ERSAP: Towards Better HEP/NP Data-Stream Analytics with Flow-Based Programming

V. Gyurjyan, D. Abbott, N. Brei, M. Goodrich, G. Heyes,
E. Jastrzembski, D. Lawrence, B. Raydo, C. Timmer

***Abstract*— This paper presents a reactive, actor-model and flow-based programming framework that we developed to design data-stream processing applications for HEP and NP. This framework encourages a functional decomposition of the overall data processing application into small mono-functional artifacts that are easy to understand, develop, deploy and debug. The fact that these artifacts (actors) are programmatically independent means they can be scaled and optimized independently which is impossible to do for components of monolithic applications. One of the advantages of this approach is fault tolerance where independent actors can come and go in the data-stream without stopping or crashing the entire application. Furthermore, it is easy to locate any faulty actor in the data pipeline. Due to the fact that actors are loosely coupled and data carries context, they can run in heterogeneous environments and utilize varied accelerators. This paper describes the main design concepts of this framework. It also presents a proof-of-concept application and the results of its processing on-beam calorimeter streaming data.**

I. Introduction

The volume and complexity of data being produced at HEP and NP research facilities has been growing exponentially, and there is an increased need for new approaches to process it in real or near real-time. Physics data processing is mentioned as one of the areas that cause most concern especially in the light of upcoming accelerator and experiment upgrades[1]. The main concern is the ability to analyze, store and understand data sets using contemporary algorithms and technologies.

Due to the increasing demand for high throughput computing, existing data processing architectures need to be reevaluated in terms of their adaptability to adopt new technologies and process streaming data. Despite success, reported by some, with using high throughput computing that incorporate heterogeneous accelerators and ML, the pace of these developments is rather slow. We believe that one of the reasons for the slow deployment of heterogeneous hardware and ML lies in the fact that the majority of these efforts are being developed on top of existing software built to fulfill unrelated goals. Data stream processing poses an entirely new set of challenges and new frameworks are needed which better fit them. In this paper, we consider flow-based programming[4] as a paradigm for developing an environment for real-time streaming, acquisition and processing (ERSAP) in order to design data-stream processing applications capable of incorporating new and future technologies, allowing natural evolution of the application and efficient data management.

II. Flow-Based Programming

The flow-based programming (FBP) paradigm was first introduced by J. Paul Morrison in the late '60s and used a "data processing factory" metaphor for designing and building applications. FBP defines applications as networks of "black box" processors or actors, communicating via exchange of data chunks (data quanta) traveling across predefined connections where the connections are specified externally to the processors. These actors can be reconnected to form different applications without having to be changed internally. FBP is innately component-oriented. Actors are coupled by data, the loosest form of coupling between software components, thus promoting a flexible component-oriented software architecture. In this architecture, data and not control flows through actors to meet the functional requirements of a system. In essence, actors are asynchronous message-driven entities, where messages are the boundary between actors that ensure loose coupling, isolation and location transparency. The naturally collaborative form of these actors, distributed, loosely bound, very resilient, and independently scalable makes them ideal for streaming architectures.

A notable feature of FBP 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 impossible in case of object oriented programming (OOP) or service oriented architectures (SOA) and allowing visual no-code programming.

III. Streaming Data Acquisition and Processing

ERSAP is a reevaluation effort at JLAB to develop a streaming readout and data processing system that will satisfy future experiments at the lab. Our goal is to build a framework that will be used for not only streaming data acquisition, but also to build data-stream processing systems: a unification effort to use a single common framework for both.

Our design of the streaming system assumes use of several tiers of data storage or data cool down stations (see Fig. 1). Each tier, except the last one, will temporarily store data with data processors between tiers reducing data volume permanently stored in the final tier. Capacity and performance of these

storage-tiers will depend on linked processors' latencies and will provide retention or cooling of the data long enough to give the processors enough time to complete their tasks.

