

Overview of the DarkSide-50 Data Handling System

Kurt Biery, Chris Green, Jim Kowalkowski and Marc Paterno
Scientific Computing Division / Fermilab
Boris Baldin, Stephen Pordes, Jin-Yuan Wu and Jonghee Yoo
Particle Physics Division / Fermilab
Alessandro Razeto
LNGS / INFN

Revision 8 October 30 2012

Contents

1	Introduction	2
1.1	Purpose	2
1.2	Scope	2
2	Overview	2
2.1	Detector	2
2.2	Electronics	2
2.3	DAQ	3
3	DAQ Requirements	3
3.1	Functional Requirements for the data path	4
3.2	Functional Requirements for the control and monitoring path	5
3.3	Quality and Robustness Requirements	5
3.4	Development and Operating Environment requirements	6
3.5	Performance requirements	7
3.6	Quantities and items of DAQ system monitoring	7
3.7	Scheduling	7
4	System Architecture	8
4.1	Distribution of Functions	8
4.2	Major Software Components	8
4.3	Required communication protocols	12
5	Deployment and testing strategy	16
5.1	Phase 1	17
5.2	Phase 2	17
6	Additional Figures	18
	Bibliography	18

1 Introduction

1.1 Purpose

This document describes and defines a project for the data-handling part of the DarkSide-50 data acquisition system for the TPC detector of the DarkSide-50 experiment. The DarkSide-50 TPC data acquisition is a collaborative effort of the Fermilab Scientific Computing and Particle Physics Divisions, and the Laboratori Nazionali del Gran Sasso Physics Division.

1.2 Scope

The project consists of the provision of computer hardware and software systems for the transfer, event building and aggregation of the data relevant to the TPC detector from the DarkSide-50 experiment digitizing hardware to the data-cache at the experiment. The project also includes the facilities for the necessary interactions with externally provided System Control and data quality monitor programs and other items including

- system configuration parameters,
- calibration data,
- metadata describing the collected event data, and
- system performance data.

2 Overview

2.1 Detector

The DarkSide-50 experiment is a dark matter search located in Hall C of the Laboratori Nazionali del Gran Sasso (LNGS) in Italy. The detector has three major components: a) the central TPC containing 150 kg (50 kg active) of low radioactivity argon sitting in b) a four meter diameter sphere containing liquid scintillator which sits in turn in c) a 10 meter high, 11 meter diameter water tank. The TPC contains the target which is viewed by 38 photomultiplier tubes; devices b) and c) are used to veto background events and contain a total of 190 photomultiplier tubes.

2.2 Electronics

The TPC detector electronics reside in a clean room (required for the construction and operation of the experiment) above the experiment water shield. The output of the TPC photomultiplier tubes pass to home-made front-end amplifier and discriminator boards which produce two analog outputs, one at low gain and one at high gain, and a discriminated time-over-threshold signal. The analog outputs of the front-end boards are passed to a set of five CAEN V1720 (high gain) and five CAEN V1724 (low gain) 8 channel continuous wave-form digitizers. The discriminator outputs from the front-end boards

are connected to a CAEN V1495 general purpose FPGA board. A trigger is formed when a (programmable) minimum number of channels produce discriminator outputs within a (programmable) time window. (Typical values may be 5 channels and 100 nanoseconds.) On receipt of a trigger, a fixed length (~ 300) microsecond block of data starting a fixed time (~ 20) microseconds before the trigger is stored in the digitizer memories for readout. The discriminator outputs for the same period of time are also recorded in a CAEN V1190 TDC. Discriminator output signals from the veto system are recorded in CAEN V1190 TDCs residing in a VME crate in the experiment control room.

2.2.1 Data Header

The header of the data from each event in the CAEN 1720 and 1724 modules will contain an externally generated 8 bit trigger ordinal, and internally generated trigger counter and “time stamp” (in units of 8 ns). The V1495 is the source of the 8 bit trigger ordinal, and has a trigger counter which should match the trigger counter in the digitizers. The V1495 will also have a “time stamp” for each trigger—with the same zero as the digitizers but a different clock rate. These data will be available to ensure the integrity of the event building process.

2.3 DAQ

Each CAEN digitizer is connected by an individual optical fiber to one of 4 channels of a CAEN A3818C PCIe card sitting in a computer in the clean room. The V1495 and the V1190 TDCs are read through a CAEN 2718 VME controller which is also connected by optical fiber to an A3818C PCIe card. The data for one event sent along one fiber is called a ‘fragment’. Three Fragment Receiver computers, each with one A3818C board, receive the data fragments from the digitizers and electronics in the clean room and prepare them for transfer via an InfiniBand switch to a set of computers in the control room. A single Fragment Receiver computer is located in the control room for readout of data from the veto system. A total of five Event Builder computers will be located in the control room to assemble the fragments into complete events and perform a first level of processing; such processing includes lossless and possibly lossy data compression, and possible data reduction. The output data stream will be written in time-ordered sequence to storage in a local short-term cache.¹

3 DAQ Requirements

The experiment is expected to run in two main modes—WIMP Search and Background Study. The amount of digitized data per event is expected to be similar in the two modes. With a data acquisition window of 300 microseconds, the amount of digitized data per event is ~ 8.5 MB, the overwhelming fraction of which is pedestal. The trigger rate for the WIMP search is expected to be a few Hertz at most. For the background studies, however, the experiment needs to be able to take data at greater than 50 Hz and the data

¹The data will be transferred to long-term storage at LNGS and Fermilab but that is not part of the present project.

acquisition system should be able to handle the maximum throughput available from the digitizing hardware.

3.1 Functional Requirements for the data path

3.1.1 Data Readout

The system has to perform the following functions:

REQ DP1 *Hardware configuration:* configure the data acquisition hardware based on information received from the (external to this project) run control.

REQ DP2 *High-gain PMT Reading:* read data from the five CAEN V1720 250 MHz through A3818C PCIe boards into the clean-room Fragment Receiver computers.

REQ DP3 *Low-gain PMT Reading:* read data from the five CAEN V1724 100 MHz boards through A3818C PCIe boards into the clean-room Fragment Receiver computers.

REQ DP4 *Trigger Data Reading:* read trigger information from the (1 or 2) CAEN V1495 board(s) through a CAEN 2718 VME controller connected to an A3818C board into a clean-room Fragment Receiver computer.

REQ DP5 *TPC TDC Data Reading:* read TDC data from the (1) CAEN V1190 board in the clean room through a CAEN 2718 VME controller connected to an A3818C board into a clean-room Fragment Receiver computer.

REQ DP5.1 *Veto TDC Data Reading:* read TDC data from the (1 or 2) CAEN V1190 board(s) and V1495 in the control room through a CAEN 2718 VME controller connected to an A3818C board into a Fragment Receiver computer located in the control room.

