SOFTWARE DEVELOPMENT FOR HIGH SPEED DATA RECORDING AND PROCESSING*

D. Boukhelef†, J. Szuba, K. Wrona, C. Youngman
European XFEL GmbH, Hamburg 22761, Germany

Abstract

The European XFEL beam delivery defines a unique time structure that requires acquiring and processing data in short bursts of up to 2700 images every 100 ms. The 2D pixel detectors being developed produce up to 10 GB/s of 1-Mpixel image data. Efficient handling of this huge data volume requires large network bandwidth and computing capabilities. The architecture of the DAQ system is hierarchical and modular. The DAQ network uses 10 GbE switched links to provide large bandwidth data transport between the front-end interfaces (FEI), data handling PC layer servers, and storage and analysis clusters. Front-end interfaces are required to build images acquired during a burst into pulse ordered image trains and forward them to PC layer farm. The PC layer consists of dedicated high-performance computers for raw data monitoring, processing and filtering, and aggregating data files that are then distributed to on-line storage and data analysis clusters. In this contribution we give an overview of the DAQ system architecture, communication protocols, as well as software stack for data acquisition pre-processing, monitoring, storage and analysis.

INTRODUCTION

European XFEL [1], a new research facility currently under construction in Germany, will generate ultra-short and extremely intense X-ray flashes. The facility will open spectacular research opportunities for scientific and industrial users when operation starts in 2016.

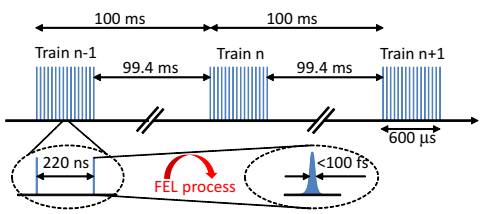

Figure 1: XFEL bunch structure.

Pulses are delivered at 4.5 MHz in pulse trains of 600 us duration with a nominal 10 Hz repetition rate. Each train consists of up to 2700 pulses (see Fig. 1).

Large area 2D pixel detectors are being developed(e.g. LPD [2]) which will be able to capture and send up to 10 GB of image data per second to backend DAQ systems. Other detector technologies (TOF, Fast-CCD, 1D spectrometers, MCP delay lines, etc.) produce smaller amounts of data at the same rate. Additionally diagnostic and control devices (GigE commercial cameras, positioning motors, vacuum sensors, etc.) produce small data volumes at lower rates. The requirements placed on the DAQ architecture and the processing and recording software used are thus driven by the large 2D pixel detectors and their input is highlighted in the results presented here.

XFEL beam delivery defines a unique time structure that requires acquiring and processing large amount of data in short bursts of time. Efficient handling of such huge and fast data streams requires large network bandwidth and computing capabilities.

In this paper the data acquisition (DAQ) and processing system being developed at the European XFEL is described by giving an overview of the architecture of the DAQ system, followed by a description of the design and implementation of prototype used to test the software developed.

DATA ACQUISITION AND PROCESSING

The DAQ system is designed to handle large volume data streams coming from multiple fast and slow detectors and sensors, perform on-line data monitoring, rejection and reduction, and forward raw data for storage and to scientific computing farm for analysis. Additionally, the system publishes pertinent near real-time information about quality of data and services to be used by the users and scientists.

General Architecture

XFEL's DAQ system is structured into five layers (see Fig. 2) with well-defined inter-layer APIs and communication protocols . Following a data-driven model, data are pushed from their sources (e.g. 2D detectors, triggers, etc.) through FEIs (e.g. train builder) to the PC layer farm and then on to storage stack and processing clusters.

Detector Front-ends

2D mega-pixel cameras, digitizers, and diagnostic sensors acquire and digitize analogue signals.

Front-end Interface

Detector interfaces are needed to transfer detector data to the backend, for the large 2D detectors a train builder [3] is used. This custom design FPGA-based ATCA board assembles and processes data generated by large-area 2D image detector modules (16 per Mpixel). The train builder, as

* Work partially supported by European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement nº 283745
† djelloul.boukhelef@xfel.eu

ISBN 978-3-95450-139-7

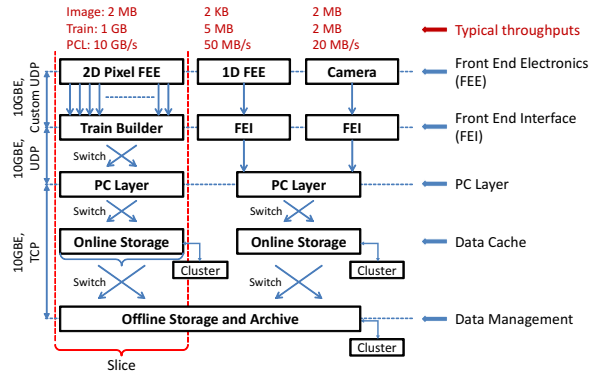

Figure 2: Multi-layer DAQ system at XFEL.

all FEIs, outputs pulse ordered images from a single train to PC layer servers via 16 10 Gbps (SFP+) switched links. The train builder pushes using plain UDP protocol.

