High-Speed Data-Injection for Data-Flow Verification at LHCb

Olivier Callot, Markus Frank, Jean-Christophe Garnier, Clara Gaspar, Guoming Liu, Niko Neufeld, Alba Sambade Varela, Andrew Cameron Smith, and Daniel Sonnick

***Abstract*—The High Level Trigger (HLT) and Data Acquisition System select about 2 kHz of events out of the 40 MHz of beam crossings. The selected events are consolidated into files in 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 is needed in the absence of real data. In order to test the system as much as possible under identical conditions as for data-taking, the solution would be 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. Data can come from several input streams, which are selected according to probabilities or frequencies. Therefore the emulator offers runs which are not only identical data-flows coming from a sequence on tape, but physics-like pseudo-indeterministic data-flow, including lumi events and candidate b-quark events. 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 optimize the emulator for throughput from the Storage Area Network. The emulator can be run in stand-alone mode, but even more interesting is that it can emulate any partition of LHCb in parallel with the real hardware partition. In this mode it is fully integrated into the standard run-control. The architecture, implementation, and performance results of the emulator and full tests will be presented. This emulator is a crucial part of the ongoing data-challenges in LHCb. Results from these Full System Integration Tests (FEST) will be presented, which helped to verify and benchmark the entire LHCb data-flow.**

***Index Terms*—Benchmark, data acquisition, high level trigger, simulation.**

I. Introduction

THE LHCb [1] 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 abundance of matter over anti-matter in the observable universe. The effect of CP-violation can be particularly well studied in decays of particles that 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 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 [2] experiments. On the other hand the total event size, determined by average number of active electronics channels, is quite small: 35 kB according to estimates from simulation. The data acquisition and data handling in LHCb are hence faced with a very high rate of 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. In order to understand how this project was carried out, it is necessary to have a rough understanding of the Data Acquisition System (DAQ) [3]. Fig. 1 gives a schematic view of the LHCb DAQ and the way data flows.

The readout network relies on Ethernet and IP. The 300 readout boards (TELL1) [4] get data from the detector and build Multi Event Packets (MEP) [5]. Each readout board typically buffers between 1 and 15 event fragments corresponding to the relevant sub-detector. A fragment is a set of banks [6], which consists of an header and physics data.

The events are then assembled in the High Level Trigger (HLT) [7] farm. An HLT farm node notifies its availability for data processing, sending a MEP request packet to the readout supervisor.

The readout supervisor, known as the Timing and Fast Control (TFC) [8], selects an available node. It sends information via a Timing, Trigger and Control interface (TTC) [9] to the readout boards in order to indicate the selected destination node. It sends a MEP to the destination node in order to supply it with the trigger information needed for events analysis and reconstruction.

First, the requirements of such a system will be presented, then the constraints and the specification of the system, its architecture, the quest for performance optimization and its use in the Full Experiment System Test (FEST [10]).

II. Requirements

The implementation of a test module arises from two needs. The first LHC beams of the 2008 startup would have been used for testing and validating the full experiment system, from the detector to Offline analysis. Unfortunately, due to a failure in one of the LHC magnets, first LHC collisions have been delayed until late 2009. Moreover the LHC is not expected to run continuously. Such shutdown periods would be opportunities to

Manuscript received May 21, 2009; revised July 17, 2009 and October 22, 2009. Current version published April 14, 2010. This work was 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.

O. Callot is with LAL, Orsay, France (e-mail: Olivier.Callot@cern.ch).

M. Frank, J.-C. Garnier, C. Gaspar, N. Neufeld, A. Sambade Varela, A. C. Smith, and D. Sonnick are with CERN, Geneva, Switzerland (e-mail: markus.frank@cern.ch; jean-christophe.garnier@cern.ch; clara.gaspar@cern.ch; niko.neufeld@cern.ch; alba.sambade.varela@cern.ch; samvar14@hotmail.com; andrew.cameron.smith@cern.ch; daniel.sonnick@cern.ch).

