Readout software for the ALICE integrated Online-Offline (O^2) system

Sylvain Chapeland[1,*], *Filippo* Costa[1], on behalf of the ALICE collaboration

[1]CERN, Geneva, Switzerland

Abstract. ALICE (A Large Ion Collider Experiment) is a heavy-ion detector studying the physics of strongly interacting matter and the quark–gluon plasma at CERN's LHC (Large Hadron Collider). During the second long shutdown of the LHC, the ALICE detector will be upgraded to cope with an interaction rate of 50 kHz in Pb–Pb collisions, producing in the online computing system (O^2) a sustained input throughput of 3 TB/s. The readout software is in charge of the first step of data-acquisition, handling the data transferred from over 8000 detector links to PC memory by dedicated PCI boards, formatting and buffering incoming traffic until sent to the next components in the processing pipeline. On the 250 readout nodes where it runs, the software has to sustain a throughput which can locally exceed 100 Gb/s. We present the modular design used to cope with various data sources (hardware devices and software emulators), integrated with the central O^2 components (logging, configuration, monitoring, data sampling, transport) and initiating the online data flow using the standard O^2 messaging system. Performance considerations and measurements are also discussed.

1 Introduction

1.1 ALICE and the O^2 project

ALICE [1] is the heavy-ion detector designed to cope with very high particle multiplicities to study the physics of strongly interacting matter at CERN's LHC. It is optimized to study the properties of the deconfined state of quarks and gluons produced in such collisions known as quark–gluon plasma [2]. It is also well suited to study elementary collisions such as proton–proton and proton–nucleus interactions.

The detector will be upgraded [3] in the next LHC long shutdown, planned 2019–2020, and will produce an increased data throughput reaching 3.4 TB/s. The Online-Offline system, named O^2 [4], will be in charge of reading out these data and processing them on-the-fly, in order to reduce the volume down to 90 GB/s initially recorded to storage. These demanding data acquisition and processing steps will be handled by a computing farm consisting of ~250 nodes for readout (named FLPs – First Level Processors) and ~1500 nodes for online reconstruction (named EPNs – Event Processing Nodes).

[*] Corresponding author: sylvain.chapeland@cern.ch

In O^2, detector data are sliced into chunks spanning ~50 ms each, called timeframes. Based on the centralized timing information provided by the trigger system, the FLPs create synchronously for each timeframe period the sub-timeframe associated to the fraction of the detector they are connected to. All sub-timeframes with a given timestamp are sent to the same EPN (changing for each timeframe), which builds the corresponding full timeframe containing all data for this period of time, and then used for reconstruction. Data distribution to the EPNs is typically done round-robin, but may also take into account load balancing requirements (network and CPUs availability).

1.2 Readout hardware

The detector outputs data through ~8000 optical links connected to PCI readout cards installed in the FLPs, in the surface counting room 100 m away from the experiment cavern. These are mainly GBT links [5], a radiation-hard bi-directional 4.8 Gb/s connection transmitting data from the detector and delivering trigger and slow control information. Some sub-detectors will also reuse the custom DDL1 and DDL2 links [6] as before the ALICE upgrade, running respectively at 2.125 Gb/s and 5.3 Gb/s.

The CRU board [7] is used to accommodate the large number of GBT fibres and implement a high-density readout system. This is a dedicated PCIe Gen. 3 x16 card hosting up to 48 links. It provides a typical throughput of 110 Gb/s thanks to a dual DMA engine Gen3. x8 running on an Intel Arria10 FPGA. For the DDL links, O^2 will reuse some C-RORC cards currently used in ALICE, based on a Xilinx VIRTEX6 FPGA hosting up to 6 links and connected to a PCIe Gen.2 x8 bus. Each FLP will host from 1 to 4 readout cards, depending on the balance between detector links throughput, network bandwidth output, and CPU processing needs. Both card types can also be used to perform some data processing on-the-fly in the FPGA (e.g. cluster finding, as done already in ALICE [8]), in order to reduce the amount of CPU resources needed for the reconstruction later in the chain.