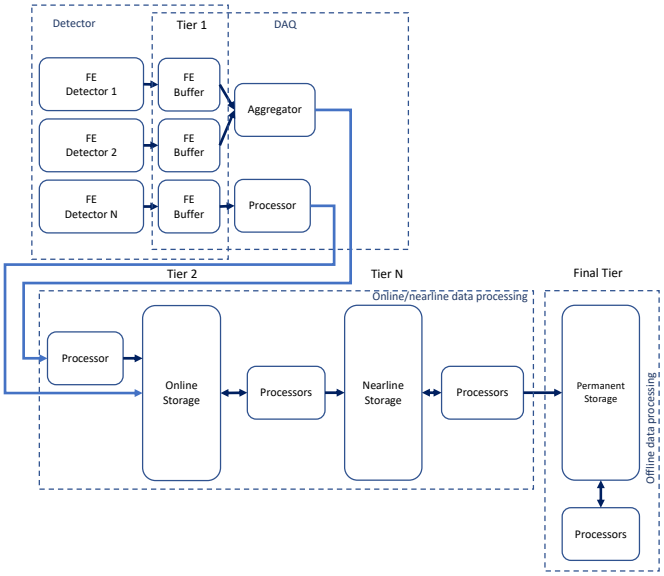

Fig. 1. Tiered data storage system

The green rectangles denote data processors or actors that harbor user defined algorithms including stream aggregators, event builders, virtual triggers, or event processors for calibration, reconstruction, etc. It is useful to note that the concepts of aggregation, building, and processing in parallel streams of data map very well onto a FBP based micro-services architecture. In such a system, well defined data types, encapsulated algorithms, and data driven system controls allow large computational tasks to be broken down into sub tasks between which data flows. This flexible model allows advanced algorithms, including ML and AI techniques, to be encapsulated in independent software services.

IV. Design Requirements and Framework Architecture

ERSAP's end product is a flexible data processing application with the ability to easily evolve over time. It is an environment that will encourage implementation of new ideas and technologies while preserving the integrity of existing data pipelines. The three basic components of this framework are: a reactive actor, a data-stream pipe (the communication channel between actors), and an orchestrator or the work flow manager of the application (see Fig. 2). Tiered memory is considered a linkable actor that provides data storage and retrieval functions. During operation, a stream of data-quanta will flow through a directed graph of reactive micro-services, where the accumulated effect of these actors define application logic.
The basic difference between ERSAP and other frameworks is that instead of instructions moving to actors or functions, the data moves, triggering the execution of actors on it. This makes actors programmatically independent. Another important design aspect is that data quanta messages are exchanged across externally predefined connections.

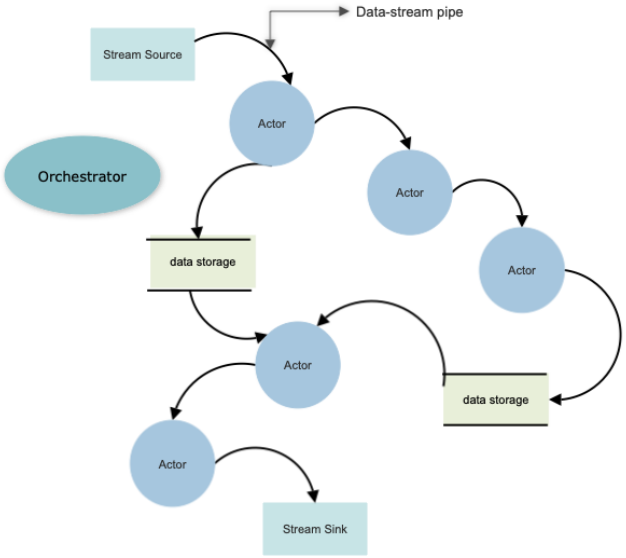

Fig. 2 ERSAP basic components.

As you can see from the Fig. 2, there are actors that have both inputs and outputs and there are actors that have only inputs or only outputs. Hence, ERSAP defines two types of actors, namely stream source/sync actors, and data processing actors. However, we use the same actor abstraction to integrate user algorithms for both types of components.
ERSAP has a 3-layer structure (Fig. 3). The lowest, transport layer defines transient metadata and transports messages containing meta-data and data (data-stream pipe). The transport layer is data format agnostic and will transport any data assuming proper serialization and de-serialization routines are provided by the user. Currently we use ZMQ and a home grown shared memory library (simple off heap memory where data is accessed by actors) for transports. Using POSIX shared memory and an in-memory distributed data grid are in development.

