Leveraging the Run 3 experience for the evolution of the ATLAS software-based readout towards HL-LHC

Serguei Kolos[1,*] on behalf of ATLAS TDAQ Collaboration

[1]University of California, Irvine, CA 92697-4575, USA

Abstract. The High-Luminosity Large Hadron Collider (HL-LHC), scheduled to start operating in 2030, aims to increase the instantaneous luminosity by a factor of 10 compared to the LHC. To match this increase, the ATLAS experiment has been implementing a major upgrade program divided into two phases. The first phase (Phase-I), completed in 2022, introduced new trigger and detector systems that have been used during the Run 3 data taking period which began in July 2022. These systems have been used in conjunction with the new Data Acquisition (DAQ) Readout system, based on a software application called Software Readout Driver (SW ROD). SW ROD receives and aggregates data from the front-end electronics via the Front-End Link eXchange (FELIX) system and passes aggregated data fragments to the High-Level Trigger (HLT) system. During Run 3, SW ROD operates in parallel with the legacy Readout System (ROS) at an input rate of 100 kHz. For the Phase-II, the legacy ROS will be completely replaced with a new system based on the next generation of FELIX and an evolution of the SW ROD application called Data Handler. Data Handler has the same functional requirements as SW ROD but must be able to operate at an input rate of 1 MHz. To facilitate this evolution the SW ROD has been implemented using plugin architecture. This contribution presents the design and implementation of the SW ROD application for Run 3, along with the strategy for its evolution to the Phase-II Readout system. It discusses the lessons learned during Run 3 and describes the challenges that have been addressed to accomplish the demanding performance requirements of HL-LHC.

ATL-DAQ-PROC-2025-001
29 January 2025

1 Introduction

In preparation for LHC Run 3, which commenced in July 2022, several new detector and trigger components were integrated into the ATLAS experiment [1]. A novel Readout system has been developed to facilitate data collection from these detectors. This system incorporates a new element known as the Software Readout Driver (SW ROD) [2], specifically engineered to receive data from the newly implemented detectors through the Front-End Link eXchange (FELIX) framework [3]. Upon acquisition of detector data, the SW ROD is responsible for event fragment construction, which may involve the aggregation of data across multiple FELIX streams. Subsequently, it sends the completed data fragments to the High-Level Trigger (HLT) and buffers them until they are explicitly discarded following the generation of the HLT decision.

*e-mail: serguei.kolos@uci.edu

2 FELIX in Run 3

FELIX is a custom-designed PCIe card that receives data from the detector FE electronics through several types of optical links and transfers the data to the memory of a commodity computer via the PCIe bus. During Run 3, FELIX is used to receive data either via the GigaBit Transceiver (GBT) [4] or the in-house designed FULL mode protocol, which allows input links to operate at a rate of 9.6 Gb/s. Run 3 FELIX cards feature 24 optical input links, which, in GBT mode, can be split into up to 192 virtual links known as E-Links. The optical links can operate at speeds of up to 4.8 Gb/s in GBT mode and 9.6 Gb/s in FULL mode. However, the overall bandwidth of a FELIX card is limited to 128 Gb/s by the 16-lane PCI Express 3.0 interface.

3 SW ROD in Run 3

The SW ROD facility is implemented by a number of homogeneous software processes running on a set of commodity computers. Each process uses a specific configuration that defines a set of E-Links for data reception, along with data processing parameters, such as the fragment-building algorithm. In Run 3, the SW ROD provides two data-handling algorithms, named after the FELIX card modes they are typically used with: the FULL Fragment Builder and the GBT Fragment Builder.

The FULL Mode Fragment Builder is typically used to handle data received from FELIX cards operating in FULL mode. In this mode, FELIX optical input links cannot be subdivided into virtual E-Links, making it particularly suitable for cases that require the transmission of large data packets, typically several kilobytes in size. These packets usually contain data aggregated by the FE electronics from multiple detector channels and, therefore, do not require further aggregation before being sent to the HLT. This mode doesn't represent any difficulty for the corresponding Fragment Builder as all it has to do is to store incoming data packets in the internal buffer until they are processed by the HLT.

3.1 GBT Mode Fragment Builder

In contrast to the FULL Mode, the GBT mode poses a considerable challenge for the SW ROD. Typically, FELIX cards operating in GBT mode receive numerous small data packets that require aggregation before being transmitted to the High-Level Trigger (HLT). Consistent with the established conventions of the legacy Readout System, a channel responsible for generating aggregated data segments from a specific set of E-Links is referred to as a Readout Buffer (ROB). Consequently, the data segments produced by such a channel are designated as ROB fragments. The data aggregation process utilizes identifiers generated by the ATLAS Level 1 (L1) Trigger to ensure that each fragment comprises data packets associated with distinct collision events originating from every E-Link overseen by the SW ROD application. A single SW ROD application possesses the capability to manage an arbitrary number of ROBs.

