The D0 Data Acquisition System and its Operational Control

G. Briskin[1], D. Cutts[1], A. Karachintsev[2]. S. Mattingly[1], G. Watts[3], R. Zeller[2].

[1] Brown University, Department of Physics, Providence, RI 02912 USA
[2] Zeller Research Ltd., Bristol, RI
[3] University of Washington, Seattle

Abstract

The D0's data acquisition system for the Fermilab Collider's Run 2 features a unique design for the free flow of data blocks from digitization crates to Level 3 trigger processor nodes. In addition to buffer memory boards, upgraded from Run 1, the new system includes collectors that receive blocks from the crates and transmit them over fiber to the first of a group L3 farm segment branch modules. The data blocks are constantly moving, either to an appropriate L3 node in a segment, to the next segment branch, or returning in a recirculation loop to the initial collector.

The DAQ control software will coordinate overall data acquisition operation, and provide quasi-real-time monitor information. The control software is based on a distributed system with specific processors coordinating each custom data path module, and two separate machines for overall system supervision and for monitoring.

Keywords Data acquisition, Data handling, realtime, online control, PC farms, software trigger

1. Introduction

As shown in Figure 1, the upgraded D0 Level-3 trigger and data acquisition system hardware consists of five primary elements:

1. Front-end VME Buffer Driver (VBD): these custom memory boards contain two buffers and two list processors one of which can load a buffer over the VME backplane while at the same time the other processor outputs a buffer through the external port.
2. Front-end VBD Readout Concentrator (VRC): these modules serve to coordinate the readout of groups of digitizing crates. Each VRC handles two independent readout paths from VBD's, coupling the two 48 MB/s data cables to a single 100MB/s path driven by Fibre Channel hardware.
3. Segment Bridge (SB): in Run 2 the Level-3 processor farm is subdivided and these new modules bridge between the primary data path and the individual processor farm segment's data cables.
4. Event Tag Generator: this module receives the L2ACCEPT signal from the hardware trigger framework. It uses information loaded for the run in progress, defining the triggers and the L3 farm, to create a tag, which is circulated to each Segment Bridge. This "event tag" permits a highly flexible node assignment for that event, driven by the specific trigger bits received with the L2ACCEPT.
5. Level-3 analysis node: the individual nodes are commercial multiprocessor SMP systems running Windows NT OS. They are connected to the data acquisition data paths via a PCI-VME bus adapter (pVBA). Data blocks arrive through external ports of the VME multiport memory boards, and then are transferred over the VME backplane and into processor memory via the pVBA.

A block diagram of the system and the relationship of these elements is shown in Figure 1. The acquisition system is intended to be 'free running'. Recirculation loops provide intrinsic rate limiting for data and event tag distribution without excessive buffering. Data flow in the system is segmented into five primary paths:

- *Front-end data collection path.* Data flows from a group of VBDs into a VRC via a token arbitrated loop, with two 48Mbyte/s loops per VRC.

- *Primary Fibre Channel path.* This path is a recirculation loop linking the VRCs to the SBs. Data flows from each of the 8 VRCs via a 100 Mbytes/s link based on Fibre Channel components to the first segment bridge. Blocks not accepted by this segment are passed to an adjoining SB, and if necessary, from the last SB are returned to the originating VRC.

- *Level-3 data distribution bus.* Each of the initial 4 segments of the L3 farm nodes has 4 unidirectional direct address parallel busses that link the SBs to the MPMs in Level-3 nodes. There is a total of 16 busses each operating at 48 Mbytes/s.

- *Event tag path.* This recirculation loop links the event tag generator to the segment controllers. The ETG creates and circulates a tag in response to the L2 trigger, to provide information needed by the SBs for node assignment and readout crate mapping for the event.

- *Level-3 event output path.* The event rejection accomplished by the Level-3 trigger's filter algorithms reduces the 1 KHz input rate expected in the initial system to an output rate below 100 Hz, where standard commercial Ethernet connectivity to the online system is appropriate.

As can be seen in Figure 1, the VBD in a particular digitizing crate outputs a data block onto a path to a readout controller (VRC) that handles its crate group. The data block proceeds through buffers in the VRC onto a fiber line to the first of the L3 farm Segment Bridges. The Segment Bridge (SB) recognizes the event as one assigned to a L3 node in its segment, and buffers the block (given that space is available); otherwise it passes the block to the next Segment Bridge. If any SB does not accept the block, it is returned to a recirculation buffer in the originating VRC.

Assignment of an event to a specific node (or nodes) is initiated by the response of the Event Tag Generator (ETG) to the L2ACCEPT signal and its associated trigger information. The ETG circulates an event tag, and a Segment Bridge receives the tag and determines, from its contents and from the availability of L3 nodes in its segment, if it can commit to accepting and bridging the blocks for that event to its segment. Within each Segment Bridge, an event tag interface (ETI) uses a 16K deep Content Addressable Memory, which, together with the use of sidebar bits and other logic, permits real time decisions on event assignment. The design permits flow of the blocks associated with an event to one (or more) of specific nodes, determined by the event's trigger bits, as well as the operation of *shadow nodes* in which a specific node or nodes receives a copy of events in which specific trigger bit(s) have fired.

