Journal of Physics: Conference Series **119** (2008) 022024 doi:10.1088/1742-6596/119/2/022024

Data Stream Handling in the LHCb Experiment

M. Frank[1], B. Jost[1], N. Neufeld[1], A. C. Smith[1], R. Stoica[1,2] and Sai Suman[1]

[1]CERN, Geneva, Switzerland
[2]IFIN-HH, Romania

E-mail: niko.neufeld@cern.ch

Abstract.

This paper presents the event data handling in the LHCb experiment, starting from the acceptance of events by the software trigger farm to making the data available for offline re-processing.

In the data acquisition system, events accepted by the High Level Trigger farm are assembled into streams according to their type. Each stream will be written to disk as a 2 GB file for optimal storage and ease of reconstruction. Under nominal conditions about 2 such files will be produced per minute. The life cycle and state transitions of each file are managed by means of a dedicated database. These files must be copied to the tape storage and simultaneously be made available online to various calibration and monitoring tasks. Additionally, to initiate the start of the offline re-processing all file parameters must be propagated to the offline Grid data analysis system.

This system has been designed for extreme robustness and reliability with redundancy in almost every component and supports various fail-over mechanisms. The software, hardware, and protocols designed for this purpose will be discussed in detail. Performance figures of data-challenges done in 2007 will be shown.

1. Introduction

LHCb is one of the four large experiments, which will start taking data in 2008, when the LHC goes into 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 hence faced with a very high rate of rather small events. All these events have to be passed through a complex, rather time-consuming algorithm for selection. The data acquisition and trigger are described elsewhere, this paper will focus on what happens to events *after* they have been selected.

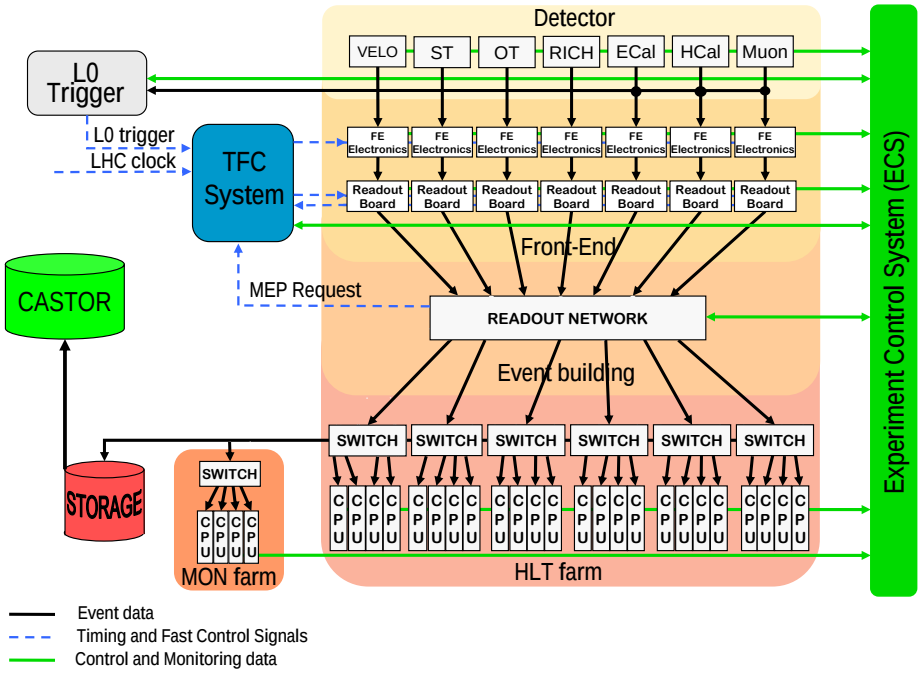

Figure 1. The LHCb Data acquisition

Nevertheless it is necessary to have a rough overview of the Data Acquisition System (DAQ) as a whole, namely the large processing farm, to understand the requirements in the next section. Figure 1 gives a schematic view of the LHCb DAQ, where a large farm of commodity servers is shown at the bottom. From a data handling point of view up to 2000 such 1U servers can be considered the original sources of the data.

2. Requirements

Of the original 10^6/s events, between 5000 and 2000 will be chosen for storage[1] by the selection processes running on the High Level Trigger (HLT) farm.