Given that the granularity of ROBs can be affected by various external software and hardware constraints - such as data quality monitoring, data analysis, and network bandwidth - the GBT Fragment Builder must exhibit sufficient flexibility to aggregate data into a single ROB from an arbitrary number of E-Links, as defined in its current configuration. Simultaneously, this flexibility should exert minimal influence on algorithm performance in order to maintain low costs for the new Readout System.

In Run 3, the main user of this algorithm was the new set of Muon System detectors, known as the New Small Wheel (NSW) [5], which was built and commissioned during the

ATLAS Phase-I Upgrade. The NSW readout system consists of approximately 7,500 readout E-Links, distributed across 30 FELIX cards. The mapping of E-Links to ROBs varies across different NSW subsystems, with some ROBs receiving data from up to 160 E-Links. Handling such ROBs with the minimal usage of computing resources represented a significant challenge, which was addressed by splitting the algorithm implementation into two steps, as illustrated in Figure 1.

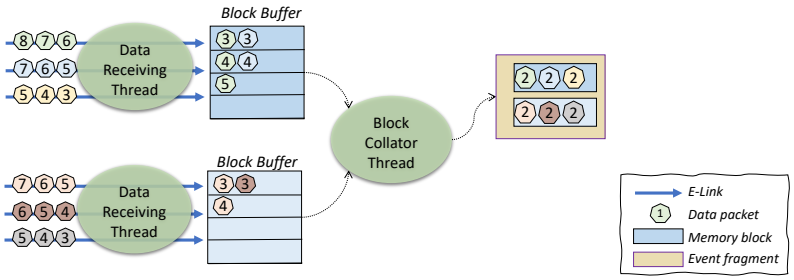


Figure 1: GBT Fragment Builder algorithm.

In the first step, all input E-Links associated with a given ROB are distributed among a configurable number of data-receiving threads. Each thread aggregates data from a unique subset of these E-Links into a dedicated memory block. In the second step, the individual blocks produced by different threads are combined to form complete ROB fragments. During the first step, the threads operate independently and do not interact until the fully aggregated data blocks are submitted to the Block Collator.

The Block Collator uses a *TBB concurrent hash map* container [6] to aggregate data blocks that correspond to the same event by using the L1 Trigger ID as the key of this map. Each key in the container corresponds to a vector of data blocks with a capacity equal to the number of data-receiving threads. When a new block is added to the appropriate vector, and the vector's size reaches its predefined capacity, the fragment is considered fully built. At this stage, the Block Collator thread moves the fragment to another buffer, where it is held until explicitly cleared by the HLT.

This implementation efficiently scales with the number of input E-Links, by distributing the bulk of the workload across multiple independent threads.

3.1.1 Performance of the Initial Implementation

The assessment of the performance and scalability of the GBT Mode Fragment Builder was conducted through a series of tests executed on a computing system identical to that employed by the new ATLAS Readout System during Run 3. This system is configured with dual Intel® Xeon® Gold 5218 processors and equipped with 96 GB of DDR4-2667 RAM. Two series of tests were performed:

- **Single Data Receiving Thread:** In this series the Fragment Builder used a single data receiving thread to process 40-byte data packets. The number of E-Links varied from 10 to 160.
- **Multiple Data Receiving Threads:** In these tests the same algorithm employed one data receiving thread for every 40 E-Links. The number of E-Links varied from 40 to 160.

In both test series one *Block Collator* thread was used. The results of these tests are presented in Figure 2.

The outcome of the tests indicated that the algorithm was fully capable to handle NSW ROBs with the maximum number of input E-Links. It also demonstrated excellent scalability

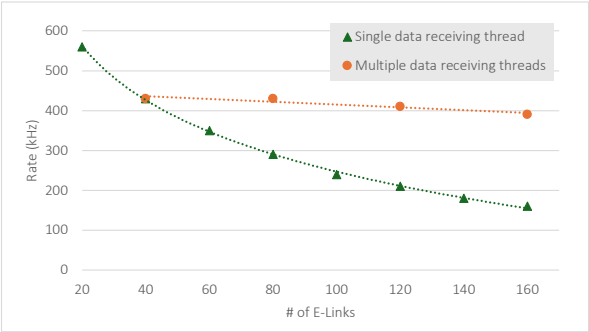


Figure 2: GBT Fragment Builder Performance: Input Rate vs. Input Channels

with an increasing number of input threads. Nonetheless, during commissioning of the Readout System an entirely unexpected yet recurring issue has been revealed: data packets were being lost at the start of a new high-rate run. The nature of the issue and the corresponding solution will be discussed in the next section.

