Journal of Physics: Conference Series **119** (2008) 022023 doi:10.1088/1742-6596/119/2/022023

The LHCb High Level Trigger Infrastructure

M.Frank[1], C.Gaspar[1], E.v.Herwijnen[1], B.Jost[1], N.Neufeld[1], S.Cherukuwada[2], R.Stoica[3]

Markus.Frank@cern.ch

Abstract. The High Level Trigger and Data Acquisition system of the LHCb experiment at the CERN Large Hadron Collider must handle proton-proton collisions from beams crossing at 40 MHz. After a hardware-based first level trigger events have to be processed at the rate of 1 MHz and filtered by purely software-based trigger applications executing in a high level trigger farm consisting of up to 2000 CPUs built of commodity hardware (HLT). The final rate of accepted events is around 2 kHz. This contribution describes the architecture used to host the selection algorithms of the high level trigger on each trigger node, which is based on shared memory event buffers. It illustrates the interplay between event building processes, event filter processes and processes sending accepted events to the storage system. It describes these software components that are based on the Gaudi event processing framework.

1. Introduction

The Large Hadron Collider (LHC) at CERN is scheduled to begin operation in 2008 and will deliver proton-proton collisions at a centre of mass energy of up to 14 TeV to the LHCb detector [1] at a rate of 40 MHz. This data rate is reduced with a two-level trigger system to roughly 2 kHz of particle collisions to facilitate the handling of these data with the available computing and long-term data storage resources. The first level trigger, which reduces the rate of accepted events to 1 MHz, is hardware based and located in the frontend electronics. The second level or High Level Trigger (HLT) is purely software based. The infrastructure to access event data, to collect error messages and to control the HLT will be discussed in the following sections.

2. Environment

The HLT hardware consists of up to 2000 processors, which are grouped into 50 subfarms of up to 40 nodes each. As shown in figure 1, the subfarms are connected to the main readout switch, which forwards the data from the frontend readout boards to the HLT nodes. Each subfarm, its nodes and the processes executing within, is controlled by a subfarm controls node. These nodes are connected to the experiment control system, which also collects all monitoring information of these processes. A strict separation between the information flow and the control flow has been applied.

An attempt has been made to design an open architecture where the basic building blocks of the data acquisition are kept as independent as possible. These building blocks include implementations of

[1] CERN, Geneva, Switzerland

[2] This research project has been supported by a Marie Curie Early Stage Research Training Fellowship of the European Community's Sixth Framework Programme under contract numbers MEST-CT-2004-007307-MITELCO and

[3] MEST-CT-2005-020216-ELACCO

functions to abstract differences between operating systems, the buffer manager and the network data transfer mechanism.
This gives flexibility in tailoring the applications necessary for event filtering, data storage and data monitoring. With these basic building blocks, all participating tasks are built to comply with the Gaudi component architecture [2] used also in the LHCb offline environment to perform data analysis after the data are written to long term storage. The possibility of either using existing Gaudi components or replacing offline components with implementations specialized for the online environment during the data aquisition facilitates:

- the customization and parameterization of the various processes,
- transparent access to event data like in the offline environment,
- redirection of monitoring entities like histograms and output logging,
- the embedding of the HLT filter code, which is developed in an offline environment.

To include the required functionality for the online environment, specialized components were developed to assemble fragments of event data sent by the front-end readout boards (TELL1 boards) connected to the sub-detectors, to transfer event data originating from single particle collisions between the different processing elements and to allow event processing applications to access event data in a transparent way. The applications are monitored using a Gaudi- compatible online monitoring component [3]. Operating system dependent code fragments were encapsulated to support both, the Linux and Windows platform.

