THE HIGH LEVEL FILTER OF THE H1 EXPERIMENT AT HERA

A. Campbell, S. Levonian, DESY, Hamburg, Germany
M. Vorobiev, ITEP, Moscow, Russia

Abstract

We present the scheme in use for online high level filtering, event reconstruction and classification in the H1 experiment at HERA since 2001.

The Data Flow framework (presented at CHEP2001[2]) will be reviewed. This is based on CORBA for all data transfer, multi-threaded C++ code to handle the data flow and synchronisation and fortran code for reconstruction and event selection. A controller written in python provides setup, initialisation and process management. Specialised java programs provide run control and online access to and display of histograms. A C++ logger program provides central logging of standard printout from all processes.

We show how the system handles online preparation and update of detector calibration and beam parameter data. Newer features are the selection of rare events for the online event display and the extension to multiple input sources and output channels.

We dicuss how the system design provides automatic recovery from various failures and show the overall and long term performance.

In addition we present the framework of event selection and classification and the features it provides.

BACKGROUND AND MOTIVATION

The electron proton collider HERA at the DESY laboratory in Hamburg and the H1 experiment[1] completed major upgrades in 2001. To cope with increased demands on data processing a new high level filter sytem was developed combining the functionality of the specialised VME (L4) and mainframe systems (L5) to a single system based on networked linux PCs (L45).

REQUIREMENTS

The framework must be capable of transferring a stream of events from a source (or multiple sources) through multiple processing tasks and out to one or more sinks. A source may be a selection of events from a sequence of files or online data from the H1 experiment. A sink may be disk or tape files or a special data logging task. It must be possible to require that certain groups of events from the input stream remain grouped on the output stream, for example events from different data taking runs must not be mixed despite the large variations in processing time between events. The online L45 filter task must be capable of maintaining a continuous input data rate of 7 MB/s and output rate of 4 Mbytes/s. The data throughput should be independent of individual event sizes which vary greatly from several hundred kilobytes for complex e-p interaction events to just a few bytes for special calibration records or compressed data summary format events.

In addition to event data, calibration constants must be distributed to the processing nodes ahead of the data to which they apply. For the L45 filter a few calibration constants may be changed on a short timescale for immediate use from a particular event number within a run. The L45 filter program is based on the full reconstruction program (h1rec) which consists of large Fortran code for charged particle and calorimeter shower identification. Event data BOS banks are transfered in the H1 specific machine independent FPACK format and monitor histograms are stored in the H1 specific LOOK package. Especially for online L45 processing we require to access for immediate display the monitor histograms filled within the Fortran code. In addition histograms must be collected from all processing tasks, summed and stored on a per run basis.

As the L45 processing and filtering step is run on raw data from H1 before storage on persistent media a certain degree of robustness against program and machine failure is required. In particular a crash of the full reconstruction program h1rec should not lead to a loss of events and a new process should be started automatically. Failure of complete machines should be recoverable without halting the data flow, and it must be possible to add additional processors at any time. In the rare case of the failure of a complete machine some data loss is tolerable as long as it can be quantified.

IMPLEMENTATION TOOLS

We choose to base our implementation on CORBA. This has several major advantages over raw sockets and message passing infrastructures as parameter marshaling code is generated automatically, connection setup is handled completely by the ORB, the remote calls are machine architecture and language independent, and calls are unaware if the objects are local or remote (location independence). A performance comparison of several freely available CORBA implementations led us to choose omniORB[4] as basis for the main dataflow infrastructure due to the excellent data transfer throughput and the small footprint as well as the strict standards compliance. We use omnipy python scripts and jdk1.4 and JAS[5] for online histogram browsing.

