An Online Reconstruction CPU Farm for the BLAST Experiment at MIT-Bates Lab

Timothy Paul Smith, Ben Yoder

Massachusetts Institute of Technology - Bates Lab, Middleton, MA (USA)

Abstract

The BLAST detector and experimental program at MIT-Bates Lab will start to take data in early 2001. The analysis of the raw data is complicated by the large acceptance, the non-homogeneous magnetic field and the 500 to 1000 real event rate. Yet an online reconstruction gives the experimentalist a handle on the status of the polarized target, the detectors and over-all experiment. Therefore, we are constructing an online reconstruction CPU farm. In the prototype we used 50 Linux CPUs and analyzed 20 million Monte Carlo events at a rate of 1500 event per second. In the next stage we are analyzing data from the partially instrumented BLAST detector. The key to the farm is a data server which buffers the raw data and server it to the analysis CPUs via sockets. It then collects back the results and also maintains histograms of raw and analyzed data. Since there are a number of time dependent experimental parameters, the correct sequence is vital. We can also serve histograms of data or analysis to client-monitors. Our client-server CPU farm performs well and we expect to be able to handle the data from the full detector in just over a year.

Keywords: CPU farm, Data Analysis, client-server

1 Introduction

One of the major difficulties being faced in Nuclear and High Energy physics is that with a new generation of detectors coming online, the data rate is crushing. In the BLAST[1] detector, presently being built at MIT-Bates Lab, we expect a data rate of 500-1000 events per second after all levels of the trigger. BLAST is designed for internal polarized target experiments. The analysis needs a unique “Data Server” for two reasons. First, the order of the data is critical to maintain, this is because the analysis depends on the slow control variables (such as target polarization, which is periodically flipped) that are injected into the data stream to maintain time order. The second reason for a unique Data Server is that it can also serve as an online monitor. With real-time reconstruction it can also be used as a diagnostic for the status of the target and detector.

Our strategy is to create a Data Server which can either read raw data live from the experiment or from a file off-line. Our reconstruction code can process roughly 30 events per second. This is too slow for our expected data rate by a factor of 30. The solution clearly involves using 30 CPUs, and a Data Server which controls the data: and can also work as an online monitor.

2 The Data Server and CPU Farm

The initial design criteria for our Data Server and CPU Farm were that they must:

- Be able to reconstruct up to 1000 events per second.

