# Streaming DAQ software prototype at J-PARC hadron experimental facility

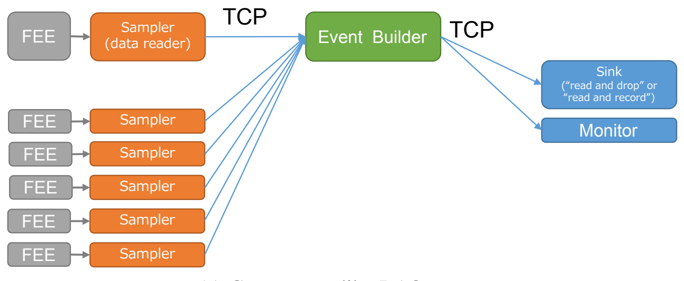
Tmonori Takahashi, Ryotaro Honda, Youichi Igarashi, Hiroshi Sendai

***Abstract*—In recent years, particle physics and nuclear physics experiments require faster data collection systems and advanced trigger systems as the beam intensity increases. The current DAQ system at the J-PARC hadron experimental facility (HEF) uses a fixed- and low-latency trigger with dedicated hardware to reduce data and event-building software that merges data into a single endpoint. This conventional DAQ system is expected to be inadequate for the requirement of future experiments and has to be replaced with a new simplified but powerful one, called streaming DAQ, which collects the whole detector signals and filters them by software running on many computers. Therefore, we have developed a prototype of streaming DAQ software for the J-PARC HEF using FairMQ and Redis as middleware. The key features are that it is simple and has a low learning cost to develop and operate by a small number of people. The software can be used not only in a fully streaming readout system but also in the triggered DAQ and the combined DAQ of the hardware and software trigger.**

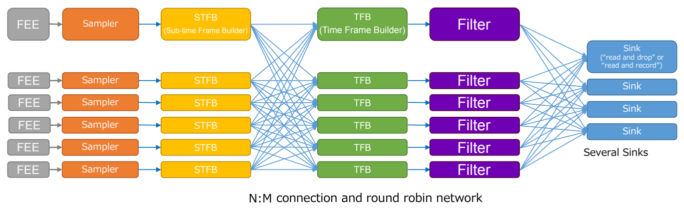
***Index Terms*—Control systems, Data acquisition systems.**

## I. Introduction

THE J-PARC Hadron Experimental Facility (HEF) was constructed to conduct various particle and nuclear physics experiments using secondary particles produced by the intense proton beam and started operation in 2009. Recently, experiments at the HEF require faster data collection and advanced trigger systems as the beam intensity increases. The current data acquisition (DAQ) system at the HEF [1] was developed more than 10 years ago for small- to medium-scale experiments. The DAQ system uses event selection with fixed- and low-latency triggers generated by NIM logic modules and an FPGA module to reduce the amount of data to be recorded. The event data selected by the trigger are collected on a single computer over a TPC/IP network as shown in Fig. 1a. However, the current system has limitations for future experiments with high event and data rates such as E16 [2] and E50 [3] experiments. For example, it is challenging for the limited number of people in the experimental group responsible for DAQ to develop trigger circuits and logic in hardware to achieve the complex selections required in future experiments that use many detector signals (e.g., more than thousands) for triggering. Furthermore, the configuration of only one event builder process in the system causes data



(a) Current tree-like DAQ system.


FEE
Sampler
STFB (Sub-time Frame Builder)
TFB (Time Frame Builder)
Filter
Sink ("read and drop" or "read and record")
Sink
Several Sinks
STFB
TFB
N:M connection and round robin network

(b) Multiple endpoint DAQ.

Fig. 1: Schematic views of the DAQ topology used in the current experiments (a) and required for future experiments (b).

collection to slow to about 1 GB/sec. Therefore, to solve these problems, it is necessary to design and develop DAQ software that reads detector data in a self-triggered manner and performs event selection by 100 to 1000 processes with load balancing software (Fig. 1b). In addition, the software must be simple and have low learning costs so that it can be quickly developed and operated by a small number of people when developing an application for each experiment.

