CODA Performance in the Real World

D.J. Abbott, W.G. Heyes, E. Jastrzembski,
R.W. MacLeod, C. Timmer, E. Wolin

Thomas Jefferson National Accelerator Facility (TJNAF)
12000 Jefferson Avenue, Newport News, Virginia 23606 USA

Abstract

The most ambitious implementation of the Jefferson Lab data acquisition system (CODA) to date is for the CLAS spectrometer in Experimental Hall B. CLAS has over 40,000 instrumented channels and uses up to 30 front-end (FASTBUS/VME) crates in the DAQ subsystem. During the initial experiments we found that performance of the fully instrumented DAQ system did not scale as expected based on single point to point benchmarks. Over the past year we have been able to study various performance bottlenecks in the CLAS DAQ system including front-end real time performance, switched 100BaseT Ethernet data transport, and online data distribution and recording. Performance tuning was necessary for components on both real time (VxWorks) and UNIX (Solaris) operating systems. In addition, a new efficient Event Transfer System (ET) was developed to provide faster online monitoring while having minimal impact on data throughput to storage. We discuss these issues and efforts to overcome the real world problems associated with running a high performance DAQ system on a variety of commercial hardware and software.

I. INTRODUCTION

The CODA Data Acquisition system has been used in production experiments at Jefferson Lab since 1995. It was developed as a modular and extensible software toolkit which allows for rapid construction of DAQ systems of varying complexity using a wide variety of commercially available front-end hardware, CPUs, and network links [1]. As a software-based system CODA, can be easily adapted to run on many different types of hardware. CODA Version 2 is supported on the SUN/Solaris [2] and Intel/LINUX [3] based operating systems, and the VxWorks [4] real-time operating system. Existing CODA DAQ systems in use at Jefferson Lab and other facilities range in complexity from a single self-contained VME crate keeping time-based scalar statistics to the CEBAF Large Angle Spectrometer (CLAS) which uses over 30 distributed CPUs for physics data acquisition, detector monitoring and control.

One of the primary problems with such a software-based DAQ is that the overall system performance is governed largely by the "slowest" hardware. This can be an ADC with a long conversion time, a slow CPU, a poor or congested network link, or perhaps an "antiquated" disk or tape drive. However, as systems become more complex new problems can arise. Tracking down the bottleneck may not be as trivial as identifying the "slowest" hardware component. This was the case with the CLAS DAQ system during the first production experiments. With hardware in place that should have allowed for trigger rates in excess if 2 kHz and/or aggregate data rates from 10-15 Mbytes/sec, the CLAS DAQ system could not be pushed beyond 500 Hz ($\approx$1 Mbytes/sec) without suffering a significant increase in acquisition dead time.

In the time that followed we examined the performance of the DAQ system as a function of complexity and found that some of the features that were introduced into CODA Version 2 to manage large systems with parallel data streams were not behaving as predicted in past simulations [5]. We briefly discuss the CODA architecture followed by the solutions found to bring the CLAS DAQ system up to experimental performance specifications.

II. CODA ARCHITECTURE

CODA consists of a set of software objects or components which communicate via a common network accessible database. The three primary CODA components are the readout controller (ROC), the event builder (EB), and the event recorder (ER). Additional software services or components can be implemented to customize and extend the acquisition system. These include the trigger supervisor (TS) component, the CMLOG error/message reporting system, and the event transfer (ET) system which is a replacement for the older shared-memory data distribution system (DD). Each component can be run on the same CPU or distributed among many different hosts.

In a typical data acquisition system one or more ROCs would be running on VxWorks single board computers resident in FASTBUS, VME, or CAMAC crates. Data collected by the ROCs is buffered and sent to the EB (via a standard network link - Ethernet, ATM) running on a UNIX host. The EB sorts the ROC event fragments and builds a complete event which is in turn submitted to the ET or DD system for consumption by the ER and other optional user processes (*i.e.* online analyzer, event display).

Control and monitoring of all CODA components can be handled directly via a built-in Tcl (v7.4) [6] shell or through a Run Control server that is responsible for configuration, run/state transitions, and monitoring of all active CODA components. All the information for experimental configurations, hosts, downloadable modules for components, and scripts to be executed are stored in a network accessible mSQL[7] database. This database is the key to transparent object-based communication between all components in a CODA DAQ system.

Figure 1 shows the implementation of CODA for the CLAS data acquisition subsystem. Twenty-one ROCs accept physics triggers from the Trigger Supervisor subsystem [8].