3.1.2 Data Fragment Handling

Once the data are received from the A3818 PCIe cards, the Fragment Receiver computers have to

REQ DP6 *Data labeling and packaging:* label and package the data from the various A3818C boards using the header information as described in [2.2.1](#)

REQ DP7 *Fragment Routing:* route each labeled data fragment to the appropriate Event Builder computer via the InfiniBand switches.

3.1.3 Event Building

The Event Builder system has to:

REQ DP8 *Event Building:* build complete events from the fragments it receives.

REQ DP9 *Compression*: be able to perform lossless compression of the event data for storage.

REQ DP10 *Data reduction*: be capable of executing modules to perform data reduction before storage.

REQ DP11 *Local event cache*: write a time-ordered event stream to local disk storage.

REQ DP12 *Data quality monitoring stream*: deliver a time-ordered event stream, at possibly a reduced rate, for data quality monitoring.

REQ DP12.1 *Data quality monitoring stream*: be capable of accepting event analysis modules and transmitting the results of the analyses along with the event data to the data quality monitoring stream.

3.2 Functional Requirements for the control and monitoring path

The reportable system status items are those necessary for ensuring that the data acquisition system is running correctly. They are enumerated in section 3.6.

REQ CP1 *Reporting to system control*: To ensure that the DAQ system is operating according to specification, all software subsystems in the system must be capable of reporting performance results and status information to system control.

REQ CP2 *Reporting to storage*: To allow diagnosis of problems identified in previous runs, all software subsystems in the system must be capable of reporting performance results and status information to a database.

REQ CP3 *Reporting to system control*: The system must report, to system control, the status of all hardware connected to the A3818C boards.

3.3 Quality and Robustness Requirements

REQ Q1 *Robustness against defective data*: Events which are identified as defective will be marked as defective and propagated through the system to a configurable data stream.

REQ Q2 *Reporting defective data*: The event builder will report each instance in which it detects a defective event.

REQ Q3 *Robustness against mis-identified fragments*: The event builder must be able to detect mis-identified fragments in which more than one fragment appears internally identified as coming from the same hardware source (waveform digitizer).

REQ 94 *Robustness against missing fragments:* The event builder must be able to detect events for which some fragments do not arrive within a configurable time window.

REQ 95 *Robustness against EB node failure:* In the event of a failure of an EB node, the system shall be reconfigurable to allow data collection at a possibly reduced rate on the remaining resources.

REQ 96 *Ability to run with partial detector:* The system must be able to collect event data when the detector readout system is incomplete, down to a single active board.

REQ 97 *Reporting loss of computing resources:* The system will report the loss of clean-room computing resources (Fragment Receivers) or control-room computing resources (Event Builders and one Fragment Receiver).

REQ 98 *Report loss of a data source:* The system will report the loss of an expected data input source (e.g., input from one board).

REQ 99 *Data buffering:* The event builder must recognize that the trigger ordinal sets limit to the number of events which can be buffered.

3.4 Development and Operating Environment requirements

The DarkSide-50 DAQ will be operating at the LNGS. The computing systems will be setup at Fermilab and run with simulated data to validate the operation of the DAQ system before shipping to LNGS. A system with one FR and one EB will be maintained at Fermilab. Members of the Fermilab Scientific Computing Division will assist in the deployment of the DAQ system on site at LNGS. Arrangements need to be made for the duration of the experiment for appropriate access by people at Fermilab to the systems at the LNGS and for appropriate access from LNGS to the required Fermilab environment for developing, building, distributing, and testing the software from which the system is composed.

The environment for developing, building, and distributing the DarkSide-50 software will be the system used to develop, build, and distribute the **art** framework and used by the experiments currently using **art**. This includes the **cetbuildtools** build system and the **ups** product distribution system.

REQ OE1 *OS:* The supported operating system shall be Scientific Linux Fermi, version 6 (SLF6).

REQ OE2 *administration interface:* IPMI must be available on the 1G network for low-level administration functions and monitoring system function on all DAQ nodes.

REQ OE3 *OS software updates:* The operating system packages must be capable of update via yum servers.

REQ OE4 *Application software updates:* The application software, including updates to new releases and release candidates, must be loadable in binary form from FNAL servers.

REQ OE5 *Development access to software products:* The same software delivery system used to provide the software to the DAQ computers shall be usable to make the software available to software development computers at LNGS.

REQ OE6 *Development access to DarkSide-50-specific source code:* The source code repository used to store the DarkSide-50-specific code shall be accessible from software development computers at LNGS. Note: DAQ machines shall not require such access, and are intended for use neither as software development machines nor as software building machines.

REQ OE7 *System test:* The system needs to support testing the data path by reading files containing simulated or real fragment data, and passing them through the main data path.

3.5 Performance requirements

REQ P1 *Front-end data throughput:* The system shall be capable of sustained transfer at a rate of 900 MB/s from the Fragment receiver computers to the Event Builder computers. This matches the expected maximum data transfer rate from the digitizers.

REQ P2 *Event output rate:* The system shall be capable of sustained event-writing rate of $900/5$ MB/s = 180 MB/s to the local event cache [DP11]. This take into account an expected lossless compression ratio of five.

REQ P3 *Event Builder computer input rate:* Each Event Builder computer shall be capable of receiving front-end data at a rate of 180 MB/s.

REQ P4 *Fragment Receiver output rate:* Each Fragment Receiver computer shall be capable of delivering front-end data to each Event Builder computer at a rate of 60 MB/s for a total transfer rate from each Fragment Receiver of 300 MB/s.

3.6 Quantities and items of DAQ system monitoring

(Kurt and group)

3.7 Scheduling

REQ S1 *Software provisioning:* The computer system shall have all necessary software installed before shipping from Fermilab. The necessary software components are enumerated in section 4.2.

REQ S2 *Shipping:* The computer system shall be prepared at Fermilab ready for air shipment to LNGS by November 16 2012.

REQ S3 *Commissioning at LNGS:* The system shall be commissioned from Jan 2013 through March 2013.

4 System Architecture

To meet the functional requirements listed in section 3.1, we have designed a multi-process event-building program, which uses MPI[1] as the communication protocol for event-data in the main data path, and which uses a variety of communication protocols (specified below) for control and monitoring.

The programs run as part of the event-builder MPI program include: a) the fragment receiver, b) the event builder, and c) the aggregator. These processes are distributed across the Fragment Receiver and the Event Builder computers.

4.1 Distribution of Functions

The DAQ software runs on a set of nine servers, each of which contains a total of 32 processor cores. Three of the Fragment Receiver nodes reside in the clean room, one in the control room; each of these contains one of the CAEN A3818 boards. Five Event Builder computers reside in the control room. Each of the nine nodes is on both an 1Gb/s Ethernet network and an InfiniBand network. Figure 1 shows the distribution of the computing hardware and the networking connections between the computers.