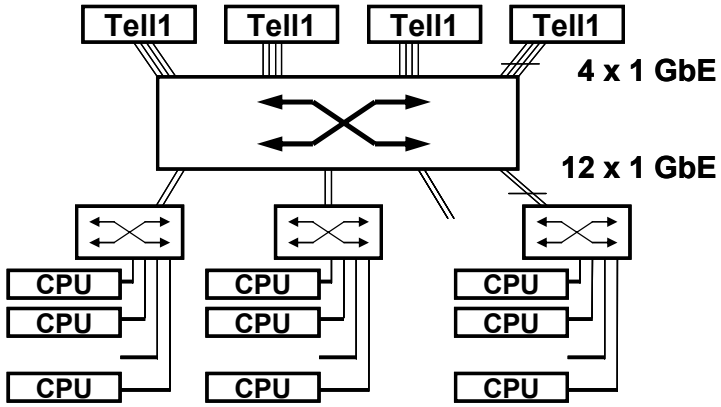


Figure 1: The HLT hardware layout. The Tell1 readout boards connected to the readout network send the event data to up to 2000 processing elements hosted by maximal 50 subfarms.

3. Data Processing Architecture

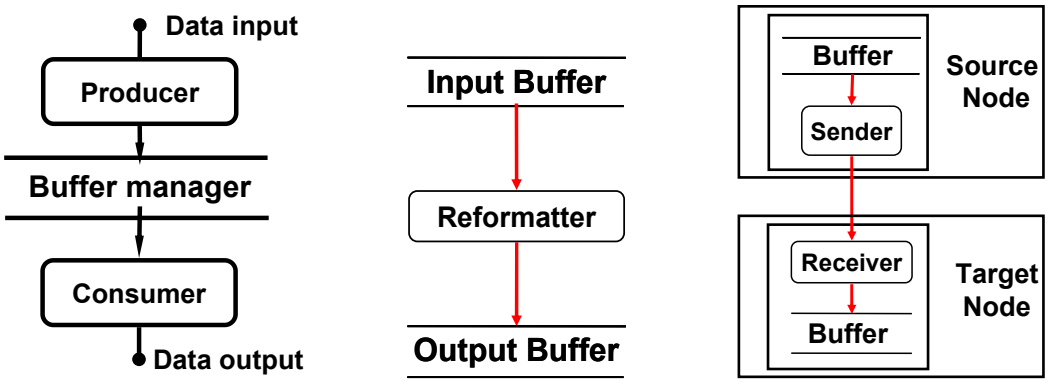
The basic pattern of the task architecture in each processing element is shown in figure 2a [4]. Every readout function is separated into independent tasks running asynchronously in order to derandomize the flow of events. The producer task is responsible for reading/assembling data from data sources. As soon as the read operation is finished, the data block is declared to a managed shared memory area, the buffer manager (BM) and the producer is ready for the next read operation. The consumer task is activated each time an event is declared in the BM. The consumer is responsible of processing the data and the releasing of the space occupied as soon as it has finished. Thus, the two activities of reading and processing can proceed asynchronously, provided there is always sufficient space in the buffer to accommodate at least one read operation. The BM is managed by a dedicated task executing on each node, which creates and initializes the shared memory area. The synchronization between the various participating processes has been realized using shared semaphores.
Any task using the BM registers itself with a name and a partition identifier. Consumers will only receive data declared by producers with the same partition identifier and a boolean accept and veto

mask of a width of 128 bits, which allows them to be specific in the type of data they wish to receive. Consumers may request data from the BM in three different ways:

- The consumer sees all data declared to the BM according to the request mask.
- A group of consumers see the data exactly once. This mode allows executing multiple instances of an application and taking advantage of multiple CPU cores.
- A consumer is served on a best effort basis. In this mode producers are always able to declare new data. Possibly pending data, which has to be released to satisfy the space requests, will not be seen by these consumers.