Readout data is delivered directly from the PCI cards to the PC host memory by DMA, without involving the FLP CPUs. A RoC (Readout Card) device driver and libraries [9,10] allow this flow to be controlled by the readout software.

2 Readout software

In this paper, the focus is on the readout software. It consists of a runtime process running on each FLP, dealing with moving the data from the detector electronics into the memory of the computer hosting the readout cards, and streaming those to further FLP processes in the O^2 online pipeline (e.g. quality control, sub-timeframe building, local processing, transport to EPN). In the present paper, we will refer to this runtime process simply as *Readout*.

Readout tasks consist in particular of:
- Initializing the hardware (CRU and C-RORC) using the common RoC driver interface.
- Allocating the memory buffers used for DMA.
- Providing data pages to be filled by the PCIe device.
- Aggregating, slicing, and formatting the data received.
- Checking the data consistency.
- Distributing the data to consumers.
- Reporting performance and errors.

2.1 *Readout* architecture

In order to accommodate the various needs and keep the flexibility to evolve with time, *Readout* consists of multiple interconnected modules each assigned to a task. In particular, two general-purpose classes of components have been defined: producers of data (also named equipments) and consumers of data, organized around a central core taking care of the data handling. Multiple instances of each module may be created depending on the configuration. Figure 1 shows a typical runtime setup.

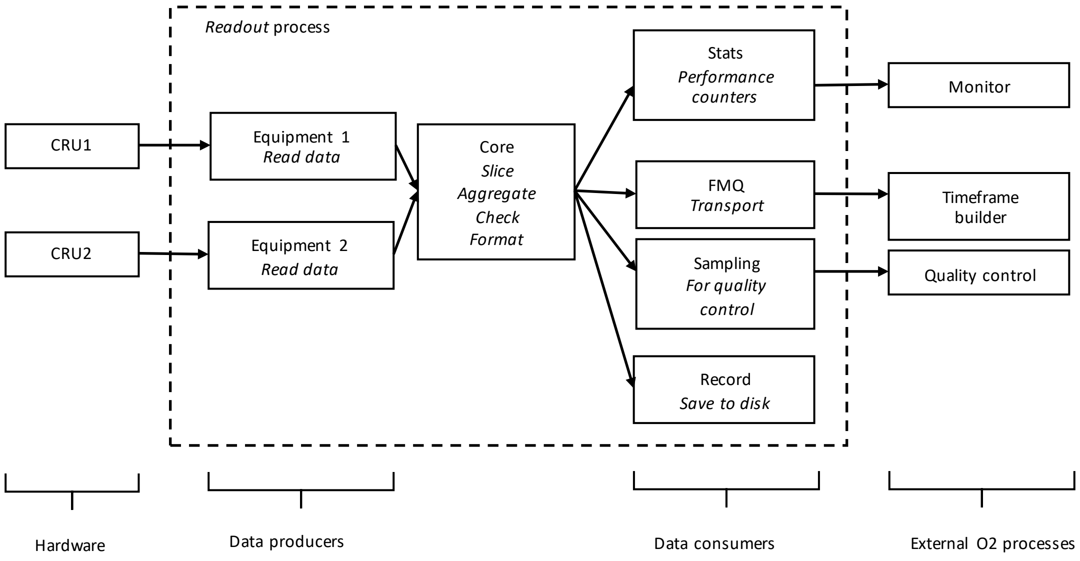

Fig. 1. *Readout* architecture and internal data flow (example runtime configuration).

2.2 Implementation

2.2.1 Multi-threaded pipeline

The *Readout* process is a multi-threaded application, to benefit from the multiple CPU cores available on the FLPs and run the different tasks in parallel. It starts one thread per component, internally connected by lock-less FIFO buffers to implement the flow of data from producers to consumers. This ensures a low-overhead transport between the modules and independent operation of each thread. The building blocks developed in a previous many-core data streaming application [11] were reused, allowing inter-thread messaging up to a Megahertz rate without significant CPU usage, while allowing non-blocking operation when there is space in the buffer. A main thread takes care of distributing the data between threads and to synchronize them on process startup and shutdown. This approach is slightly different from the previous ALICE data-acquisition system [12], where there was one process per equipment and one process for each other task. This has the advantage of simplifying code, component communication and control.