3.1.2 The Data Loss Issue

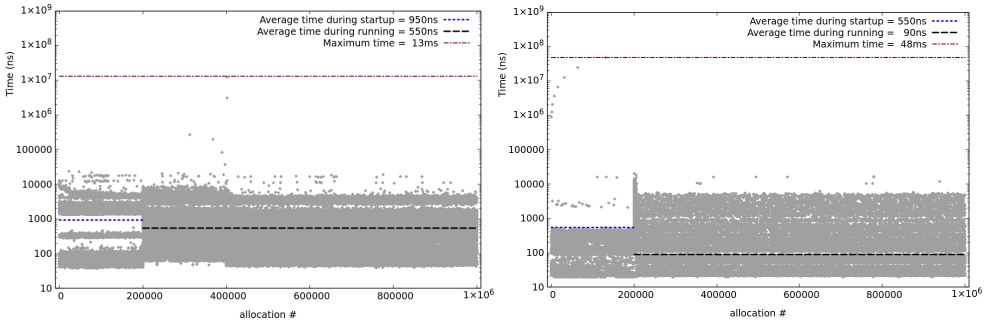
The SW ROD application experienced a brief disruption during the initial seconds of a new high-rate data-taking session. As a result, a few incomplete ROB fragments were produced due to data reception timeouts. This occurs when data from certain E-Links cannot be processed within the time frame specified by the timeout value, causing the ROB fragments to lack data from these E-Links. Any delayed data packets that arrived after the corresponding ROB fragments had already been constructed were consequently lost.

A study was conducted to identify the root cause of this issue. Several potential factors were investigated and ruled out, including network latency and sporadic delays from the operating system scheduler. Network latency was consistently below 100 microseconds, while OS scheduler latency did not surpass one millisecond. Ultimately, the disruption was traced to the *std::malloc* memory allocation routine, which was found to occasionally require more than 10 milliseconds to complete its execution, as revealed in a specifically conducted test.

In this test, the time required by the *std::malloc* function to allocate one million 2 KB memory blocks was measured. Initially, the allocated memory was retained until 200,000 blocks had been allocated. After reaching this threshold, another thread began releasing the allocated memory while memory allocation continued. This pattern mirrors the behavior of the SW ROD at the start of a new data-taking session, where fully built ROB fragments are temporarily stored in the SW ROD internal buffers during the time that HLT requires to process the first pile of events. Memory is only released once the HLT begins generating event acceptance decisions. Results of this test are presented in Figure 3a.

To address this issue, *std::malloc* was initially replaced with the Boost.Pool[7] memory pool implementation, which proved to be well-suited for the task. Since the GBT Fragment Builder could not determine the full fragment size in advance, it always pre-allocated memory blocks of a fixed size. While using Boost.Pool noticeably improved the average allocation time, the data loss issue became even more pronounced. As illustrated in Figure 3b, some memory allocation calls under this approach took even longer - up to 50 milliseconds to complete.

Fortunately, the behavior of the Boost memory pool could be easily traced, as Boost.Pool is an open-source implementation consisting of only a few hundred lines of code. The oc-



(a) std::malloc (glibc 2.34)

(b) Boost.Pool (Boost version 1.82.0)

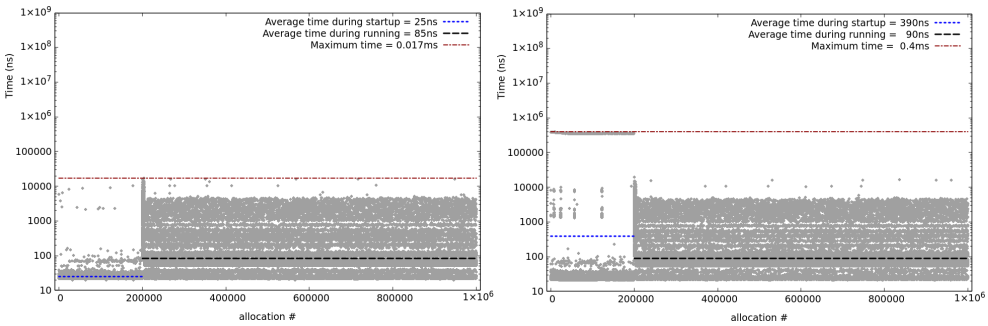
Figure 3: Memory allocation time for one million 2 KB blocks

casional delays were found to occur when the memory pool requested a new large block of memory from the operating system. Once the block was acquired, the memory pool implementation split it into smaller chunks of the requested size and organized them into a singly linked list by storing a pointer to the next chunk at the start of each one. This process was the primary cause of the high latency.