The other building blocks can be constructed by connecting different data sources or data sinks to the input and output data connections of producers and consumers:

- Connecting the input of a producer to another buffer manager yields a *Data Reformatting Unit* (Figure 2b). The reformatting process consumes data from an input buffer, then transforms or copies the data and declares the result into an output buffer. Not all events received from the input buffer necessarily need to be declared in the output buffer. Hence, this unit is ideal to perform HLT event filtering. The output buffer and the input buffer may not be identical to avoid deadlock situations.
- Connecting the output of a consumer to the input of a producer on a remote node suing a network connection yields a *Data Transfer Unit* (Figure 2c), used to evacuate the accepted events from the HLT farm to the storage system.



(a) **(b)** **(b)**

Figure 2: The components of the basic building block for HLT data processing (a) allows to construct data reformatting units (b) and data transfer units (c) capable to transfer data between buffer managers on two nodes.

With these building blocks the process and buffer architecture in an HLT node was constructed as shown in Figure 3. The readout boards send data blocks containing fragments of multiple events, called *Multi-Event Packets* (MEP), containing the detector response of several beam interactions accepted by the first level trigger to the *Event Builder* processes (EB) executing in each node. The EB assembles the fragments into a contiguous block, a *Multi Event* and deposits the events in the "MEP" BM. After the assembly of a multi-event is finished, the EB sends an event request to the trigger supervisor, which steers the data transfer between the front-end boards and the EB. Typically 10 events are sent simultaneously to reduce the overhead of the transport protocol like Ethernet headers etc. The data payload has a bank like format with a common bank header and subdetector specific payload. The *Event Producer*, subscribed to the "MEP" BM analyses the data format and builds descriptors to all banks representing one single event (Figure 4), which are declared to the "EVENT" BM. Event descriptors are very small compared to the data and provide a way to avoid unnecessary copies of event data. This optimization is important, because the contribution of one frontend source to the total event size of 30 kB accounts for typically 100 Bytes and unaligned memory copies of very small pieces of data are avoided [5]. The event filter processes (*Moore*) access the event data using the

descriptors and compute the decision if the event is to be kept. On a positive decision the descriptor data are copied to the "RESULT" buffer, where the *Sender* process collects the individual banks into contiguous memory and sends the data to the storage system. At the time the filter processes declare the accepted events, they also specify the trigger mask which later allows tasks in the storage or the monitoring system to classify the events according to basic physics properties without analyzing the data content. Data in the MEP buffer are blocked using a reference count mechanism until all consumer requests to access event data using the descriptors are satisfied.

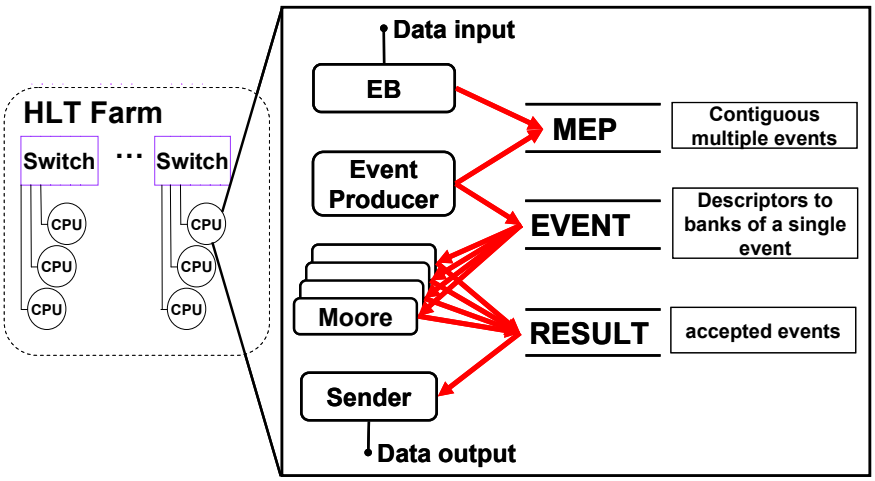


Figure 3: The different flow of event data through different buffer managers between the tasks executing in a HLT farm node.

