ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUPV013

BACKEND EVENT BUILDER SOFTWARE DESIGN FOR INO mini-ICAL SYSTEM

Mahesh Punna[1], Narshima Ayyagiri[1], Janhavi Avadhoot Deshpande[1], Preetha Nair[1] , Padmini Sridharan[1],Shikha Srivastava[1], Satyanarayana Bheesette[2], Yuvaraj Elangovan[2], Gobinda Majumder[2], Nagaraj Panyam[2]

[1]BARC, Mumbai, India
[2] TIFR, Mumbai, India

Abstract

The Indian-based Neutrino Observatory collaboration has proposed to build a 50 KT magnetized Iron Calorimeter (ICAL) detector to study atmospheric neutrinos. The paper describes the design of back-end event builder for Mini-ICAL, which is a first prototype version of ICAL and consists of 20Resistive Plate Chamber (RPC) detectors. The RPCs push the event and monitoring data using a multi-tier network technology to the event builder which carries out event building, event track display, data quality monitoring and data archival functions. The software has been designed for high performance and scalability[chronous data acquisition and lockless concurrent data structures. Data storage mechanisms like ROOT, Berkeley DB, Binary and Protocol Buffers were studied for performance and suitability. Server data push module designed using publish-subscribe pattern allowed transport & remote client implementation technology agnostic. Event Builder has been deployed at mini-ICAL with a throughput of 3MBps. Since the software modules have been designed for scalability, they can be easily adapted for the next prototype E-ICAL with 320 RPCs to have sustained data rate of 200MBps.

INTRODUCTION

The Indian-based Neutrino Observatory (INO) collaboration has proposed to build a 50 KT magnetized Iron Calorimeter (ICAL) detector to study atmospheric neutrinos and to make precision measurements of the neutrino oscillation parameters. The detector will look for muon neutrino induced charged current interactions using magnetized iron as the target mass and around 28,800 Resistive Plate Chambers (RPCs) as sensitive detector elements [1]. The mini-Iron Calorimeter (mini-ICAL) detector, a prototype of the ICAL detector is being set up at the Inter Institutional Centre for High Energy Physics' (IICHEP) transit campus at Madurai. The mini-ICAL detector has 20 glass Resistive Plate Chamber (RPC), which act as sensors and are stacked in between 11 iron plates of 4 metre x 4 metre size. The iron plates are magnetised by passing electricity through copper coils wound around. This is expected to serve the purpose of understanding the engineering issues in constructing the main ICAL, and at the same time provide important inputs on the ICAL's operating parameters and physics measurement capabilities. E-ICAL with 320 RPCs is planned to be setup in Madurai, India. Max throughput expected for E-ICAL is around 200MBps with 10% hit rate and 10k trigger rate.

SYSTEM OVERVIEW

The system consists of several sub-systems: RPC DAQs, Backend Data Acquisition System (BDAQ), Trigger System, Calibration System (CAU), Magnet System, Gas System, and LV/HV System as shown in Fig. 1. Description of each system is beyond the scope of the paper [2].

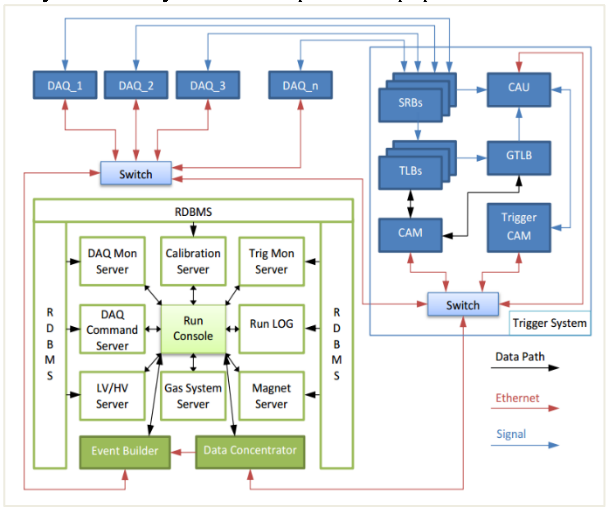


Figure 1: System overview.