4.2 Major Software Components

4.2.1 MPI program start-up and management

The System Controller is responsible for starting several processes that form the MPI event-building application, done using the process management tool *pmt*. *pmt* will accept as input a list of task/node/port triplets, which denote each task (fragment receiving, event building, etc.) is to be run on which node, listening for system control messages on the given port.

pmt is responsible for starting and holding together the processes of the MPI program, and for obtaining their exit statuses. *pmt* is run on an Event Builder computer.

4.2.2 artdaq fragment receiver

The fragment receiver processes are run on the three clean-room computers and the fragment receiver in the control-room. The fragment receiver, shown in figure 4, is a multi-threaded program. It is responsible for a) initializing and configuring the CAEN A3818 board, b) reading data (*fragments*) from the board, c) routing each fragment to the correct event builder using MPI, and d) accepting commands from the System Controller *via* XMLRPC. When started, the fragment receiver reads, from its command line, the port on which it is to listen for XMLRPC commands. The information necessary for configuring the fragment receiver will be sent to it through the XMLRPC protocol.

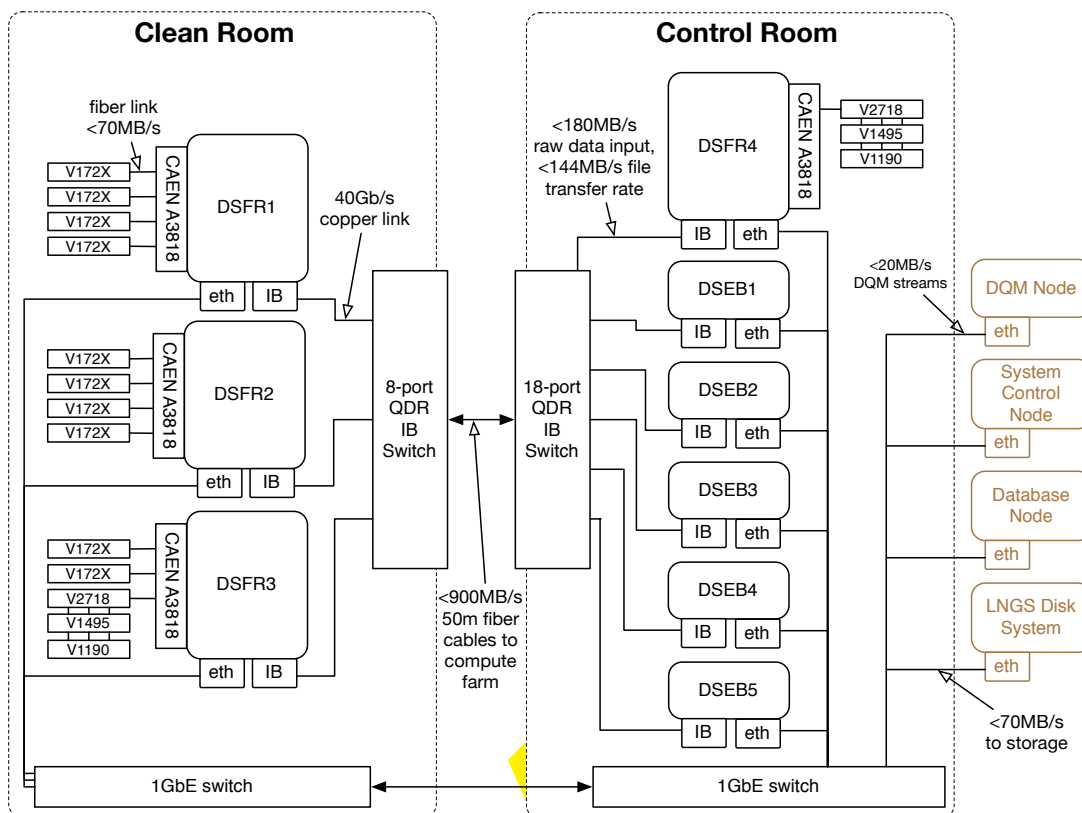


Figure 1: The DAQ computing platforms and networking.

4.2.3 artdaq event builder

The job of the event builder is to assemble the fragments of a given event into a whole event, and to run data compression and data reduction algorithms on the whole events. Each event builder processes, as shown in figure 5, receives fragments from each fragment receiver process; each event builder process sees one in N_{EB} events, where N_{EB} is the number of event builder processes being run. The event builder processes are run on the Event Builder computers in the control-room. The number of event builders run on each computer is configurable.

Each event builder process is multithreaded, and contains at least the following threads: a) one thread responsible for receiving fragments through the MPI protocol, and for assembling them into complete events, b) one thread responsible for processing completed events with the **art** framework, c) one thread responsible for sending messages *via* XMPP to the Message Logger, and d) one thread responsible for communication *via* XMLRPC with the System Controller. The **art** framework may run additional threads.

The event builders will use the output facilities of the **art** framework to write data files to local disk.

The size of each file written by the event builders should be limited so that the intercalated files written by the aggregator are not too large. When an event builder closes an output

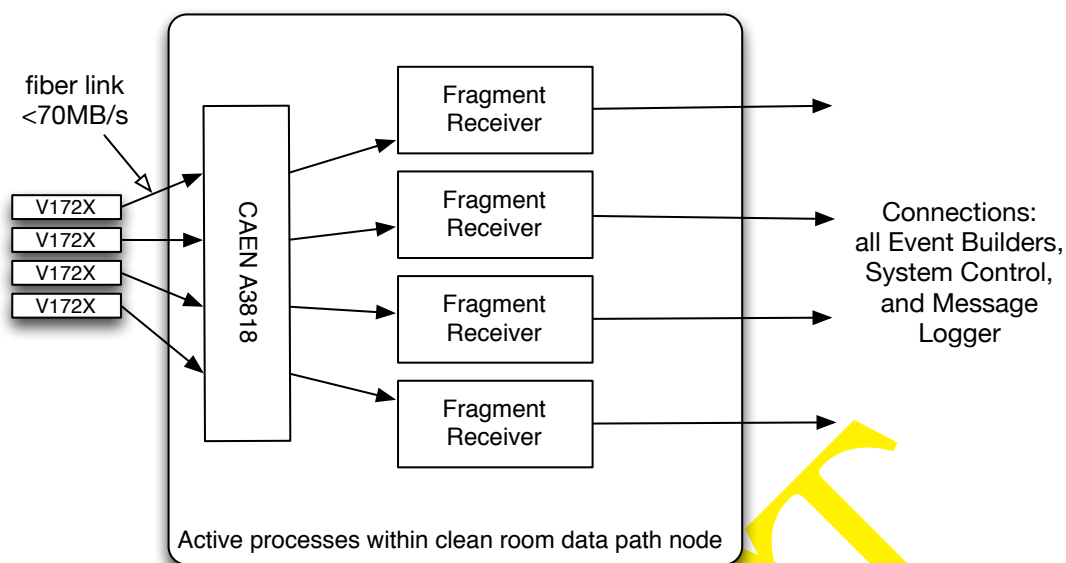


Figure 2: The active processes of the main data path run on each of the Fragment Receiver computers.