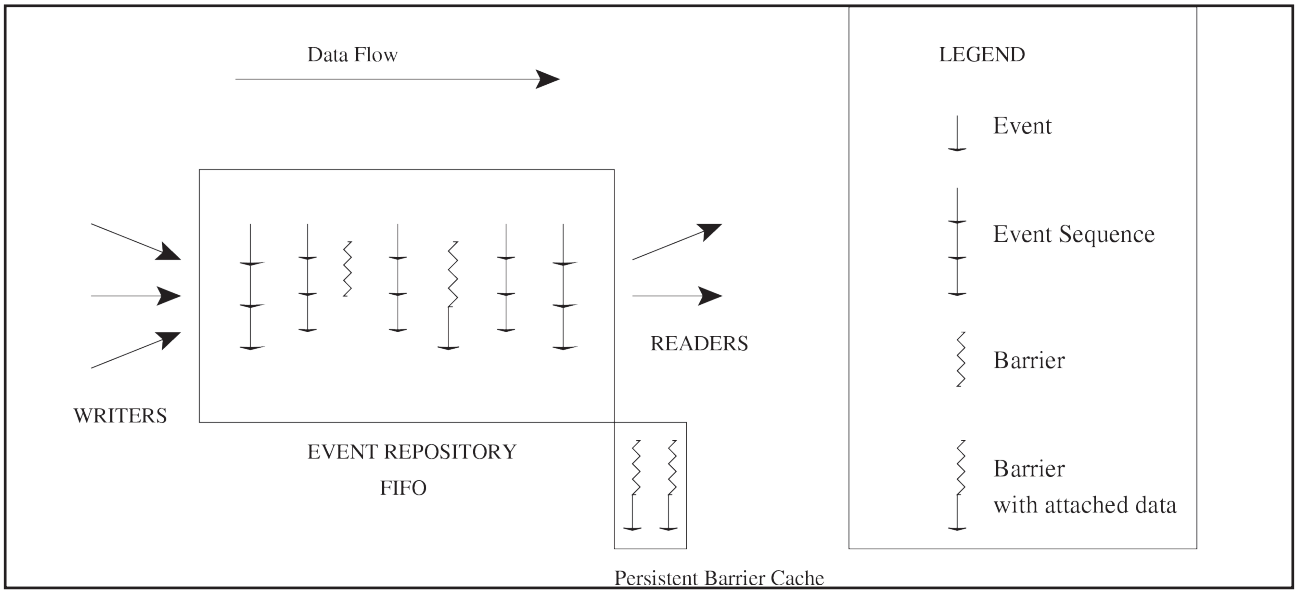

Figure 1:CORBA Event Repository

THE EVENT REPOSITORY

The dataflow is organized by data transfer between CORBA event repository objects[3]. The event repository is basically a first-in first-out event store which may be read and written in a manner similar to sequential file access. Events are actually stored in the repository as suitably large sequences instead of individually and entire sequences are transfered between repositories over the network to enable high throughput, independent of event size. An event repository with its reader and writer tasks is depicted in Figure 1.

Multiple readers can read simultaneously from the same repository to receive subsequent event sequences. The multiple reads are handled by separate ORB server threads per client machine with only a short mutually exclusive section to remove a sequence pointer from the repository FIFO buffer. No expensive data copying is required. Similarly multiple writers can hand over event sequences to the repository simultaneously. Hence repositories are used for both event distribution and collection.

A method is needed to separate the event flow into blocks such that events stay within a block. All output events within a block must reach their final destination before events from the next block, although events within a block may be dropped as is often the case in the L45 filter application. Practically block boundaries are start and finish of data taking runs in the case of online processing and the start and end of data files for offline processing. Additional block boundaries may mark eg every hundredth calibration pulser event to trigger the preparation of a new calibration.

To support this synchronisation of the event flow the repository implements barriers. Like event sequences barriers are inserted into the repository FIFO store. A barrier is assigned an incremental number when it is first written to its source repository and retains this number as it is transferred from repository to repository. Synchronisation is obtained by requiring that all writers write a barrier before any readers can read it. Further writers can continue to write event sequences in front of a barrier which has already been written by another writer. As soon as all writers have written a particular barrier that barrier becomes readable. Unlike event sequences barriers are not removed from the repository until they have been read by all readers.

However a given reader can read event sequences (or further readable barriers) behind barriers which they have already read. Only when all readers have read a barrier is it removed from the repository. Hence each barrier is distributed to all processing units and recombined at the final sink repository. As data events cannot skip across a barrier this ensures the required synchronisation. Of course the repositories must be sufficiently large and the barriers sufficiently infrequent so that the overall data flow is not hindered.