Neutrino interacts with the iron plates along its line of travel, triggering events in several RPCs along its path. Orthogonal strip channels (X&Y) on RPCs pick up the charged particles, which are produced from the interaction of neutrino with iron plates. RPC-DAQ modules are connected in hybrid network topology to backend system. Trigger System detects events of interest and notifies RPC-DAQs to transmit event data event data which consists of strips hit and timing information over TCP socket to the designated Data Concentrator (DC) node. Data Concentrator nodes collect event data packets from all the triggered RPC-DAQs and assigns the timestamp and Event Number to the data packet. The updated RPC-DAQ data packets from the data concentrators are pushed to event builder

Backend Data Acquisition System (BDAQ)

The BDAQ system as shown in Fig 2. comprises of several subsystems that are intended to acquire event data and monitor data from the RPC-DAQs. The system also provides event building, event display, data quality monitoring, data archival mechanisms and run manager. BDAQ is a distributed system consisting of several subsystems; Data Concentrator, Event Builder (EB), Run Manager, Data

Quality Monitoring, Data Visualization, Long term data archival. The scope of the paper is limited to the development of Event Builder module.

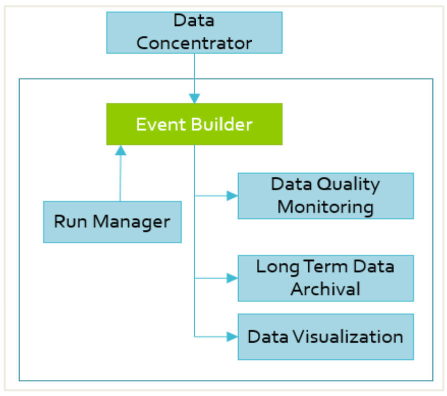


Figure 2: Backend Data Acquisition system.

EVENT BUILDER

Each muon interaction triggers several RPC events; Event builder node is responsible for collating the individual RPC event data packets based on the event number and storing the built-event collection in the required data format. The main functional requirements of Event Builder software include:

- Event data acquisition from Data Concentrator;
- Monitoring of data from RPC DAQs;
- Event Collation from the RPC event data;
- Local data archival in the selected data format;
- Pushing the collated event data to remote consoles;
- Online muon track visualization.

Software Architecture

The software has been developed following layered architecture as illustrated in Fig. 3. Communication layer implements Asynchronous TCP/IP module for receiving variable length RPC data from Data Concentrator, Data push module for forwarding the complete built event to the other remote nodes. The event building layer consists of Data Serializer that parses the data bytes to create valid message object which is binned into a concurrent data structure based on the event number. A high-performance lockless data structure has been used for event collation.

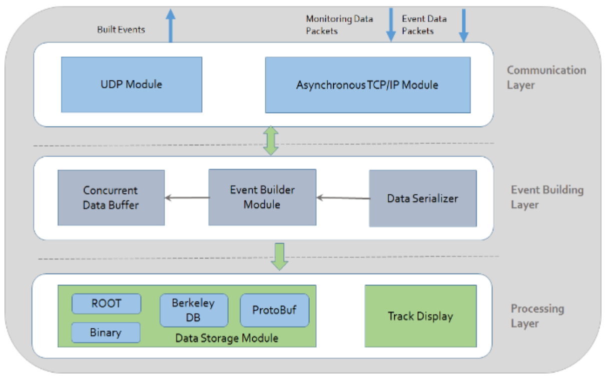


Figure 3: Software Architecture.

The collated event is published to Server push module and processing layer. Processing layer handles the storing the event in the required data format and displaying the track information. Data storage module implements different storage schemes; ROOT, binary, Berkeley DB, ProtoBuf and XML.

Core Modules of EventBuilder

Data acquisition module Data acquisition provides high performance, catering to the system data throughput requirements. The module is optimized to eliminate any possible memory allocation overheads.

Event building Event building algorithm is crucial part of the software. Event building module collates the complete event from out of order RPC data. The module design implements the necessary concurrency control mechanism to handle multi-threaded data acquisition.