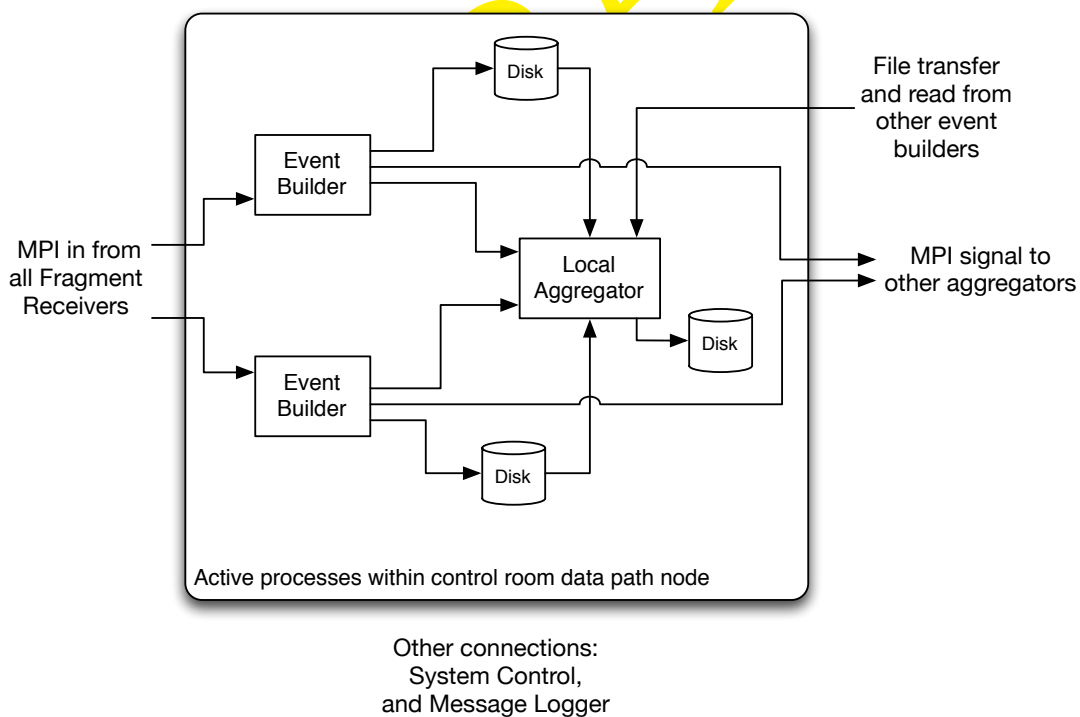


Figure 3: The active processes of the main data path run on each of the Event Builder computers.

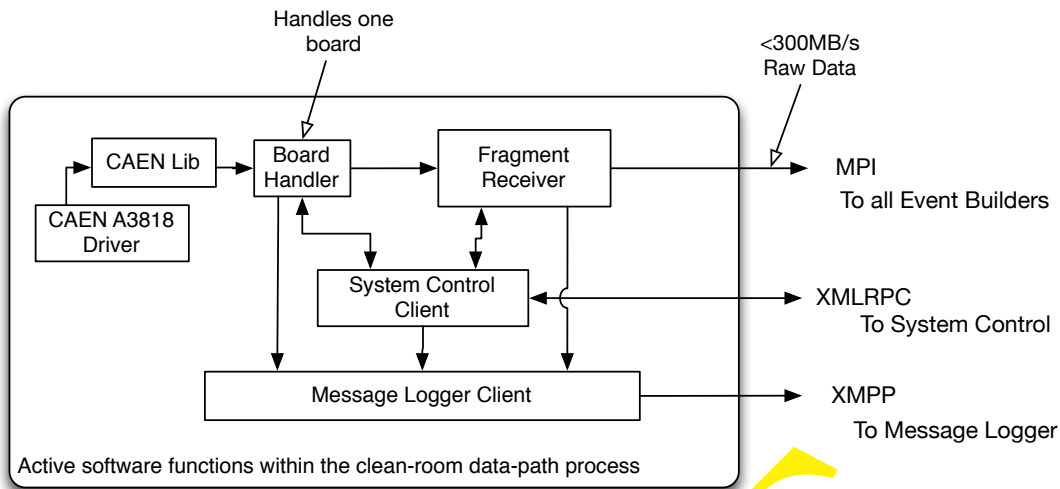


Figure 4: The functions and communication paths of the fragment receiver processes.

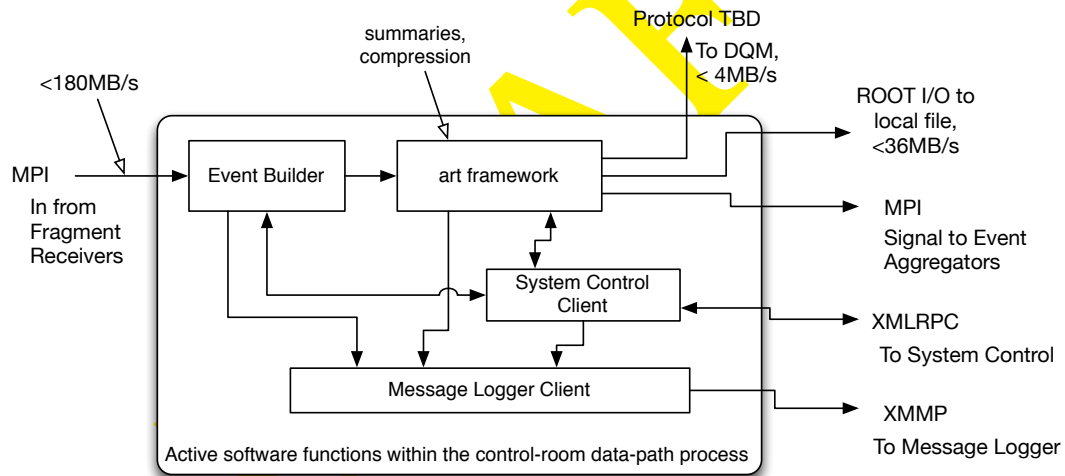


Figure 5: The functions and communication paths of the event builder processes.

file, it sends a message *via* MPI to the appropriate aggregator, identifying the file that is ready for transfer.

4.2.4 artdaq aggregator

The job of the aggregator is to create files of time-ordered, contiguous blocks of events. Since each of the event builder processes sees only a sparse subset of the full event stream (one in N_{EB} events, where N_{EB} is the number of event builder processes), each aggregator process must obtain the output files from each event builder for a given period of data collection. This period should always be an integral number of subruns, to avoid complications in dealing with intervals of validity of metadata.