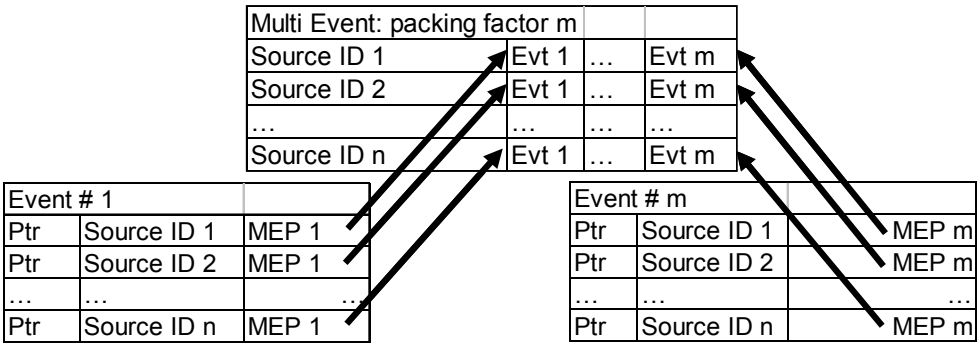


Figure 4: The event descriptor mechanism. Multiple events are assembled in a contiguous block. The descriptors referencing single events are then built by the Event Producer process.

4. Output Logging

The processes executing in the HLT farm produce log messages indicating error conditions or additional information for debugging. These messages in the offline environment are sent to standard output, an approach, which is not feasible in the online environment, where many task instances execute in a highly distributed system. Detector problems spotted by the event filter tasks however are likely to be seen and reported by many filter processes. Hence, additional filtering is required before messages may be presented to the operator who wants to be notified by one message describing the problem and not an avalanche of identical messages hiding possible other errors. Figure 5 describes the hierarchical approach, which is used to collect and filter output messages from all processes contributing in the HLT. Similar collection and filtering mechanisms are then applied to sample the

output of all nodes in a subfarm and finally to collect all messages from the entire HLT farm. Messages passing the final filter are then published and can be presented to an error logging screen. Message filtering is possible both, at the level of each reporting process, where the Gaudi infrastructure allows to e.g. suppress messages occurring several times and at the Error Server, which may block identical messages resulting from independent processes.
Each process on the HLT node publishes messages using the standard Gaudi output logging facility. Standard output/error streams are collected by the process management infrastructure and published by a dedicated *Error Server* process [6] using the DIM protocol [7]. At the level of the subfarms and the HLT the published information is collected similarly from the lower levels.

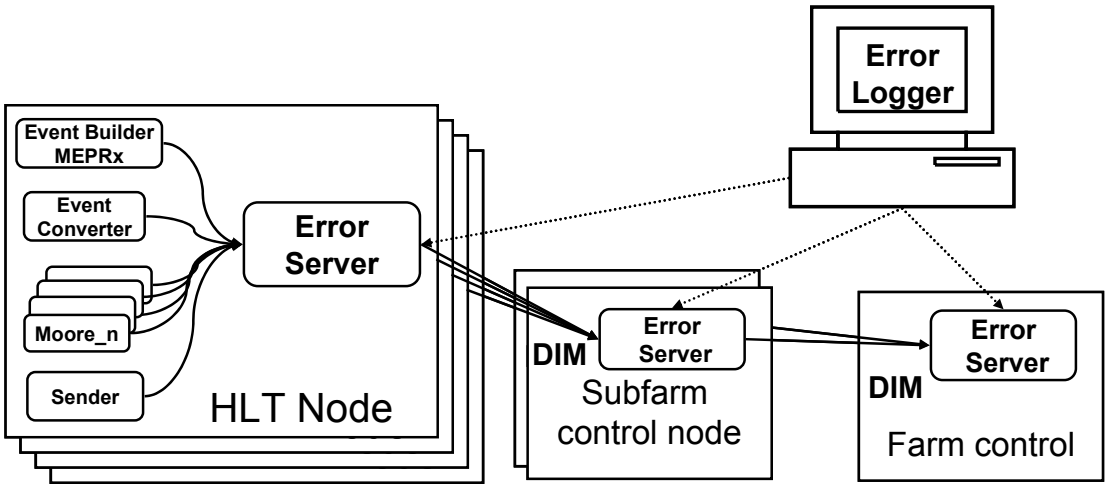


Figure 5: The output logging hierarchy. Messages are first collected and filtered at each HLT farm node. Then messages are collected from all nodes of a subfarm and finally from all subfarms to be presented to the operator.