G. Liu is with CERN, Geneva, Switzerland, and also with the University of Ferrara, Ferrara, Italy (e-mail: guoming.liu@cern.ch; gliu@cern.ch).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNS.2009.2038216

0018-9499/$26.00 © 2010 IEEE

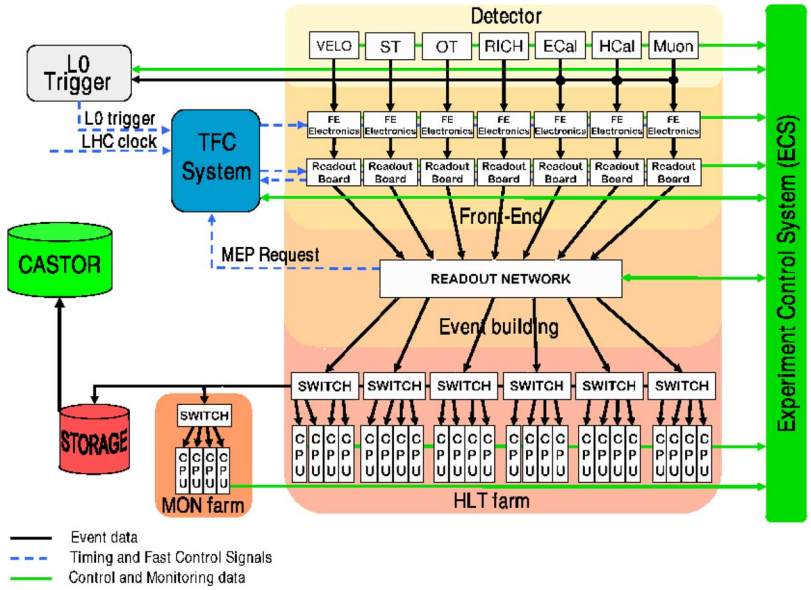

Fig. 1. LHCb data acquisition.

make modifications to the system, but they should be validated before the next production activity.

A solution for this kind of test would be a program which injects simulated IP packets into the readout network. Injected data could be both real physics data taken from previous detector activity, and simulated data produced Offline. It would be set up for regular use, involving the LHCb collaboration once a month for exercises close to a normal operation of the experiment. This is called the Full Experiment System Test.

The main requirement of the FEST is to test everything in the way it should be during true data acquisition. Therefore the development of the test module (or injector) and its integration in the system is made in order to emulate real runs. This is why the module is integrated on top of the DAQ (Fig. 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.

The aim is not to benchmark performance of each part of the processing, but to test the various features of the DAQ and HLT for long term runs.

To carry out our objectives the injector needs:

- To be supplied with simulated physics data files, in order to inject events into the DAQ.
- To assume the role and characteristics of all readout boards in use by the experiment.
- To produce a minimum output rate of 2 kHz, so an HLT farm operated in "pass-all" mode can produce an equal output rate. This calls for the injector throughput to be at least 70 MB/s, for the standard event size of 35 kB.
- To be triggered by the TFC in order to be used as the detector.
- To be able to mix different input data streams according to the TFC triggers.
- To be fully integrated into the Experiment Control System (ECS) [11].
- To be used in parallel with every other activity at LHCb.

All these requirements impose constraints on the software implementation, on the hardware setup and on system operation.

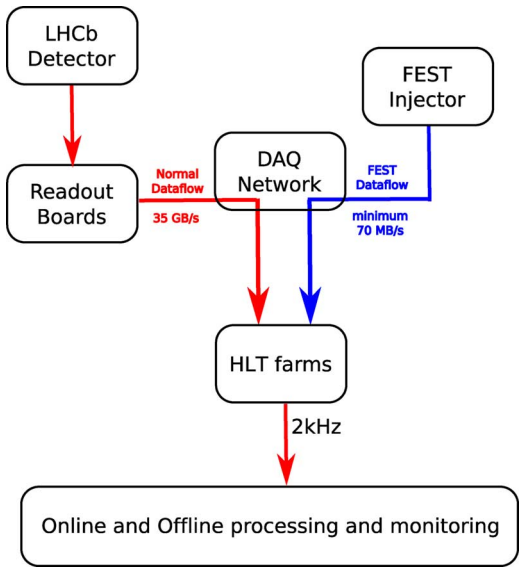

Fig. 2. LHCb data injection.

III. Architecture

To address these requirements, software has been developed using the standard LHCb C++ Gaudi framework [12].

It consists of a set of Gaudi services, which are normal Unix processes, and which encapsulate a set of POSIX threads. The Gaudi framework allows the injector to interpret commands received from the Experiment Control System. Moreover, it allows the implementation to re-use existing solutions, such as reading events from files.

This section will focus on the system operation and software implementation. The hardware setup relies on a general purpose server node, which provides enough resources to run the implemented software. Its configuration has been chosen in order to meet requirements. More details on the choices will be given in Section IV.