PC Layer

The PC layer, a core element of the DAQ system, is a homogeneous high-performance computer farm located between FEI and on-line data cache. Each PC layer node receives periodically a full train of images acquired during a given burst. The PC layer is required to allow full speed data receiving and writing through to on-line storage and analysis, this capability will be used to debug the data rejection and reduction systems.

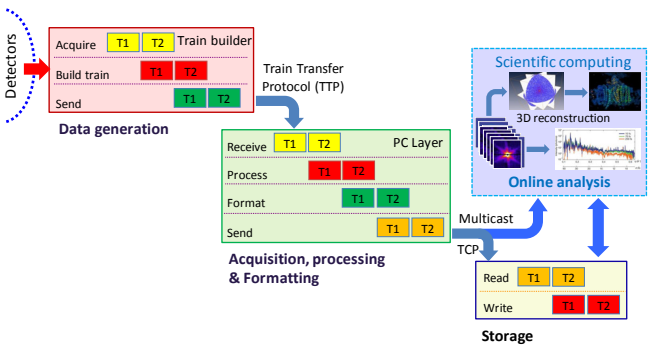

Figure 3: Data acquisition, processing and storage systems.

PC layer node software is organized as a pipeline that can handle multiple trains concurrently at different stages (see Fig. 3). Each PC layer node will be fed by train builder with one full image train periodically in round robin fashion. After receiving a full train in memory, and while waiting for next one, PC layer node performs several checks, pre-analysis and transformation (e.g. image rejection, formatting) on the train data received before sending it to permanent storage. The various steps are described below.

Readout: The most important task of a DAQ is to allow for high speed data receiving from different data sources and safely send it to storage and analysis. The largest amount of data comes from 2D pixel detectors through train builder which feed the PC layer with up to 10 GB of image data every second per detector Mpixel. Data transfer is based on plain UDP protocol. Sophisticated tuning of software and hardware are required in order to minimize traffic jam and data loss. Other fast and slow *monitor* data streams (e.g. digitizers and sensors) are also sent to PC layer through dedicated TCP channels. Traffic management should ensure that any train-related data needed by PC layer activities (e.g. monitoring, fast analysis) is available at the corresponding node without data loss.

Monitoring: Another mandatory feature of the DAQ system is to continuously monitor the quality of data streams and gather statistics about system and software status and make it available to other components and users. Monitoring tasks performed by the PC layer are grouped into four classes: control quality of data streams (availability, validity, etc.); monitor train data content (train builder and computational algorithms,); publish system health and resources status; and gather statistics on application activities and performance. Monitor data is used by operators and scientists to understand whether the experiment is performing correctly, and allows any necessary fine tuning of the hardware or software parameters at runtime, as well as detecting and fixing problems quickly.

Rejection and reduction: The DAQ system is designed to enforce data reduction and rejection in all layers aimed at reducing data storage and transfer requirements. At the electronics level for instance, the VETO system evaluates pulse information and distributes a trigger decisions to detector front-end ASICs and FPGAs allowing them to remove poor quality images. To further improve the data quality and reduce the amount of data to be stored, additional data rejection and reduction algorithms are also run at the PC layer.

Pre-processing: Depending on time and resources available for each train, PC layer nodes can also perform additional data pre-processing tasks, such as hit and Bragg peak finding. These algorithms, which are mainly be provided by scientists, should be fast and reliable and be capable of removing unwanted low quality pulse data before the storage layer.

Formatting: PC layer prepares data for writing to storage by formatting and aggregating raw input train data and any data produced by the PC layer processing into a common file format. The latter is required to be easily exploitable (i.e. read/write, retrieve) later by the scientific computing pipeline used as well as additional external software tools. Currently, the system exports data using HDF5 standard [4], a self-descriptive and portable data model widely used by the scientific community. HDF5 is open source and provides rich support for handling complex data types and integrated performance features. Additionally a unified hierarchical file structure to store most of data types (i.e. values, attributes, annotations, etc.) according to their sources, frequency and usage, etc. has been developed.

