otsdaq: An Innovative Framework for Efficient Data Acquisition and Analysis

Brenda Najjuma
Wilbur Wright College - City Colleges of Chicago
(Dated: August 10, 2023)

ABSTRACT

otsdaq is an advanced data acquisition and analysis framework designed to optimize data processing and streamline scientific research processes. This research paper provides a comprehensive overview of the system's architecture, features, and its applications in diverse fields. The otsdaq system offers a scalable, high-throughput, and user-friendly platform for handling vast volumes of data and extracting meaningful insights, thereby revolutionizing the way data is acquired and analyzed in modern research.

INTRODUCTION

Otsdaq is significant in facilitating data-driven research. It presents the motivation behind its development, outlining the challenges faced by traditional data acquisition methods and the need for a more efficient and streamlined approach. Otsdaq is significant in facilitating data-driven research. It presents the motivation behind its development, outlining the challenges faced by traditional data acquisition methods and the need for a more efficient and streamlined approach. Otsdaq is an acronym for "off-the-shelf data acquisition". It is a data-acquisition (DAQ) solution that is ready to use and designed for the test beam, detector development, and other rapid deployment settings. The Mu2e experiment makes use of this online DAQ software framework. Otsdaq comes as a web page, offering a web interface for configuring, controlling, and monitoring online DAQ software entities. Over time, upgrades, bug fixes, and the development of new features are necessary to enhance the design and functionality of certain aspects of otsdaq's graphical user interface.

ONLINE DAQ SOFTWARE DEVELOPMENT

C++ is used on the server side. Through plugins (C++ classes deriving from the relevant class), developer code is added. Here is the user online DAQ software plugin categories: Front-end interfaces are the programs used to interact with external devices, such as the plugins that are typically available for each type of FPGA readout card. Trigger modules, online monitor modules, and artdaq fragment generators are examples of art modules that use code to decode data and transfer it to artdaq event builders. Code for custom data handling in data processors, such as data stream-to-ROOT for Visualizer Configuration table handlers are programs that handle configuration data in a certain way, for as by outputting FHiCL or by offering auxiliary functions like getVolume() for objects whose sizes are determined by configuration parameters. HTML and JavaScript are used on the web side of the system. Web apps are created by adding developer code through .html files (together with the necessary .js and .css files). If you wanted to overlay the calorimeter FPGA temperature color-coded on a 3-D model of the detector with slider controls to define thresholds, this would be a customized user web application. There are examples and ots JavaScript libraries that control functionality, like altering the configuration, reading from slow controls, or running a series of front-end reads and writes.

ARCHITECTURE AND COMPONENTS

For its implementation, otsdaq employs the artdaq DAQ framework, which provides flexibility and scalability to address increasing DAQ needs. For the experiment, otsdaq incorporates technologies from the artdaq toolkit and serves as the bond that allows for a consistent experience for all users. The Fermilab Scientific Computing Division developed otsdaq and artdaq, and developments are divided into two sections: the server side and the web side. otsdaq offers a library of front-end boards and firmware modules that implement a custom UDP protocol. In addition, an integrated Run Control graphical user interface (GUI), and readout software that is preconfigured to communicate with otsdaq firmware are included. Front-end interfaces - code that communicates with an external device, such as the Data Transfer Controller (DTC) and each type of Readout Controllers (ROC). Front-end interfaces are plugins that define how to connect to a device that is not connected to otsdaq. The executables that start when otsdaq is launched are those that are enabled in the configuration tree for that node. The executables' children are then instantiated depending on the specifications specified by the selected configuration alias when the state machine later moves to the Configured state. A configuration tree that fully defines the online DAQ configuration is represented by a configuration alias

WEB DESKTOP

The ots web desktop environment serves as your entry point to all of otsdaq's potential. Here are the desktop features in brief:

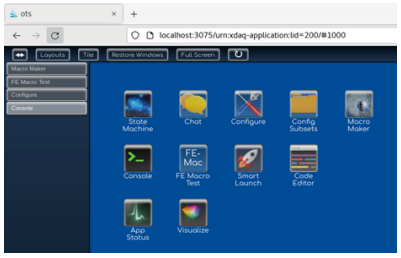

FIG. 1. Homepage of the Graphical User Interface (GUI) of otsdaq with all the available icons.

- Same user across different monitors, computers, and browser tabs.
- Access permissions-enabled configurable desktop window icons and folders.
- Window manipulation including tiling, resizing, moving, minimizing, maximizing, refreshing, and closing are available.
- Global and individual user presets for window layout.

STATE MACHINE

Each of the entities that follow the state machine has a defined behavior according to the states and transitions. Otsdaq supports several state machines, each of which has a set of configuration aliases and user access authorization settings. At any given time, only one state machine can be in the configured state. It is possible to operate separate otsdaq instances of partitioned state machines in parallel (could be shared read-only configuration).

