A Lightweight, Efficient Data Aggregator for SNO+

Andy Mastbaum[*]
University of Pennsylvania

July 7, 2011

1 Introduction

In the SNO+ experiment, data from 9728 PMT channels spanning 20 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. This paper provides a detailed technical description of this event builder.

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 (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Event-Level Headers Event level headers may be produced at any time, and apply to all following events, until superceded by another event-level header of the same type.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 that no lists ever need to be looped over. 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

[*] mastbaum@hep.upenn.edu

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, a packet representing data from PMT 37 associated with GTID 42 is stored in `Buffer[42].PMTBundle[37]`.

The internal data structures `PMTBundle`, `MTCData`, and `CAENData` are identical to those used by the front end (XL3), obviating any translation. The detector event, run-level header, and event-level header structure match the specification given 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 storage efficiency, although tradeoffs remain possible.

The event builder must buffer relatively few run- and event-level headers, and so storage requirements for those 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 events, or 3.5 minutes of triggering at 1 kHz.

It is possible to improve this number by noting that events, on average, have approximately 1000 hit PMTs, not 10000. Given the 96 bit size of PMTBundles, it is more efficient for NHIT < 3333 to store in the Event an array of 64-bit pointers to PMTBundles, and allocate on-demand. However, such an approach requires that the PMTBundle pointer array be flattened before writing to disk or a socket, a time-consuming task. Additionally, memory allocators must store some metadata to keep track of allocations, and hence typically have a fixed per-allocation overhead, disincentivizing many small allocations. The standard GNU glibc malloc commonly used in Linux programming, for example, has a 4 byte overhead on each alloation, meaning that replacing PMTBundles with pointers may actually double the data stored: a 64-bit pointer + 32 bit malloc overhead + the original 96 bit data = 192 bits consumed.

In light of these concerns, this event builder allocates space for 19 × 16 × 32 PMTs for every event. Structures written to disk are hence largely empty; on-the-fly gzip compression with zlibc results in files of the expected size given N hits.

A final memory-management concern is allocator performance in a thread-based environment. 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.

[1] http://www.canonware.com/jemalloc/

The latter point here represents a significant improvement over the SNO event builder. The event builder keeps track of the position of each crate's XL3 in its readout loop and the position of each sequencer. By tracking when the XL3 and sequencer loops wrap around, the event builder can determine when an event has been fully built without relying on a fixed timout. This provides a gain in both speed (readout is usually much faster than the timeout) and robustness (fewer "orhpan" events).

A timeout is still provided as a failsafe for pathological events and rare corner cases not covered by the above (e.g. events with an entire crate unhit).

Once an event is ready to be shipped, the shipper sends any stored 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.

For example, say that run 42 begins with GTID 84. When the Event with GTID 84 is ready to be shipped, the run header, CAST header, CAAC header waiting in the run-level header buffer, each with a starting GTID of 84, will be written out before the Event.

Headers and events are written to:

- Disk in a packed binary format (CDAB)
- Via a TCP/IP socket to another machine running a "dispatcher" process
- Optionally, via a UNIX socket to a lightweight RAT process running the "UnPackEvent" processor, reading raw data from a network socket and writing to disk in RAT format.

3.1 CDAB Packed Binary Format

The packed binary format, CDAB, consists of C structures written directly to disk with gzip compression. This format is space efficient, fast, and requires minimal CPU time. The C structures may be serialized for enhanced portability if it is deemed necessary. As implemented, CDAB is similar in spirit to SNO's ZDAB format, which were based on Fortran ZEBRA structures written to disk in binary form.

CDAB format is preferable to ROOT for several reasons. First, writing ROOT files with the event builder requires that the builder link to ROOT libraries. This adds substatial overhead to what should be an extraordinarily lightweight program. Second, ROOT I/O calls C I/O at a low level. Hence it is necessarily slower to write the same data to disk with ROOT than with a C fwrite(). Third, ROOT trees require a small but nonzero amount of storage overhead for metadata. C structures are stored with no overhead.

The present nominal packed format for SNO+ abuses the ROOT TTree to create a serial arbitrary object store: each "event" in the tree may be detector event data or one of the several types of run headers. Hence, since events are not identical, would-be features such as TTree::Draw cannot be used to display data from packed ROOT format. Serial data streams were not the intended use of TTrees, and are much better-implemented in a traditional binary file such as ZDAB or CDAB.

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. This allows the use of blocking system calls (each thread can safely wait for data), minimizing CPU load. 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. A mutex is stored at the Buffer level, and enforced when a thread is creating/removing an event/header. Under normal operating conditions, only one PMT bundle per PMT per event can be received, and locking conditions have been tuned for best performance with this in mind.

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. The amount of memory required will depend on the average trigger rate of the detector. ECC memory DIMMs up to 48 GB are readily available, which could provide buffering of over 500000 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.

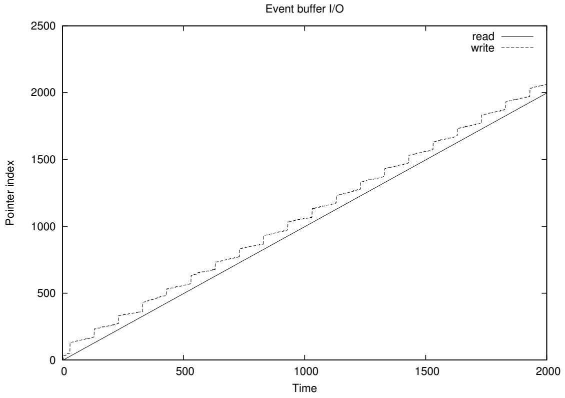

Figure 1: Detector event buffer status

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
- Modern, thread-optimized memory allocators

to achieve a lightweight, simple, and robust design. Excluding the data structures, the code base of the event builder comprises approximately 500 lines of pure C. The event builder runs on inexpensive commodity hardware in Linux or UNIX environments, enabling rapid deployment, easy maintenance, and assurance of long-term platform support.