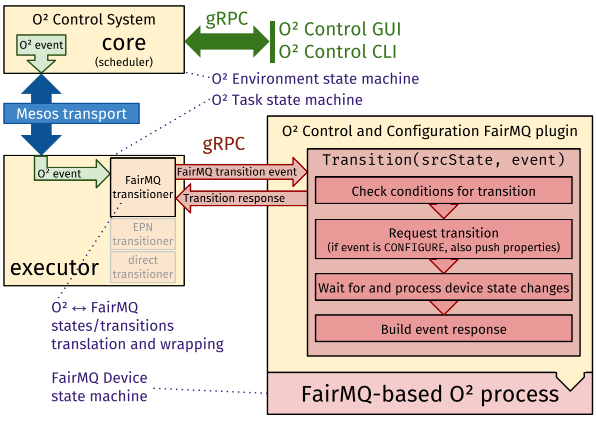
O² Control System
O² event
core
(scheduler)
gRPC
O² Control GUI
O² Control CLI
O² Environment state machine
O² Task state machine
Mesos transport
gRPC
O² Control and Configuration FairMQ plugin
Transition(srcState, event)
Check conditions for transition
Request transition
(if event is CONFIGURE, also push properties)
Wait for and process device state changes
Build event response
O² event
FairMQ transitioner
FairMQ transition event
Transition response
EPN transitioner
direct transitioner
executor
O² ↔ FairMQ states/transitions translation and wrapping
FairMQ Device state machine
FairMQ-based O² process

**Figure 3.** The O$^2$ control executor integrates modular components called *transitioners*. These units act as translation wrappers between O$^2$ control states and events, and the states and events of the state machine of a specific controlled process. In the figure above the executor has loaded the FairMQ transitioner, which drives the state machine of a FairMQ-based process.

### 4.6 O$^2$ process control

Most O$^2$ processes are also FairMQ devices, i.e., programs that make use of the FairMQ library for its state machine and I/O facilities. FairMQ provides a plugin system, which is capable of loading the purpose-built O$^2$ control and configuration plugin for FairMQ devices. This plugin enables any FairMQ device to accept control commands from an O$^2$ control executor (see Fig. 3). The OCC plugin takes control of the process on startup, and starts a gRPC server on a specific TCP port as instructed by the O$^2$ control executor. When the OCC plugin receives a remote procedure call from the executor, it drives the state machine of the FairMQ device and reports back. The OCC plugin is also capable of pushing configuration key-value pairs as FairMQ properties to the FairMQ configuration map of the device.

O$^2$ tasks are started on demand when an environment's roles require them, but they are generally not killed when an environment is disbanded. Instead, they are kept in an *idle tasks pool*, ready to be reconfigured and used without additional deployment steps (see Fig. 4). Thus, since automatic port assignment and other crucial data flow setup operations happen at task configuration time rather than at task startup, it is possible to stop an environment, add or remove a subtree of roles, and resuming operation without having to redeploy all tasks.

## 5 Conclusion

We propose a new, custom built, microservices oriented solution for synchronous control in the O$^2$ computing system. We assert that the leap to O$^2$ is an opportunity for a broad technical

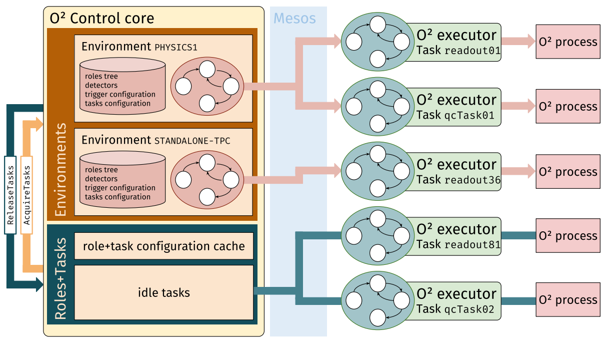

**Figure 4.** Example of how O$^2$ control handles task deployment. Tasks in an idle state are generally not killed when their parent environment is torn down, instead they are kept running and tracked in an idle tasks pool. If and when necessary, these active but idle tasks can be reacquired into an environment if they can satisfy a role in that environment's workflow. They are then reconfigured and reused.

refresh by leveraging modern cluster resource management and IPC technologies for a high performance, low latency O$^2$ control.

By taking advantage of Apache Mesos, we gain resource management, control message transport, events, and more, with the goal of achieving improved operational flexibility. On top of this framework, we implement a distributed state machine mechanism, with an expressive configuration format and a modular process control stack for maximum compatibility in an inevitably heterogeneous context.

We aim to minimize the waste of LHC beam time while ensuring optimal usage of the new O$^2$ facility for both synchronous and asynchronous data-driven workflows. With the O$^2$ control system we make provisions for self-contained WLCG-compatible environments for asynchronous operation. With our design approach we aim to achieve substantial performance improvements and operational benefits in mission critical use cases compared to the previous system.

## References

[1] K. Aamodt et al. (ALICE), JINST **3.08**, S08002 (2008)

[2] B. Abelev et al. (ALICE), Journal of Physics G: Nuclear and Particle Physics **41**, 087001 (2014)

[3] J. Adam et al. (ALICE), Tech. Rep. CERN-LHCC-2015-006 / ALICE-TDR-019, CERN (2015)

[4] J. Mitra, S. Khan, S. Mukherjee, R. Paul, Journal of Instrumentation **11**, C03021 (2016)

[5] A. Borga, F. Costa, G. Crone, H. Engel, D. Eschweiler, D. Francis, B. Green, M. Joos, U. Kebschull, T. Kiss et al., Journal of Instrumentation **10**, C02022 (2015)

[6] *FairMQ C++ Message Queuing Library and Framework*, https://github.com/FairRootGroup/FairMQ, accessed: 2018-10-08

[7] M. Al-Turany, D. Bertini, R. Karabowicz, D. Kresan, P. Malzacher, T. Stockmanns, F. Uhlig, Journal of Physics: Conference Series **396**, 022001 (2012)

[8] M. Al-Turany, P. Buncic, P. Hristov, T. Kollegger, C. Kouzinopoulos, A. Lebedev, V. Lindenstruth, A. Manafov, M. Richter, A. Rybalchenko et al., Journal of Physics: Conference Series **664**, 072001 (2015)

[9] F. Carena, W. Carena, S. Chapeland, V.C. Barroso, F. Costa, E. Dénes, R. Divià, U. Fuchs, A. Grigore, T. Kiss et al., Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **741**, 130 (2014)

[10] P. Chochula et al., Proceedings of the 16th International Conference on Accelerator and Large Experimental Control Systems pp. 323–327 (2018)

[11] S. Bagnasco, L. Betev, P. Buncic, F. Carminati, C. Cirstoiu, C. Grigoras, A. Hayrapetyan, A. Harutyunyan, A.J. Peters, P. Saiz, Journal of Physics: Conference Series **119**, 062012 (2008)

[12] *Apache Mesos*, http://mesos.apache.org/, accessed: 2018-10-04

[13] D. Berzano, G. Eulisse, C. Grigoraş, K. Napoli, Journal of Physics: Conference Series **898**, 082043 (2017)

[14] *The O$^2$ Control System*, https://github.com/AliceO2Group/Control, accessed: 2018-10-08

[15] *Consul by HashiCorp*, https://www.consul.io/, accessed: 2018-10-08

[16] *gRPC A high performance, open-source universal RPC framework*, https://grpc.io/, accessed: 2018-10-04

[17] *ALICE O$^2$ software*, https://github.com/AliceO2Group/AliceO2, accessed: 2018-10-08