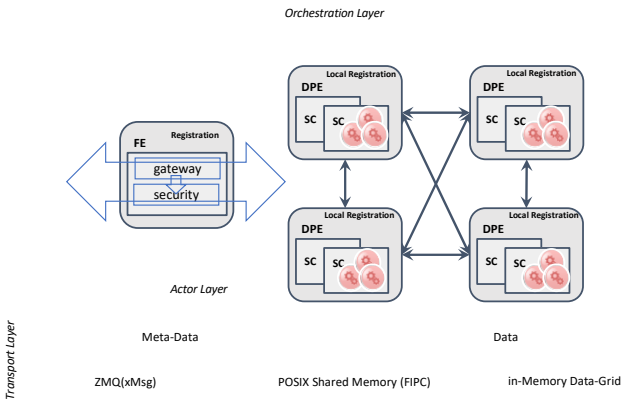

Fig.3 ERSAP structure.

The middle or actor layer is a repository of actors that can be discovered and registered using ERSAP's registration and discovery service. Actors are deployed in data processing environments (DPE) where there is a single DPE/per technology on a single node. Actors are further grouped logically within containers in the DPE. Logical grouping allows separation of parallel running domains within the same DPE. In addition, ERSAP provides multiple normative actors that, for example, take care of data encryption and control access to actors for security reasons. The third layer is the orchestration layer where ERSAP based data-processing application orchestrators reside.

V. Framework Components

A. Data processing station

A data-stream processing application is a network of interconnected actors, each abstracted as a data processing station. Each station provides for a running a user algorithm (engine) in a run-time environment and handles all data communication. As a result, a user engine is relieved of network programming, data serialization and I/O in general and always gets a data object as its input. The engine's only requirement is that it must 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, and potentially frees a user from writing multi-threaded code (see Fig. 4).

B. Data-stream pipe

The data-stream pipe provides asynchronous publish/subscribe and point-to-point communications as well as defining a data structure for transport.

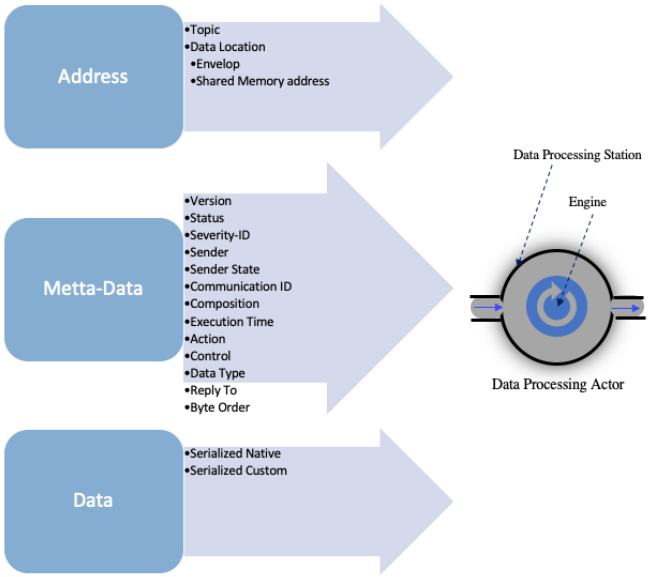

Fig.4 ERSAP transient data envelope.

Note that ERSAP data transport is agnostic to user provided data formats, and will transfer any data, assuming proper serialization and de-serialization routines are provided for custom user data. However, the user can also use the native data format of the framework (currently proto-buff or flat-buffers which support primitive types and arrays of primitive types) to represent the results of their data processing. For the native data format the framework takes care of all serialization and de-serialization. Figure 4 illustrates the transient data envelope including a data address section, a meta-data section and an actual serialized data section. To minimize or completely avoid data copying and serialization, the framework uses built-in shared memory. If using shared memory, the actual serialized data section of the transient data envelope will be empty as a result of orchestrator optimizations which, in turn, are based on the actor deployment mechanism.

C. Orchestrator

The main disadvantage of FBP systems is that they can become a very complex directed graph of actors distributed across a network of heterogeneous hardware and software structures. In this case, the role of the application orchestration becomes critical. We gave considerable effort in designing a data-quantum level workflow orchestrator as well as an API allowing domain experts to design their own workflow orchestration systems. First, the orchestrator is responsible for locating user engines and presenting them as actors and deploying them based on a domain expert's requirements. A domain expert defines the application graph or composition using a graphical UI or simple YAML file.