An aggregator process, as shown in figure 6, will run on each of the Event Builder computers. When an event builder process closes an output file, it will signal the aggregator (using the the MPI protocol) that a file is ready for processing. The aggregator will then transfer the file from its original location to the disk local to the aggregator.

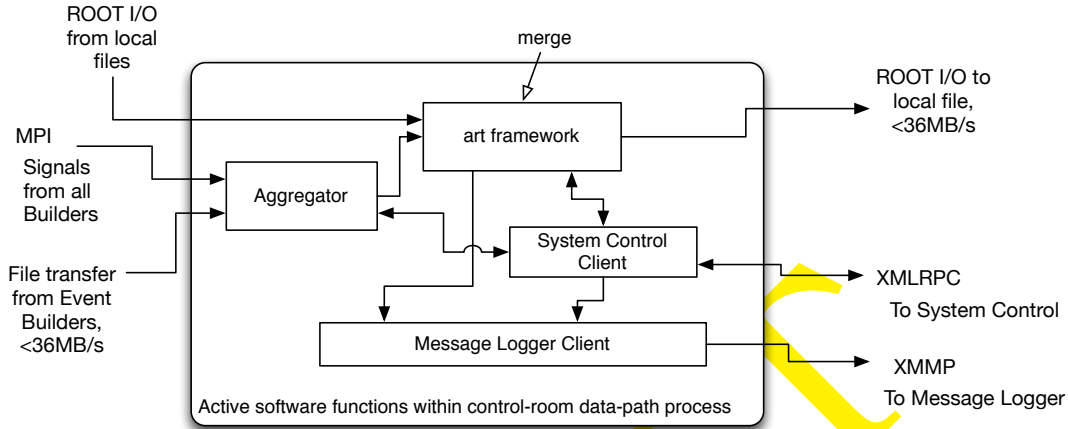


Figure 6: The functions and communication paths of the aggregator processes.

When an aggregator has transferred all N_{EB} data files corresponding to one run period, it then simultaneously reads all the files to merge the events into a single file of time-ordered, contiguous events.

4.2.5 messagefacility

The **messagefacility** package will be used to send all error, warning and informational messages from the DAQ software to the Message Logger. This package supports the addition of plug-in *message destinations*, which provide the protocol by which messages written using the package's interface are delivered to the places the messages are bound.

The destination used will support the XMPP protocol, and allow run-time configuration of the locations to which messages will be delivered. Specification of the Message Logger, which aggregates the messages, is outside the scope of this document.

4.2.6 Configuration facilities

We will use FHiCL[2] as the format for configuration information. In most cases, the FHiCL *document* (the textual specification of an independent portion of program configuration) will be communicated through the system control protocol (section 4.3.1).

4.3 Required communication protocols

4.3.1 System control using XMLRPC

The run control messages the system must respond to are:

1. initialization (configuration): `init`. This will contain the FHiCL document to configure the relevant system.
2. begin run: `start`. This will contain the run number.
3. end run: `stop`.
4. pause run: `pause`
5. status report: `status` The application will return a value indicating its status. The valid status are: ready, initialized, running, paused, and stopped. If there are multiple active subsystems within the application, each can report a separate status in addition to the overall process status.
6. current conditions report: `report`. The application will produce a report containing performance data and error data that it has collected. *FIXME: Does the information come back in the XMLRPC response or just an indication that the report has been shipped out to the message logger?*
7. reset statistics: `perfreset`. Reset performance numbers and error counters.

The format of the returned reports will always be a collection of name-value pairs.

4.3.2 MPI Program Start-up Protocol

The `pmt` program will accept the process layout file described in section 4.2.1. `Pmt` will use an underlying `mpirun` program to start data path processes on all participating nodes and will remain running while the entire distributed application is alive and operational. If any of the data path programs exit or abort, this program will first stop all the other data path programs, calculate a status code and return it.

Table 1: Example `pmt` configuration file.

<code>progFr</code>	<code>DSFR1</code>	11080
<code>progFr</code>	<code>DSFR1</code>	11081
<code>progFr</code>	<code>DSFR1</code>	11082
<code>progFr</code>	<code>DSFR1</code>	11083
<code>progFr</code>	<code>DSFR2</code>	11080
<code>progFr</code>	<code>DSFR2</code>	11081
<code>progFr</code>	<code>DSFR2</code>	11082
<code>progFr</code>	<code>DSFR2</code>	11083
<code>progEb</code>	<code>DSEB1</code>	11080
<code>progEb</code>	<code>DSEB1</code>	11081
<code>progEb</code>	<code>DSEB2</code>	11080
<code>progEb</code>	<code>DSEB2</code>	11081
<code>progEb</code>	<code>DSEB3</code>	11080
<code>progEb</code>	<code>DSEB3</code>	11081
<code>progAg</code>	<code>DSEB1</code>	11085
<code>progAg</code>	<code>DSEB2</code>	11085
<code>progAg</code>	<code>DSEB3</code>	11085

The configuration file example of table 1 will generate four unique fragment receiver processes for each of the two specified front-end nodes. It will generate two event builder processes on each of three specified computational nodes and one aggregator process

on each of three specified computational nodes. All processes will be brought up in an unconfigured *ready* state.

Multiple copies of *pmt* can be active at the same time. Each instance will be a distinct set of cooperating data path processes and will be treated as such. Each instance requires a unique set of ports assigned for command and control. Each piece of hardware can only be associated with one *pmt* instance.

4.3.3 Message Facility messages

Each message sent through the message facility contains a header with both user-supplied and system-supplied information. The logging API accepts a *severity* and *category* from the caller. The *severity* must be Error, Warning, Info, or Debug. The *category* can be any string and is used to classify the message. Routing and filtering are performed using the *category*. Summary statistics are also gathered using the *category*.

The header of each message contains the following information.

- timestamp
- host name
- host address
- severity
- category
- application name (optional)
- process ID (optional)
- current run and event number (optional)
- software module name (optional)

Whether or not the message headers contain the optional information is up to the application. Software frameworks such as art fill in the optional items.

4.3.4 Message Facility communications

FIXME This section is not complete. We still need to talk through the details.

```
<message
  from='fragment_receiver_3@dsfr2'
  id='ktx72v49'
  to='message_server@dsfr4'
  type='normal'
  xml:lang='en'>
<subject>Should this be used for the category?<\subject>
<thread>Maybe use the severity here?</thread>
<body>
  header
    timestamp=xxx
    hostname=xxx
    hostaddr=xxx
    severity=xxx
    category=xxx
```

```
extendedheader
  appl=xxx
  pid=xxx
  run=xxx
  event=xxx
  module=xxx
  Here is the body of the message.
</body>
</message>
```

Each of the running programs will be given a unique login name that will include the name of the application and the instance number (which can be a pid or a rank). The presence messages will be used to indicate the status of the application instance. The global message log listener will subscribe to all known application instance names.