The event data streaming and storage system described in this paper must ensure that each event accepted by the HLT farm will be reliably written to a file. Events will be put into *streams* depending on their classification by the trigger. Each stream of events can be written to several physical *files* in parallel. The files must be checksummed, transferred to permanent storage, registered in the Grid file catalog, and published for offline re-processing. Also, files must be immediately made available in the online system for prompt processing tasks (such as quality checking, monitoring, calibration). Finally, this system must be *partitionable*. A *partition* is a part of the experiment, which can be run independently and in parallel with other parts. A partition is usually one sub-detector, e.g. the muon system, but can comprise several

[1] The luminosity of the accelerator will decrease with time, as will the rate of interesting events.

sub-systems. Partitions are mutually exclusive; a subsystem can only be member of a single partition at a time. Partitions are therefore useful for parallel debugging and commissioning of the overall experiment. A component in the LHCb Online system is said to be partionable, if it can be instantiated as often as is required by the number of partitions running in parallel.

When events have been accepted by the HLT farm a considerable amount of CPU resources have been invested in the selection process and they are likely to be important physics events. Moreover, exact knowledge of the trigger efficiencies is extremely important in LHCb, hence random loss of events after the trigger must be avoided to ensure physics event data is not biased. The software and hardware at this stage must hence be reliable to avoid data loss due to failures as far as possible[2]. This reliability can be achieved by redundancy of components and fail over mechanisms.

3. Architecture

To address these requirements a multi-layered distributed architecture has been devised. Each layer is implemented by *Tasks*, which can be started in parallel on multiple servers and their implementation ensures load-balancing and fail over. These three layers in the order following the data-flow are the Receiving Layer, the Streaming Layer, and the Storage Layer.

The details of the tasks running on the individual servers is described elsewhere [1]. Data are exchanged between processes via buffers implemented in shared memory. Processes can act on buffers as producers or consumers. Buffers are marked by a *Trigger Mask*, which is used later to separate events into streams.

Keeping track of the files associated with the streams is done through a database, the *Run Database*. This is the central repository for all information on event data files and runs. This database must contain everything necessary to successfully export the file to the offline processing Grid system.

3.1. Receiving Layer

Considering the number of servers in the farm, the number of accepted events and the individual event size we arrive at the following requirements for the Receiving Layer:

- It needs to receive events from up to 2000 processes running on the HLT farm and
- Receive between one and five 40 kB events per second from each server

Recapturing the requirements for the the Receiving Layer one finds

- It needs to consolidate a large number of TCP connections
- It needs to be capable of managing multiple output streams
- It needs to feed events directly to online consumer tasks

3.2. Streaming layer

The Streaming Layer consists of tasks that receive events for specific streams. Each such stream produces a number of files, each optimally sized for subsequent handling by data management software, and eventual archival to tape. Every accepted event is written to at least one file. The Streaming Layer connects to the Storage Layer as a fault-tolerant, redundant service. Each Streaming Layer server is capable of failing over to a different Storage Layer server in case one such server experiences a failure. This fail-over takes place without any loss of events.

[2] In the large farm used for event selection, servers equipped with redundant power supplies, multiple network paths, and multiple hard-disks cannot be afforded. As a result, full redundancy cannot be achieved here.

3.3. Storage layer
The Storage Layer has to implement the other side of the failover mechanism with the Streaming Layer servers. Apart from that, it needs to write files as efficiently as possible. Unique file names are generated based on the stream, the current run, and the current time. Information about these files (number of bytes, events, checksums[3] , etc.) are stored in the Run Database.

3.4. Run Database
The Run Database is central to bookkeeping and file management in the online data acquisition system. The inherent serialization of the database is used to ensure locking and uniqueness in all file related operations. Because of this serialization all components that are involved in file handling can be run in multiple instances without the need to perform explicit synchronization between themselves. The database is also useful for generating unique identifiers, for example creating file and run identifiers. The Run Database is also used to manage the states of the files during their life-cycle. Finally, all information required to register files in the Grid Logical File Catalog (LFC) [2] and in the LHCb Bookkeeping database is kept here.

The Run Database is used by several important processes involved in file management, notably the Data Mover and the File Transfer Agent that replicates the files to the CASTOR system, both of which are explained later. The Run Database can also be controlled and monitored by the Experiment Control System through its API. Details on all these processes and the Run Database can be found in [3].