## II. The prototype architecture

The software prototype is developed in C++ language using FairMQ [4], [5] and Redis [6] (Fig. 2). FairMQ is a toolkit developed at GSI for large-scale data processing and is an implementation of a parallel processing workflow with an actor model. The core building block of FairMQ is called FairMQDevice, which is the base class for implementing user code. FairMQDevice has a finite state machine and provides methods to implement the configurations of data paths and a user task and task execution. Each process in the DAQ system corresponds to one FairMQDevice and performs a user-defined task. FairMQ also provides an abstract communication utility called FairMQChannel or FairMQSocket, which wraps messaging libraries such as ZeroMQ [8]. ZeroMQ has many advantages in DAQ applications, for example, broker-less, lightweight, and low latency. It supports advanced

Manuscript received August 22, 2022. This work was supported by JSPS KAKENHI Grant Number JP20K04005.

T. N. Takahashi is with Nishina Center for Accelerator-basd Science, RIKEN, Wako, Saitama 351-0198, Japan (e-mail: tomonori@riken.jp).

R. Honda, Y. Igarashi, and H. Sendai are with Institute of Particle and Nuclear Studies, High Energy Accelerator Research Organization, Tsukuba, Ibaraki 305-0801, Japan (e-mail: rhonda@post.kek.jp; yoichi.igarashi@kek.jp; hiroshi.sendai@kek.jp ).

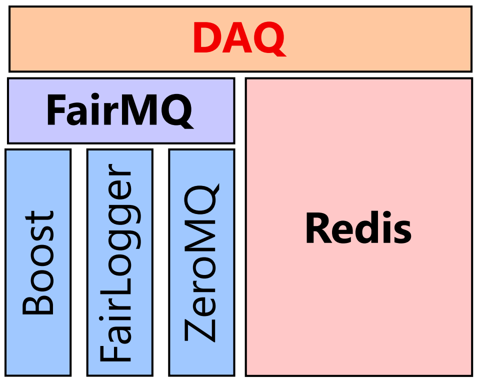


Fig. 2: DAQ software stack.

communication patterns such as publish/subscribe (Pub/Sub), request-reply, push-pull pipeline, and exclusive pair compared to direct use of the raw sockets. In addition, FairMQ imposes no restrictions on the data format of messages, allowing for freedom in how to use it.

The configuration, control, and monitoring of the FairMQDevice are handled by a dynamic library called Plugin, which is loaded at runtime. To operate multiple FairMQDevices (e.g., ∼100 k user processes), external libraries called DDS (dynamic deployment system) [9], [10], and its extension ODC (Online Device Control) [11] are available for FairMQ. In this work, however, simpler, Redis-based Plugins have been developed to implement web-based user interfaces. Redis is an open-source in-memory type data structure server that is fast. Redis can handle various data types: strings (i.e., simple key-value data), lists, hash maps, and sets. Furthermore, Redis also functions as a message queue broker and can be used for DAQ command distribution. It also offers flexibility in extending the user interface for each experimental group or application, as it has libraries for many programming languages.

Fig. 3 shows the typical internal block diagram of each DAQ process of the software prototype. Three types of Plugins have been developed.

## A. DAQ service Plugin

The DAQ service Plugin provides the user interface for controlling DAQ states. It receives state transition and state check commands, which are formatted in JSON structure and distributed from a controller process to the DAQ process through the Redis pub/sub channel. It returns the current state to the controller process. The controller process can be any redis client and can be operated from the command line with redis-cli or via a web graphical user interface, as shown in Fig. 4. A simple web-based controller with an HTTP server and WebSocket using the boost library is available.