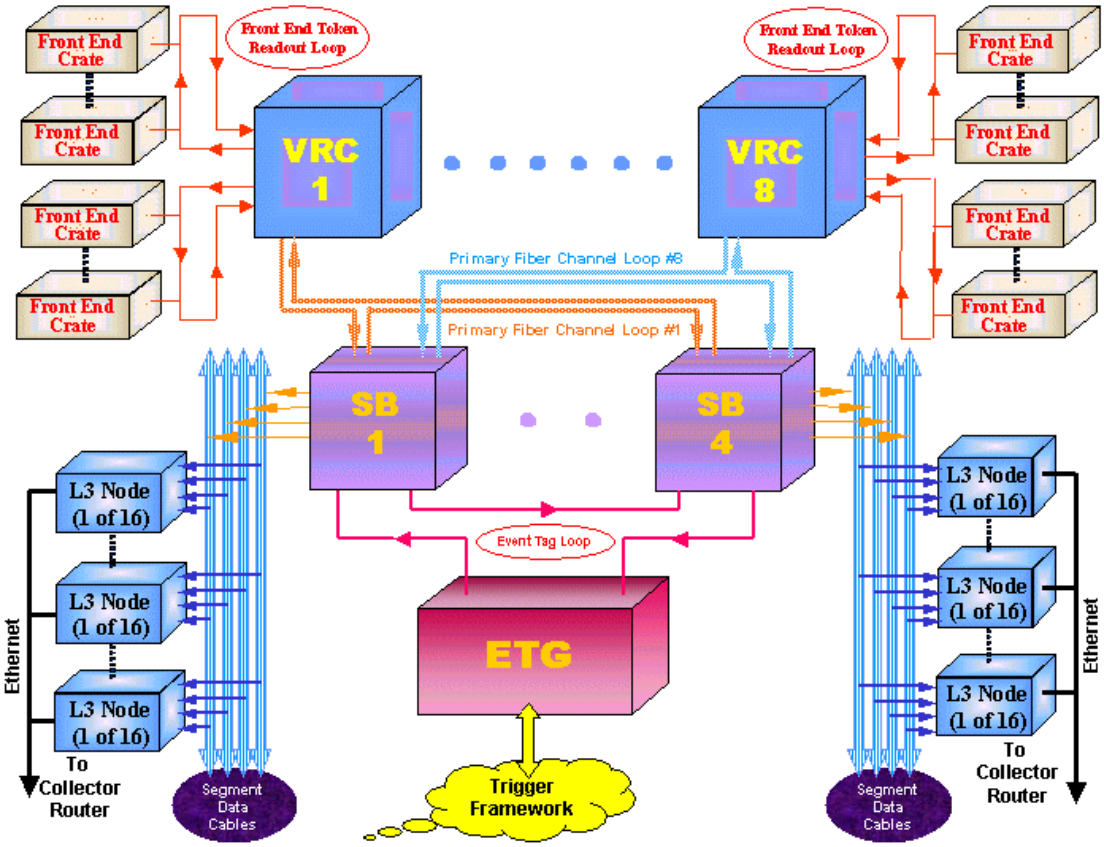

Figure 1. Schematic view of D0 Data Acquisition System.

2. Control of D0 DAQ System

The Level 3 DAQ/Trigger is a complex distributed system as was described in the introduction. Figure 2 is a block diagram of the system, each block representing a different DAQ control component. The Level 3 DAQ software can be split in two major parts: *data handling* and DAQ *control*. Data handling involves operations that must occur for every event that is read into the farm, while control functions execute much less frequently. The data handling software is mostly present in the L3 farm nodes; it manages the readout, the filtering, and the transport of accepted events to the online system's Collector/Router. The DAQ control software manages the Level 3 DAQ system as a whole: the 50 L3 farm nodes, the 10 or so individual hardware components (Segment Bridges (SB), VBD Readout Collectors (VRC), *etc*…), and the interaction with the online system. The following two sub-sections provide a very brief introduction to the two; later sections of this document describe the control framework in more detail.

2.1. The L3 Node Framework Software

The L3 Node Framework software manages all data operations in an L3 Farm Node. It is responsible for setting up the MPMs to read in the data and check its integrity. The data is then

distributed to one of several L3 Filter Processes for physics analysis. Once the L3 Filter has rendered the physics decision on the event, the data is either thrown out or is packaged and sent to the online system for recording and distribution. The framework is designed to survive crashes in the physics analysis code. The framework can also take advantage of a multiprocessor machine by running several copies of the analysis code at the same time.