3.5. File Transfer to the Offline Re-processing System
Once files are successfully written on the storage system, they are ready to be transferred to the tape storage center. Permanent storage is achieved at CERN's central storage facility, CASTOR [4]. However, simply copying a file is not enough; the file needs also to be registered in the LFC and in LHCb Bookkeeping Database.

LHCb already has a workload management system for dealing with these tasks. It is called the Distributed Infrastructure with Remote Agent Control (DIRAC [5]) framework and it is normally used to manage reconstruction and data analysis tasks on the LCG [6] for the LHCb experiment. To maximize reuse of existing expertise and code, the DIRAC framework is also used for transferring and registering files originating in LHCb's Online system.

Contrary to a standard offline scenario, the creation of the files is not managed from DIRAC, so a glue-layer is required. DIRAC's modular structure makes this task relatively easy. A dedicated daemon, the *Data Mover* polls the Run Database at regular intervals to search for files ready for transfer. It then hands over a move request to a DIRAC based agent for copying a file to CASTOR and registering it in the LFC.

After a successful transfer the Data Mover creates an XML request for entering the file information in the LHCb Bookkeeping Database. This is sent to a daemon running on a dedicated server at CERN that acts as an interface to the LHCb Bookkeeping Database and performs the actual database update.

4. Implementation

4.1. Receiving Layer
The Receiving Layer consolidates the $O(2000)$ TCP connections from the farm into a small number of streams, which are then sent on to the next layer. At this point the events are not yet

[3] The standard checksum used for all data files in LHCb is an MD5 checksum over the entire file. While MD5 is a good hash-code, it is computationally quite expensive. The tape hardware behind CASTOR calculates therefore a different, much simpler, checksum of the Adler32 type. When files are written in the storage system both checksums are calculated on the fly and stored in the Run Database. Using the Adler32 checksum avoids the need of having to re-read the files from tape to verify their integrity.

categorized; all incoming and outgoing streams contain all types of events. The categorization is based on information in the event header and can be based on the "Run Type"[4] and an "Event Class" or "Trigger Type"[5] assigned by the selection algorithms in the event-filter farm. Based on this category, events are now sent on to a specific stream in the Streaming Layer. Events are buffered at the Receiving Layer nodes. Should a streaming layer node be unavailable, then buffers in the receiving layer nodes will eventually fill up. Back-pressure will finally reach the HLT farm nodes, which will ultimately disable the trigger. However, the control system is configured to react to such conditions much before that. The small event size allows several seconds worth of data to be stored in memory, which should be enough to establish a connection to a different streaming layer node. This layer also serves events of a specific type to monitoring tasks. The architecture of the monitoring infrastructure is described in [7].

The receiving layer can be scaled up to reach any desired ratio of incoming to outgoing streams. Initially, two servers will provide the required performance. The servers are dual socket servers with Xeon (Nocona) processors at 2.8 GHz with HyperThreading enabled and 2 GB of RAM. They are connected to the DAQ network using Gigabit Ethernet links.

4.2. Streaming Layer

Based on the event type mask the data is sent by the Receiving Layer to specific tasks running on the Streaming Layer servers. Each task on the Streaming Layer receives only one type of events. The events are consolidated into a file format and sent to the Storage Layer for writing to disk. The file size is adjustable (currently 2 GB) and is chosen to fit the needs of the tape mass-storage and the replication on the Grid. While the event data files are created at this level, the MD5 and Adler32 checksums are calculated without a need for duplicating the I/O to the online storage system. Care has been taken that failover and failback between the processes on the Streaming Layer and Storage Layer works without data loss or file corruption.

4.3. Storage Layer

The file storage layer is provided by a cluster of servers attached to a shared disk subsystem. This cluster is henceforth referred to as the 'Storage Cluster'. Servers in the streaming layer write events to nodes in the storage cluster, and do not discard any of these events unless they are confirmed to be written on disk by the storage cluster nodes. All the nodes in this cluster export the same file system namespace to the Streaming Layer. In the event of a failure occurring on one of the servers in the cluster, any streaming layer servers that are connected to it can simply resume by sending unacknowledged events to be written to the same file, but through a different cluster node. This ensures continuity of the file writing service even in the event of a complete failure of one of the storage cluster nodes. Automatic load balancing occurs when storage cluster nodes are added or removed. The file writer service also aligns disk write operations appropriately to ensure the highest possible throughput.

