Online testbench for LHCb High Level Trigger validation

M. Frank\textsuperscript{1}, J. Garnier\textsuperscript{1}, C. Gaspar\textsuperscript{1}, G. Liu\textsuperscript{1}, N. Neufeld\textsuperscript{1},
A. S. Varela\textsuperscript{1}

\textsuperscript{1}CERN, Geneva, Switzerland

E-mail: jean-christophe.garnier@cern.ch

Abstract.

The High Level Trigger (HLT) and Data Acquisition (DAQ) system selects about 2 kHz of events out of the 40 MHz of beam crossings. The selected events are consolidated into files on an onsite storage and then sent to permanent storage for subsequent analysis on the Grid. For local and full-chain tests a method to exercise the data-flow through the High Level Trigger when there are no actual data is needed. In order to test the system as much as possible under identical conditions as for data-taking the solution is to inject data at the input of the HLT at a minimum rate of 2 kHz. This is done via a software implementation of the trigger system which sends data to the HLT. The application has to simulate that the data it sends come from real LHCb readout-boards. Both simulation data and previously recorded real data can be re-played through the system in this manner. As the data rate is high (100 MB/s), care has been taken to optimise the emulator for throughput from the Storage Area Network (SAN). The emulator can be run in stand-alone mode or run as a pseudo-subdetector of LHCb, allowing for use of all the standard run-control tools. The architecture, implementation and performance of the emulator will be presented.

1. Introduction

LHCb \cite{1} is one of the four large experiments, which will start taking data in 2009, when the LHC starts operation at CERN. LHCb is dedicated to the study of CP-violation, a subtle asymmetry in the fundamental laws of nature, which is thought to be responsible for the striking absence of any anti-matter in the observable universe. The effect of CP-violation can be particularly well studied in decays of particles, which contain a b-quark. The distinguishing characteristic of b-decays is the relatively long lifetime of these particles, which allows them to fly a measurable distance from the original collision point before they finally decay. LHCb selects events mostly by reconstructing these secondary decay vertices. This reconstruction requires a large fraction of the detector data, which means that LHCb must read out the detector $10^6$ times per second; 10 times more often than the other LHC experiments. On the other hand the total event size, determined by average number of electronics channels activated by a collision, is quite small: 35 kB according to estimates from simulation. The data acquisition and data handling in LHCb are therefore dealing with a very high rate of rather small events.

The purpose of this paper is to present the tool which emulates the LHCb detector and which is used to perform commissioning and validation tests of the complete online and offline
system, from the Data Acquisition System to the Event Reconstruction. In order to understand how this project was carried out, it is necessary to have a rough understanding of the DAQ. The operations to trigger the acquisition are usually:

- Readout boards [2] get data from the detector and build Multi Event Packets (MEP). A MEP is a buffer which stores from 1 to 15 part of events [3].
- A HLT [4] farm node sends a MEP request to the Timing and Fast Control supervisor (TFC [5]), in order to tell that it is available for data processing.
- The TFC selects an available HLT node. It sends information via a TTC [6] interface to the readout boards and via a MEP to the destination node. It indicates, among others, the destination node.

Figure 1 gives a schematic view of the LHCb DAQ and the way data flows top-down.

First, the requirements of such a system will be presented, then the constraints and the specification of the system, its architecture, the tests which have been performed, the quest for performance optimization and its use in the Full Experiment System Test (FEST [7]).

2. Requirements
LHCb teams are scheduling Full Experiment System Tests for commissioning the online and offline systems. They want to test, commission and validate the various monitoring, data quality, alignment tools which are used in the complete processing, performing correctly at the expected
Figure 2. The LHCb Data Injection: A new data-flow comes from a software emulator implemented on commodity hardware. Despite its lower throughput, it is a perfect copy of the detector data-flow.

The online output rate of 2 kHz. Normally data from the first LHC beams would have been used for testing and validating the full experiment system, but as more time is now available before real data taking starts, it has been decided to use a “data injector” to perform these indispensable tests. It acts as a detector emulator.

This is why the module is integrated into the DAQ, as a new data input (Figure 2). The HLT farms get data as if they were coming from the readout boards. Then, every part of the process, starting from the HLT, can be tested: Monitoring, file selection and transfer, data quality, etc. It needs also to be fully integrated into the Experiment Control System, and it should be usable in parallel to other detector tests.

The aim is not to perform highspeed performance tests of each part of the processing. It is to test the various features of the DAQ and HLT for long term runs. Therefore the injector reads simulated physic data files. In order to make various test runs, the injector should be able to mix different input data streams according to experiment control triggers.

The event rate needed for input to the HLT is about 2 kHz. This calls for a throughput of about 70 MB/s.

All these requirements impose constraints on the software implementation and on hardware operation.
3. Architecture

To address these requirements, a multithreaded software has been implemented on a powerful dedicated node of the DAQ network. This software has been developed using the standard LHCb C++ Gaudi framework [8].