[1]Bates Large Acceptance Spectrometer Toroid, `http://mitbates.mit.edu/blast`

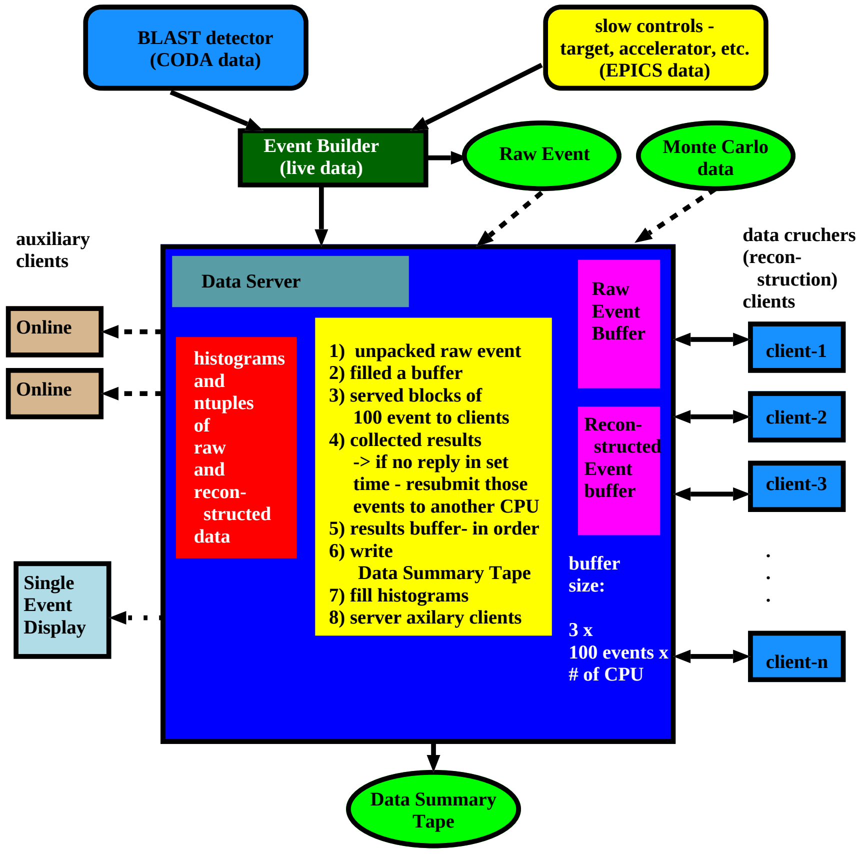
BLAST detector (CODA data)
slow controls - target, accelerator, etc. (EPICS data)
Event Builder (live data)
Raw Event
Monte Carlo data
auxiliary clients
data cruchers (recon- struction) clients
Data Server
Online
Online
histograms and ntuples of raw and recon- structed data
1) unpacked raw event
2) filled a buffer
3) served blocks of 100 event to clients
4) collected results -> if no reply in set time - resubmit those events to another CPU
5) results buffer- in order
6) write Data Summary Tape
7) fill histograms
8) server axilary clients
Raw Event Buffer
Recon- structed Event buffer
buffer size:
3 x 100 events x # of CPU
client-1
client-2
client-3
client-n
Single Event Display
Data Summary Tape

Figure 1: The **Data Server** and related processes.

- Each event containing 500 bytes to 1 kilobyte.
- Reconstruction means going from raw ADC and TDC values to the initial momentum vector of each particle and the events vertex position.
- Our reconstruction algorithms can process about 30 events per second on a 400 megahertz Linux computer.

- Maintain the time order of the events. This is because “slow controls” data, such as beam current, target flow rates and target polarization, will directly, on an event-by-event bases, effect how we use the data.
- Expand to many computers and platforms.

After the first prototype test of our Data Server we recognized that the server as specified above could analyze data fast enough such that a real-time, “online” reconstruction was possible. But in order for it to serve as useful online monitor and diagnostic utility, a number of other features needed to be added, including:

- Being able to acquire a number of histograms of raw data.
- Being able to serve these histograms to users independent of the CPU farm.
- Being able to serve up individual events for a user to view independent of the CPU farm.
- Being able to build and serve ntuples for quick “on the fly” problem solving.
- Being able to process archived data from tape (or disk) and live data from the data acquisition system.

To achieve these goals we needed to implement three major sections: socket communications, raw and reconstructed event buffers, and histograms/ntuples.

2.1 Sockets Communications

Sockets allow us to perform high speed data transfer between the Data Server and a large number of clients. We open the socket connection and initiate the communication with a few tagging messages such as a request for data, a request for a histogram or a request for a single event. Since the sockets we use are designed with a 64 kilobyte buffer, we generally transfer about 100 events at a time, which minimizes overhead.

2.2 Raw and Reconstructed Event Buffers

When data is read in from the DAQ system, or from an archive tape or disk, it is unpacked and placed in our Raw Event Buffer. From here 100 events at a time are sent to a client. After the client has reconstructed the events, the results are sent back to the Data Server and placed in the Reconstructed Event Buffer. The Reconstructed Event Buffer is then flushed to the Data Summary Tape. The reason that results are staged in the Reconstructed Event Buffer is that results can come back from the clients in a different order then they were submitted, and we want to guarantee that events are correctly ordered. Because of the difference in the CPU speed or the complexity of the events, they may be returned to the Data Server in a different order.