Backend data store The collated event is processed by back end store, archiving the data locally. Data storage module write throughput should satisfy required system throughput, otherwise it would create back pressure in acquisition pipeline.

Server push module Server data push module publishes the data to the interested subscriber nodes like data quality monitor consoles and data visualization consoles.

Data Acquisition Module

In general, there are several IO models followed for developing network application, which are broadly divided into thread based and event based models.

Threaded Server Creating a thread per connection with blocking IO calls is the simplest solution for server application. This multi-threaded approach provides concurrent processing of requests. This approach is not scalable due to limited CPU/Memory resources

Synchronous non-blocking IO consists of a single threaded event loop waiting for the readiness of the multiple subscribed socket handles, and triggered socket events will be synchronously dispatched to the corresponding event handler. It is based on synchronous event demultiplexing mechanism. Although this approach can handle more number of sockets than thread-per-connection, but still this is not scalable.

Asynchronous IO is based on asynchronous event demultiplexing. The operations are initiated asynchronously by the application and they run to completion within the I/O subsystem of the OS. Once the asynchronous operation is initiated, the thread that started the operation becomes available. The completed events are inserted into completion event queue and Asynchronous event de-multiplexer removes the completed events and returns to the caller by invoking the corresponding completion handler. Concurrent Asynchronous Event Demultiplexer can improve the performance by making use of thread pool for demultiplex and dispatch completion handlers concurrently [3] [4][5].

The data acquisition module has been developed using multi-threaded *async* IO and is a state full TCP server that keeps track of all the clients connected to it. To reduce memory allocation/deallocation overhead, the server has a pool of *SocketAsyncEventArgs*, which reuses the same object for every receive. Buffer Manager handles allocation with dynamic resizing keeping a pool data buffers. The complete message received is propagated to the next module in the pipeline through event trigger mechanism. The module has been implemented using F#. *Async* workflow in F# helped to improve code readability avoiding traditional call back approach. The use of functional features like Active Patterns and Pattern Matching improved expressiveness.

Event Building Module

Event building is a multi-producer single consumer problem, where many RPC message events are being pushed simultaneously. The main responsibilities of the module include;

- Out of order individual RPC events need to be collated to form the complete event.
- Having a provision for accommodating the delayed events if any.
- Collated events propagation to the event processor module should be serialized based on the event number

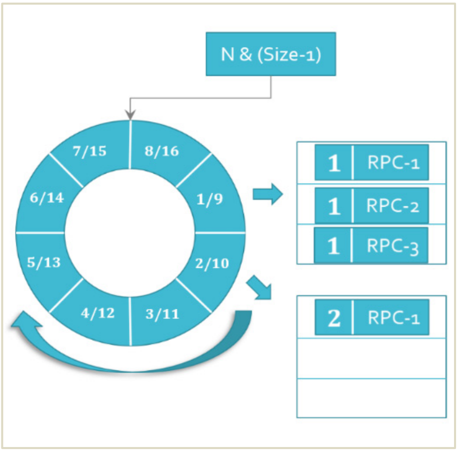


Figure 4: LMAX Disruptor based ring buffer.

Event building module has been developed following LMAX disruptor algorithm [6] as shown in Fig. 4. Disruptor algorithm is based on a bounded ring buffer that allows multiple producer threads without using locking. Each item in ring buffer can hold a list of RPC events, so the ring buffer is a collection of collections. Putting the messages into the buffer is a 2-phase process. First the producer needs to claim a slot in the ring buffer. The sequence/slot number to claim is calculated from the event number, which gives access to a slot in the buffer and the event data is inserted into RPC collection based on RPC ID. All the events with the same event number are collected in the same event slot. The events occupy corresponding entry within a RPC collection based on RPCID. Hence, it does not require locking.