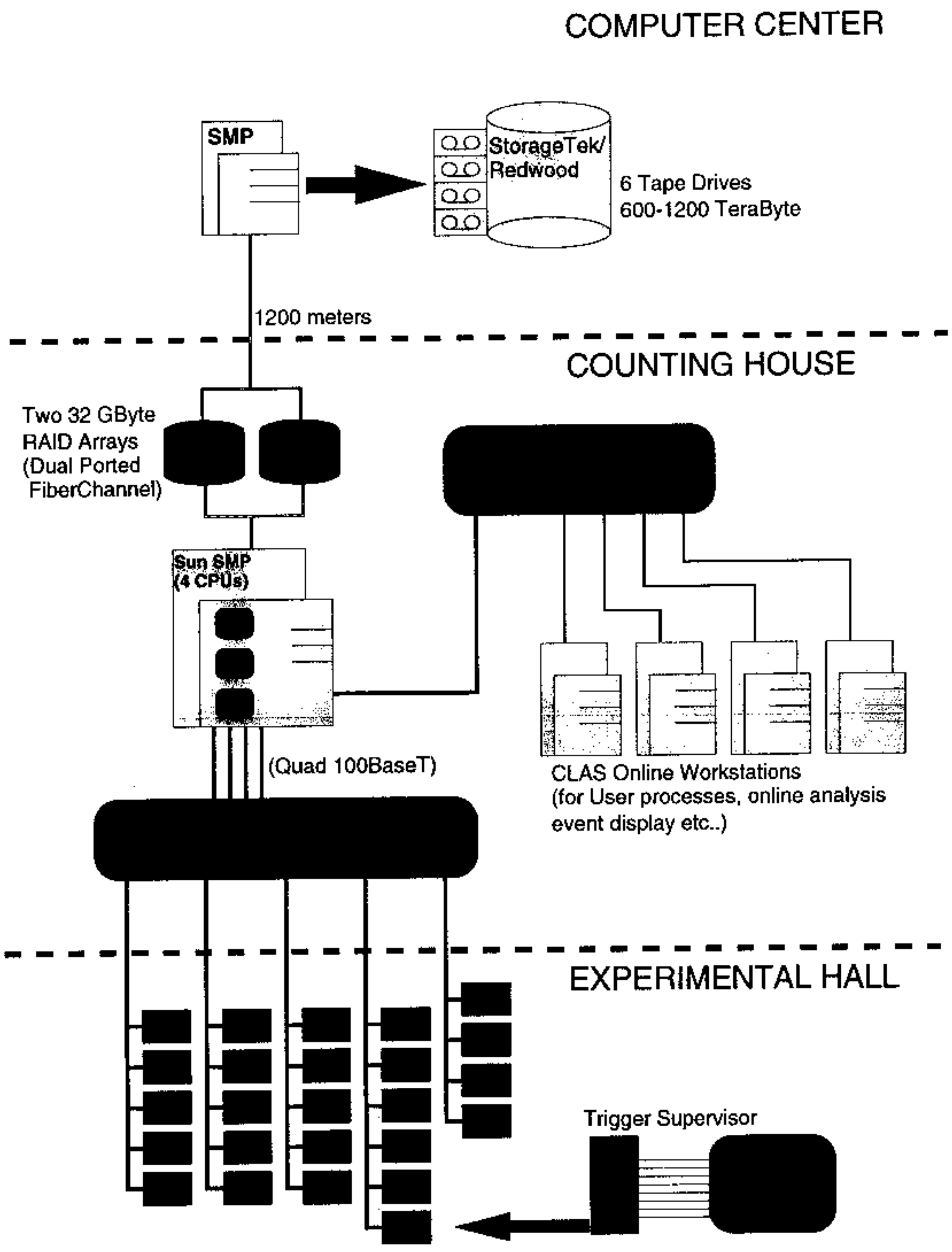

Figure 1: Schematic showing the CODA implementation for the CLAS DAQ system.

Four to nine additional ROCs are available for control and monitoring of the various CLAS detector subsystems. Data from "physics" ROCs are sent via switched 100BaseT Ethernet to an EB running on Sun SMP (4 CPUs). Built events are written to one of two dual-ported fiber channel RAID storage arrays. As one array is filled the second is backed-up to a StorageTek/Redwood tape silo [9] in the Computer Center (about 1 km away). Additional Sun workstations are available on a separate subnet for online monitoring and analysis.

