Fast Transfer of Shared Data

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

Thomas Jefferson National Accelerator Facility, 12000 Jefferson Ave. Newport News, VA 23606 USA

Abstract

The Event Transfer (ET) system enables its users to produce events (data) and share them with other users by utilizing shared memory on either Solaris or Linux based computers. Its design emphasizes speed, reliability, ease of use, and recoverability from crashes. In addition to fast local operation, the ET system allows network transfer of events. Using multithreaded code based on POSIX threads and mutexes, a successful implementation was developed which allowed passing events over 500kHz on a 4 cpu Sun workstation and 150kHz on a dual cpu PC.

Keywords: event, transfer, data, POSIX

1 Introduction

At the Thomas Jefferson National Accelerator Facility (Jefferson Lab), there are three experimental halls each with different data acquisition (DAQ) requirements. The CEBAF Large Angle Spectrometer (CLAS) in Hall B is the detector whose operation places the highest demand on the DAQ system of any detector being used. With over 40,000 channels and 30 FASTBUS/VME crates, a data rate of 10 Mbytes/sec is not uncommon. To handle this data, the CODA data acquisition toolkit[1][2] has been developed to run on Solaris and Linux systems.

CODA is composed of software components that communicate via the network and with a common database. The first of the three main CODA components is the readout controller (ROC) which runs in embedded controllers in FASTBUS or VME crates collecting raw data. ROCs send their data to the second component, the event builder (EB), which constructs complete events out of these data fragments. The EB, in turn, passes complete events to the last component - the event recorder (ER).

It is the Event Transfer (ET) system that is responsible for passing events (data buffers) between the EB and ER. This system is also used to pass the data to other processes which may, for example, monitor the data quality or do some preliminary analysis. In fact, the ET system is a general software package which may can be used independently of CODA. It is a replacement for the data distribution (DD)[3] system developed at BNL which was previously used by CODA.

2 ET System Design

Software for the transfer of data is such a basic and important building block for Jefferson Lab's DAQ system, it had to meet some stringent requirements. Namely, it had to be fast enough not to be a bottle neck, it had to be extremely reliable, it had to be flexible enough to accomodate many possible uses, and it had to be usable by the average physicist (the most difficult of the four conditions).

Previously, CODA's data transfer system had used UNIX Sys V semaphores to arbitrate accessing events in shared memory. However, in order to achieve optimal data flow, POSIX mutexes and condition variables were used instead, which are generally much faster. Another improvement was to place the transfer of events and the handling of shared memory (memory mapped file) into a single, multithreaded process using POSIX threads instead of into several unrelated processes. When transfering events through the ET system, only pointers to events were passed.

Reliability was mainly achieved by having the users' processes and the ET system monitor each other's heartbeats. Each can tell when the other has crashed or disappeared for some reason. A user is capable of waiting for the return of the ET system and continuing where it left off. The ET system, on the other hand, can recover the events that a crashed user was holding - placing them where the user previously specified - and continue on.

Flexibility and ease of use is due in part to making the ET system software completely reentrant. This means that multiple copies may run on the same or different machines. No environmental variables are used, eliminating a whole class of headaches. In addition, processes on remote nodes may receive events over the network.

3 ET System Implementation

The basic idea behind the flow of events in the ET system (borrowed from the DD system) is to have a sequential series of event repositories called ”stations” (see Figure 1). Each station is primarily composed of two lists sitting in shared memory. One list contains events available for use (input list) and the other contains events a user process is finished with (output list). The first station (GrandCentral) is a special repository of all unused events which users can request, fill with data, and put into the system. Users that create these new events are call producers. Once produced, the ET system process places events in other stations ”downstream.” Users wanting to read or modify the previously created events are called consumers and may ”attach” themselves to stations downstream from GrandCentral where they can get and put these events based on user programmable criteria. Once events reach the last station, they are recycled back to GrandCentral.