5. Control Flow

Data taking normally proceeds in cycles. Firstly all hardware components and processes participating are configured. This is followed by a data taking period and finally a closedown phase. The system used to control data taking must be capable of sequencing these actions to ensure that each component has been set to its correct state at each step of the cycle. In developing the control architecture it was found convenient to model the entire system in terms of a Petri-Net in which the behavior of each component is represented by a finite state machine [8]. This consists of specifying the possible states in which each component can be found and the possible transitions between these states. Each transition is characterized by a condition, which causes it to be invoked and an action which should be performed on making the transition. Each transition can be made dependent on another component being in a particular state, which allows constructing a hierarchy of controlled components.
The control of these tasks was realized with a tree-like architecture, where firstly all processes in an HLT node are grouped, then nodes are grouped to subfarms and finally subfarms to the HLT farm [9]. Figure 6 shows the model of the finite state machine executed by each task.
All processes in the HLT farm must be started in a well defined way to ensure that infrastructure dependencies such as buffer managers are present and network ports are open before clients try to use them. Table 1 shows the actions that each task type performs during the transitions shown in Figure 6 to ensure coherent access to the required resources.

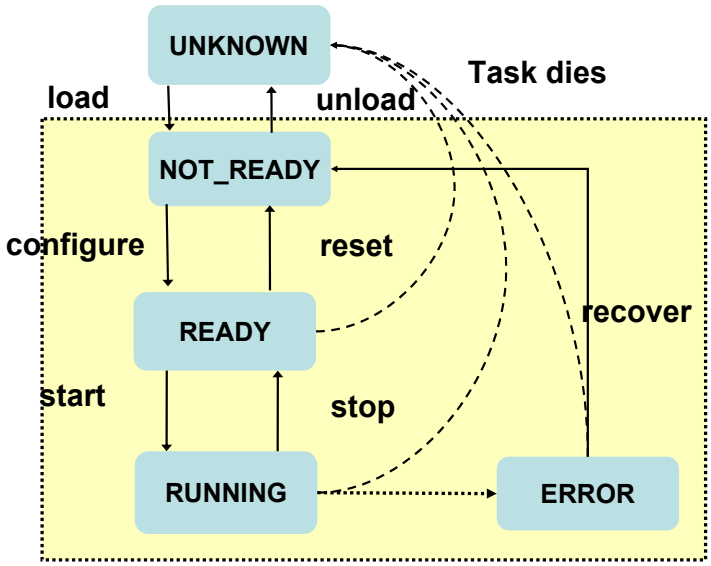


Figure 6: The state diagram of the processes executing in an HLT farm node.

Task type \ Task state	load	configure	start
Buffer managers	Create BM		
Error Servers	Listen to messages		
EB, HLT filters & data reformatters		connect to BM	
Network Receivers		connect to BM open data port	
Network Senders			connect to BM connect to data port

Table 1: The different types of processes executing in the HLT farm and the actions each task performs at a given transition.

At the highest level the operator issues commands to the Run Control, which is responsible for the overall orchestration of the state transitions. The Run Control as shown in Figure 7 solely deals with macroscopic entities like units representing subdetector hardware or a control unit called *DataFlow*, which represents all Gaudi based processes participating in data taking. Internally the *DataFlow* unit consists of the sub-units *Monitoring* [10], *Storage* [11] and the HLT farm. The controls hierarchy was constructed using the commercial SCADA system PVSS [12], and SMI++ [13], a state manager interfaced to PVSS. The Gaudi based processes accept transition requests from SMI++ and publish their state to SMI++ using the DIM protocol.

