EPJ Web of Conferences **295**, 02025 (2024) https://doi.org/10.1051/epjconf/202429502025
CHEP 2023

Streaming Readout and Data-Stream Processing With ERSAP

Gyurjyan Vardan[*1], *Abbott* David[1], *Goodrich* Michael[1], *Heyes* Graham[1], *Jastrzembski* Ed[1], *Lawrence* David[1], *Raydo* Benjamin[1], *Timmer* Carl[1]

[1]JLAB, CST Division, 12000 Jefferson Avenue, Newport News, Virginia, 23606, USA

Abstract. With the exponential growth in the volume and complexity of data generated at high-energy physics and nuclear physics research facilities, there is an imperative demand for innovative strategies to process this data in real or near-real-time. Given the surge in the requirement for high-performance computing, it becomes pivotal to reassess the adaptability of current data processing architectures in integrating new technologies and managing streaming data. This paper introduces the ERSAP framework, a modern solution that synergizes flow-based programming with the reactive actor model, paving the way for distributed, reactive, and high performance in data stream processing applications. Additionally, we unveil a novel algorithm focused on time-based clustering and event identification in data streams. The efficacy of this approach is further exemplified through the data-stream processing outcomes obtained from the recent beam tests of the EIC prototype calorimeter at DESY.

1 Introduction

Data stream processing poses new challenges, and new frameworks are needed to fit them better [1]. Due to the increasing demand for high-performance computing, existing data processing architectures must be reevaluated in their adaptability to adopt new technologies and process streaming data. ERSAP is a reactive actor model and FBP [2] paradigm-based framework to design a data-stream processing application for nuclear physics (NP). This framework encourages the functional decomposition of the overall data processing application into mono-functional artifacts that are easy to understand, develop, deploy, and debug. Since these artifacts or actors are programmatically independent, they can be scaled and optimized independently, which is impossible for monolithic application components. A critical advantage of this approach is fault tolerance, where independent actors can come and go in the data stream without causing the entire application to stop or crash. Furthermore, locating any faulty actor in the data pipeline is simple.

Since the actors are loosely coupled, and the data carries the context, they can run in heterogeneous environments utilizing different accelerators. The actors can be physically separated, so deploying a new set of actors does not put the whole system on hold, and the updates are less intrusive. Independent development teams can develop

[*1] Corresponding author: gurjyan@jlab.org

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

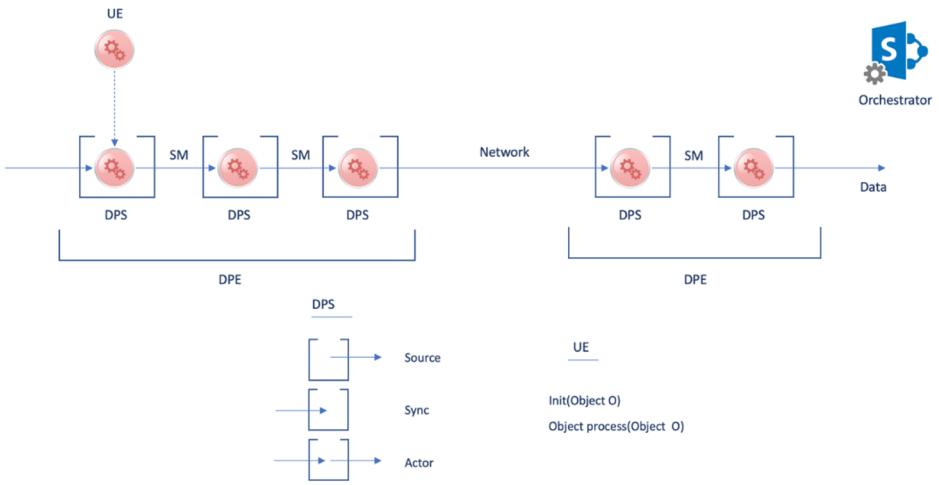

Fig. 1. ERSAP core components. DPE: Data Processing Environment, SM: Shared Memory, DPS: Data processing Station, UE: User Engine.