This Plugin also has the function of service discovery using Redis. The Plugin registers a variable called *presence* with a timer (time-to-live, TTL) in Redis. The timer is periodically updated to prevent the timeout from occurring. When the timer expires, Redis notifies the controller and other subscribers that the expiration occurs via the Redis Pub/Sub channel, a lightweight and low latency messaging protocol. In addition,

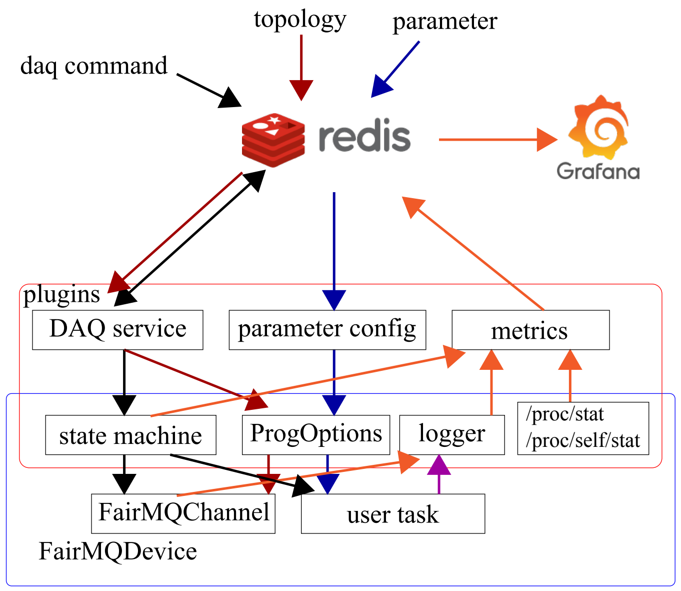


Fig. 3: Overview of the Redis-based DAQ user interface for a FairMQ processes.

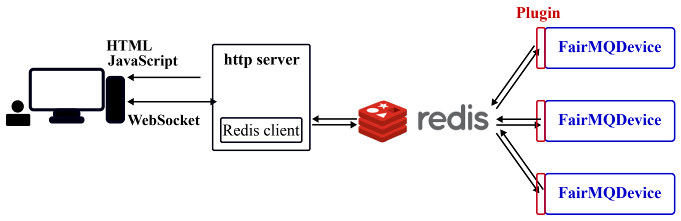
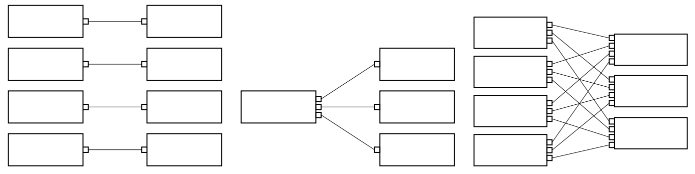


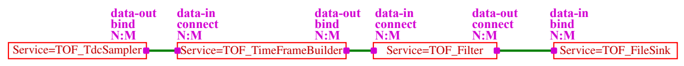
Fig. 4: Controller UI using Redis Pub/Sub channel.

the Plugin registers process information to Redis: unique ID in DAQ system, process group ID, IP address, and properties of FairMQChannels. FairMQChannel has properties to be configured, such as communication pattern (PUB-SUB, PUSH-PULL, REQ-REP, PAIR), socket type (bind or connect), and the socket address. These properties determine the communication topology between FairMQDevices. Fig. 5 shows examples of topology diagrams commonly used in a DAQ system. Fig. 5a shows the case where pipelines processing independent data are configured in parallel. Fig. 5b shows an example of load balancing with a static round robin. The load balancing functionality built into ZeroMQ's push-pull pipeline does not allow strict control of data destinations. Connecting sockets with individual addresses make it possible to control the destination. Fig. 5c shows an example of a topology configuration for aggregating data fragments called time frames (or time slices) or event-building with load balancing. The topology of the entire DAQ system is composed of the combination of the partial diagrams described above. However, if the sockets are configured one by one, the effort required to set the whole topology will grow rapidly as the number of sockets increases. Therefore, instead of describing the connection between individual sockets, the software prototype primarily describes the connection between process groups called a *service*. Then, the connection settings between individual sockets are generated from topology hints

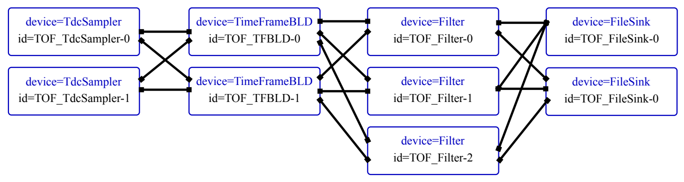


(a) 1-1. (b) 1-N (N-1). (c) N-M.