3. The DAQ Control System Software

The DAQ control software is responsible for coordinating all aspects of the Level 3 Trigger and DAQ. Unlike the software in the node, the DAQ control software is not active for every event. Rather, its job is to configure the farm and DAQ hardware as requested by the online system. Most of this activity occurs during run initialization, when the farm must be configured to take data, and at the end of a run when statistics and other information must be gathered. The DAQ control system also gathers monitor data. Figure 2 shows a simplified diagram of the control system.

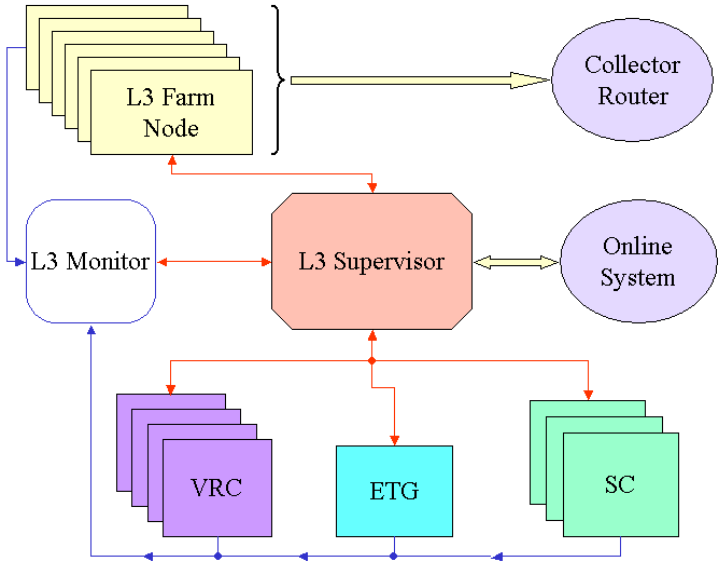

Figure 2 The block diagram of the DAQ Control System.

3.1. Interprocess Communication

The trigger and DAQ system is distributed: over 50 computers will comprise the initial system. The requirements aren't so much speed – these are control messages we are passing around. We did need something that was potentially rich (*i.e.* more than simple strings). We were also actively looking for ways to better test the system, so any communication protocol had to be integratable into a test system. We have looked at several technologies to manage the communication: TCP/IP streams, CORBA, and COM. Streams, it was decided, were not rich enough. We liked the idea of CORBA, but were unable to make it work easily in the NT environment. We are currently in the final stages of testing COM as the solution.

COM has several advantages. First, it is well integrated into the MS Visual Studio environment. The editor writes the housekeeping code required for a COM object for you. It is

also well integrated with the NT operating system, integrated with scripting languages like JavaScript and Visual Basic. If COM is adopted, it is likely many of the test cases for the Level 3 control system will be written as scripts.

3.2. Farm Initialization

All computers are assigned a task in a central database. For example, farm nodes are assigned the farm task, a VRC node is assigned a VRC task. Upon a boot up, each system automatically checks with the central database for its task. It then, if required, downloads the necessary software, installs it, and runs it. This system is installed on all nodes in the farm. The central database is stored on the supervisor node.

This system is used by the supervisor to change the software assignment of any node. This most often happens during initialization or loading test executables.

Note that if a farm node crashes during a run, it is possible for it to recover and then return to the data taking run currently in progress.

3.3. The Supervisor

The supervisor program is very much like a resource allocator. It receives requests from the DØ Run Control (RC) program and then sequences out a much more detailed list of commands to the farm and DAQ computers. For example, if RC asks to alter or add trigger a bit during a run, the supervisor first halts the data flow, alters the ETG's event routing tables, and then restarts the data flow.

The supervisor is a single C++ program. Internally, it keeps a database of the farm's current configuration. After a new set of commands from RC, a desired configuration database is created. The two are compared and commands to change from the current configuration into the desired are issued. If an error occurs during the change, the object that reported the error is marked in an unknown state, and the error is reported back to RC. If RC attempts the configuration change again, it will be starting where it left off. In practice, once an error has occurred during a configuration change the whole farm and DAQ system has to be reinitialized. However, based on our Run 1 experience, this is not the case.

3.4. Monitoring

Monitoring is especially important for the Level 3 system. Run 1 experience has shown that trouble in the DAQ system often first shows itself in the Level 3 displays. It will be vital to have enough monitor information to tell if it is the Level 3 system that is the source, or one of the systems that feeds Level 3.

The monitor node centrally manages monitor information. Is responsible for acquiring information from all other nodes and relaying it to outside sources. The driving concept behind this was to make sure that external monitor programs could not affect the farm's performance. Each node in the farm and DAQ system will be equipped with a small monitor module that will relay information to the monitor node. This system has been written in C++ in a way to make declaration of a monitored variable as easy as declaring an integer or string.