The actors and engines handle specific system areas with their release cycles and at their own pace. A notable feature of ERSAP is the ability to represent the whole data processing application as a data-flow graph. An important implication of the graph-like structure is the ability to reason about the entire application in a unique way that is often difficult in the case of object-oriented programming or service-oriented architectures and allows visual no-code programming [3]. With this type of programming, you can add, remove, and connect different components to get different outcomes or functionality. Also, having a visual representation of the algorithm makes it more accessible to users so they can easily create applications without writing multiple lines of code.

2 Framework

ERSAP is a flexible data-stream processing framework that allows the design and deployment of data-stream processing applications that can evolve. It is an environment that will encourage implementing new ideas and technologies while preserving the integrity of existing data pipelines. The three essential components of this framework are a reactive actor, a data-stream pipe (the communication channel between actors), and an orchestrator or the workflow manager of the application (see Fig. 1).

During operation, a stream of data-quanta will flow through a directed graph of reactive actors, where the accumulated effect of these actors defines application logic.

The fundamental difference between ERSAP and other frameworks is that instead of instructions moving across functions, the data drives, triggering the execution of actors' engines. The data-oriented nature of this environment makes actors programmatically independent. Another important design aspect is that data moves across actors through externally predefined connections. As shown in Fig. 1, some actors have inputs and outputs, and some only have inputs or outputs. Hence, ERSAP defines two types of actors: stream source/sync actors and data processing actors. However, we use the same actor abstraction to integrate user algorithms for both components. A data-stream processing application is a network of interconnected actors, and each is abstracted as a data processing station (DPS). Each station provides a run-time environment for user algorithms and handles all data communication. As a result, an engine developer is relieved of network programming, data

serialization, and I/O in general and always gets a data object as input. The engine's only requirement is to implement the data-in/data-out interface to be considered an actor in ERSAP. The station also provides a means for engine configuration and scaling, potentially freeing users from writing multi-threaded code.

3 The dataflow model

The ERSAP model represents a data processing application by a directed graph [4]. The graph nodes are actors encapsulating user data processing algorithms (also referred to as an engine within the framework's terminology). Directed arcs between the nodes represent the data dependencies between actors. Arcs that flow toward an actor are input arcs, while those that flow away are said to be output arcs from that actor. Conceptually, data flow as tokens (data quanta) along the arcs and behave like unbounded first-in, first-out queues.

When a program begins, stream source actors place data onto certain key input arcs, triggering the rest of the application. The actor is said to be active whenever an actor's specific set of input arcs (called an activation set) has data. As a result, a busy node/actor reacts to the input data quantum, performs its operation, and places a new data token on some or all of its output arcs. It then ceases execution and waits to become active again. The essential advantage is that, in a data pipeline, more than one actor can become involved at once. Thus, if several actors become active simultaneously, they can be executed in parallel. This simple principle allows massively parallel execution at the data quantum level. Another advantage is that the message-passing mechanism in the actor model makes it easy to support data computation via data messages and debugging requests via control messages. Streaming control messages in the same pipeline as data messages leads to application elasticity and fault tolerance.

4 Event identification in the streaming data

In the growing field of data analytics, density-based clustering stands out as a significant method for identifying patterns and associations in datasets. Traditional approaches, such as DBSCAN [5], have provided foundational understanding. However, with the exponential rise in real-time data streams, particularly time-series data, a need emerges for more efficient algorithms to process this data swiftly and accurately. The Adaptive Crawling Window Event Clustering (ACWEC) algorithm is introduced to address this precise need. Distinct from DBSCAN, ACWEC's design is such that it traverses a one-dimensional time-series dataset just once, ensuring a high degree of efficiency. At its core, ACWEC leverages two primary features: adaptive window size and the timestamp of every incoming event. By doing so, it can discern and group events that are temporally proximate to each other within a user-defined time frame. As data flows, for every event that enters the system, the algorithm evaluates its nature. It determines whether the event can be classified as a core event. The criteria for this classification is based on the presence of a minimum number of events (as specified by the N_n 'minimum samples' parameter) within a specific time, referred to as T_h. Expanding from these identified core events, the algorithm then includes events that can be reached within the same T_h time, forming a dense cluster.

Conversely, suppose an event neither qualifies as a core event nor can be accessed from other core events within the stipulated Th duration. In that case, it is labeled as noise and remains un-clustered. The operational dynamics of ACWEC come to the fore when the adaptive window is deployed. The right edge of this window is designed to slide and encompass events

guided by input parameters. As it moves, the algorithm constantly checks the time difference between the window's earliest and most recent events.