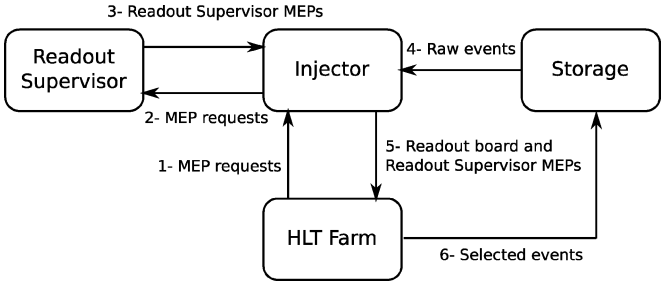

Fig. 3. Integration in the DAQ system.

A. Operating the Injector in the Experiment System

We have chosen to operate the injector and FEST the way it is presented in Fig. 3. This is done in order to answer a technical issue: the injector needs to be triggered by the TFC, and thus it needs some trigger information for the simulation processing. Unlike the readout-boards, however, it cannot be connected to the TFC TTC interface. The only way to get the trigger information is to receive the MEPs sent by the TFC. This is achieved by 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 farm nodes in order to record their MEP requests (Fig. 3, step 1). In parallel, it pretends to be an HLT node to a TFC board, in order to forward the HLT node MEP requests (Fig. 3, step 2), and to receive the answer (Fig. 3, step 3). Once the injector has received this MEP and read enough raw events (Fig. 3, step 4), it fragments them in order to build MEPs as any TELL1 board would do. It then sends each of the 300 produced MEPs to one of the requesting HLT nodes (Fig. 3, step 5). For each packet it modifies the IP header, in order to pretend to be the real TELL1 which would have built the MEP. In the end, the HLT nodes reassemble the events and send them to the storage system (Fig. 3, step 6).

B. Software Architecture

In order to increase performance, and to manage each input and output whilst taking care of real time constraints, the various injector interfaces need to be managed asynchronously. The injector software will thus rely on different threads and processes which share their information using Inter Process Communication tools. Fig. 4 shows the various threads, Gaudi job instances, main process entities, and their method of communication.

First, the injector runs two dedicated threads to handle its network interfaces with the TFC and the 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 used immediately. MEP requests are stored in a map, so the injector knows how many events have to be sent to each requesting farm.

The injector reads events from files. Events are stored in the Master Data File (MDF) format. It is necessary to convert them to the MEP format. It is a computationally complex task because of the software and hardware operations implied in the procedure. Then the injector also has to produce and send MEPs through a network interface to the HLT.

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

- A task dedicated to reading events from files and writing them into a buffer.
- A task dedicated to consuming events, building MEPs and sending them to the network.

It has been chosen not to separate the MEP building from sending as this would have required one more layer of synchronization and another layer of buffering. This additional buffering layer would have needed specific synchronization and memory management, which would have cost more than just sending the produced MEPs directly.

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 and to answer the requirement of mixing them according to TFC trigger types. The injector needs at least one reader, and can use up to eight. This limit comes from the number of different trigger types implemented by the TFC. Performance limits about the input of this architecture will be detailed in Section IV-A.

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 trigger information, the injector selects a buffer and reads events, produces the MEPs and sends them to the selected HLT node. It then asks for another trigger from the TFC to continue the processing.

IV. Implementation

A. Storage Access

Simulated data used as injection input have been produced Offline and were moved to the LHCb Storage Area Network (SAN) [13]. This SAN provides a Network File System export (NFS), and fiber channel access to some selected nodes. There are two ways for reading data: NFS over Gigabit Ethernet and StorNext File System (CVFS) [14] via fiber channel.

The injector needs to send events to the HLT farm at a minimum rate of 2 kHz. Since the injector is based on the producer-consumer pattern, the overall performance depends strongly on how fast events can be produced by reading them from storage.

Consequently, several tests were performed, using the two available data access configurations and various implementations of the reading algorithm. To determine the performance offset of the reading algorithm, each reader was also tested against a file from residing on a RAM disk. Performance tests were made using Linux monitoring tools and the following readers:

- The standard Unix "dd" command was used to get raw performance results. It highlighted the maximum throughput of 300 MB/s via CVFS, and 100 MB/s for NFS.
- The EventSelector Gaudi Service, which reads input files and puts event objects into a transient store.
- Our own C/C++ reader in which buffering use has been improved, which handles events as memory buffers instead of C++ objects, and pushes them into an LHCb standard buffer manager.