III. PERFORMANCE ISSUES

CODA uses standard network protocols (TCP sockets) for transporting data fragments from the ROC to the EB. Ideally each event fragment should be sent as soon as it is produced on the ROC; however, to avoid the overhead associated with TCP/IP transport, fragments are buffered in sufficiently large records to make the network transport as efficient as possible.

In CODA version 2.0 we implemented a "pull" architecture for the data transport. The EB sends a "token" to the ROC indicating it is ready to accept data. In the case where there are multiple EBs, the ROC queues tokens from each EB and can then send data to the next one available. The EB operates as two threads. The main thread is responsible for polling the ROC connections for data, reading the data, buffering it, and then generating the next token after it has received a record from every ROC. The second thread is responsible for building the events from the ROC fragments, and writing this data to the DD system for analysis and/or recording by the ER.

For a single ROC to EB link the original hardware was capable of 2 kHz and/or 3-4 Mbytes/sec throughput. However, for the case where more than twenty ROCs were sending data to the EB the performance degraded to less than 500 Hz and/or about 1 MByte/sec. We looked at many issues including, TCP send/receive buffer sizes and alternate token passing schemes, all of which had nominal effects on the performance, but not the breakthrough we were looking for. The primary problem was in the EBs handling of multiple ROC data streams. Although the ROCs were simultaneously sending data, there was only a single thread servicing all the connections (via select()). The EB was still serially reading data from all the ROCs as well as grinding though the overhead of monitoring the status of all the connections.

The solution was to provide a mechanism for insuring that data from the ROCs are read in parallel. The first option would be to run multiple EBs on the same or different nodes. However, even with a second EB one could only read two ROCs in parallel, and increasing the number of active EBs seemed an unnecessary complication to the DAQ system given that the network links were perfectly capable of handling the throughput. In addition, there was more than enough processing power by running the EB on a quad-CPU Solaris SMP. Hence, we modified the architecture of the EB to provide a separate "reading" thread for each ROC connection. Each thread simply blocks on a read call to its socket. This solves several problems. First, one avoids calling select(). The main thread remains unblocked and is still responsive to outside inquiries. Second, tokens can be avoided. ROCs now "push" the data to the EB as soon as they are ready. The EB only reads the data when it is ready. Each thread queues data from its ROC into a FIFO buffer. If the buffer for one ROC is full, then no more data is read until the FIFO is emptied by the EB "Build" thread. TCP/IP essentially provides the "backpressure" flow control between the ROC and EB.

Letting a modern SMP UNIX operating system manage 20+ threads in a single process proved to be much simpler than trying to run multiple processes on one or more hosts. Even for the case where one runs an EB on a single processor machine, the multiple thread-based architecture proved to be more efficient than the original two-thread event builder. The EB performance now truly scales with the host (*i.e.* the number of processors, network links) on which it is running. Its current limitation is the use of only a single build thread.

Concurrent with the software changes, new hardware was put in place for the front-end processors (Motorola MVME2306 300MHz VME single board computers). This boosted the performance of the single ROC to EB data link to 11 Mbytes/sec. In tests with multiple ROCs sending data to the new EB running on a Sun Ultra 450 (four 250MHz CPUs and a quad 100BaseT Ethernet card), rates of 30-35 Mbytes/sec were achieved.

Using the components of the revised release of CODA (v2.1) the CLAS data acquisition system has been able to reach its data acquisition design goals, and is now routinely running at 2.5 kHz, 10.5 Mbytes/sec at about 10% dead time.

Event Transfer and Online Analysis

For most data acquisition systems efficient data transport to permanent storage is only half the battle. Users also need the ability to monitor the data content for quality assessment and perhaps preliminary analysis. This should be done with no measurable impact to the data flow.

With CODA version 2, we supported a data distribution system (DD) written for experiments at Brookhaven National Lab [8]. It is an event buffer manager based on UNIX System V shared memory and semaphores. Event Producers can attach to the system, request a buffer and insert an event. Event Consumers can attach to the DD, request some fraction of events for monitoring, analysis, etc. In the CODA implementation, the EB is a permanent DD Producer and the ER is a DD Consumer requiring all events placed into the system.