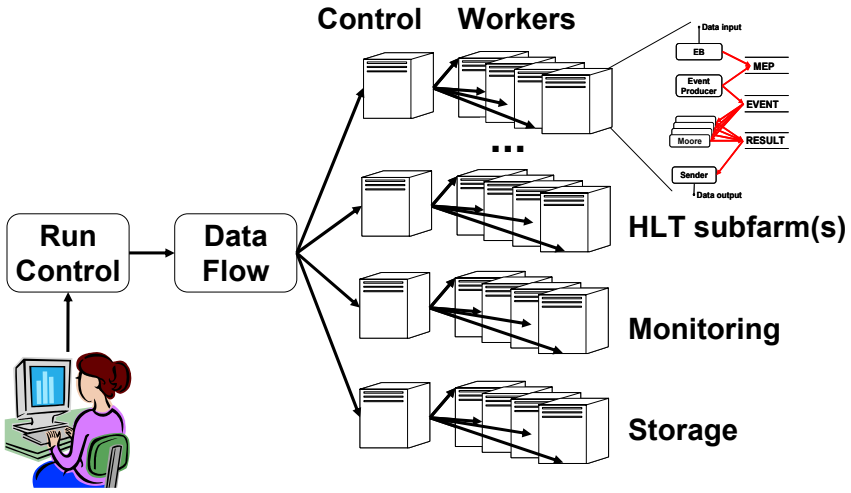


Figure 7: The view of the control hierarchy as seen by the operator.

6. Conclusions

Two techniques, the buffer manager concept and a dedicated network library allowed us to construct in a modular way all components to enable HLT event filter processes executing in the LHCb online environment. The filter processes, which configure both in the offline and online environments dynamically, load the components without the need of modification and hence allow for a seamless coexistence of the development process of the event filter code and its deployment in the HLT farm. The integration of all participating processes into the state manager which also configures and manipulates the experiment hardware allows for a coherent control of the data taking process.

References

[1] LHCb Collaboration, "LHCb Technical Proposal", CERN-LHCC-98-04, February 1998.
[2] G. Barrand et al., "GAUDI - A software architecture and framework for building LHCb data processing applications", CHEP 2000, Proceedings, Padova, Italy, February 2000.
[3] P.Vannerem et al., "Distributed Control and Monitoring of High Level Trigger Processes on the LHCb Online Farm", ICALEPCS'2003, Gyeongiu, Korea, October 2003.
[4] A.Belk et al., "DAQ Software Architecture for ALEPH, A Large HEP Experiment", IEEE Trans. On Nucl.Sience, Vol 36, No.5, Oct 1989
[5] B.Gaidioz et al., "Low Level Gigabit Ethernet Analysis for the LHCb Computing Farm", LHCb 2005-091
[6] D.Galli et al., "The Message Logger for the LHCb On-Line Farm", LHCb 2005-050 DAQ
[7] C.Gaspar et al., "DIM, a Portable, Light Weight Package for Information Publishing, Data Transfer and Inter-process Communication", CHEP 2000, Padova, Italy
[8] J.L.Peterson, "Petri Nets", Computing Surveys, Vol 9, No 3, September 1977
[9] E.van Herwijnen, "Control and Monitoring of on-line trigger algorithms using Gaucho", CHEP 2006, Mumbai, India, 2006
[10] O.Callot et al., "Data Monitoring in the LHCb Experiment", CHEP 2007, Victoria, BC; these proceedings
[11] M.Frank et al., "Data Stream handling in the LHCb experiment", CHEP 2007, Victoria, BC; these proceedings
[12] ETM, "PVSS II, Version 3.0", Eisenstadt, Austria, 2004.
[13] C.Gaspar et al., "SMI++ - Object Oriented Framework for Designing Control Systems for HEP Experiments", CHEP 97, Berlin, Germany, 1997