First, we had to look for the storage access media which was the most suitable to meet our performance requirements. NFS was quickly dismissed because the maximum injection rate barely

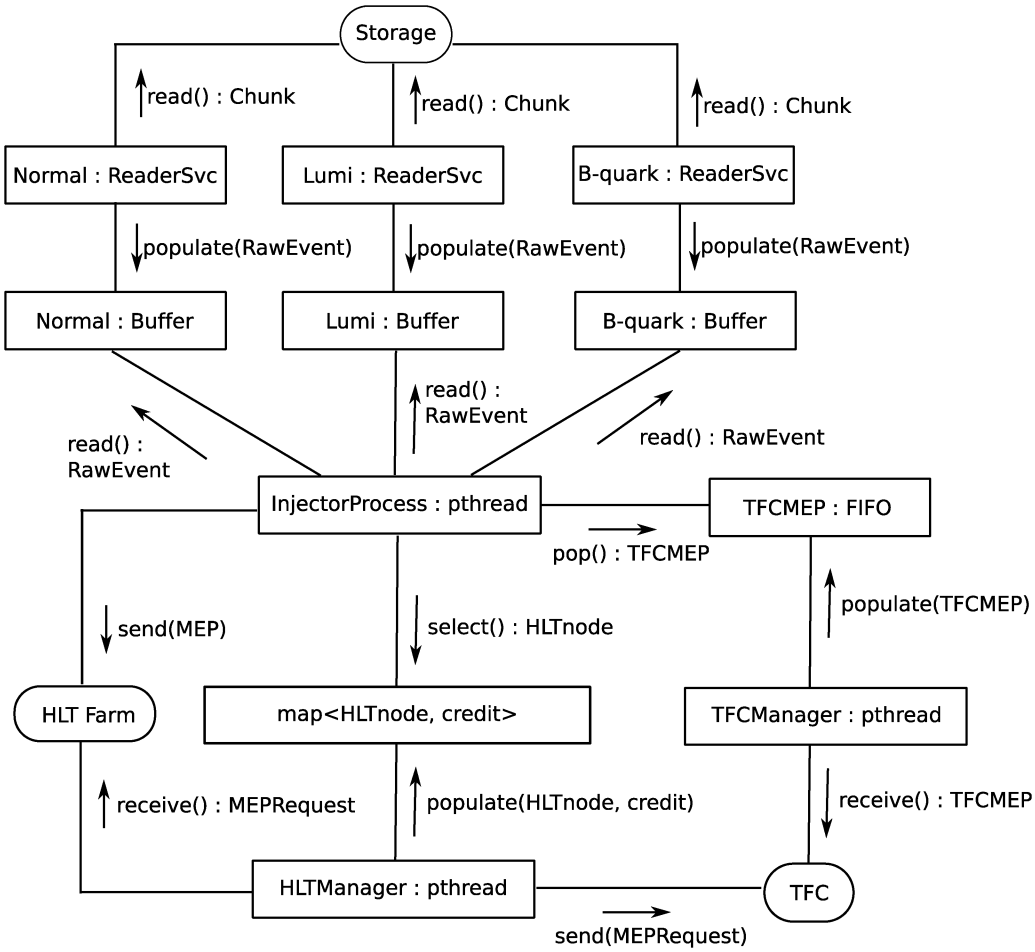

Fig. 4. UML collaboration diagram of the injector.

matched the 2 kHz requirement. It was also subject to strong performance fluctuations, since it has to share the networking resources with other exports. It was then decided to connect the injector directly to the SAN and to connect it to CVFS.

For completeness, however, further tests were also performed with NFS.

We had to choose a software implementation in order to read events. The input file format consists of a sequence of events, with each event comprised of a header followed by data encapsulated in banks.

The standard EventSelector reads events one by one and builds C++ objects for each encapsulation layer, i.e. it reads 35 kB chunks before processing them. Therefore the hardware access is not optimized at all. Then, data is stored in a Transient Store, from which they have to be extracted before use. Reading such small chunks usually incurs a huge performance penalty. The maximum read-rate that could be achieved was 2.5 kHz. However the data were left in a format that could not be processed efficiently afterwards.

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 parse the events in it. Secondly, the data format on disk is very close to the final MEP format and it is more efficient to copy data directly into an output buffer instead of using the indirection of C++ objects in the transient store.