4.3.5 File handling and formats

There are two types of output files: in-order event files and out-of-order event files. This system contains many event builder writing streams. Although each stream writes a time-ordered event stream, the streams will not write consecutive events. An out-of-order file does not contain consecutive events (in time, using an integral event counter). To produce the required file containing consecutive events, an n-way merge step is necessary to produce the in-order event files. The in-order event files will use the **art** ROOT output file format. The out-of-order event files can use any format that captures all relevant metadata and data products. We will be initially using the same format as the in-order files to shorten integration and deployment times.

The ROOT output file is a well-established format that is documented elsewhere and in use within several experiments. It can be simply described as a file of ROOT data product trees. Each product tree identifies the type of data it contains. In addition to the data trees, there are metadata tree, that describe the configuration of each of the software entities that produced or processed the data products. These files can be examined directly at the ROOT interactive prompt.

The out-of-order file names will be of the form “OO_Writer_Stream_Run_StartingSubRun.root”. The in-order file names will be of the form “IO_Writer_Stream_Run_StartingSubRun.root”.

FIXME Please make sure there is a requirement indicating that subruns cannot span files.

4.3.6 Data source

Describe the `FragmentGenerator` base class, and the use of the `Fragment` class.

FIXME This section needs to be completed.

4.3.7 art filter modules

Art module interface for `EDFilter` and `EDAnalyzer` and `EDProducer`. These interfaces are covered in the **art** tutorial.

4.3.8 Common configuration language

Program and **art** module configuration will be done using the FHiCL language and the **fhiclcpp** package.

FIXME This section should include a typical configuration for the fragment receiver process, the event building process, and the aggregator. It should also include a suggested management structure for the fhicl file fragments within the configuration system.

4.3.9 DQM protocol and event streaming

An **art** output module will be needed that is capable of packaging the event data for delivery over a network connection to the data quality monitoring application. The details of this protocol need to be worked out. An **art** application can be configured with additional DQM paths that will form DQM data streams. Paths within **art** can include filtering modules that can accept events. A simple standard filtering module is the prescaling module that can be configured to accept m out of n events. An output module can be attached to a path. Only events that pass the path are seen by the associated output module. In addition, an output module can be configured to only write a specific set of data products.

The **art** instance running inside the event builder sees all the raw data within each event. Summary statistics can be readily calculated and stored within each event alongside the compressed raw data. The system can be configured to supply several DQM data stream. Two important streams will be the prescaled compressed raw data stream and the summary statistics stream. It may be possible to send summary statistics at full rate to the DQM application.

There are two sources of DQM events: directly from the event builder and out of the aggregator. The event builder output can be delivered in real-time. The output from the aggregator will be delayed until there are enough events ready from all event builder to allow a merging operation to run efficiently.

FIXME If the real-time feed is desired, the event reader within the DQM application must be able to accept events from all event builders and buffer the data until consecutive event IDs are present.

5 Deployment and testing strategy

FIXME This section is not yet complete.

The deployment of the hardware will happen in two phases. The first phase is the installation and configuration at FNAL. The second phase is after shipping, installation, and configuration at Gran Sasso. This section provides details concerning the location of equipment and major connectivity, the location and organization of the application software, file systems, and operating system. Also contained here is a discussion of available tests on the various installations

There are four roles that were considered during the development of this section.

- Developer - someone who will be making changes to source code, compiling it, and running tests for it. Each developer will use his own account to perform this function.
- Operator - someone who will be using parts of the installed software for either production running or testing. There will be a shared account available to perform this function.
- Release Manager - someone who will be building products and installing them for use in one of the systems. There will be a shared account available to perform this function.
- System Administrator - all low-level system configuration and repair will be performed through the root account.
- Debugger - someone that needs access to many of the about functions.

There is an implicit assumption that nodes within a dashed box will be interconnected via 1Gb ethernet. This network will be used for IMPI (low-level remote administrative functions such as reboot), operating system administration, and login terminals.

The Infiniband network (IB) can carry TCP/IP traffic by using the IP over IB driver. A TCP/IP network interface will be enabled to permit standard high-speed communications over the IB network in addition to the ethernet interfaces.

There will be an NFS filesystem configured and running on one of the DAQ nodes. This NFS filesystem will hold all of the released software products that are not part of the standard Linux platform. This includes artdaq, art, and all the products they depend upon. This NFS filesystem will also contain the user account home directories. This filesystem will be mounted on all of the DAQ nodes.

Infiniband requires a software component to be active called the *subnet manager*. The primary subnet manager will be located on one of the Fragment Receiver computers. A secondary back-up subnet manager will be located on one of the computational nodes in the control room.

5.1 Phase 1

Figure 7 depicts the configuration of the DAQ system components during initial configuration at FNAL.

FIXME Important things to note here: first round of installation uses the currently-running grunt IB switch, all nodes are configured the same way using dsfr1 as the NFS file server.

5.2 Phase 2

Figure 8 depicts the configuration after shipping the major DAQ system components to LNGS.

FIXME Before bringing up the nodes at Gran Sasso, the subnet manager will be installed and configured as shown. The LNGS development nodes will be severed from the FNAL network and configured with an NFS server and subnet manager for independent use. The DAQ nodes remaining at FNAL will be attached to the existing IB equipment. The DarkSide-50 analysis node will be used as a gateway for login access.

6 Additional Figures

These figures may help describe the system design:

Figure 9 shows a schematic of the overall DAQ system.

Figure 10 shows a schematic of the connections and their protocols between various processes.

Figure 11 shows a schematic of the trigger architecture.

Bibliography

- [1] The MPI specification is available at <http://www.mcs.anl.gov/research/projects/mpl>.
- [2] FHiCL is the Fermilab Hierarchical Configuration Language; the FHiCL home page is <https://cdcvcs.fnal.gov/redmine/projects/fhicl>.