On-line and Off-line Storage

On-line storage allows for full speed writing of raw data to disks as soon as it comes out of the PC layer. Its aim is

ISBN 978-3-95450-139-7

to decouple real-time data readout from scientific computing and off-line storage tasks. The capacity and bandwidth of on-line storage should be sufficient to handle all data produced during an experiment. Off-line storage is mainly used to store raw and processed data (results of scientific analysis) for longer period of time.

DAQ PROTOTYPE IMPLEMENTATION

A simulation prototype of the DAQ software to be used for handling the large amounts of data coming out of detectors at very high rate has been developed (see Fig. 4) and tested on hardware acquired for testing the DAQ readout system. The implementation targets mainly high performance data recording with special focus on scalability, reliability, and flexibility.

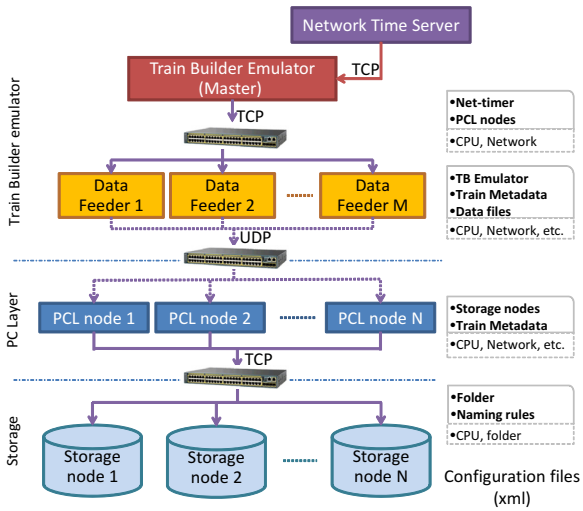

Figure 4: Prototype of the DAQ system.

Karabo Device and Device-server Model

XFEL's DAQ system is designed as a hierarchical and modular solution around a central message broker service. Following a component-oriented development, tasks in the system are delegated to independent software components called devices. Control commands, property change events, and slow and small data are exchanged between system components and users through the broker. Dedicated point-to-point TCP connections are established on need for large or time critical data transfers such as those used between layers in the DAQ architecture.

Devices are developed using the Karabo [5] software framework, a homogeneous pluggable distributed application environment currently under development at the European XFEL. Developed natively in C++ (with a binding to Python), Karabo provides a rich API landscape including device self-description and configuration, program-flow organization, logging and communication, etc. Karabo is intended for use in control, data-management, DAQ and scientific workflow tasks.

Train Builder Emulator

The top level component of the simulation prototype, those that feed data into the simulation, consists of a set of devices that emulate the function of train builder board. Entirely distributed, the emulator is composed of a set of data feeders and a master device that synchronizes data generation and sending to PC layer nodes. Data feeder device generates detector-like image data, formats it into trains, and sends it out using the train transfer protocol (see next section) to one PC layer node whose address is designated on-the-fly by the master device.

The network timer device emulates the timing system and generates one train-tagged network packet every 1.6 second which corresponds to the smallest number of PC layer nodes likely to be used with a 1 Mpixel detector. For every signal received, the master instructs one data feeder to generate a new train data and send it to the next PC layer node in round robin fashion.

PC Layer

PC layer node application is implemented as a multi-threaded C++ application, where each thread is responsible for one activity such as data readout, formatting, processing, etc. Common data are stored in a shared memory block, where data from each train are handled together as one envelope identified by the train number. To optimize memory bandwidth data copying and moving is avoided and only references are exchanged between worker threads. Inter-thread communication is done through high-performance thread-safe queues, a generic C++ templated queue implemented for this purpose.

The PC layer node application is designed as a series of tasks under strict control. Each task is well defined and assigned to dedicated thread. Data receiver thread ensures the correct reception of UDP packets from train builders, stores data into the associated train envelope and notifies other worker threads which will start the processing (e.g. data rejection) when all needed data are available.

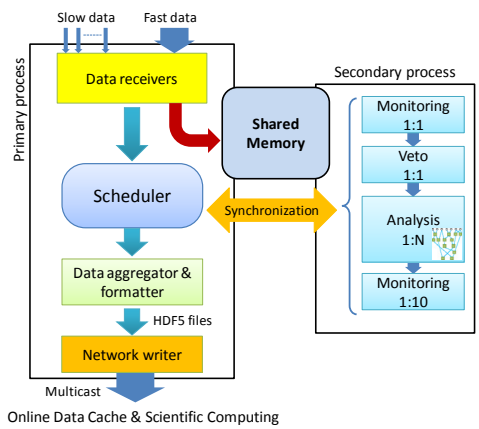

Figure 5: PC layer node internals.