Generally we make both buffers at least

$$6 \times 100\; events \times \#ofCPUs.$$

This means that a CPU can fall behind the other CPUs by a factor of three. If a CPU has not returned its results in a normal period of time, the event block can be resubmitted to a second CPU. This allows for the first CPU to have conflict not related to the data (it is off-line or there are higher priority jobs). If the event block fails on a second CPU it is assumed that there is something unusual in that block, it is logged for later analysis and the Data Server continues with the rest of the data stream.

This method works well when working from archived data, where we read data only as fast as we can process it. For live data we need to have enough CPUs to keep up with the data rate, or there will be data lost in our online reconstruction. For our full experiment in 2001 we expect that to be about 30 CPUs. We can also optimize this to match our expected varying data rate.

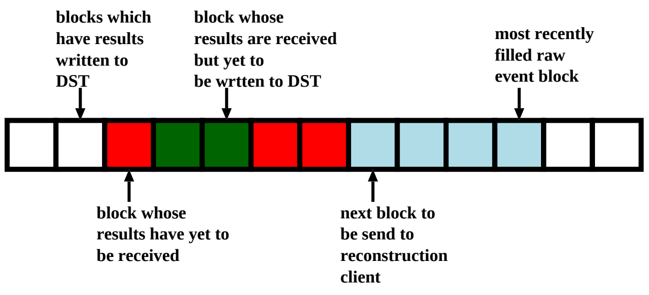
blocks which have results written to DST
block whose results are received but yet to be wrtten to DST
most recently filled raw event block
block whose results have yet to be received
next block to be send to reconstruction client

Figure 2: Raw Event Buffer

2.3 Histograms and Ntuples

There are certain quantities which we know ahead of time we would like to histogram, and certain quantities which we may think of during a run which we would like to investigate on the fly. We allow for both. We accumulate pre-configured histograms such as: the TDC or ADC values on every channel of electronics, the scalar values, the slow control values of beam current, target flow rate over the last hour, and the reconstructed electron momentum. We accumulate these histograms for the entire run. We also accumulate an ntuple of selected quantities for the last 10,000 events. This ntuple can be sent to a client which could then perform “on the fly” analysis. Although the “on the fly” ntuple is on a limited data set, it is very useful for real-time trouble shooting.

3 Initial Test With Monte Carlo Data

Our first Data Server was developed in the summer of 1999. We designed the server to read and distribute data generated by our GEANT simulation of our detector. In the earliest trials we tested our server with half a dozen Linux CPU which were available within the lab. This offered us the opportunity to see how our system worked with CPUs of various vintages and speeds. We found that our reconstruction rate scaled with CPU clock speed, and that we incurred about a %7 overhead in the clients from the socket I/O.

In the second phase we transplanted the server and clients to the University of New Hampshire’s Nuclear Physics Farm. Here we processed 10 million monte carlo events on a farm of 50 Linux CPUs. In this test we only had analysis clients and a control GUI client.

Table I: Test on UNH 50 CPU farm

Results
33 events/sec/process.
50 processes
10 million events total
Celerons - 450 MHz.
1,700 events per second for all processors combined

4 Cosmic Ray, Prototype Detector Test, and Final Design

At present the BLAST collaboration is preparing for a test of prototype detectors and a sample of our final detectors. Before the test run in February 2000 there is a cosmic ray study. We are using our Data Server with the additional features of serving data to auxiliary clients, such as the “online” monitor and the single event display. Both of these clients have been developed and tested in a 1998 prototype run, and the modification to take data from the Data Server is simple.

5 Conclusions

The Data Server which we are developing at MIT-Bates Lab, for the BLAST detector has performed well in its initial 10 million event challenge. At that time, with 50 CPUs, its reconstruction rate was even greater then the expected rate for the full BLAST. Those studies also indicated that the computer which ran the Data Server had enough spare CPU cycles that it could also be an analysis client, or that we could greatly expand the Data Server’s role to include the collection of histograms and an ntuple, as well as serving auxiliary clients. Early test this winter confirm that this expanded role is viable.