The writer processes run on the servers directly attached to the Storage Area Network (SAN). Currently we use two servers, which are connected with a Gigabit Ethernet link to the same router as the Streaming and Receiving Layer servers. The storage servers are dual-socket, dual-core Xeon (Woodcrest) servers with 4 GB of RAM and are equipped with a dual-port 4 Gigabit Fibrechannel card to attach via redundant paths to the storage backend. Details on the storage implementation can be found in [8].

[4] A Run Type can be calibration data, physics data, data from only a specific subsystem or a combination of subsystems ("partition"), etc.

[5] This broadly identifies the category of physics signatures: single muon, di-muon, multi-hadronic, etc.. . .

4.4. Run Database
The Run Database is implemented on an Oracle 10g server [6]. The schema contains the most important properties of runs and files as well as their relationships. Also, the schema is extensible using generic self-describing fields. Access to the database is possible by means of a Python API, which supports both the XMLRPC and DIM communication protocols. The same functionality is available through both protocols. The DIM interface is used to connect the Run Database to the global LHCb Experiment Control System (ECS) [9], while XMLRPC is used for the rest of communication in the online system. Also a web based interface is available for simple administrative tasks. Direct access to the database is only foreseen for administrative purposes. More details on the implementation can be found in [3].

4.5. Transfer of Files to Permanent Storage
The interactions involved in the migration of a file to the offline system are presented in fig. 2. The Data Mover polls the Run Database at a low frequency and searches for newly created files. For each file a transfer request will be sent to the DIRAC agent. The DIRAC agent saves the request to disk as an XML encoded file and the transfer will be later executed by a different daemon. The actual protocol used for the file copy is RFCP, a CASTOR internal protocol. It is foreseen to be replaced by Storage Resource Management (SRM) protocol to allow using the X.509 certificate based Grid authentication scheme also for file transfers. After a file is copied to CASTOR the DIRAC transfer agent will publish the newly created file in the Grid domain. This is accomplished using a the DIRAC interface to the LFC.

After the above steps are completed, the transfer agent reports the results back to the Run Database through an XMLRPC call. The checksum calculated by CASTOR will be encapsulated inside the call and this will be compared with the initial one. In case of a failed transfer or a mismatch in the checksum the data-mover will re-initiate the transfer for a programmable number of times. An alarm in the ECS will be created, if the transfer fails more than a specified number of times (typically two). When the transfer is successful, the file will be marked as *migrated* [7] in the Run Database.

The Data Mover will then propagate the file meta-data the LHCb Bookkeeping Database. A similar approach to the CASTOR transfer is used to decouple the online system and the Bookkeeping Database. A XML request containing all the necessary informations is sent to a service running outside the online system. The request is saved to a file and a daemon responsible for updating the LHCb Bookkeeping Database asynchronously processes it .

5. Performance
Recalling the performance requirements it can be seen that under nominal conditions (2 kHz of of accepted events to tape, 35 kB event size) 2 GB files are written every 30 s. Including overheads and a safety factor of two made us set the required performance at 150 MB/s I/O consisting of 2 GB files. Before an integrated test in a data challenge, the individual components have been independently tested.

The storage system has been tested successfully for up 550 MB/s of I/O from 32 simultaneous read and write streams. This is almost a factor two above the requirements[8] and leaves sufficient margin to cope with re-tries of transfers and file-reads by online processes. The system has been verified to scale with the number of disks in the underlying hardware, so an upgrade can be done easily, if need arises.

[6] The full power of an Oracle DB is not actually required here, but Oracle is used throughout in the LHCb experiment. Portability is not an issue for this application.
[7] This means that the file can be deleted if it is not in use by any process in the online system, and disk space is required.
[8] Two, because files need to be written *and* read at least once.