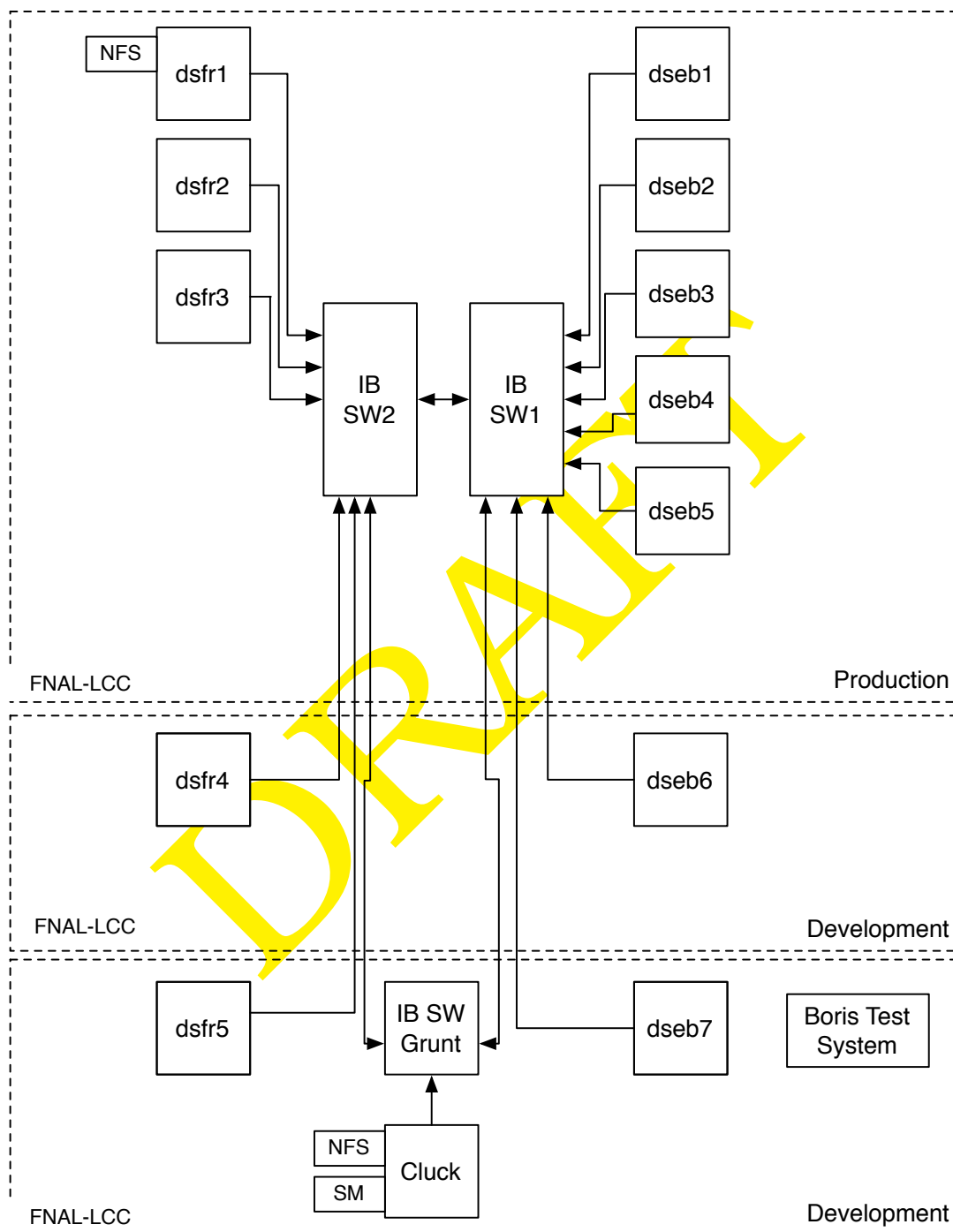


Figure 7: The computing hardware organization for phase I.

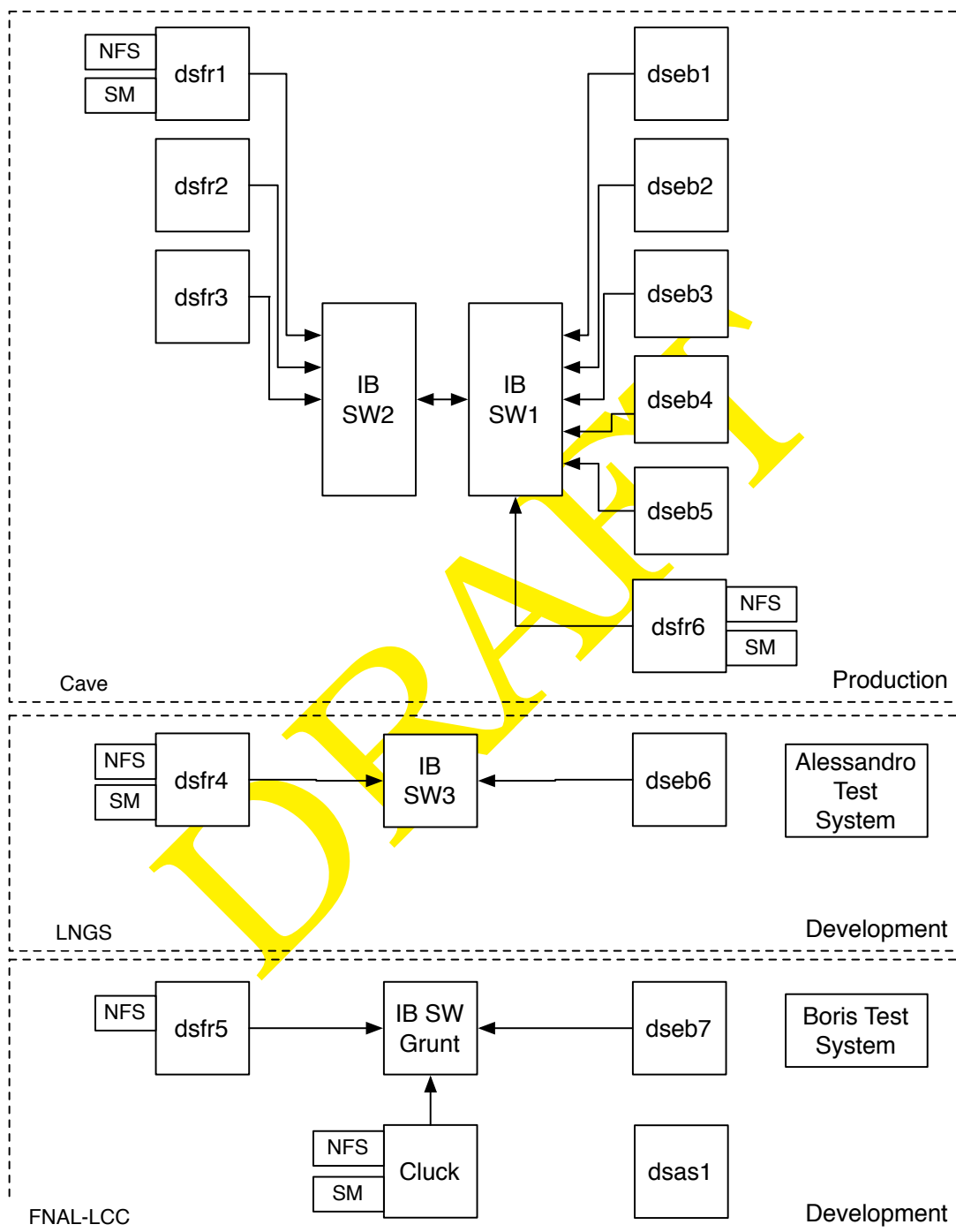


Figure 8: The computing hardware organization for phase 2.