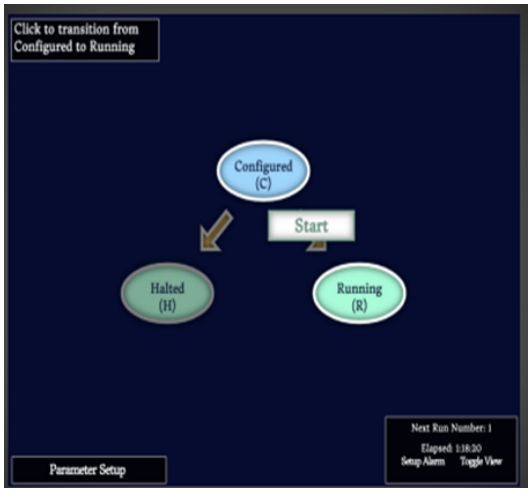

FIG. 2. Image of the State machine after being configured.

What transpires during system transitions?

1. Starting up. This creates executable applications.
2. Initializing. This resets application properties.
3. Configuring. After loading the configuration tree based on the configuration alias parameter, all plugins are instantiated and the configuration sequence for each entity in the system is run according to priority. Slow controls are then activated (polling threads are started).
4. Starting. Write startup registers, front ends go live in priority order as data processing begins (files/sockets are opened, buffers are instantiated).
5. Pausing. Front ends become inactive, and data processing pauses (threads may become idle, files/sockets may not be closed).
6. Resuming. Data processing resumes, and front ends become active.
7. Stopping. Front ends become inactive, and data processing stops (threads are killed, files are closed).
8. Halting. Deallocating plugins.
9. Shutting down. Exit executable applications.
10. Soft erroring. When a plugin throws a soft-error exception, for example, all entities pause until the exception is resolved, at which point experts might try to recover and resume the run.
11. Erroring. Asynchronous error conditions cause all entities to reach a failed state (for example, a plugin throwing an exception due to an error).

What occurs systemically when in each state?

1. Initial: This one is in idle mode and can interact with other users, monitor EPICS, change configuration settings, and more.
2. Halted: Some application properties have been set up.
3. Configured: This is the same as the initial state, but now additionally polling/reading slow controls data and publishing to EPICS.
4. Running: Data processing is active (files/sockets are in use, buffers are filling and emptying), and front ends are running.
5. Paused: Front ends are inactive, and data processing is suspended (idle files/sockets, threads sleep).
6. Shutdown: Idle except for the Gateway application, for example, cannot monitor EPICS.
7. Failed: This one is the same as Halted.

DATA PROCESSING

The online DAQ's main function is data processing. Event data can be processed using artdaq modules or data processor plugins. Plugins for data processors can extend the capability of generic data handlers by adding their own specialized handling. One feature of the ots visualization tools, for instance, uses specialized data processor plugins to produce ROOT objects that can be viewed on the web desktop. Users are free to create ots data processor plugins for whatever idea they have. Users have access to artdaq's versatility and scalability when they utilize the Artdaq data processor plugin in otsdaq. An artdaq Board Reader with a Fragment Generator plugin is created by the artdaq data processor plugin. Additionally, the user can instantiate artdaq Event Builders, Dispatchers, and Data Loggers depending on how the online DAQ system is configured. As an illustration, Mu2e will have a Board Reader for each DTC, one Tracker/Calorimeter Event Builder per server (each running the trigger algorithm with as many art analyzer processes as fit on the server, approximately 20), a second-level Event Builder which will integrate CRV data, several Data Loggers on dedicated nodes for writing data to online storage, and several Dispatchers to provide real-time data quality monitoring. A Data Processor Producer/Consumer creates ROOT objects that are displayed in a Visualizer after being forwarded by Monitors that subscribe to a Dispatcher. The user can simultaneously send a subset of metrics to EPICS, all of them to Ganglia, and only the most crucial ones to a file. Artdaq tracks a vast number of metrics that essentially cover everything regarding event rate and data flow.

DATA STORAGE AND MANAGEMENT

FRONT-END INTERFACES

Front-end interfaces are thought to be the specifics for how to interface to a device not part of otsdaq (i.e., C++ to write and read). For instance, during development, a detector readout software emulator or a front-end interface plugin may connect to an FPGA card for detector readout. Here are a few examples of front-end interface plugins.

MACRO MAKER

A tool called Macro Maker enables users to create macros by executing front-end interface writes and reads and building sequences of the writes and reads. Users can save macros individually or make them accessible to all users. Early development and low-level front-end interface debugging can both benefit from the use of macro maker. A target plugin or C++ can be used to export macros directly as FE Macros.