This flow of events is accomplished by multithreading the ET system process. Each station has its own event transfer thread - or conductor - which is waiting for output events. When events are ”put” by the user, the conductor is woken up and reads all events in the output list, determines which events go where, and writes them in blocks to each station's input list. A key optimization in the transfering of events from station to user and vice versa is to transfer an array at one time. This reduces the contention for mutexes by the number of events in the array and can result in an increase of speed by over an order of magnitude.

The use of threads has made complete error recovery possible essentially all of the time. ET system and user processes each have a thread which sends out a heartbeat (increments an integer in shared memory). Simultaneously, the system monitors each process and each process monitors the system in other threads. If the system dies, user processes automatically return from any function calls that are currently pending and can determine if the system is still alive. If it is not alive, the processes can wait until it returns. If a user's heartbeat stops, the system removes any trace of it from the system while all events tied up by the dead user process are returned to the system. These events can be sent to either: 1) the station's input list, 2) the station's output list, or 3) grandcentral station (recycling them).

As for the actual events themselves, there are a number of ways to determine which are accepted by a station. Each event has an associated header containing an array of integers whose values may be set, effectively tagging them. Stations may choose to select events based on those tags using either a default algorithm or a user supplied routine. It is also possible to prescale so

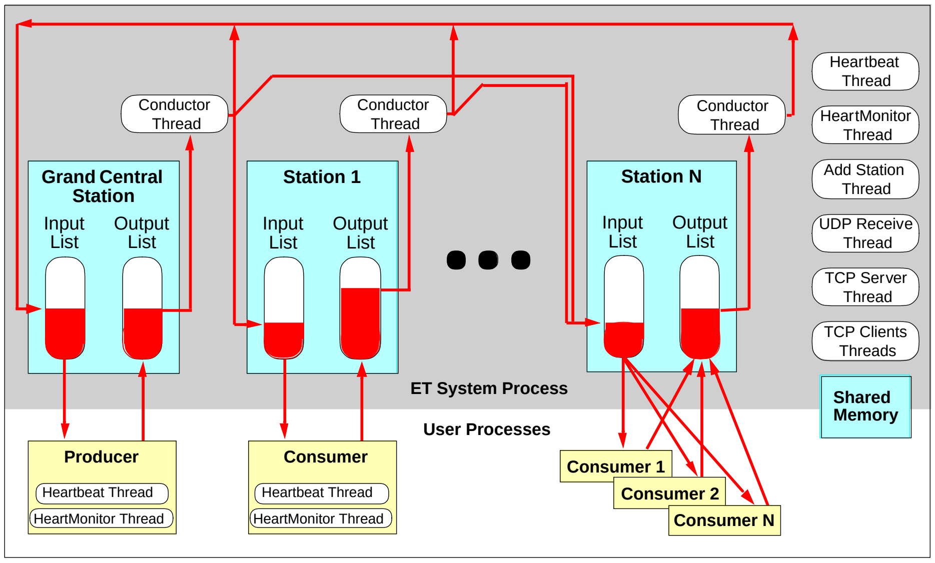

Figure 1: ET System Architecture and Event Flow

that every Nth event is chosen. Another method of selection is to make the station ”blocking” in which case it receives all events that match its selection criteria or ”non-blocking” in which case it only receives a set number of those events before its cue is full. Furthermore, events can be either high or low priority. High priority events that are placed into the system are always placed at the head of stations' input and output lists. That is, they are placed below other high priority, but above all the low priority items.

Occasionally, a user will need an event to hold a large amount of data - larger than the fixed space allocated for each event when the ET system was started and the event size was determined. In such cases, a request for a large event will cause a file to be memory mapped with all the requested space. When all users are done with it, this temporary event will be disposed of - freeing up its memory. This is all transparent to the user.

Part of the ET system's flexibility is its remote capabilities. Users can interact with ET systems over the network since each system has two threads dedicated to that purpose. One thread responds to the UDP broadcasts of remote consumers trying to find an ET system of a particular name somewhere on the network. The response simply sends back the port number of the socket that the second thread is listening on. The second thread, is the listening port of a tcp server. The server, in turn, creates other threads which connect with consumers and handle general and event I/O with them.