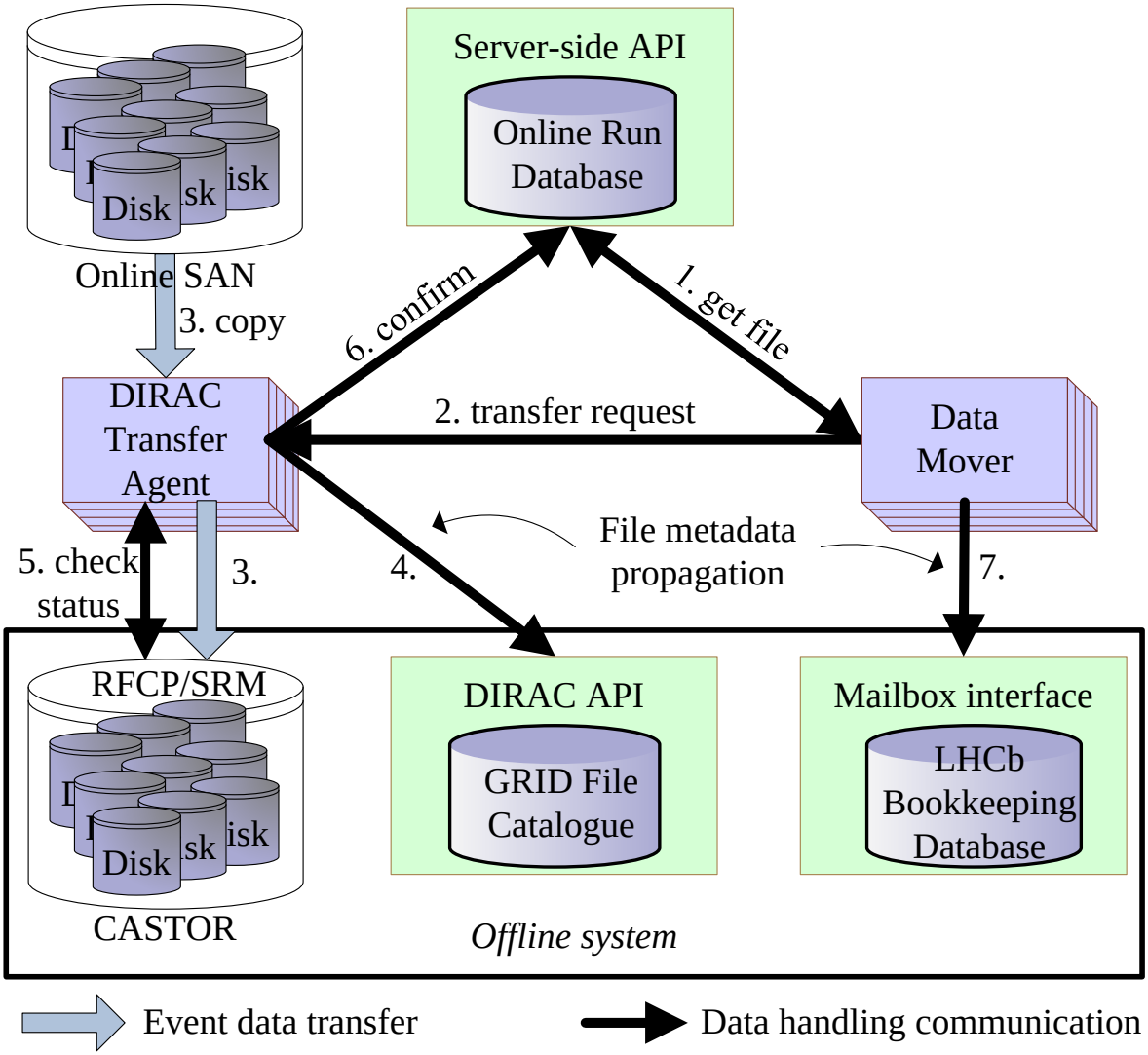

Figure 2. The logical flow of events in the file-transfer and registration, tested in the data-challenge.

The Run Database has been tested to be capable of tracking files through all states from creation to migration at rates in excess of 100 Hz. This is well above the requirement of 1 Hz, reflecting partly the fact that the hardware on which it is running is designed for much more demanding applications.

The Receiving and Streaming Layers have been tested to be able to handle 5000 events per second per process.

Since a minimum of two processes run on two servers this amounts to at least 10000 events per second. Consolidation of 500 TCP connections per process works well. Both numbers exceed the currently anticipated requirements and the performance can be scalled by starting more processes and/or adding servers. The server node in the tests was an Intel dual-Xeon(Nocona),

2.8 GHz with 2 GB of RAM.

In the challenge events were created by injector processes replacing the standard sender processes. Files were then allocated in the Run Database, written onto the storage and transferred to CASTOR using DIRAC. This was done on a dedicated 10 Gigabit Ethernet network segment directly connecting the storage system to the CASTOR disk pool.

All standard tests were successful and the file management was proven to function over extended periods of time.