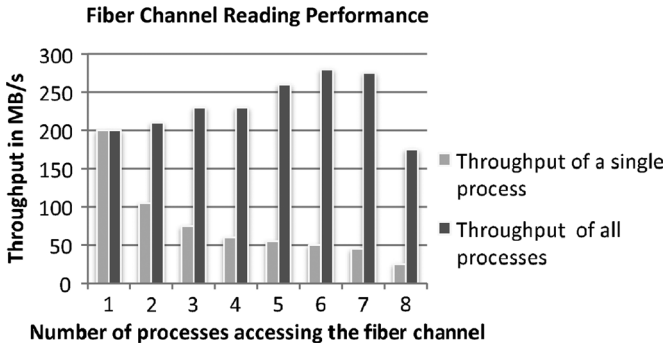

Fig. 5. Storage access performance via fiber channel, using from 1 to 8 dedicated reader process, accessing different set of files.

Therefore, a dedicated reader which follows these improvements has been implemented for the injector. The throughput reaches 200 MB/s via CVFS for a single reader thread. This translates into 5.8 kHz of events. This dedicated reader has been integrated into a Gaudi Service, in order to get the flexible architecture presented before and for easier integration in the standard LHCb software. Adding more readers increases the performance even more, as shown in Fig. 5, and allows the injector to fulfill its requirement for injecting different data for different trigger types. Care has to be taken though because performance

starts to drop at eight readers, due to the available number of CPUs inside the injector machine.

Nevertheless, good performance is ensured while the injector is using a reasonable number of readers. The injector input rate will not be a limiting factor on the process to meet required performance.

B. Multi Event Packet Construction and Sending

The injector has to send MEPs as if they were originating from the TELL1 readout boards. This has given rise two problems.

1) Building MEPs From Read Events: The data files assembled events, while the HLT farm expects event fragments orginating from distinct TELL1 boards. The injector splits each event into several MEPs, according to which TELL1 it wants to emulate. It will typically build 300 MEPs for the full detector. As the DAQ protocol is IP, each MEP is identified by the IP address of the TELL1 that it was sent from. However, events read from the storage do not contain this information. Instead it contains the type of the sub-detector connected to the TELL1, and a source channel identifier. From these two values, the injector computes the TELL1 IP address used in production.

Subsequently, several events have to be put into one MEP. This is done by parsing the MEP received from the readout supervisor, and pushing event fragments into MEP buffers until the requested number of events has been reached. In the end, all the generated MEPs are sent from the TELL1 IP addresses.

2) Sending Data From Multiple IP Addresses: The injector has to send MEPs as if they were coming from real TELL1s. Therefore it performs IP address spoofing, writing the TELL1 source IP addresses itself. It is done using a raw socket with specific options requesting management of the IP header in our user space.

However this disables the IP fragmentation [15] 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 (typically 9000 Bytes). For this reason the injector has to implement the IP fragmentation itself.

As for the first stage of the processing, particular care has been taken for this implementation. As the algorithm relies mainly on iterations, mapping, and memory copy, the sources were profiled several times using the Valgrind's Tool Suite [16].

C. Networking

At LHCb, raw data are stored on a Storage Area Network [17] accessible from the HLT farm via a dedicated Ethernet Local Area Network. As the storage LAN is physically separated from the DAQ LAN,[1] 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 almost the line rate of the 1 Gb network interface. For an average size of 35 kB per event, it represents a rate of about 3.8 kHz, which is close to twice the requirement of 2 kHz. This allows us to operate the farm in a normal mode, and to test various triggers.