At run-time, an orchestrator listens and reports each actor's performance, status and errors. It is the contact point for users trying to design or expand a data-stream processing application by providing access to the actors' registry and discovery services. Most importantly it optimizes stream quanta communication by optimizing actor deployment to minimize data copying, serialization, and deserialization. It will prefer deployment of the micro-services to take advantage in-process or intra-process shared memory infrastructures. It is able to adapt in order to guarantee optimum performance, and minimize underutilization of resources in a shared heterogeneous cluster.

D. The dataflow model

In the ERSAP model, a data processing application is represented by a directed graph. The nodes of the graph are actors encapsulating user data processing algorithms (engines). Directed arcs between the nodes represent the data dependencies between actors. Conceptually, data flow as tokens (data quanta) along the arcs and behave like unbounded first-in, first-out (FIFO) queues. Arcs that flow toward an actor are said to be input arcs to that actor, while those that flow away are said to be output arcs from that actor. When a program begins, stream source actors place data onto certain key input arcs,

triggering the rest of the application. Whenever a specific set of input arcs of an actor (called a activation set) has data in it, the actor is said to be active. As a result, an active node/actor reacts on 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 key advantage is that, in dataflow, more than one actor can become active at once. Thus, if several actors become active at the same time, they can be executed in parallel. This simple principle provides the potential for massive parallel execution at the data quantum level. Another advantage is that the message-passing mechanism in the actor model makes it easy to support both 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.

VI. Proof of Concept

An ERSAP based application was designed to perform an on-beam test of the Electron Ion Collider (EIC) prototype calorimeter, reading its streaming data output and processing that in real-time. The 3 × 3 PbWO4 matrix was placed downstream of a secondary electron/positron beam generated by the primary photon beam in JLAB's Hall-D. Leptons were identified and their energy measured by a pair spectrometer (PS) tagger. The prototype was installed in Hall-D downstream of the PS, as shown in the figure 5.

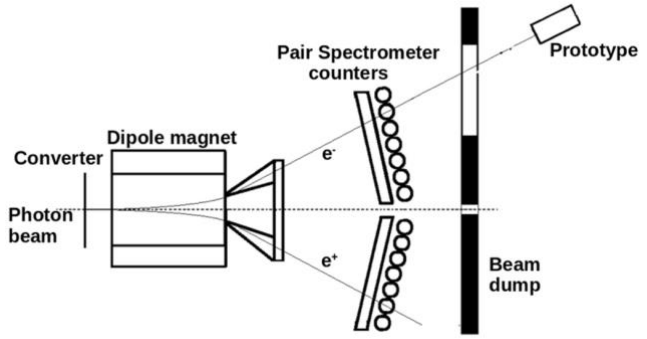

Fig. 5 EIC prototype calorimeter beam-test setup.

The flash ADC board (FADC250, 250 MHz sampling rate, 16 channels) designed at JLAB was used t**o** readout the prototype calorimeter. The firmware of the ADC has been adapted for streaming readout by utilizing the VXS serial links, which were previously used for trigger outputs. The FADC250 firmware detects pulses using a software-defined threshold. When a threshold crossing is found, the pulses are integrated in a programmable-sized window, a pedestal is subtracted, and a gain is applied. The result is a calibrated charge and time for the found pulse, which is sent over the VXS interface to the next stage, a VXS Trigger Processor (VTP) module (see Fig. 6). When running in this mode, the FADC250 supports up to 30 MHz of hits per channel.

The VTP board is another custom JLAB design that is used in conjunction with FADC250 modules when fast readout is needed. This is a fairly generic and flexible module (with several firmware implementations to match the requirements of different experiments) whose conversion to streaming readout was a simple and natural extension. The resources on the VTP provide reasonable serial connectivity between the VXS and the optical links, as well as significant buffering capability.

A. Application design

Figure 6 shows the EIC prototype calorimeter stream readout and processing application design. It is represented as a directed graph with 3 parallel pipelines branching out from the root pipeline dedicated to decode VTP data frames. The common actor (source vertex) for these branches presents a set of FADC250 hits per timestamp on three outgoing edges directed to destination actors that perform beam and cosmic event identification, and persists data for the future off-line data analysis. Real-time event identification pipelines of the application include histogram actors that visualize cosmic and beam events online (see Fig 7).