In order to support insertion of additional processing units in an established data flow readers and writers must perform an open (or login) operation on the repository. This makes a contract with the repository to provide/read all barriers from a particular barrier number. Attachment to the dataflow is a 3 step process; first the downstream repository is contacted and a contract made to deliver the front-most non-readable barrier (or next barrier to come if no barrier is present in the repository), then the upstream repository is opened and a promise made to read the front-most barrier, and finally the contract with the downstream repository is modified to correspond to the barrier which is now known from the upstream repository. Similar care must be taken to cleanly detach from the dataflow and a method is required to logout

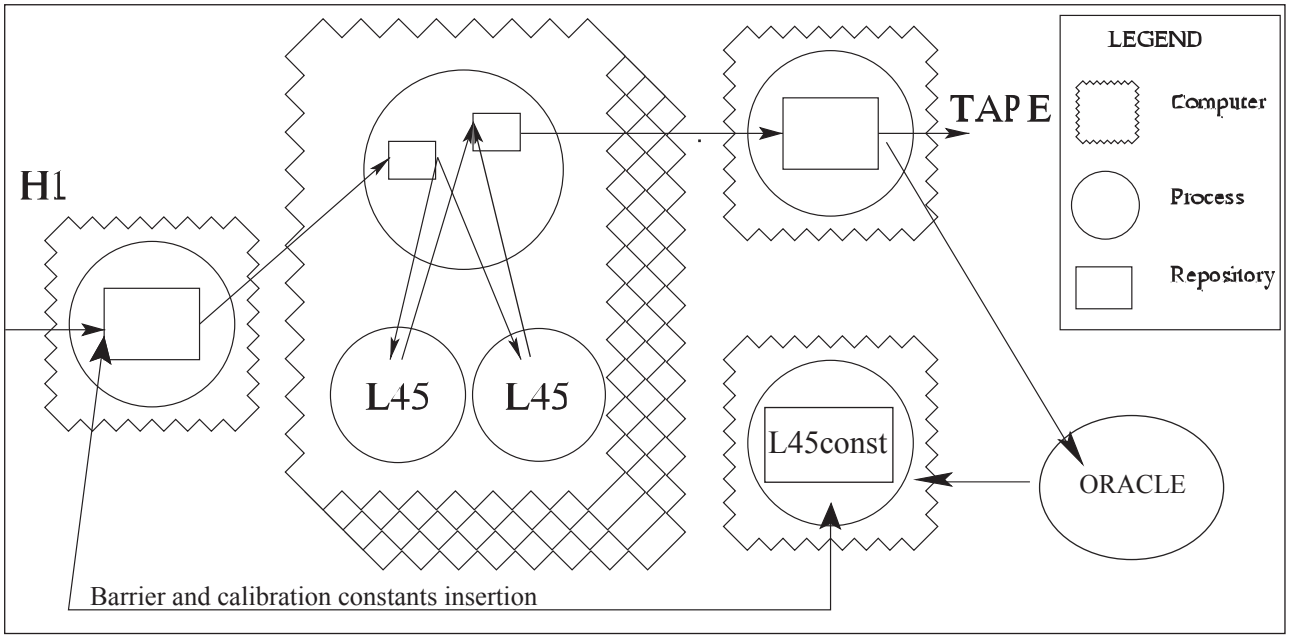

Figure 2:Overall L45 Data Flow

readers and writers from repositories in the case of abrupt abortion of a task. This is one of the responsablities of the controller framework described below.

Each barrier contains a type indicating its meaning eg start-of-run, end-of-run, end-of-dataflow. As a barrier flows through the processing code it can trigger special actions, for example the end-of-run barrier triggers the creation and output of run histogram records to the output repository for subsequent summing and logging at the data sink. The special end-of-dataflow barrier causes the shutdown of the containing process on its removal from all its repositories.

As barriers are "broadcast" to all processing units and mark a timestamp in the dataflow we can use them also as an elegant mechanism for distribution of calibration and geometry constants to the processing nodes by attaching data records to them. If we introduce a barrier with attached calibration data this data can be received and kept in the analysis program and is therefore available for the processing of subsequent events. Hence we avoid separate requests to the central database from the many processing tasks and ensure that the same constants are used by all processing units even if the database is updated in the meantime. This mechanism allows also the timely distribution of run settings for online data which are not yet stored in the central database, as well as the introduction of new "constants" from a particular event number within an ongoing run as is for example required in the case of a sudden shift of the collision vertex within the H1 detector.

In order to ensure the availability of the correct calibration constants to processing units inserted into an established dataflow barrier records with attached data are not deleted when they are removed from a repository but instead are inserted in the associated persistent barrier cache replacing there any older version of the barrier data. A new reader reads first all barriers from the barrier cache before continuing to read from the repository,thus receiving all necessary calibration data.