The basic workflow consists of several steps: data are produced by readout equipments, a data aggregator aggregates their output based on common data block identifiers (also performing checks and header formatting), and the result is distributed to consumers. Data chunks are shared, each consumer may use (read-only) the incoming data before releasing it. Pushing data to consumers is by default a blocking operation (if FIFO full, *Readout* waits until being able to push). Backpressure is applied upstream between the *Readout* threads (output FIFO of step N-1 is not emptied anymore when input FIFO of step N full).

The code [13] is written in C++14, using the typical building blocks of the standard library (in particular the *shared_ptr*, *unique_ptr*, and *thread* classes simplifying the development of multi-threaded applications).

Polymorphic classes are defined for producers and consumers, so that new implementations can easily be added while keeping the same interface. The corresponding runtime instances are created from configuration (local file or central O^2 repository). *Readout* can handle any number of instances of each producer and consumer class.

For the time being, there are 3 types of producers: a simple software generator to push data to memory without a hardware readout card, a more elaborated emulator generating data with realistic LHC clock rates and formatted data, and finally the readout class able to get data from C-RORC and CRU devices using the RoC library. As a side remark, from the driver and software point of view, each CRU x16 card is actually seen as two independent x8 devices, so *Readout* instantiates two RoC equipments to read out one CRU card.

On the consumers' side, a wide variety of features are provided:
- a *Stats* class, to count the number and size of blocks produced by *Readout* (among other statistics). The counters can be published to the O^2 monitoring system for live or historic views of the rates and performance.
- a *FileRecorder* class, allowing the readout data to be written to a local file. This is useful for front-end electronics development and debugging, but not in production.
- a *DataChecker* class, with extensive data checks in the header and payload, verifying systematically all fields integrity and conformance to CRU format specifications. It is used in particular for debugging and stability tests, but might not be used for production depending on the available CPU resources.
- a *DataSampling* class, providing data (full or subset) to the Quality Control system [14].
- a *FMQ* class, pushing the data outside *Readout* as a FairMQ device. FairMQ [15, 16] is the common transport library used in O^2.

Readout is also integrated with the base O^2 packages providing facilities for logging [17], configuration [18], and monitoring [19, 20].

2.2.2 Memory management

Readout provides to the CRU, by means of a FIFO buffer, the free memory pages where incoming data can be written. When a data page is full, the CRU gives it back through a second FIFO buffer, holding the ready pages to be picked up by *Readout* for the data consumers.

The CRU card transfers data over PCIe DMA in chunks of 8 kB. To reduce the rate of control messages between host and card when providing the empty pages and getting back the ready pages, larger data pages (typically 2 MB) are given and filled with contiguous 8 kB blocks. This effectively reduces the FIFOs rate from ~1.7 MHz to ~7 kHz, relieving both the CPU and PCI bus. As each 8 kB block has a header, it is still possible to easily browse the data within each page.

Large (multi-gigabyte) buffers are needed to cope with the optical links throughput. *Readout* therefore creates a pool of memory pages on start-up. Large physically contiguous memory regions are allocated, and registered to the driver in order to set up the DMA for the whole address range(s). These big (~GB) regions are split by *Readout* into smaller (~MB) data pages of selected size, with the proper byte-alignment needed to optimize the DMA transfers. *Readout* also enforces the allocation of the memory on the same NUMA node connected to the PCIe card. At runtime, free pages are taken from the pool, given to the CRU, taken back when filled with data, provided to the consumers (and possibly shared by them), and put back in the pool when not used anymore, ready again for a new write cycle.

The pages themselves are allocated from different support types, depending on the needs: simple malloc(), memory mapped files (provided by the RoC library, based on HugeTLBFS), and shared memory (provided by FairMQ, and used for inter-process or network zero-copy transport).