The ET system's network capability is what allows it to run on Linux (Redhat 6.0). Currently, the Linux kernel does not allow the sharing of pthread mutexes and condition variables between processes. This makes it impossible to access the shared memory of the ET system safely between processes. However, this problem can be circumvented by treating local Linux producers and consumers as remote. The server built into the ET system handles all ET routines that require handling these mutexes and sends users pointers to events that can then be used to access events in shared memory. This makes ET systems on Linux slower than those on Solaris.

4 ET System Performance

Measurements of the ET system's speed in the handling of events can be seen in Figure 2. Tests were made on both a 4x250 MHz cpu Sun workstation running Solaris and on a 2x450MHz Xenon PC running Linux. Solid lines denote consumers reading and writing events from a single station, while dashed lines indicate each consumer is attached to a different station. In all cases, a single producer was running to create events with 3000 total events in the system and with each read and write containing 100 events. No copying or manipulation of data was done. The stations were made to accept every event. As can be seen, with just a single consumer, an event rate of 550kHz is possible on the Sun, and a rate of 150kHz is possible on the PC. These numbers are a bit arbitrary as the rate can be increased or decreased depending on how many events are written or read at one time (100 in this case). The large difference in the rates is due in part to the treatment of local users as remote on Linux and the greater efficiency of mutex handling on Solaris. The network throughput of data to remote consumers was measured as high as 10 MB/sec on 100 Mb ethernet.

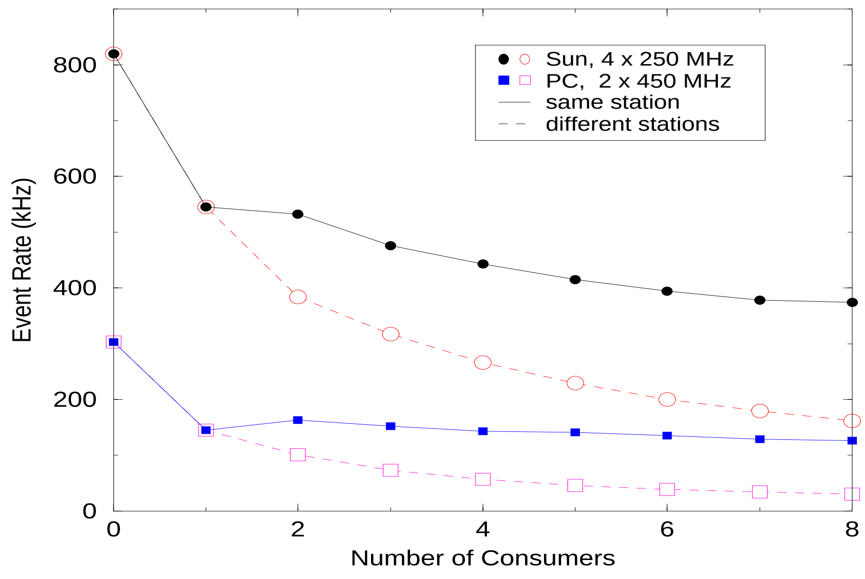

Figure 2: Event Transfer Speed

Operation of the ET system with Jefferson Lab's CLAS detector has been a great success as it has been running for 9 months handling over 16 billion events with no problems. The ET system has been shown to be extremely reliable, it has been easy enough for physicists to use, and it has been flexible enough to meet all the demands placed on it at Jefferson Lab. Most importantly, the bottle neck on the event rate due to the previous data transfer system was removed, allowing rates limited currently by front end hardware.

References

1 G.W. Heyes, et.al., “The CEBAF On-line Data Acqusition System”, Proceedings of the CHEP'94 Conference, April 1994, pp. 122-126.

2 D.J. Abbott, et. al., “CODA Performance in the Real World”, Proceedings of the IEEE NPSS Real Time '99 Conference, Santa Fe, June 1999.

3 Private Communication, Chris Witzig, Brookhaven National Laboratory