OVERALL DATAFLOW

The overall dataflow is shown in Figure 2 for the L45 filter application with online data input from the H1 experiment. Data flows from the data acquisition system via a TCP socket as a stream of FPACK physical records. The input node converts this stream into event sequences and writes it to its repository. A set of 20 dual processor nodes run 3 processes each; a single i/o task containing both input and output repositories for the 2 reconstruction and filter processes L45. Each L45 process reads just a single event at a time and a copy of the event is kept within the i/o task so that it is not lost should the L45 process abort. The output repository handles the main output event stream and extra calibration records created within L45. Additional threads within the i/o task pull data from the input node(s) and push data to the output nodes. The output node can update calibration constants and insert them as barriers to the current dataflow by request to the input node. For offline analysis programs the i/o and processing tasks can be easily combined in a single process due to the location independence of CORBA objects.

ENVIRONMENT

To facilitate the launching and control of the processes over all nodes a controller framework has been developed. A master controller process is started on the initial node and a slave controller is launched on each node on which dataflow processes should be started via ssh. Requests to the master controller from a python script initiate all processes. The local controller daemons can take appropriate action in the case of death of child processes such as informing the connected repositories and restarting. In addition the controller daemons distribute initialisation parameters, assign unique process identifiers, and maintain lists of all CORBA objects such as repositories and histogram server objects. The master controller object is entered in the CORBA name service and thus acts as a single access point for all objects within the distributed job.

We have developed a comfortable histogram display tool based on JAVA and JAS[5] capable of collecting in real time the histograms from the running reconstruction programs for comparison to reference histograms allowing immediate data quality checks. In addition web access to histograms is developed via a python script based on twistd[6], biggles[7] and SVG.

Further monitoring of the data flow is provided by timestamped log files recording state changes in the repositories (empty , full , barrier entry). This allows for diagnosis of bottlenecks and hangups in the dataflow.

FILTER ALGORITHM STEERING

The filter algorithm executed in L45 is defined in text supplied along with the calibration and geometry data and stored in the database for bookkeeping. The text consists of some definition sections and a sequence of trigger statements which define the algorithm.

First code modules are named (eg CJC for central jet chamber track reconstruction) together with a list variables which can be computed after execution of the module and a list of dependent modules (eg QT must run before CJC). Next trigger masks are defined to allow statements to act on a set of L1 triggers eg all track triggers.

The algorithm is specified by a sequence of trigger statements. A trigger statement has a name, a list of conditions on variables (true/false, ><= a value) and an action. Conditions are evaluated from left to right by executing the modules associated to the specified variables (unless they have already been executed for this event) and requesting the evaluation of the variable by the module. As soon as a condition is false the algorithm jumps to the next trigger statement. If all conditions are true the specified action is taken: accept,reject or reset_mask. The reset_mask action causes reset of the mask bits in a copy of the L1 trigger result and leads to a reject if all resulting bits are zero. In addition the action can specify a fraction to provide output of rejects for monitoring purposes (or scaledown of accepts) and an optional continue condition which causes only evaluation of the statement but no action (for evaluation of new statements). Hence the algorithm can be modified without recompilation of the L45 code and it is guaranteed that execution time is minimised as only the modules needed to take the trigger decision are executed.

Finally actions to be taken on accepted events are specified (typically execution of all modules) and a histogram section specifies ranges for timing histograms of modules and variables which should be histogrammed either on evaluation or within a specific trigger statement only. Histograms are automatically filled to record the result of all conditions in all statements allowing full monitoring and the evaluated variables are added to the event data.

STATUS

The tools described are in production usage since the startup of HERAII datataking in 2001. Since then 27TB have been processed. The system has proven to be very stable and typically runs without manual intervention constantly for several weeks.

ACKNOWLEDGEMENTS

We gratefully acknowledge in particular the enormous contribution of J.Nowak in implementing the event repository.

REFERENCES

[1] H1 Collaboration,"The H1 detector at HERA",NIM A 386(1997)310.
[2] A.Campbell et al,"A Dataflow Meta-computing Framework for Event Processing in the H1 experiment", Proceedings CHEP'01
[3] J.Nowak,"Data distribution system for the DESY/H1 experiment analysis",MSc Thesis,University of Mining and Metallurgy,Krakow,Poland
[4] http://omniorb.sourceforge.net
[5] A.S.Johnson,"Java Analysis Studio", http://jas.freehep.org
[6] http://twisted.sourceforge.net
[7] http://biggles.sourceforge.net