The event slot will be open for event collection until the buffer has no available slot or a timeout occurs. This gives provision for accommodating any delayed events. This event collation process continues for remaining events. A new event claims an occupied slot by publishing the event occupying the slot and updating the cursor variable. Cursor variable tracks the next slot number available for event processing. The event processor handler will be notified which copies the published event data. The algorithm ensures the collated events are propagated to the event processor in order based on the event number. The size of buffer and timeout is calculated based on event trigger rate

Backend Datastore Module

The following are the desirable features of backend data store

- Efficient data access, particularly data write speeds need to match with the max system throughput, as it would create back pressure in the system pipeline
- Cater to large sets of data to suit physics runs which continue for several hours
- Lightweight and embeddable to prevent inter process communication overhead
- Efficient object to event byte stream encoding to save disk space
- Columnar data access/vertical storage techniques that enables retrieving only the selected fields from event record without loading the complete event object
- Forward and backward message format version compatibility as the schema is likely to change over the time, but the archived old format data should still be readable
- Provide extensive querying and data analysis features

Data storage schemes The following data storage schemes were studied and implemented in the software.

- Binary Serialization: Event object is serialized to data byte stream including meta data describing the class. This is the simplest solution [7][8].
- XML: XML Serialization converts object public fields and properties to readable xml stream, which creates a verbose xml file [9].
- BerkeleyDB: This is a light weight embeddable data management library. The memory footprint is just 300KB, but can manage the databases up to 256TB in size. It is a NoSQL database based on key-value pairs. It runs in the address space of the application, hence no communication overhead and all the data access methods are defined through function-call interface [10]
- Google Protocol Buffers(ProtoBuf): It produces faster and smaller byte stream as schema is defined external to the data. It is based on base128 varint encoding. It can support data versioning [11]
- ROOT: CERN developed ROOT framework [12] has been used for event data archiving in mini-ICAL system. It is an Object-Oriented framework for large scale data handling applications. It provides efficient data storage and access system designed to support huge structured data sets. The main reason for choosing ROOT as a backend is its edge in analysis and visualization features.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUPV013

Server Data Push Module

There are other remote consoles that require the built events for track visualization and data quality monitoring. The server proactively pushes the data to the interested nodes without clients polling for the data. This supports implementation of remote consoles in a technology agnostic manner; browser based, web applications, desktop application or mobile application depending on the usage. This is met with the following design approach as illustrated in Fig. 5.

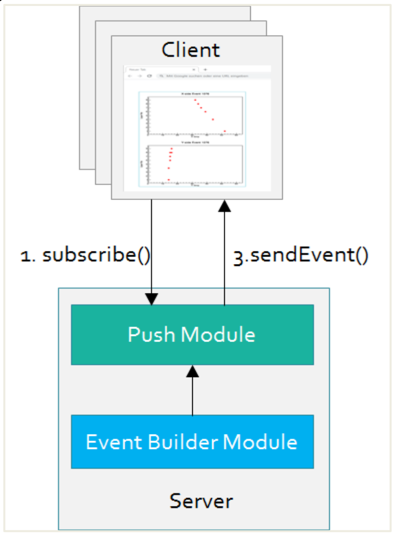


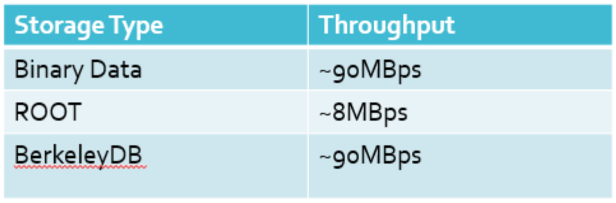
Figure 5: Server data push module.

The remote consoles get subscribed to the back-end server publisher module for various types of events and the server manages the clients into groups based on the subscription type. When the data is ready, the push module will publish it to all the subscribed nodes.