Fig. 5: Examples of partial connection diagram in DAQ system.



(a) Service discovery input



(b) Service discovery ouptut

Fig. 6: Example of service autodiscovery.

such as 1-1, 1-N, N-1, or N-N.

Fig. 6 shows an example of the service autodiscovery. It is assumed that a workflow with four different process groups is executed:

- Sampler, reading and buffering data from frontend electronics, then dividing the data into a certain time slice
- Time-frame builder, merging the data fragments in the same time slice
- Filter, performing data reduction
- Sink, storing or monitoring the analysis results

Fig. 6a illustrates the input parameters for the topology setting. They are six settings of service input/output endpoints (one for sampler output, two for time frame builder input and output, two for filter input and output, and two for sink input) and three settings of connections between services (sampler and time frame builder, time frame builder and filter, and filter and sink). It corresponds to nine lines of the redis-cli command. When processes of each service start, they find the connection partners based on the information registered in the service registry. Fig. 6b shows an example of the results of automatic service detection. The user does not need to change configuration parameters even if the number of processes increases.

### *B. Parameter Config Plugin*

The parameter config Plugin provides an interface to set program options via Redis. Data of strings, lists, hash maps, or sets registered in Redis are passed to the application. The plugin checks Redis at each state transition and updates the application parameters.

### *C. Metrics Plugin*

The metrics Plugin checks the DAQ state, collects input/output message transfer volume, message transfer rate, and process statistics such as CPU load and memory usage, and sends them to Redis. These measurements are taken at the rate-logging cycle of the FairMQDevice (e.g., 1-second intervals) and passed to the data monitoring tool Grafana [12] via Redis. Grafana [12] displays the state of the DAQ process and visualizes time series data using the time series extension of Redis.

## III. Performance test

Two rack-mount servers were used for deployments of the prototype software. Each server had 24 cores of Intel Xeon Gold 6126 (2.6 GHz CPU), 64 GB memory, a network card with 10-gigabit interface, and Scientific Linux 7 OS. The two servers were connected across a 10 GbE switch. In this test, three tasks (sampler, worker, and sink) were connected in a 1-N-1 topology and two different process deployments

- On the PC. All processes including Redis run on one host (Fig. 7a).
- Via network. The workers run on a different host from the others (Fig. 7b).

and two types of workloads

- Without load. Workers do nothing with the received data.
- With load. Workers shuffle the received payload for 1 msec as the dummy load.

were evaluated.

In the measurements, the size of event fragment data generated by each sampler was fixed at 50 KB.

## IV. Test results

The measured throughput is shown in Fig. 8. The lines with cross and asterisk markers are measured without workload, while the formers correspond to cases where the data transfer is closed to the same host. It was found that the network card's upper limit of 10 GbE was reached when there was no workload, and performance of 10 Gbps or higher was also obtained when the same host was used.

The lines marked with squares represent measurements with the dummy workload. The results scaled linearly with the number of CPUs both for the same host and for different hosts.