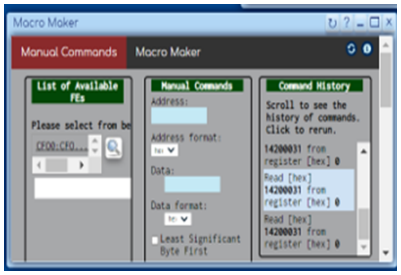

FIG. 3. Image of the Macro Maker

FE MACROS

A front-end interface plugin class's C++ member functions are known as FE Macros. The main benefit is that FE Macros are easily accessible through the web interface using either the FE Macro Test web app or custom user web apps. In the FE Macro Test web app, the input and output arguments (strings or numbers) for FE Macros are displayed on the right. Generic private and public macros from Macro Maker are also supported by the FE Macro Test online application.

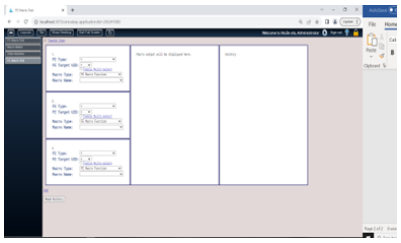

FIG. 4. The FE Macro Test layout with the output and history sections.

MACRO MAKER MODE

The idea behind Macro Maker mode is that anyone (such as a firmware developer) can utilize this streamlined mode without keeping track of configuration changes or using the state machine if they only wish to use front-end interface plugins with FE Macros or generic macros. The configuration is imported through the usage of an FHiCL parameter file. The state machine immediately moves to the Configured state when Macro Maker mode is activated.

DATA ANALYSIS AND VISUALIZATION

Visualizer: The visualizer web application allows users to navigate both live and stored ROOT objects. With

refresh rate as an option, the web application can display several ROOT objects in various desktop windows. It is possible to store and load pre-made views. Additionally, the visualizer web app has a 2-D and 3-D mode that communicates with the visualizer server-side app using a unique ots protocol (although suffers from some bit rot). Users of the 3-D display in their browser can rotate and fly through it.

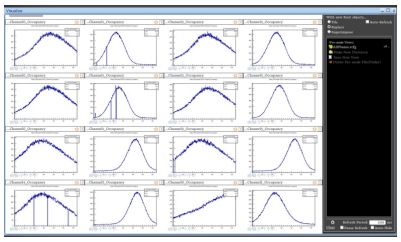

FIG. 5. The web application displaying several ROOT objects.

SCALABILITY AND PERFORMANCE

Console: Another ots utility that eliminates the need to enter the Linux terminal is the console web app, which enables users to exist remotely. The foundational features of the console are based on the Artdaq message facility. Messages can be filtered, and user preferences are maintained per user. They also contain labels, line numbers, and severity. The ots output macros can be used to generate printouts from any user plugin code to the terminal, log files, or web console.

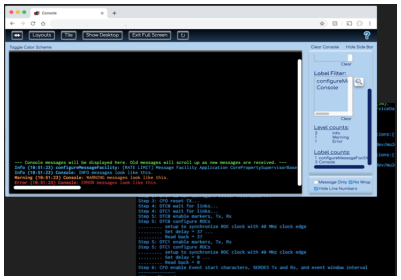

FIG. 6. The console web app.

CONCLUSIONS

oIn conclusion, otsdaq represents a cutting-edge framework revolutionizing data acquisition and analysis in research. Its adaptable architecture, encompassing C++ and web-based components, enables seamless integration and customization. With versatile plugins, otsdaq empowers researchers to optimize data handling and visualization. This innovation propels scientific exploration towards efficient, impactful insights in a rapidly evolving landscape.

By addressing the limitations of traditional data acquisition methods, otsdaq underscores its significance as a catalyst for data-driven research. Its ability to seamlessly integrate with various scientific applications, such as the Mu2e experiment, demonstrates its versatility and adaptability in diverse settings.

The utilization of plugins, employing C++ classes derived from base categories, facilitates dynamic code integration, ensuring flexibility and extensibility for developers.

The internship aimed to comprehend the system, grasp HTML, and enhance the GUI, contributing to otsdaq's continuous evolution.

With dedicated emphasis, a notable portion of the internship time was directed toward the ambitious objective of creating an all-encompassing history within the FE macro test plugin. Despite encountering challenges that resulted in a partial outcome, this experience yielded valuable insights into the intricacies of otsdaq's graphical user interface.

ACKNOWLEDGEMENTS

1. Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

2. This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers, and Scientists (WDTS) under the Community College Internships Program (CCI).

3. I would like to express my gratitude to my supervisor, Micol Rigatti, for teaching me such exceptional problem-solving techniques. Furthermore, she always asks us to come along with her so we may view the projects she is working on and pick up new skills.

REFERENCES

1. Rigatti, M.,2020, Analysis, design, and test of an HW/SW electronic system for data acquisition and control of the Mu2e Fermilab Experiment, University of Pisa, p. 46-113.

2. Rivera, R.A., Flumerfelt, E., 2023, otsdaq Software Features for Mu2e, Workshop; Fermi National Accelerator Laboratory, p.1-62.