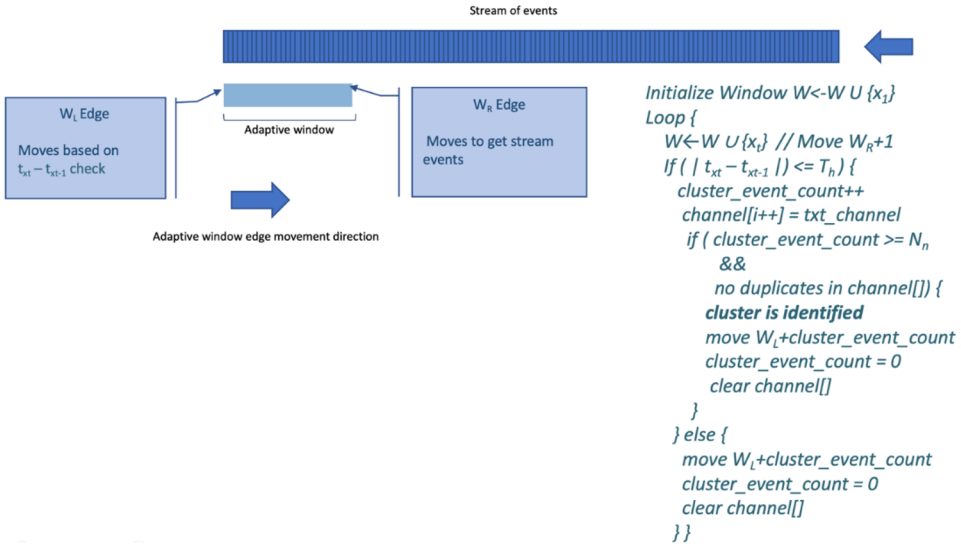

Fig. 2. ACWEC algorithm

A cluster is recognized if this time difference aligns with the predetermined window. If not, the algorithm doesn't halt but instead makes an intuitive adjustment: it shifts the left edge of the window to exclude certain events, ensuring that the time difference criterion is satisfied. In the meantime, the right edge progresses forward, continuously integrating new events. This seamless back-and-forth, characterized by adaptability, is what empowers ACWEC. Though iterative, the entire process is carried out in a single pass through the data. The outcome is a dynamic algorithm highly responsive to the ever-changing nature of data streams and adept at pinpointing event clusters with remarkable precision in real-time scenarios. The ACWEC algorithm is illustrated in Fig 2.

5 EIC prototype calorimeter ERSAP pipeline at DESY

The EIC prototype calorimeter's stream readout and processing application is a directed graph with two parallel pipelines that branch out from the root pipeline and are dedicated to decoding VXS trigger processor (VTP) [6] data frames (see Fig 3). The common source (actor) for these branches presents a set of FADC250 [7] hits per timestamp on two outgoing edges directed to destination actors that perform beam and cosmic event identification. Real-time event identification pipelines of the application include histogram actors that visualize cosmic muon and beam lepton events online.

Fig 4 portrays the process of lepton identification through the accumulation of signals emanating from crystals within the electromagnetic calorimeter (The X axis is the ADC channel, and the Y axis is the hit count). Simultaneously, the graph exhibits the linearity of the response generated by the electromagnetic calorimeter. This representation effectively demonstrates the discernment of leptons and validates the consistent behavior of the calorimeter's signal output.