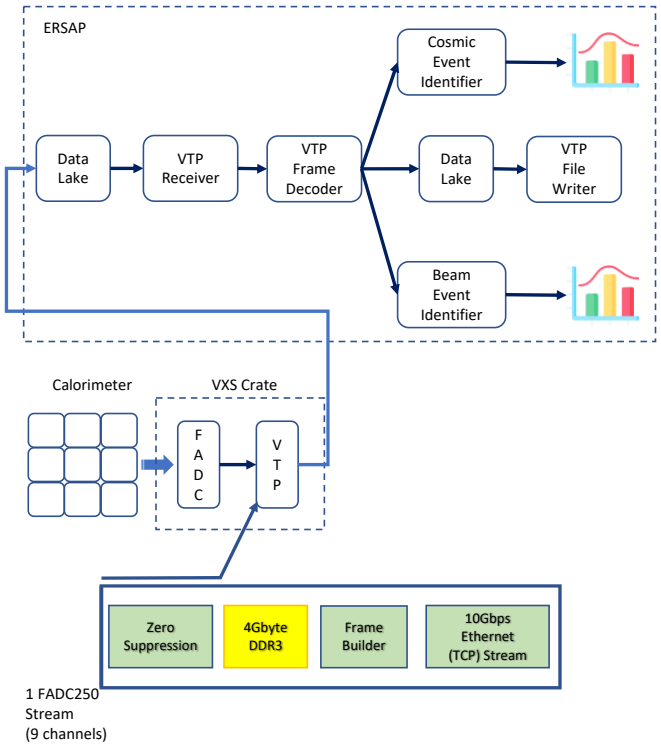

Fig. 6 EIC prototype calorimeter data-stream processing application.

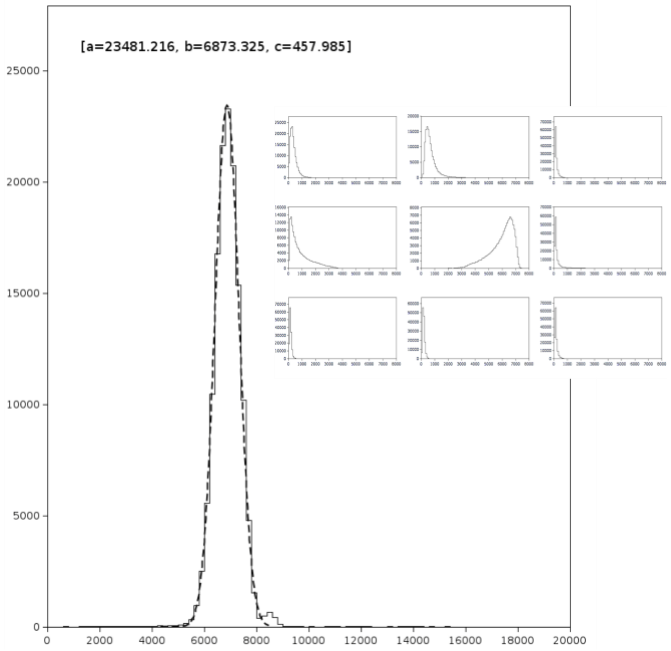

Fig. 7 Real-time beam lepton identification and energy deposition in the calorimeter crystals.

VII. Conclusion

We developed a reactive, actor-model and FBP paradigm framework to design a data-stream processing application for NP. This framework encourages functional decomposition of the overall data processing application into small mono-functional artifacts that are easy to understand, develop, deploy and debug. Due to the fact that these artifacts or actors are programmatically independent, they can be scaled and optimized independently which is impossible to do for components of monolithic applications. One important 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, it becomes simple to locate any faulty actor in the data pipeline. Due the fact that 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 from each other, so the deployment of a new set of actors does not put the whole system on hold, and the updates are less intrusive. The development of the actors and engines handling specific areas of the system can be performed by independent development teams with their own release cycles and at their own pace.

References

[1] "A Roadmap for HEP Software an Computing R&D for the 2020s", arXiv:1712.06982v3 [physics.comp-ph] 11 Feb 2018

[2] " JANA2 Framework for Event Based and Triggerless Data processing", EPJ Web of Conferences 245, 01022, 2020

[3] "CLARA: A Contemporary Approach to Physics Data Processing", 2011, J. Phys.: Conf. Ser. 331 032013 doi:10.1088/1742-6596/331/3/032013

[4] "Flow-based Programming, 2nd Edition: A New Approach to Application Development", CreateSpace Independent Publishing Platform, 2010