2.3 Performance

Readout has been developed and benchmarked on various platforms to understand and optimize the numerous tuning parameters, both for software and hardware. The system consistently showed good performance well above 6 GB/s per PCIe x8 CRU end-point, and routinely reaches the maximum throughput of 6.75 GB/s per x8 end-point (i.e. 13.5 GB/s per CRU) with the appropriate settings, even with multiple CRUs on the system.

The most critical settings are usually related to hardware, in particular the memory pages used for the DMA, which need to be aligned and take into account the system NUMA configuration for minimal PCI to memory latency.

On the software side, the size of the FIFOs between threads is not critical, and the system works well with FIFOs having ~100 entries, providing enough back-buffer to allow the threads to be idle when they are empty (instead of an aggressive polling misusing CPU resource). The CPU deep sleep features are disabled by software (just for the *Readout* process) for best DMA performance (otherwise latency may increase when *Readout* goes sleeping, waiting the FIFO to fill in).

The standard development setup consists of a DELL R740 server equipped with 1 or 2 CRUs. For the detector commissioning, some ASUS ESC4000G3 machines have been set up with 2 CRUs running at a total of 26 GB/s (6.5 GB/s per equipment). As shown in Figure 2, long-running tests show excellent stability over 5 days, using only 12% of one CPU virtual core (the machine has 2x Xeon E5-2690v2 @ 3GHz, seen as 40 processors by Linux when hyper-threading is enabled), i.e. around 0.3% of the system CPU resources are used for *Readout*. This result validates the efforts done by *Readout* to minimize CPU usage for the flow control and memory management, effectively reducing polling between threads and leaving the CPU for other local tasks.

Readout was also tested with up to 8 CRUs in a server, using a Supermicro 4029GP-TRT server providing 8 PCIe slots x16, half-duplex, allowing a measured aggregated throughput of 40 GB/s for the 16 equipments (no optimization). Although this setup does not reach the maximum bandwidth of each CRU (there are not enough PCIe lanes on the CPUs, so the PCIe x16 does not run at full speed), this is a convenient system for batch testing of the CRUs (over 300 will be produced for the ALICE upgrade).

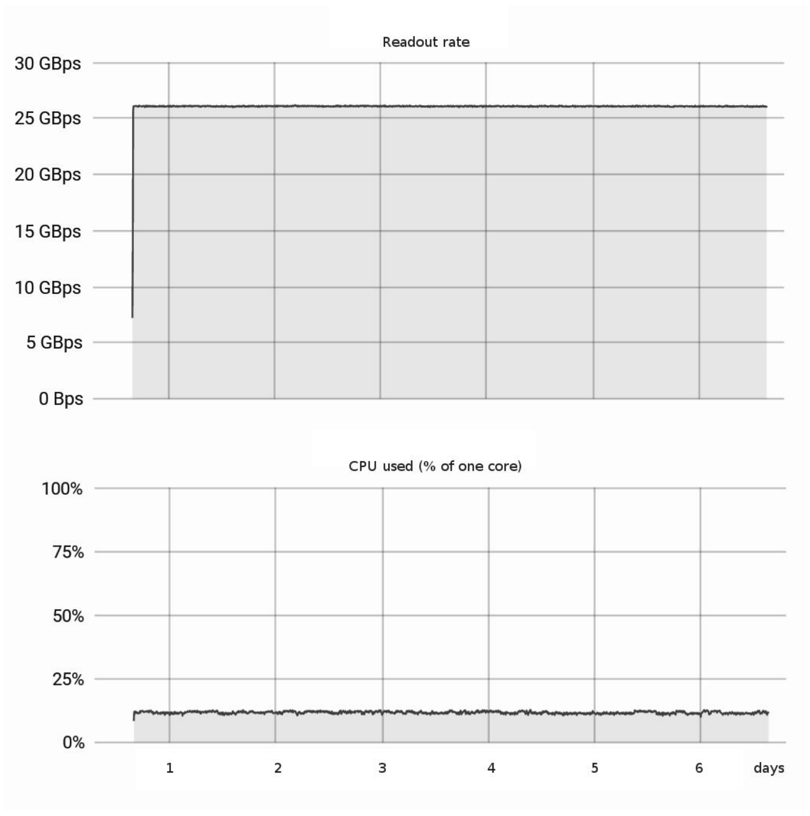

Fig. 2. CRU throughput and CPU resources of *readout* running on a dual-CRU system.