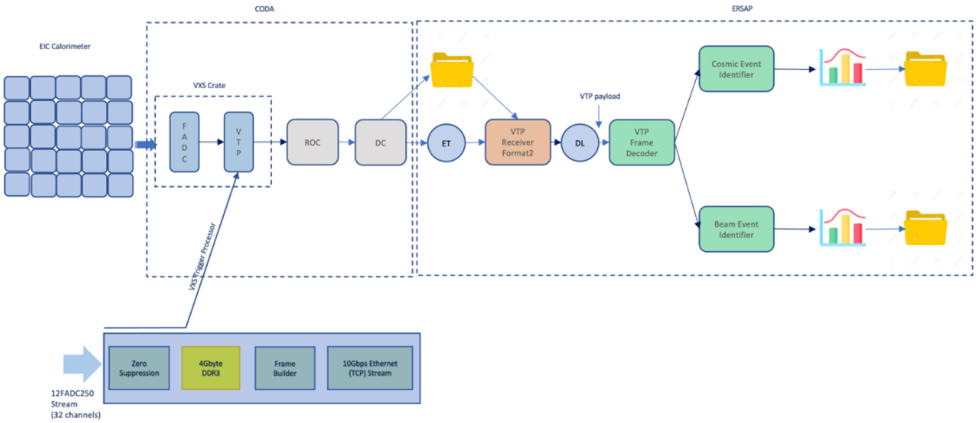

Fig 3. Data-stream processing pipeline for the EIC prototype calorimeter.

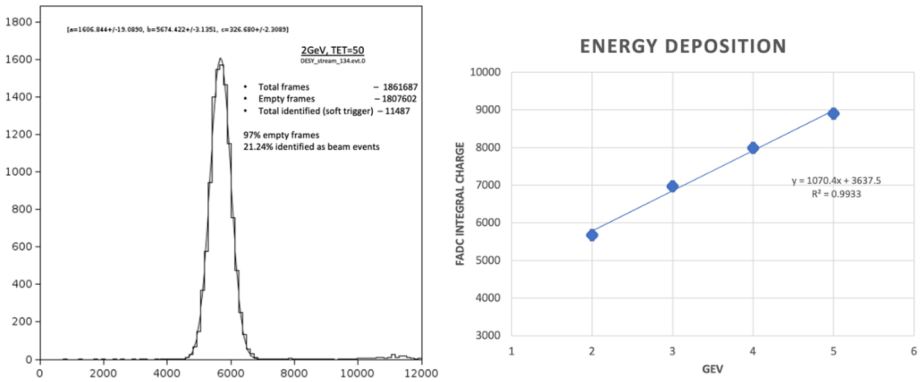

Fig 4. Energy deposition of leptons in the 5x5 calorimeter and response linearity.

6 Summary

The paper addresses the challenges posed by the growing volume and intricacy of data produced by high-energy physics and nuclear physics research facilities. The pressing need for enhanced real-time data processing is discussed, emphasizing the urgency to rethink the flexibility of current data processing architectures. The central highlight of this paper is the introduction of the ERSAP framework. ERSAP synergistically combines flow-based programming and the reactive actor model, propelling the efficiency of distributed, real-time data stream processing applications forward. Within ERSAP, the data processing application is represented as a data-flow graph, promoting visual no-code programming. This visual representation simplifies the development process, enabling users to establish applications without extensive coding. The paper also introduces the adaptive crawling window event clustering (ACWEC) algorithm. This new algorithm, designed for time-series data, efficiently clusters events based on their temporal proximity. The real-world application of this framework is demonstrated through the EIC prototype calorimeter's data-stream processing at DESY, showcasing its real-time capabilities in distinguishing cosmic muon and beam lepton events.

References

[1] Ameli, F., Battaglieri, M., Berdnikov, V.V. et al. Streaming readout for next-generation electron scattering experiments. Eur. Phys. J. Plus 137, 958 (2022). https://doi.org/10.1140/epjp/s13360-022-03146
[2] "Flow-based Programming, 2nd Edition: A New Approach to Application Development", CreateSpace Independent Publishing Platform, 2010
[3] Jasak H., Jemcov A., Tukovic Z. et al. Extended Visual Programming for Complex Parallel Pipelines in ParaView. Eurographics 2023, DOI: 10.2312/pgv.20231084
[4] Josh Fischer, Ning Wang. Grokking streaming systems. Manning publications. March 2022
[5] Combining DBSCAN and Grid-Based Clustering For Performance Analysis by Vijay Rai and LAP Lambert Academic Publishing, June 2019, ISBN 978-6200102256
[6] VTP Manual. [Online]. Available: https://coda.jlab.org/drupal/system/files/pdfs/HardwareManual/VTP/VTP-HallB-Manual.pdf
[7] FADC manual. Jefferson Lab, March 2008. [Online]. Available: https://www.jlab.org/HallB/ftof/manuals/FADC250UsersManual.pdf Accessed on November 30, 2022