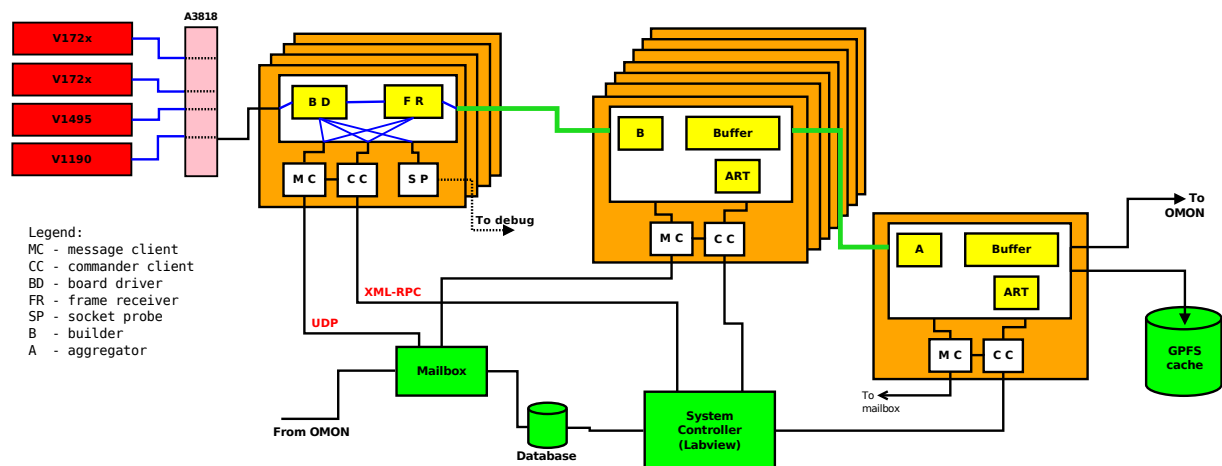


Figure 9: DAQ Components with art Framework.

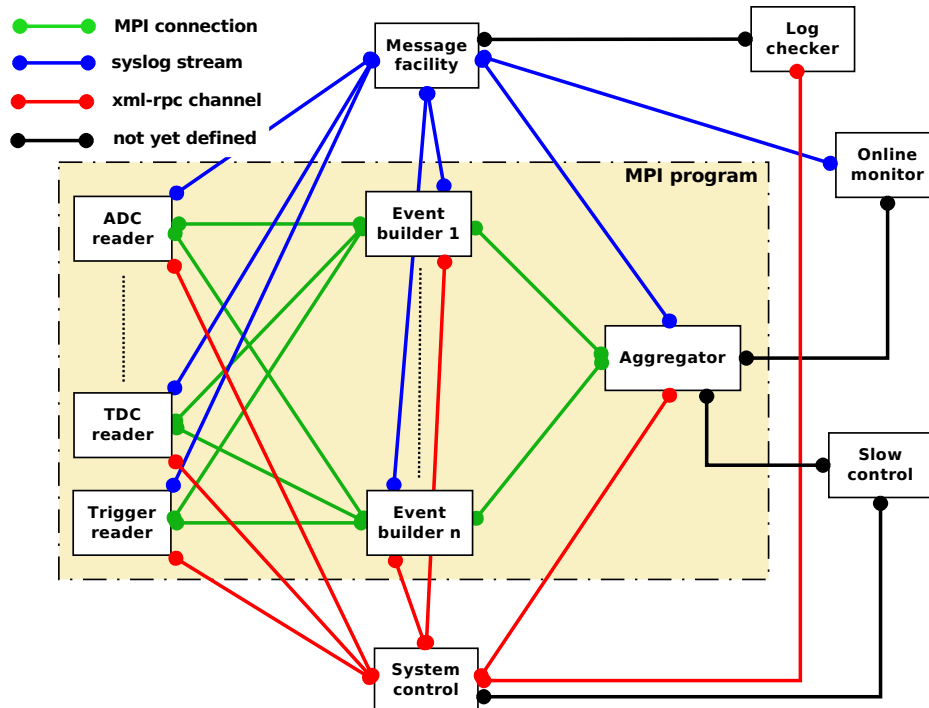


Figure 10: Communication Paths and Protocols.

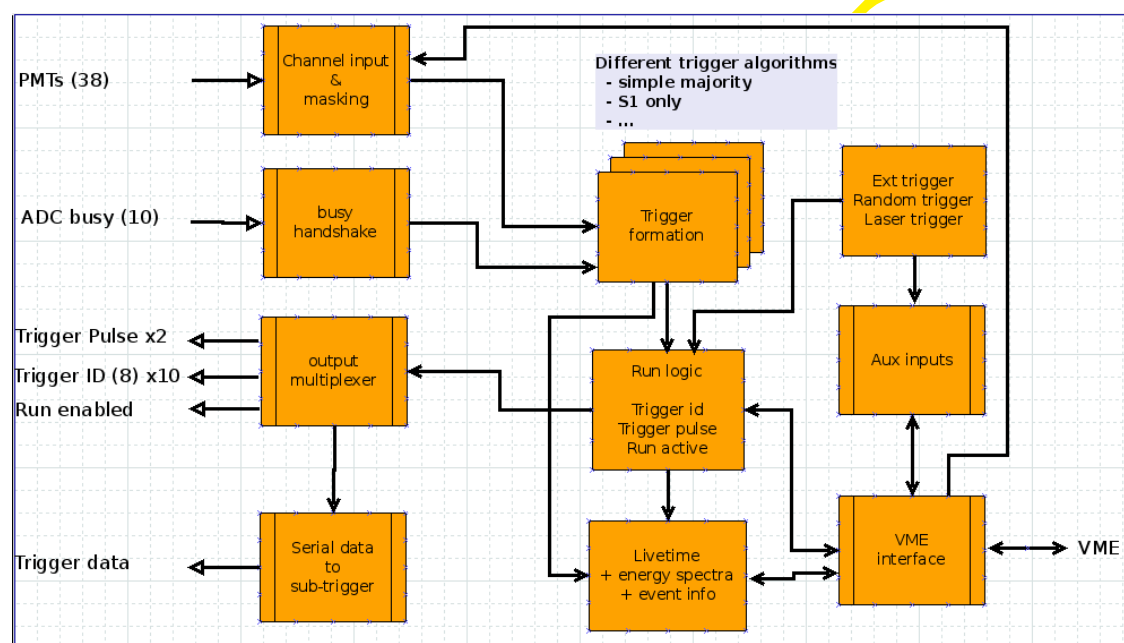


Figure 11: Trigger structure