Instead of developing the transport mechanisms for HTTP/TCP from scratch, open source server push technology SignalR [13] is considered. SignalR creates persistent connection between client and server and pushes the server content to clients instantly. It is an abstraction over http and TCP protocols. It supports the following transport techniques for handling real-time communication; http transports websockets [14], SSE, Long polling. Out of which, websockets provide bidirectional persistent communication. if websockets is not available, SignalR falls back to next transport method, based on the capabilities of the server and client

PERFORMANCE

Event builder with all the integrated modules (acquisition, data serialization, event building, event store modules) has been tested for performance with various data storage types. The test setup consists of multiple DC simulator clients sending data to EB server on 1 Gbps Ethernet link. It was observed binary data and BerkeleyDB write throughput was considerably better than ROOT as shown in Fig. 6. With the designed ROOT TTree structure, the write speed observed was around 8MBps due to the write overheads (file meta data). With mini-ICAL, the maximum throughput requirement is around 3MBps. However, due to efficient columnar data access and data visualization, ROOT has been used for data archiving.



Storage Type	Throughput
Binary Data	~90MBps
ROOT	~8MBps
BerkeleyDB	~90MBps

Figure 6: Performance.

TESTING & INSTALLATION

The software was functionally tested with Data concentrator simulator & Front End Electronics (FE) simulator at BARC lab with all the modules integrated to verify the throughput and data validity. The software was installed at IICHEP, Madurai for mini-ICAL. It has been recording data since 2017.

CONCLUSION

The modules have been developed with scalability as design concern. Asynchronous IO based data acquisition module is scalable with the increasing the number of connections. Disruptor algorithm based event buffer provides lock-less data structure for event building. The optimized data structure can be scaled to work for E-ICAL by tuning the buffer parameters.

Non-blocking multi-threaded networking and event building module has been tested up-to 90MBps on 1Giga bit Ethernet network. The required network throughput (200MBps for E-ICAL) can be achieved by upgrading the hardware resources (like 10 Giga bit network and SSD drives). For improving the data write speeds, multi-threaded file writing, multi-level data writing and high performance NoSQL databases like BerkeleyDB will be explored.

ACKNOWLEDGEMENTS

Authors would like to thank the colleagues from TIFR, Mumbai for their suggestions and guidance.

REFERENCES

[1] INO Project Report, vol.1, 2006. `http://www.ino.tifr.res.in/ino/openReports/INOReport.pdf`

[2] Gobinda Majumdar, Suryanaraya Mondal, "Design, construction and performance of magnetised mini-ICAL detector module", *in Proc. 39th International Conference on High Energy Physics (ICHEP2018)*, Seoul, Korea, Jul. 2018, Paper ICHEP2018_360, doi: `10.22323/1.340.0360`

[3] `http://www.flounder.com/asynchexplorer.htm`

[4] Irfan Pyarali, Tim Harrison, Douglas C. Schmidt, Thomas D. Jordan, "Proactor; An Object Behavioral Pattern for Demultiplexingand Dispatching Handlers for Asynchronous Events", in *Proc. 4th annual Pattern Languages of Programming Conference*, Allerton Park, Illinois, USA, Sep. 1997.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUPV013

[5] https://docs.microsoft.com/en-us/windows/win32/fileio/i-o-completion-ports

[6] Martin Thompson, Dave Farley, Michael Barker, Patricia Gee, Andrew Stewart, "Disruptor: High performance alternative to bounded queues for exchanging data between concurrent threads", May 2011, http://lmax-exchange.github.io/disruptor/files/Disruptor-1.0.pdf

[7] https://docs.microsoft.com/en-us/dotnet/standard/serialization/

[8] https://docs.microsoft.com/en-us/dotnet/standard/serialization/binary-serialization

[9] https://docs.microsoft.com/en-us/dotnet/standard/serialization/xml-and-soap-serialization

[10] Michael A. Olson, Keith Bostic, Margo Seltzer, "Berkeley DB", Proceedings of the FREENIX Track, 1999 USENIX Annual Technical Conference, Monterey, California, USA, June 6–11, 1999.

[11] https://developers.google.com/protocol-buffers

[12] https://root.cern/

[13] https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-3.1

[14] V. Pimentel, G. Nickerson, "Communicating and Displaying Real-Time Data with WebSocket," in IEEE Internet Computing, vol. 16, no. 4, pp. 45-53, July-Aug. 2012, doi:10.1109/MIC.2012.64