The underlying reason was that, for newly allocated memory, the write operation triggered virtual-to-physical memory mapping. This mapping is performed once for every 4 KB memory page during its first write and involves complex coordination between the operating system and the CPU, causing substantial delays. Additionally, the total delay increased logarithmically with each successive memory block, as its size was always doubled. These delays are visible in the top-left side of the plot shown in Figure 3b.

3.1.3 Solution of the Data Loss Problem

A simple modification to the Boost.Pool memory allocation algorithm was implemented to address the issue. This change eliminated the organization of memory chunks into a list when acquiring a new large memory block from the operating system. Instead, the updated implementation initializes an empty list of free chunks in the constructor and adds a chunk to this list only when it is released. This modified memory pool demonstrated exceptional performance, as illustrated in Figure 4a.



(a) Without virtual-to-physical memory mapping

(b) With virtual-to-physical memory mapping

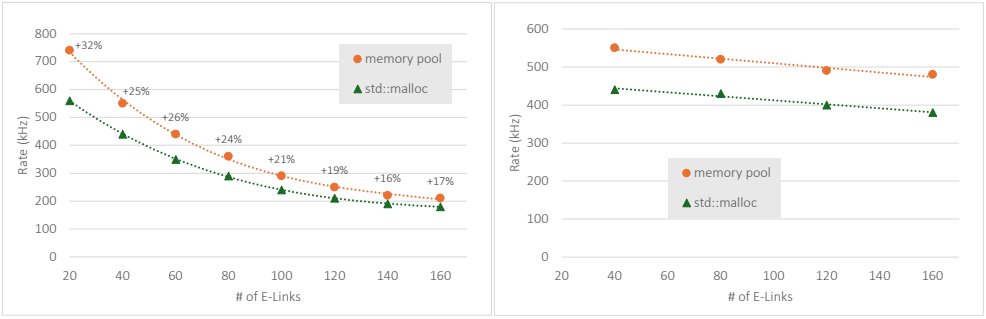
Figure 4: The modified memory pool allocation time for one million 2 KB blocks

However, it is important to note that this approach defers the virtual-to-physical memory mapping until data is written to a chunk for the first time. To ensure a fair comparison between the modified and original memory pools, the test application was updated to write a non-zero value to the first byte of each allocated memory chunk immediately after allocation. The results of this second test are shown in Figure 4b.

The test results demonstrate that the modified memory pool implementation achieves the same average allocation time as the original Boost.Pool during the main phase of the run. Additionally, it significantly reduces both the average and maximum allocation times during the initial phase, effectively resolving the data loss issue.

3.1.4 Run 3 Fragment Builder Performance

Ultimately, the optimized memory pool has been used by the Run 3 SW ROD for the GBT mode Fragment Builder, effectively fixing the data loss issue and improving the overall algorithm performance, as shown in Figure 5.



(a) Single data receiving thread

(b) Multiple data receiving threads

Figure 5: Performance of the Run 3 GBT Fragment Builder

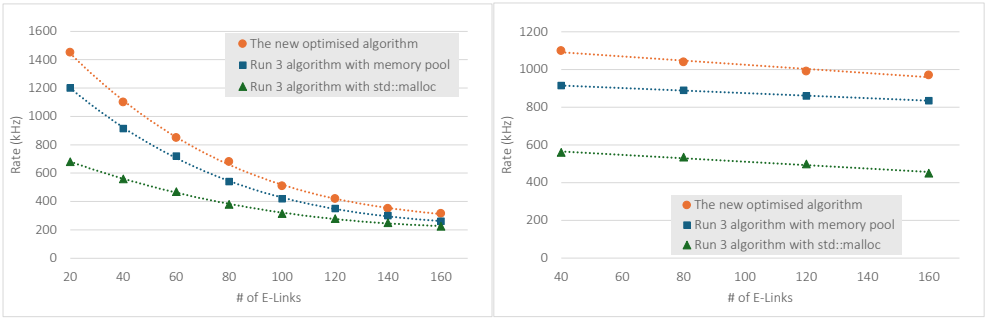
4 Readout System for Run 4

During Run 3, the new FELIX-based readout system operates alongside the legacy system, with both receiving data at a rate of 100 kHz. For Run 4, all detector systems will be upgraded to handle the increased luminosity of the High-Luminosity LHC (HL-LHC) and will transition entirely to the new readout system. This upgraded system must operate at an input rate of 1 MHz, requiring the SW ROD to meet the same functional requirements as before while accommodating a tenfold increase in input rate.