3 Conclusions

Readout is the process used in ALICE O^2 to initiate DMA transfer from PCIe readout cards to PC memory, and the first software component in the O^2 pipeline. It is connected to the O^2 subsystems (libraries and runtime processes), and provides a lightweight and extendable architecture to deliver the needed development, debugging, and production tools to inject the detector data coming at an aggregated 3 TB/s from 8000 links in the future ALICE online system. It is capable of handling 13.5 GB/s from each of the CRU boards, and it will run on the ~250 O^2 readout nodes. It is currently used in the detector commissioning, which started in summer 2018.

References

1. The ALICE Collaboration et al. "*The ALICE experiment at the CERN LHC*"JINST **3** S08002 (2008)
2. N. Cabibbo, G. Parisi Phys. Lett. B **59** pp 67-69 (1975)
3. The ALICE Collaboration et al. "*Upgrade of the ALICE Experiment: Letter Of Intent"* J. Phys. **G 41** 087001 (2014)

4. P. Buncic, M. Krzewicki, P. Vande Vyvre et al. *"Technical Design Report for the Upgrade of the Online-Offline Computing System"* Technical Design Report ALICE **19** (2015)
5. P. Moreira et al. "*The GBT project"* Proceedings of the Topical workshop on electronics for particle physics in Paris, France 342-346 (2009)
6. F.Costa et al. *"DDL, the ALICE data transmission protocol and its evolution from 2 to 6 Gb/s"* JINST **10** C04008 (2015)
7. J. Mitraa, S.A. Khana, S. Mukherjeeb and R. Paulc *"Common Readout Unit (CRU) - A new readout architecture for the ALICE experiment"* JINST **11** C03021 (2016)
8. H. Engel, T. Alt and U. Kebschull *"FPGA based data processing in the ALICE High Level Trigger in LHC Run 2"* J. Phys. Conference Series **898** 032018 (2017)
9. The O^2 project, *Readout Card (RoC) module* http://github.com/AliceO2group/ReadoutCard
10. P. Boeschoten and F. Costa "*The ALICE O2 common driver for the C-RORC and CRU read-out cards"* ACAT 2017 conference proceedings arXiv:1710.05607 (2017)
11. S.Chapeland "*A programming framework for data streaming on the Xeon Phi"* J. Phys. Conference Series **898** 072007 (2017)
12. F. Carena et al. *"The ALICE data acquisition system"* Nucl. Instr. Meth. A **741** 130-162 (2014)
13. The O^2 project, *"Readout"* http://github.com/AliceO2group/Readout
14. B. von Haller, P. Lesiak and J. Otwinowski *"Design of the data quality control system for the ALICE O2"* J. Phys. Conference Series **898** 032001 (2017)
15. M. Al-Turany, D. Klein, T. Kollegger, A. Rybalchenko, N. Winckler *"C++ Message Queuing Library and Framework"* https://github.com/FairRootGroup/FairMQ
16. M. Al-Turany et al. *"ALFA: ALICE-FAIR new message queuing based framework"* Proceedings of the CHEP 2018 conference (to be published)
17. The O^2 project, *"InfoLogger, the logging system for ALICE O2"* https://github.com/AliceO2Group/infoLogger
18. The O^2 project, *"The configuration system for ALICE O2"* https://github.com/AliceO2Group/Configuration
19. The O^2 project, *"The monitoring module for ALICE O2"* https://github.com/AliceO2Group/Monitoring
20. A. Wegrzynek et al. *"Towards the integrated ALICE Online-Offline (O2) monitoring subsystem*" Proceedings of the CHEP 2018 conference (to be published)