[1]To ensure maximum robustness and good capability

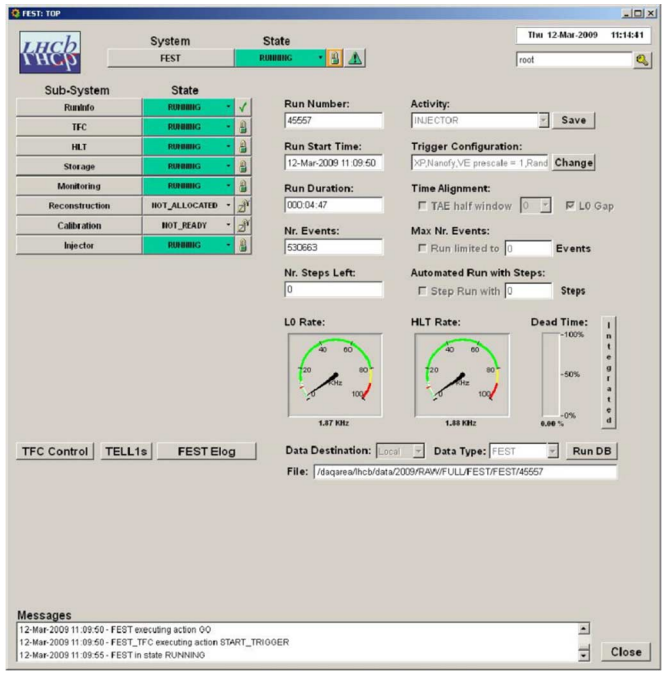

Fig. 6. Control panel of the FEST partition.

D. Integration in the Experiment Control System

Once the injection process was implemented, integration into the system was required. We needed a flexible and easy method of operation, which would allow parallel running with the detector.

The LHCb software architecture is made of a processing layer and a control layer. They communicate with a lightweight communication middleware, DIM (Distributed Information Management System) [18]. The use of the Gaudi framework to implement the injector made it receptive to such commands. The injector implements a finite state machine which is controlled by a dedicated LHCb Run Control [19] partition. Fig. 6 shows that the injector is completely integrated into the ECS, equally to all other subsystems (HLT, storage, TFC, etc.). In parallel, other partitions for normal detector runs or sub-detector runs can be used, sharing all common sub-systems.

Flexibility is also achieved through the Run Control interfaces. On injector dedicated panels, the user can configure the input files, the number of readers, the trigger type which will enable the reader, etc. The run configuration information is configured with Run Information panels and with TFC panels, like in normal detector use.

The integration is complete and a single click is enough to start the simulated run, like for the detector.

V. Full Experiment System Test

The FEST is the main objective of the injector development. At least one week per month since January 2009 is a FEST week, and this will continue at least until the LHC start-up. FEST

weeks will also be held during periods of LHC shutdown. Experts from both Offline and Online teams are working together to achieve the commissioning and the validating of the data processing and its monitoring. The injector proves its use as a key tool in the validation of the LHCb Offline and Online system. It is constantly used outside FEST weeks in order to validate single project upgrades, mainly for the HLT farm.

An example where the injector has been found useful was in the diagnosis of a system configuration issue. During FEST runs, the HLT tasks were complaining that a few MEPs were not received. This problem was occurring quite randomly. After investigation, we noticed that these errors were occurring only on the most recent nodes. Although the HLT farm nodes have all approximately the same hardware, these new nodes have a different network interface card. The unfortunate combination of these cards, a specific module version and a specific kernel version, sometimes caused troubles with the interrupt management. Such issues could have been found only during a run in which the data traffic is high enough: either a FEST run or normal detector run. However in the latter case it would have meant loss of physics data.

Other examples of FEST achievements, concerning triggers and physics analysis, are described in [10].

Therefore the injector has proved useful for debugging the full system, from the system configurations at every stage of the Online data flow, to the Offline analysis and reconstruction.

VI. Conclusion

This paper describes how in LHCb we can perform high-speed data-injection in order to validate our system in a situation similar to a normal run of the LHC, using mainly software. There are some areas in which improvements to the simulator could be made. Other experiments have also Online and Offline test modules. For example ATLAS [20] can store a few events in the buffers of the DAQ input components, which are commodity computers. Then a test run uses these data buffers instead of the detector, and the data-flow can be tested into the DAQ. However, data are limited by the buffer size.

Another example is the second stage of the CMS [21] Event Builder, which is made of a PC farm and a multi Gigabit Ethernet network. Data fragments from simulation or from previously taken data can be played into the system at the level of the Readout Units. They are then transferred through the data network to the Builder Units which deliver the complete events to the Filter Processors which are housed in the same PCs as the Builder Units. After processing, the events are saved to local storage and transfered Offline in the same way as during global data taking.

We note that each solution is dedicated to its experiment, and that flexibility was never foreseen because the implementation relied on system particularities. Testing procedures might be the same in every experiment, but the test setup has to be dedicated.

The second possible improvement is that for the next upgrade of the detector, the current solution may not be powerful enough. This is why a project is currently being carried out, involving FPGA development and 10 Gigabit Ethernet communication. A solution based on an FPGA implementation would have a better integration at the readout level. It would be possible to directly interface it with the TFC via the TTC media, as with a standard readout board. Moreover, as the integration would be better, we could easily use several FPGA injectors synchronized by TTC, in order to reach the upgraded readout boards data rate.