5.1. Global tests

The software was initially tested together with all the software layers presented in Figure 2. Simulated event data were generated, which traversed the whole online software chain and were written to disk before being migrated to the offline system (CASTOR, LFC). All software layers were run on a different server (Event Builder, Data Writer, Online Run Database, File Handler, DIRAC Transfer Agent). The tests so far targeted the software functionality, as the performance requirements or throughput will not be very high in the final configuration. In this scenario time stamp based error recovery was tested by simulating various error scenarios:

- Failure probabilities were added in the communications between the various components. This was meant to simulate possible network failures in a real life situation.
- Storage hardware errors were simulated by manually modifying existing raw event files. The checksum based error recovery behavior was targeted in this case.
- Processes were abruptly terminated for testing redundancy and load balancing after restart.

The final software tests were done on the actual hardware and network configurations used for data acquisition. The Oracle database together with its interface were run on a dual CPU 3 GHz dual core Intel Xeon (Nocona) server, 8 GB of RAM, running a 64 bit Scientific Linux CERN 4 (SLC) operating system. For performing the raw event file writing I/O we used two quad core 3 GHz Intel Xeon (Clovertown), 4 GB RAM running also a 64 bit SLC4 operating system which were directly attached to the Online SAN. One instance of the Data Writer, Data Mover and 3 DIRAC Transfer Agents processes were started on each server. Although one server can meet the necessary performance requirements, two were used for testing redundancy and load balancing. As the interface to the ECS is not yet fully implemented, the ECS was not used in this tests and the process management was replaced with a number of custom scripts. In total 10^5 2 GB files were transferred from the LHCb experiment site to the CASTOR tape system where a dedicated CASTOR disk pool was created. These tests were meant to expose any problems that might appear in practice in the final configuration. They were also meant to gather various statistics on the reliability of the storage hardware used in the Online and CASTOR storage systems.

A final extended test with full ECS integration and the new version of DIRAC and SRM will be done towards the end of this year. This will include replication of files to the secondary LHC Grid centers (Tier-1 centers). However it should be noted that, while this is essential for LHCb, the Online system is not coupled to the replication to the Tier-1 centers at all, since all data have at this stage already successfully been stored on tape.

Acknowledgments

The authors would like to thank the CERN IT department, in particular the teams of IT/FIO (Castor operations) and IT/CS (campus networking) for their tight collaboration and effective support.

This work has been supported by the Marie Curie Early Stage Research Training program of the European Community's Sixth Framework Program under contract numbers MEST-CT-2004-007307-MITELCO, MEST-CT-2005-020216-ELACCO and MEST-CT-2004-504054-DEVINFO.

References

[1] Cherukuwada S S, Frank M, Gaspar C, van Herwijnen E, Jost B, Neufeld N, Somogyi P and Stoica R 2007 *Proc. International Conference on Computing in High Energy and Nuclear Physics* (Victoria, BC)
[2] Baud J P, Casey J, Lemaitrel S and Nicholson C 2005 *Proc. IEEE 14th International Symposium on High Performance Distributed Computing, HPDC-14* pp 91–99
[3] Frank M, Neufeld N, Smith A C and Stoica R 2007 *Proc. 15th IEEE NPSS Real Time Conference* (Fermilab, Batavia, Il)
[4] CASTOR homepage http://cern.ch/castor
[5] Tsaregorodtsev A *et al.* 2006 *Proc. International Conference on Computing in High Energy and Nuclear Physics* (Mumbai)
[6] The LCG Editorial Board 2005 LHC computing Grid Tech. Rep. CERN-LHCC-2005-024 CERN http://doc.cern.ch//archive/electronic/cern/preprints/lhcc/public/lhcc-2005-024.pdf
[7] Callot O, Frank M, Gaspar C, Graciani R, van Herwijnen E and Altarelli M P 2007 *Proc. International Conference on Computing in High Energy and Nuclear Physics* (Victoria, BC)
[8] Cherukuwada S S and Neufeld N 2007 *Proc. 15th IEEE NPSS Real Time Conference* (Fermilab, Batavia, Il)
[9] Cherukuwada S S, Frank M, Gaspar C, van Herwijnen E, Jacobsson R, Jost B, Neufeld N and Stoica R 2007 *Proc. International Conference on Computing in High Energy and Nuclear Physics* (Victoria)