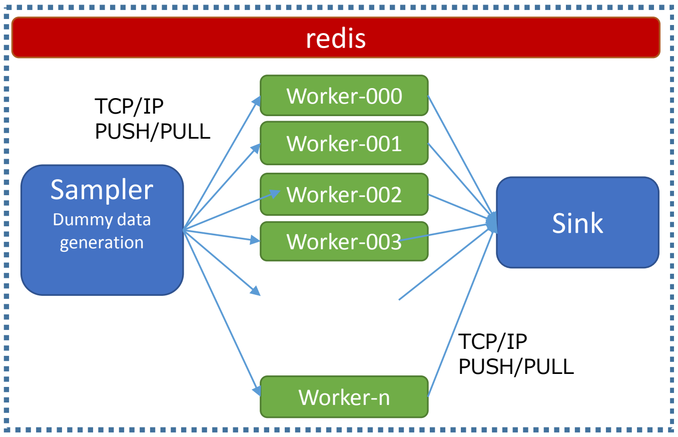
Also, we confirmed that the Redis-based controller was able to manage at least 700 processes.

## V. Applications

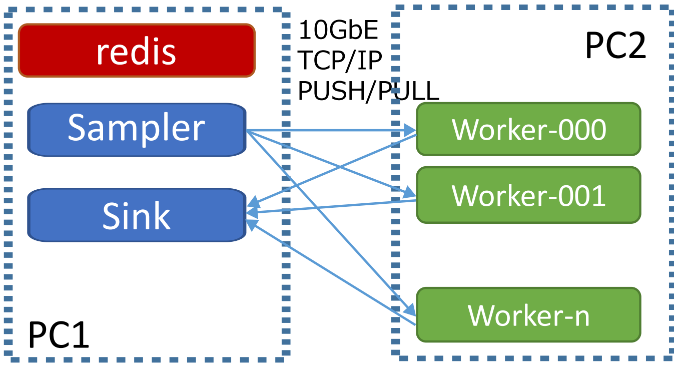
At least two experiments at J-PARC HEF adopts the software prototype and are developing their applications.

### *A. J-PARC E16*

J-PARC E16 experiment investigates the origin of the hadron mass through the systematic measurement of the vector meson mass [2]. E16 is upgrading the readout system of the silicon strip detectors from a triggered DAQ using APV25 to a streaming DAQ using SMX2 [14]. E16 will use the prototype software to combine the existing triggered DAQ [13] and a streaming DAQ for SMX2 and start a first physics run in 2023.



(a) On the PC



(b) Via network

Fig. 7: Process deployments used in the throughput evaluation.

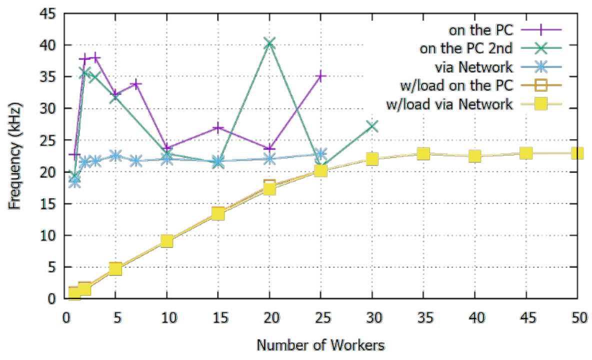


Fig. 8: Measured performance of the prototype software with 1-N-1 topology.

## B. J-PARC E50

J-PARC E50 experiment intends to study a diquark correlation through the charmed baryon spectroscopy [3]. E50 has to read approximately 25,000 channels of TDC under the high-intensity of 30 MHz $\pi^-$ beam with the reaction rate of 1.5 MHz. Momentum reconstruction and particle identification with most readout channels are required in the online analysis to reduce the raw data of 10–20 GB/spill by 100. E50 employs a fully streaming DAQ based on the prototype software to avoid the hardware trigger's development cost and utilize the flexibility of the software filtering. E50 has so far developed circuits for streaming readout and demonstrated small-scale DAQ [15]. The development of an efficient filtering algorithm is underway.

## VI. Summary

The software prototype of streaming DAQ software has been developed for high event and data rate experiments at J-PARC HEF. The software is written in C++ and employs FairMQ as the task executor of the DAQ workflow and Redis as the control, monitor and configuration interface. The topology of the DAQ system with multiple endpoints is configurable via the service autodiscovery using Redis. Test results show that the prototype has scalability up to the network bandwidth of at least 10 GbE. Also, the controller can manage at least 700 processes.