The current injector, however, has already been a major contribution to the LHCb collaboration for the few months in which it has been used in production, and hopefully this will continue in the future.

References

[1] A. J. Augusto Alves *et al.*, The LHCb Collaboration, The LHCb Detector at the LHC pp. 1–190, Aug. 2008 [Online]. Available: http://www.iop.org/EJ/article/1748-0221/3/08/S08005/jinst8_08_s08005.pdf

[2] L. Evans and P. Bryant, LHC Machine Aug. 2008 [Online]. Available: http://www.iop.org/EJ/article/1748-0221/3/08/S08001/jinst8_08_s08001.pdf

[3] P. R. Barbosa-Marinho *et al.*, LHCb Online System, Data Acquisition and Experiment Control: Tech. Design Rep., LHCb, 2001 [Online]. Available: http://cdsweb.cern.ch/record/545306/files/cer-2302560.pdf, Tech. Rep.

[4] A. Bay, A. Gong, H. Gong, G. Haefeli, M. Muecke, N. Neufeld, and O. Schneider, "The LHCb DAQ interface board TELL1," *Nucl. Instrum. Meth. A*, vol. 560, pp. 494–502, 2006.

[5] B. Jost and N. Neufeld, Raw-Data Transport Format, Tech. Rep. EDMS 499933, 2004.

[6] O. Callot *et al.*, Raw-Data Format, Tech. Rep. EDMS 565851.5, 2005.

[7] The LHCb Collaboration, LHCb HLT Homepage [Online]. Available: http://lhcb-trig.web.cern.ch/lhcb-trig/HLT

[8] The LHCb Collaboration, LHCb TFC Homepage [Online]. Available: http://lhcb-online.web.cern.ch/lhcb-online/TFC

[9] B. Taylor, "Timing Distribution at the LHC," presented at the 8th Workshop on Electronics for LHC Experiments, Sep. 2002 [Online]. Available: http://www.cern.ch/TTC/LECC02.pdf

[10] M. Cattaneo, "LHCb Full Experiment System Test (FEST09)," presented at the Int. Conf. Computing in High Energy and Nuclear Physics, Prague, Czech Republic, Mar. 2009.

[11] A. Sambade Varela *et al.*, "An integrated control system for the LHCb experiment," presented at the 16th IEEE NPSS Real Time Conf., Beijing, China, May 2009.

[12] The LHCb Collaboration, The Gaudi Project [Online]. Available: http://proj-gaudi.web.cern.ch

[13] J. Caicedo Carvajal *et al.*, "A high-performance storage system for the LHCb experiment," presented at the 16th IEEE NPSS Real Time Conference, Beijing, China, May 2009.

[14] Quantum [Online]. Available: http://www.quantum.com

[15] U. Inform. Sciences Inst. RFC791—Internet Protocol [Online]. Available: http://www.faqs.org/rfcs/rfc791.html

[16] Valgrind Developers, The Valgrind's Tool Suite [Online]. Available: http://valgrind.org

[17] S. S. Cherukuwada and N. Neufeld, "High-performance storage system for the LHCb experiment," *IEEE Trans. Nucl. Sci.*, vol. 55, pp. 278–283, 2008.

[18] C. Gaspar, M. Dönszelmann, and P. Charpentier, "DIM, a portable, light weight package for information publishing, data transfer and inter-process communication," presented at the Int. Conf. Computing in High Energy and Nuclear Physics, Padova, Italy, Feb. 2000 [Online]. Available: http://dim.web.cern.ch/dim/papers/CHEP/DIM.PDF, Teatro Antonianum

[19] C. Gaspar, "The LHCb run control," presented at the Int. Conf. Computing in High Energy and Nuclear Physics, Prague, Czech Republic, Mar. 2009.

[20] The ATLAS Collaboration, G. Aad *et al.*, The ATLAS Experiment at the CERN Large Hadron Collider Aug. 2008 [Online]. Available: http://www.iop.org/EJ/article/1748-0221/3/08/S08003/jinst8_08_s08003.pdf

[21] The CMS CollaborationS. Chatrchyan *et al.*, The CMS Experiment at the CERN LHC Aug. 2008 [Online]. Available: http://www.iop.org/EJ/article/1748-0221/3/08/S08004/jinst8_08_s08004.pdf