The DD system was originally designed for the E787 experiment at BNL. There the events were large, and the data came in bursts during a beam spill. At Jefferson Lab events are being acquired in a continuous stream. The DD, while certainly adequate for many of the experiments that CODA was designed to handle, had its limitations. We began to see these limitations as experimental event rates crept up and as more user processes were attempting to request events both on the local DD host as well as remotely. For CLAS, after the ROC to EB communication issues had been dealt with, it was clear that the next bottleneck was the DD system.

With these issues in mind we have written a new event distribution system called ET (Event Transfer) for use with CODA 2.1. The ET system is a general software package useful for efficient access and high speed transfer of data between processes using shared memory. In the ET architecture, there is one UNIX process that handles the flow of all events. This process passes event "descriptors" around a sequential list of "stations". The descriptors point to events in a shared memory file. User processes, including those on remote nodes, may attach to these stations in order to read and to write events.

To provide a system with maximum flexibility and robustness, the ET API has been made reentrant and independent of environmental variables, and crash recoverable. To maximize the speed, multithreading, shared memory, pthread mutexes, and condition variables are used. Unfortunately at this time, ET's use of POSIX extensions (shared pthread mutex semaphores and condition variables) means that some UNIX flavors may not support the main ET system (*e.g.* LINUX). This is not a restriction for an individual ET consumer process.

Great effort was put into ensuring that a crash of a local or remote ET user process will be handled gracefully by the system. A station has three choices in disposing of events left after the user process disappears. If there is a second process attached to the station, the events can be re-queued for consumption by that process. Otherwise, the events can be passed on to the next station or placed back on the free list. Conversely, in the case of an ET system crash, a user process can wait for a new ET "heartbeat", indicating that the system is running again, and can then reattach and carry on. This is particularly useful for programs such as Event Displays that the user would like to start and leave running throughout the experiment.

Preliminary measurements on a Sun SMP (running Solaris 2.6) indicate that ET is up to 25 times faster than the DD system (independent of any memory to memory copying of data). Hence, station to station buffer management as well as multiple producer and consumer overheads have been significantly reduced. For a typical CODA 2.1 application running an EB as an ET Producer generating 5 Kbyte events and an ER as an ET consumer, rates of 20-25 kHz can be achieved (the limiting factor being memory copy speeds).

IV. CONCLUSIONS

With the front-end hardware upgrades and the implementation of the CODA 2.1 updates, the current performance limitations of the CLAS data acquisition system are well balanced for the existing event structure ($\approx$4 Kbytes/event). File I/O to the fiber channel RAID is currently limited to about 12-15 Mbytes/sec. Front-end trigger/readout latency restricts event rates to 3.2-3.5 kHz.

The CLAS Online System has provided an extensive test of the CODA Data Acquisition System, encompassing all stages of event processing. Starting from the front-end where events are acquired using a buffered, multilevel hardware triggering scheme, on to a network-based event building system, and finally to a complex event distribution system. We have been able to create a high-performance flexible DAQ toolkit that has met and even exceeded the specifications for all the experimental programs at Jefferson Lab.

V. ACKNOWLEDGEMENTS

We would like to thank the members of the CLAS Online Group for their participation and assistance during software and hardware optimization, tuning, and testing of the CLAS DAQ system. This work was supported by DOE Contract #DE-AC05-84ER40150.

VI. REFERENCES

[1] G. Heyes, *et. al.*, "The CEBAF on-line data acquisition System", Proceedings of the CHEP Conference (Apr. 1994), pp. 122-126.

[2] Developed by SUN Microsystems, USA. http://www.sun.com/

[3] LINUX Software distributed by Red Hat Software Inc., Durham N.C., USA.

[4] Developed by Wind River Systems Inc., Alameda, CA, USA. http://www.windriver.com/

[5] David C. Doughty Jr. el. Al., "Event Building using an ATM Switching Network in the CLAS Detector at CEBAF", Proceedings of the International Data Acquisition Conference (Oct. 1994), FERMILAB-Conf-95-054, Batavia, IL, USA.

[6] J.K. Ousterhout, "Tcl and the Tk Toolkit" (Addison-Wesley, Massachusetts, 1994).

[7] Developed as part of the Minerva Network Management Environment. Copyright (c) 1993-1995 David J. Hughes. http://www.Hughes.com.au/products/msql

[8] E. Jastrzembski *et. al.*, "The Jefferson Lab Trigger Supervisor System", Proceedings of the Real Time 99 Conference, Santa Fe, NM, USA.

[9] Manufactured by StorageTek Inc., Louisville, CO, USA. http://www.storagetek.com/