Intermediate results generated by processing algorithms are stored along with raw data in the corresponding train

ISBN 978-3-95450-139-7

envelope. Post-processing and formatting tasks aggregate all partial results and find out which images are to be discarded and which ones should be stored in HDF5 files.

Scientists are expected to provide most of the detailed monitoring and processing algorithms to be used as plugins compiled in shared libraries. These algorithms will be installed and run in the PC layer. Careful vetting of the algorithms provided will be required to ensure that their usage does not compromise PC layer operation. Algorithms that do not pass requiremenst based on execution time, resource usage, etc. will be discarded.

As a general rule algorithms provided by third parties will be decoupled from the core PC layer application so that their failure will not stop the PC layer operating. The results from crashed applications are, of course, not available and this will usually require the run to be ended - and the problem fixed.

Data exchange between readout thread, users' algorithms, and post-processing is implemented using shared memory. A central scheduler synchronizes their activities. Raw data is always kept intact until just before formatting. In case of failure at the users' pipeline, this later is simply shortcut, and raw data is saved as avoiding any data loss.

The other reason for this separation is that the data receiver thread may require higher system privilege in order to set proper running parameters (CPU binding, thread priority, etc.). For security reasons, users' algorithms will run whithin the secondary process at normal user privilege.

Storage Servers

Data cache is built of standalone storage servers. Liteweight storage server runs on each box and constantly receives streams of HDF5 formatted files from PC layer and dumps them to disk. Advanced features such as direct IO and Linux splice function were investigated to speed-up writing to disk and achieve full bandwidth performance.

TRAIN FORMAT AND TRANSFER PROTOCOL

Train data format defines a generic layout of large data blocks interchanged within the DAQ system. Mainly used to define the laytout of data sent out by train builder to PC layer, it is also adopted by the software device that emulates the train builder when this later is not used. Train data format defines five sections in the following order: *Header* holds global properties of the entire train (e.g. train number); *Images* are blocks of data which belong to the same train; *Descriptors* are the set of properties for individual images (e.g. size); *Detector block* contains information specific for the detector and used mainly for debugging purpose; *Trailer* contains properties of the entire train which are appended on the fly (e.g. checksum).

Train Transfer Protocol

The train transfer protocol (TTP) defines a generic application-level protocol for interchanging large data blocks within the DAQ system. TTP is based on UDP protocol. The whole data block (frame) is divided into small UDP packets. Each packet is tagged with frame number (32 bits), packet sequence number (24 bits) and start-of-frame (sof), end-of-frame (eof), and padding flags. Packets belonging to the same data block hold the same frame number which distinguishes them from other frames' packets.

TTP is implemented by both train builder's FPGA and emulator software device. The trailer mode variant places the sequence numbers and flags at the end of each packet, to allow for optimized memory usage, as the trailer from current packet is overwritten by data from next packet.

SUMMARY AND OUTLOOK

A complete slice of the DAQ architecture to be used at the European XFEL, from detector front-end interface through to the data cache layers, has been implemented in hardware and software. The system has been used to evaluate the DAQ performance for the large area 2D detectors being developed for us at the facility, these detectors make the largest demands on the DAQ in terms of data handling. The use of the UDP protocol for data transfer out of the front-end interface of the detectors to the backend PC layer system required sophisticated tuning of NIC driver, OS kernel and application before acceptable low data loss rates could be achieved with concurrent 10 GE line-speed (FEI input and Data Cache output) operations.

On-line storage system is built using commodity hardware, for the test eight standalone storage servers. Six are equipped with 12 x 900GB SAS disks (internal storage), the other two with 12 x 3TB NL-SAS disks (external storage). Test results show that both storage configurations were able to achieve the desired rate, i.e. 1.1GB/s and 0.97GB/s per server, respectively.

Development of software for use in backend DAQ systems has started. Results of simulation on real hardware show that network and storage performance meet the requirements for 1-Mpixel cameras. Next steps include data rejection and monitor performance measurements.

REFERENCES

[1] M. Altarelli et al., "XFEL: the European X-ray Free-Electron Laser technical design report," DESY XFEL Project Group (2006)

[2] M. Hart et al., "Development of the LPD, a high dynamic range pixel detector for the European XFEL," IEEE NSS/MIC 2012, pp. 534–537.

[3] J. Coughlan et al., "The train Builder Data Acquisition System for the European-XFEL," TWEPP 2011, Vienna, Austria.

[4] "HDF5 User's Guide," `http://www.hdfgroup.org/HDF`

[5] B. C. Heisen et al., "Karabo: An Integrated Software Framework Combining Control, Data Management, and Scientific Computing Tasks," To Appear in ICALEPCS 2013, USA.

ISBN 978-3-95450-139-7