It consists of a set of Gaudi services, which are normal Unix processes, and of a set of POSIX threads. The Gaudi framework and a set of add-ons allow the injector to interpret commands which it receives from the experiment control system. Moreover, it allows the implementation to re-use existing solutions, for example: event reading from files.

The injector operation is shown in Figure 3. The injector itself needs TFC information, but unlike the readout-boards, it cannot be connected directly to the TFC TTC interface. Therefore the only way to get the TFC information is to retrieve the MEPs from the TFC before to build the MEPs from the readout boards. This is achieved implementing the injector like an interface between the TFC and the HLT nodes. Basically, the injector pretends to be a TFC board to the HLT farms in order to record their MEP requests. In parallel, it pretends to be a HLT farm to a TFC board, in order to forward it the MEP requests and to receive the TFC MEP which contains run information. Once the injector received this MEP and read enough event from tape, it builds MEPs the same way any TELL1 board would do. Then it sends each of the 300 MEPs to one of the requesting HLT nodes. It modifies also the IP header to simulate the real readout board, which would have build normally the MEP.

To increase performance, the various injector interfaces need to be managed asynchronously. Therefore, the injector software will rely on different threads and processes which will share their information using inter process communication tools. Figure 4 shows the software architecture, the main process and their way to communicate.

The injector runs two dedicated threads to handle its network interfaces with the TFC and with HLT farm nodes. It allows the injector to manage HLT requests and TFC answers as fast as possible. TFC MEPs are stored in a FIFO if they cannot be consumed immediately, and MEP requests are stored in a map, so the injector knows how many events have to be sent to each requesting farm.

---

Figure 3. Integration in the DAQ System: The Injector pretends to be a readout supervisor to the HLT nodes, and it pretends to be a HLT node to the readout supervisor. It converts simulated events from the storage to the network protocol.
The injector reads events from files. Events are encoded in a format called Master Data File (MDF) and it is necessary to convert them to the MEP format. It is a rather time and resource consuming task because of the software and hardware operations implied in the procedure.

It is better that these hardware consuming operations do not have to wait for each other. Therefore the processing has been split in two:

- A task dedicated to read events from files and to produce them into a buffer.
- A task dedicated to consume events, build MEPs and to send them to the network.

The separation of the MEP building from the sending would have required another synchronization and buffering layer. Once the MEPs would have been produced, it would have been necessary to copy them to another buffer in order to produce the next MEPs while the next processing step was also working. Another way would have been to manage 305 circular buffers. Anyway, this additional buffering layer would have needed specific synchronization and memory.
managements, which would have cost, in some cases, more than just sending the produced MEPs directly. Therefore, MEPs production and sending are made by a single thread.

The task which reads events from files is a distinct Gaudi process. This choice allows the input of the injector to be easily configurable, in order to get several sets of files as input. This allows also for mixing events according to the TFC trigger types. The injector needs at least one reader, and can use up to 8.

The end of the processing is the following. A TFC MEP is consumed from the FIFO in order to get run information, and a requesting farm node is selected. According to the run information it got from the TFC the injector selects a buffer and consumes events, produces the MEPs and sends them to the selected HLT node.

4. Implementation

Data used as injection input are located in the LHCb Storage Area Network (SAN). This SAN also provides a NFS export. Therefore there are 2 ways for reading data: NFS over Gigabit Ethernet and CentraVision File System (CVFS [9]) over fiber channel.

The injector needs to send events to the HLT farm at a rate of 2 kHz. Since the injector uses a small pipeline, care had to be taken in order to not slow down the injector due to slow processing in the first stage.

Consequently, several tests were performed, using the two available system configurations and various implementations of the reading algorithm. To obtain clean measurements, algorithm performances were tested by reading from an in memory file system. Performance tests were made using Linux monitoring tools and the following readers:

- The dd command.
- The Gaudi “EventSelector” service, which reads input files and produces event objects.
- A C/C++ reader in which buffering is improved. It handles events as memory buffers instead of C++ objects.

First tests were made to choose which media would be used to access the storage system. For example, reading 2 MB chunks of 5 GB files is performed at the wire-speed of 100 MB/s with NFS, and at the speed of about 200 MB/s with CVFS. According to the results, the CVFS over Fiber Channel access to the storage has been chosen.

The standard EventSelector reads event data one by one and build C++ objects for each encapsulation layer, which are stored in a transient store, before they are used. In the reading process the access to the storage hardware is not optimized. The best frequency measured were 1902 events per second with NFS and 2450 events per second with CVFS. For sure, C++ object processing is easy, elegant and reusable.

However, there are two possible improvements. Firstly, the hardware access: Instead of reading events one by one, it is possible to read a large chunk of memory and then find events in it. Secondy, the instantiation of each event is a waste of time according to the aim of the injector, as this one mainly fills in raw memory buffers.