The performance evaluation with many nodes and realistic loads is in progress. In addition, we are developing applications for J-PARC E16 and E50 experiments. The prototype software will be used in the physics run of E16 or the commissioning run of E50 from 2023.

## References

[1] Y. Igarashi, *et al.*, "An Integrated Data Acquisition System for J-PARC Hadron Experiments," *IEEE Trans. Nucl. Sci.*, vol.57, no.2, pp. 618–624, 2010. [Online] Available: https://doi.org/10.1109/TNS.2009.2037959 Accessed on: August 22, 2022.

[2] S. Yokkaichi, *et al.*, "Electron pair spectrometer at the J-PARC 50-GeV PS to explore the chiral symmetry in QCD," J-PARC E16 proposal, 2006. [Online] Available: http://j-parc.jp/researcher/Hadron/en/pac_0606/pdf/p16-Yokkaichi_2.pdf Accessed on: August 22, 2022.

[3] H. Noumi, *et al.*, "Charmed Baryon Spectroscopy via the $(\pi, D^{*-})$ reaction," J-PARC E50 proposal, 2013. [Online] Available: http://j-parc.jp/researcher/Hadron/en/pac_1301/pdf/P50_2012-19.pdf Accessed on: August 22, 2022.

[4] M. Al-Turany, *et al.*, "Extending the FairRoot framework to allow for simulation and reconstruction of free streaming data," *J. Phys.: Conf. Ser.*, vol.513, p. 022001, 2014. https://doi.org/10.1088/1742-6596/513/2/022001 Accessed on: August 22, 2022.

[5] FairMQ, https://github.com/FairRootGroup/FairMQ Accessed on: August 22, 2022.

[6] Redis, https://redis.io Accessed on: August 22, 2022.

[7] C. Hewitt, P. Bishop, R. Steiger, "A Universal Modular ACTOR Formalism for Artificial Intelligence, in Proceedings of the 3rd International Joint Conference on Artificial Intelligence," IJCAI'73, Proceedings of the 3rd international joint conference on Artificial intelligence, pp. 235–245, 1973. https://www.ijcai.org/Proceedings/73/Papers/027B.pdf Accessed on: August 22, 2022.

[8] ZeroMQ, https://zeromq.org Accessed on: August 22, 2022.

[9] A. Lebedev and A. Manafov, "DDS: The Dynamic Deployment System," *EPJ Web Conf.*, vol.214, 01011, 2019. https://doi.org/10.1051/epjconf/201921401011 Accessed on: August 22, 2022.

[10] DDS, https://github.com/FairRootGroup/DDS Accessed on: August 22, 2022.
[11] ODC, https://github.com/FairRootGroup/ODC Accessed on: August 22, 2022.
[12] Grafana, https://grafana.com/ Accessed on: August 22, 2022.
[13] T. N. Takahashi, *et al.*, "Data acquisition system in the first commissioning run of the J-PARC E16 experiment," *IEEE Trans. Nucl. Sci.*, vol.68, no.8, pp.1907–1911, 2021. [Online] Available: https://doi.rog/10.1109/TNS.2021.3087635 Accessed on: August 22, 2022.
[14] K. Kasinski, *et al.*, "Characterization of the STS/MUCH-XYTER2, a 128-channel time and amplitude measurement IC for gas and silicon microstrip sensors," *Nucl. Instr. and Meth. A* vol.908, pp. 225–235, 2018. [Online] Available: https://doi.org/10.1016/j.nima.2018.08.076 Accessed on: August 22, 2022.
[15] R. Honda, *et al.*, "Continuous timing measurement using a data streaming DAQ system," *Prog. Theor. Exp. Phys.*, vol.2021, no.12, 123H01. [Online] Available: https://doi.org/10.1093/ptep/ptab128 Accessed on: August 22, 2022.