A Lightweight, Efficient Data Aggregator for SNO+

Andy Mastbaum*
University of Pennsylvania

June 18, 2011

1 Introduction

In the SNO+ experiment, data from 10000 PMT channels spanning 20 electronics crates, trigger information, and several digitized trigger sums must be aggregated into complete events and written to disk very quickly so as not to introduce a bottleneck. Event data sent from the XL3s and trigger system (via the SBC/ECPU) are not synchronous and not necessarily ordered. 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 makes it possible for an event builder to receive data directly from the crates. 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 to disk in RAT (or other) formats(s) and to a "dispatcher" which will send data to various monitoring stations. This paper provides a detailed description of this event builder.

2 Data Structure

Events are stored in the event builder in a highly structured way, in order that no lists ever need to be looped over. Events are buffered in a simple random-access ring buffer:

Buffer Ring buffer

- **Event** Built event
  - **PMTBundle** 3 words of PMT data
  - **PMTBundle** 3 words of PMT data
  - **...** × 10000
  - **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.

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. The size of data structures is 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

*mastbaum@hep.upenn.edu

An event with 10000 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.

Memory allocators in practice must store some metadata to keep track of allocations, and hence typically have a fixed per-allocation overhead, disincentivizing many such 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.

Fortunately, alternative memory allocators exist with much lower overhead (and vastly superior performance in threaded applications). The event builder uses `jemalloc`[1], the system allocator from FreeBSD, recently optimized by engineers at 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 ring buffer, and removes it under three conditions:

1. Each crate has sent a packet indicating that the event is finished
2. A fixed timeout has occured ("orphan event")
3. The ring buffer is full

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

Once popped off the buffer, the event is sent:

- Via a UNIX socket to a lightweight RAT process running the "PackEvent" processor, converting from raw data to RAT format and writing to disk
- Via a TCP/IP socket to another machine running a "dispatcher" process

3.1 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.

3.1.1 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. Mutexes are stored at the Buffer level and at the Event level, and enforced when a thread is creating/removing an event or creating/modifying event data, respectively.

For the best performace, exclusion locking would occur at the PMTBundle level. However, this is not possible given the 40-byte overhead associated with a pthreads mutex.

4 Platform

The event builder is optimized for use in a Linux or UNIX environment running on commodity hardware. Given the high thread count, a cutting-edge 24-thread (12-core) CPU or two 12-thread (6-core) CPUs 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. Such a system could be built for approximately $5000 USD, given current market values.

5 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
- Modern symmetric multiprocessing
- 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 250 lines of pure C. The event builder runs on inexpensive commodity hardware in Linux or UNIX environments, enabling rapid deployment and easy maintenance.