A Lightweight, Efficient Data Aggregator for SNO+

Andy Mastbaum*
University of Pennsylvania

March 6, 2012

1 Introduction

In the SNO+ experiment, data from 9728 PMT channels spanning 19 electronics crates, trigger information, run- and event-level headers, and several digitized trigger sums must be aggregated into complete events and written to disk very quickly. Event data sent from the XL3s and trigger system (via the SBC/ECPU) are not synchronous and not necessarily ordered, and subject to quirks of front-end electronics. The system that collates this information is known as the *event builder*.

The new XL3s perform much of the role of the DAQ system in SNO: they push available data to a listening server over TCP/IP. This configuration makes it possible for an event builder to receive data directly from the crates; in a sense, the XL3s *are* the DAQ. Introducing an intermediate system to flow data through is unnecessary and presents a potential bottleneck and additional failure point.

In light of this, a new event builder has been designed and written that will listen for incoming "raw" data over TCP/IP, buffer events until all associated data has arrived, and send complete events both to disk in a "packed" format and to a "dispatcher" which will send data to various monitoring stations. The builder also accepts data from the ORCA data acquisition system. This paper provides a detailed technical description of this event builder.

*mastbaum@hep.upenn.edu

2 Data Structure

The event builder must handle several types of data:

Event Data Event data is collected for each global trigger in SNO+, and consists of trigger (MTC/D) data, digitized trigger waveforms, event metadata (nhits, run id, etc.), and up to 9728 PMT bundles (packed raw PMT data).

Run-Level Headers Run-level headers are produced at the start of a run and apply to all the run's events. There are three types of run-level headers in SNO+: run headers (start time, run id, etc.), CAAC headers (AV position), and CAST headers (manipulator/calibration source position).

Event-Level Headers Event-level headers may be produced at any time, and apply to all following events, until superceded by another event-level header of the same type. There exist two types of event-level headers: TRIG (MTC metadata) and EPED (specific to pedestal runs).

Data is stored in the event builder in a highly structured way, in order to minimize costly list iteration. The internal storage of the builder consists of three random-access ring buffers, one for each of detector events, run-level headers, and event-level headers.

For example, the detector event buffer is structured as follows:

Buffer Ring buffer

Event Built event

PMTBundle 3 words of PMT data
PMTBundle 3 words of PMT data
... × 9728
MTCData MTCD/TUBII data
CAENData Digitizer data

Event Built event

... × buffer length

Given the (stored) offset between global trigger ID (GTID) and buffer index, data arriving from an XL3 may be slotted into the proper place without any looping. For example, the ith packet representing PMT data associated with GTID 42 is stored in `Buffer[42].PMTBundle[i]`. With this model, it is impossible to detect whether a second PMTBundle has arrived for this GTID/PMTID – an occasional problem with front-end electronics – without costly looping. Hence, that is left for the RAT input producer or data cleaning cuts to handle.

The internal data structures are identical to the detector event, run-level header, and event-level header structure defined in RAT's DS::PackedEvent.

2.1 Memory Management

The amount of of memory required to buffer a large number of SNO+ events is nontrivial, and much care must be taken to ensure that memory is managed in a thread-safe, fast, and space-efficient way. This event builder was optimized for fast throughput rather than memory efficiency, although tradeoffs remain possible.

The event builder must buffer relatively few run- and event-level headers, and so storage requirements for that data is negligible. The sizes of detector event data structures are as follows:

PMTBundle 3 32-bit integers = 96 bits

MTCData 192 bits + unknown, small contribution from TUBII

CAENData 8 channels × 100 samples × 16-bit short = 12.8k bits

An event with 9728 hits thus consumes at least 120 KB. Hence, a PC with a relatively modest 24 GB of memory could buffer just over 200000 'full-hit' events, or 3.5 minutes of triggering at 1 kHz.

However, average events have only 1000 hit PMTs, not 10000. The RAT packed data structures used internally are varible-sized, so the actual buffering capacity is substantially larger.

GNU glibc predates Linux pthreads, and performs quite poorly when multiple threads compete for memory access. Fortunately, alternative memory allocators exist with vastly superior performance in threaded applications as well as drastically lower per-allocation overhead. The event builder uses `jemalloc`[1], the system allocator from FreeBSD which was recently optimized by engineers at FreeBSD and Facebook, in place of glibc. `jemalloc` offers approximately 2% overhead.

3 Architecture

The work of the event builder is done by two functions: a "listener" and a "shipper." The listener accepts connections from the XL3s, SBC, and TUBII, parses incoming data, and inserts it into the correct pigeonhole in the ring buffer. The shipper reads the oldest event in the event buffer, and sends it to disk and the dispatcher once the event is finished.

For compatibility, the event builder also accepts headers and event data from ORCA, a configuration more similar to SNO.

Events are shipped out after a fixed timeout, through more complex logic is simple to implement if more information on the readout state can be determined from the front end electronics. Once an event is ready to be shipped, the shipper writes out any buffered run-level headers with a starting GTID less than of equal to that of the event, then any such event-level headers, then the event itself.

Headers and events are serially written:

- To disk in the packed RAT ROOT format
- On the network via a ZeroMQ (avalanche[2])

[1] http://www.canonware.com/jemalloc/
[2] http://github.com/mastbaum/avalanche

ROOT dispatcher

3.1 ROOT Output Format

Built events are to be written out in the packed RAT ROOT format in ROOT TTrees. ROOT files provide many advantages over plain C or Fortran binaries, including built-in MD5 checksumming and convenient analysis features such as interactive histogramming.

3.2 Threads

To leverage modern commodity multiprocessing, the event builder makes heavy use of POSIX threads (pthreads). The main event builder process spawns two threads at startup: the listener and the shipper. When the listener accepts an incoming connection, it opens a new port for communication and launches a new thread. In this way, each XL3 is communicating with a single thread. As noted in the introduction, performance is best when the event builder communicates directly with the XL3s; that is, when they are not hidden behind a single DAQ computer.

3.2.1 Thread Safety

With many threads simultaneously accessing the same ring buffer, care must be taken to ensure data integrity. All functions which can modify data are protected with pthreads mutexes. Buffer elements are protected individually for high performance.

4 Platform

The event builder is optimized for use in a Linux or UNIX environment running on commodity hardware. Given the high number of threaded XL3/DAQ connections, a 24-thread CPU would be ideal, but is not a requirement. The amount of memory required will depend on the average trigger rate of the detector. ECC memory DIMMs providing up to 64 GB are readily available, which could provide buffering of over several hundred thousand events – several minutes of buffer. Such a system could be built for approximately $5000 USD, given current market values.

5 Performance

The event builder has been tested using fake data derived from a RAT simulation and put through a sequencer/XL3 simulator that randomizes GTID order and inserts known "glitches" into the data stream. Figure 1 shows the state of the detector event buffer, demonstrating the shipper's behavior as events are finished and written to disk.

Figure 1: Detector event buffer status

6 Summary

A new event builder for SNO+ has been written that takes advantage of:

- A new DAQ architecture wherein data is pushed from the front-end electronics, rather than pulled by a DAQ process
- TCP/IP as a standard for communication between DAQ components
- Symmetric multiprocessing with POSIX threads
- Thread-optimized memory allocators

to achieve a lightweight, simple, and robust design. Excluding the data structures and ORCA components, the code base of the event builder comprises approximately 1000 lines of C++ and ISO C. The event builder runs on inexpensive commodity hardware in POSIX environments, enabling rapid deployment, easy maintenance, and assurance of long-term platform support.