Therefore, a dedicated reader which follows these improvements has been implemented for the injector. It gives much better performances: 200 MB/s via CVFS. It represents about 5800 events/s. This dedicated reader has been integrated in a Gaudi Service, in order to get the flexible architecture presented before. However if several readers are working on different sets of files in order to produce events in the buffers, performance can be decreased at some point, as shown in Figure 5. This comes from hardware limits of the Storage Area Network. Anyway, it
Figure 5. Storage Access Performances via Fiber Channel, using from 1 to 8 dedicated reader process, accessing different set files.

is ensured to get good performances while the injector is using a reasonable number of readers. The injector input rate will not be a limit to the global process performances.

A normal test setup will use between one and three readers, one will provide the main information at a rate of 2 kHz. The two others will bring particular information at a lower rate, for example around 100 Hz of signal or charm simulated events.

The injector has to send MEPs as if they were coming from readout boards. This given rise two problems.

300 MEPs are built by the injector. As the DAQ protocol is IP, each MEP is identified by the IP address of the readout board which would have make it. However, the event read from the storage are made of Raw Banks [10]. A Raw Bank contains the type of the subdetector which is connected to the readout board, and a source channel identifier. From these two values, the injector compute a unique readout board IP address.

Subsequently, building MEPs is pushing each bank in the corresponding memory buffer. When the injector has read enough events, the MEPs can be sent.

The injector has to send MEPs as if they were coming from real readout board. Therefore it has to access to the IP datagram header in order to write the value of the source IP field. It is done using a raw socket with specific options to ask to manage the IP header in our software layer.

However this disables the IP fragmentation for datagrams which are bigger than the Maximum Transfer Unit (MTU) of the network interface. Though the LHCb DAQ allows the use of jumbo frames, once the MEP is completely built, its size can be bigger than the configured MTU (9000 Bytes). Therefore the injector has to implement the IP fragmentation [11] itself.

In LHCb, raw data are stored on a Storage Area Network [12] which is accessible from the Event Filter Farm via a dedicated Ethernet Local Area Network. As the storage LAN is physically separated from the DAQ LAN, to ensure maximum robustness and good capability, a special network configuration using dedicated routing paths has to be put in place, in order to make the injection of data residing in the storage network into the DAQ network possible.

The output of the algorithm after the complete processing with a maximum input rate is about 160 MB/s. It represents a rate of about 4.6 kHz, which is more than twice the requirement of
2 kHz.

5. Full Experiment System Test
The injector was developed for use in system test with the complete experiment. At least one week per month since January 2009 is a FEST week, and it will carry on at least until the LHC start-up and maybe be held during LHC shutdown. Experts from both offline and online teams are working together to achieve the commissioning and the validation of the data processing and its monitoring.

The injector implements a Finite State Machine which is controlled with a dedicated LHCb Run Control [13] partition. It can be used easily with all other subsystems as shown in Figure 6. In parallel, other partitions for normal detector runs or sub-detector runs can be used.

On dedicated panels for the injector, the user can configure among other items the reader input files, the number of readers with their corresponding trigger type, and the number of events per MEP. The run configuration information is set up with Run Information panels and with the TFC panels, like in a normal data taking runs.

FEST has been iterated already two times, and it will carry on. First results and achievements were presented in the poster [7] session.

So far, the injector has played a key role in the validation of the LHCb offline and online system. It is constantly used throughout the FEST weeks to validate also single project upgrades.
Acknowledgments
This research project has been supported by a Marie Curie Initial Training Network Fellowship of the European Community’s Seventh Framework Programme under contract number (PITN-GA-2008-211801-ACEOLE)

References
[1] The LHCb Collaboration, A Augusto Alves Jr et al., “The LHCb Detector at the LHC”, 2008 JINST 3 S08005.
[2] A. Bay et al., “The LHCb DAQ Interface Board TELL1”, Nucl. Inst. and Methods, 2006.
[3] B. Jost and N. Neufeld, “Raw-data Transport Format”, CERN, EDMS 499933.
[4] “LHCb HLT homepage”, http://lhcb-trig.web.cern.ch/lhcb-trig/HLT
[5] “LHCb TFC homepage”, http://cern.ch/lhcb-online/TFC
[6] “TTC homepage”, http://ttc.web.cern.ch/TTC
[7] M. Cattaneo, “LHCb Full Experiment System TEST (FEST09)”, Proc. CHEP09, Prague, Czech Republic, 2009.
[8] “The Gaudi project”, http://proj-gaudi.web.cern.ch
[9] “CVFS or StorNext File System”, http://en.wikipedia.org/wiki/StorNext_File_System
[10] J. Callot et al., “Raw-data Format”, CERN, EDMS565851.5.
[11] “RFC791 - Internet Protocol”.
[12] S. Cherukuwada and N. Neufeld, “High-Performance Storage System for the LHCb Experiment”, Proc. 15th IEEE NPSS Real Time Conference, Fermilab, Batavia, IL, April 2007.
[13] C. Gaspar, “The LHCb Run Control”, Proc. CHEP09, Prague, Czech Republic, 2009.