In this context, the performance advantage provided by the memory pool over the standard *std::malloc* becomes increasingly critical taking into account that memory allocation rate of the GBT Fragment Builder algorithm is proportional to both the input rate and the number of worker threads. As demonstrated in the previous chapter, the average execution time of a *std::malloc* call is approximately 550 ns, which, at a 1 MHz input rate, consumes over half of the available CPU computational budget. This observation is corroborated by the measurements presented in Figure 5a, which shows that the performance gain from the memory pool increases with the input rate.

To validate this, the same tests were repeated on a newly evaluated system as part of the requirements study for the final Run 4 Readout hardware. The test platform was equipped

with two Intel® Xeon® Gold 6444Y CPUs and 128 GB of DDR5 4800 RAM. Concurrently, a new version of the GBT Fragment Builder algorithm was tested on the same system. This updated algorithm features an optimized implementation of the *Block Collator* component, replacing the *TBB concurrent hash map* with a circular buffer. The circular buffer relies solely on atomic variables, eliminating the need for mutex locks. The results of these tests are presented in Figure 6.



(a) Single data receiving thread

(b) Multiple data receiving threads

Figure 6: Performance of the Run 3 and the New Optimized GBT Fragment Builders

The results show that, while the optimized version of the algorithm offers noticeable performance improvements, the primary performance gains were achieved through the use of the custom memory pool allocator. It's also demonstrated that the GBT Fragment Builder is approaching the capability to sustain 1 MHz input rate, while successfully aggregating data from 160 E-Links into a single ROB fragment. Furthermore, it is reasonable to assume that advancements in computer hardware over the next three years will further enhance algorithm performance, providing the necessary safety margin to reliably handle the Run 4 data rate.

5 Conclusion

The High-Luminosity Large Hadron Collider (HL-LHC), which is projected to commence operations in 2030, seeks to enhance the luminosity of the LHC by a factor of ten beyond its initial specifications. In order to facilitate this substantial upgrade, a novel readout system designed to support a tenfold increase in input rate is being developed for the ATLAS experiment as part of the Phase-II upgrade.

The first version of this system was deployed in Run 3 for the new detectors introduced during the Phase-I Upgrade. This system features a software-based component called the SW ROD, which receives collision data via the Front-End Link eXchange (FELIX) system. During preparations for Run 3, a dynamic memory management issue that impacted the new Readout system's performance was identified and resolved in the SW ROD implementation. The solution significantly reduced latency and improved the application's overall performance.

Furthermore, a new prototype of the GBT Fragment Builder algorithm of the SW ROD has been developed and tested. Performance tests conducted so far show that the updated algorithm is close to meeting the stringent requirements for Run 4. Moreover, anticipated advancements in CPU and RAM performance over the next three years should provide sufficient computational power to fully meet the demands of the HL LHC.

References

[1] ATLAS Collaboration, The ATLAS Experiment at the CERN Large Hadron Collider, JINST **3**, S08003 (2008). 10.1088/1748-0221/3/08/S08003

[2] S. Kolos, G. Crone, W.P. Vazquez, New software-based readout driver for the ATLAS experiment, IEEE Transactions on Nuclear Science **68**, 1811 (2021). 10.1109/tns.2021.3083987

[3] S. Ryu, FELIX: The new detector readout system for the ATLAS experiment, Journal of Physics: Conference Series **898**, 032057 (2017). 10.1088/1742-6596/898/3/032057

[4] P. Moreira, R. Ballabriga, S. Baron, S. Bonacini, O. Cobanoglu, F. Faccio, T. Fedorov, R. Francisco, P. Gui, P. Hartin et al., The GBT Project (2009). 10.5170/CERN-2009-006.342

[5] L. Martinelli, Atlas new small wheel performance studies after first year of operation, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **1063**, 169290 (2024). https://doi.org/10.1016/j.nima.2024.169290

[6] A.D. Robison, in *Encyclopedia of Parallel Computing*, edited by D.A. Padua (Springer, 2011), pp. 955–964, ISBN 978-0-387-09765-7

[7] S. Cleary, P.A. Bristow, Boost.Pool, https://www.boost.org/doc/libs/1_82_0/libs/pool/doc/html/index.html, accessed: 2025-01-20