BUFFER MANAGER IMPLEMENTATION FOR THE FLASH DATA ACQUISITION SYSTEM

*V.Rybnikov, A.Aghababyan, G.Grygiel, O.Hensler, R.Kammering, L.Petrosyan, K.Rehlich

Abstract

The Free Electron Laser in Hamburg (FLASH[1]) at DESY is a user facility. It produces laser light of short wavelengths from the extreme ultraviolet down to soft X-rays. To study, monitor and document the machine performance and parameters and also to collect the results of the user experiment measurements a fast data acquisition (DAQ [2]) system has been developed. A shared memory based buffer manager is the heart of the system. It arranges collected data as events for every LINAC shot. All events can be read by different consumers simultaneously. LINAC feedback and monitoring processes as well as experiment middle layer servers are typical clients of the buffer manager. Any client can also generate its own data and insert it into the same event or produce its own one. The paper will focus on the detailed implementation of the buffer manager and its main features. The experience and the achieved performance will be covered as well.

INTRODUCTION

FLASH is a complex machine containing about 1000 fast ADC channels distributed over tens of VME crates. More then 40 fast cameras are being used both for diagnostics and observation. In order to understand the machine behaviour one has to be able to correlate any diagnostics channels on the bunch-by-bunch level. In order to cope with this task we have developed a fast data acquisition system capable to write all required data with the full machine repetition rate (5Hz with up to 800 bunches per shot).

The FLASH DAQ system is dedicated to the following tasks: collecting LINAC beam relevant data in real time, providing the data to feed-back and monitoring tools as well as storing it for the off-line analysis. The DAQ system also allows storing user experiments data. The experiment can select any machine channels to be written along with the experiment data for making further correlations between the experiment measurements and the LINAC state.

DAQ ARCHITECTURE

The FLASH DAQ architecture is very flexible and scalable to satisfy the requirements of the machine diagnostics subsystems as well as the experiments. It allows integration of 'short live time' experiments.

The architecture of the DAQ system is shown in Fig.1. All collected channels are split into two groups: fast (fast ADCs, cameras) and slow (Magnet currents, etc.). Fast data is sent by means of UDP multicast and collected by the Fast Collectors. The slow collector receives data via TCP.

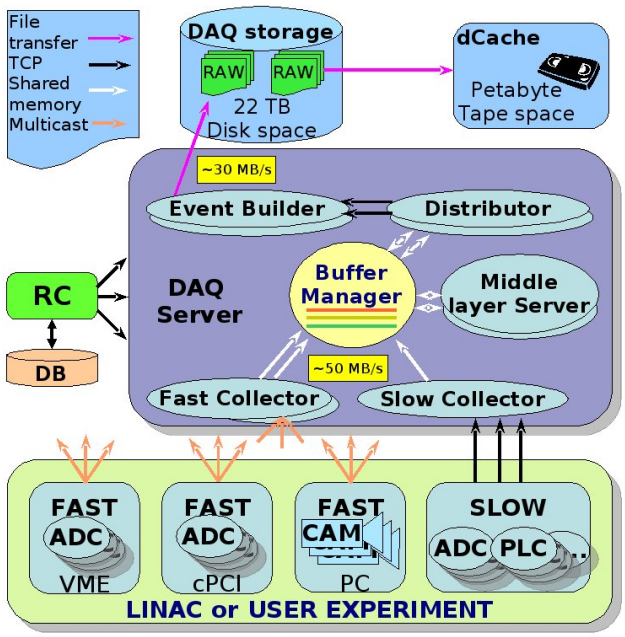

Figure 1: DAQ architecture

Both types of collectors are running on a powerful DAQ server computer (SUN Fire E2900, 16 cores, 32 GB, 6x 1GBit Ethernet). The collectors write data to shared memory segments controlled by the Buffer Manager (BM). The BM allows any process running on the DAQ server register either as data provider or as data consumer or as both. The collectors act as providers. Middle Layer servers use the collected data for special purpose operator monitors and feed-back subsystems. They usually act as data consumers, but if their data are required for off-line analysis they can also register as the data providers and write their calculated values to the BM. The Distributors extract data from the BM according to lists of channels received from the run control (RC) during the DAQ configuration. The extracted channels as data streams are pushed to the Event Builders which in turn stores the data on a local disk (1.5 TB, not shown on the picture) in a special optimized DAQ (RAW) format. The data from the local disk is transferred to a remote disk (22TB) where the machine data is kept for one month. The data for experiments is moved to a long term tape storage called Disk Cache (dCache)[3].

The RC process is in charge of the configuration and monitoring the whole system. The RC makes use of a Oracle Data Base to store all DAQ parameters. The

*Deutsches Elektronen-Synchrotron, DESY, Hamburg, Germany

Distributed Object Oriented Control System (DOOCS) [4] is employed for inter process communication.

BUFFER MANAGER

The buffer manager is the heart of the system and provides temporary data storage in memory, its synchronization on the shot-by-shot basis and a fast access for data reading to several data consumers simultaneously. The detailed BM description is given bellow.

Server Blocks

The data units the BM deals with are server blocks. They are produced by the front-end servers running on SPARC CPUs in case of VME crates with fast ADCs or on PCs under Linux with cameras or experiment specific hardware (e.g. compact PCI crates).

Fast ADCs channels digitizing devices of the same type are typically controlled by one DOOCS server. ADC clocks and triggers are generated by a hardware timing system. Trigger signals are generated by the timing system to start ADCs, trigger cameras and activate the readout by the software. The signal from the timing system contains the unique identifier (event ID) to identify data belonging to the same machine shot. Every front-end DOOCS server keeps the data for the most recent 16 accelerator shots.

Before the server sends data out to the DAQ it has to create a server block. Every server block consists of data from the channels collected by the server headed with additional information containing: time stamp, event ID, server block name (part of the DOOCS name of the server), status, trigger mask, number of channels in the block and the total block size. Every channel included into the server block consists of a channel header containing the channel name, channel type, channel status and channel data length. The channel data format depends on the device type (Beam position monitor, Toroid, Camera, etc.).

The front-end servers make use of a custom sender library to transfer server blocks via network by means of a multicast UDP protocol.

Events

All server blocks belonging to the same machine shot are merged into an event. There are two event type groups in the FLASH DAQ system: fast and slow. Fast events contain server blocks either collected by the fast collectors or generated by the middle layer servers. In both cases the data belong to one accelerator shot. The slow events consist of server blocks produced by the slow collector pulling slowly (maximum 1Hz) varying data from systems like PLC, etc.(e.g. vacuum, magnets, etc.). All events have an event header followed by a number of server blocks. The event header contains the following fields: pattern, time stamp, event ID, event type, status, trigger mask, number of server blocks and the total event size including the tail that is just the total event size.

The Architecture

The BM combines a number of shared memory segments and a library used by the buffer manager clients. The architecture of the BM is shown in Fig.2. There are two types of segments in the BM: control segments and data segments. The control segments include a client control and an event control ones. The client control segment consists of several slots and dedicated to

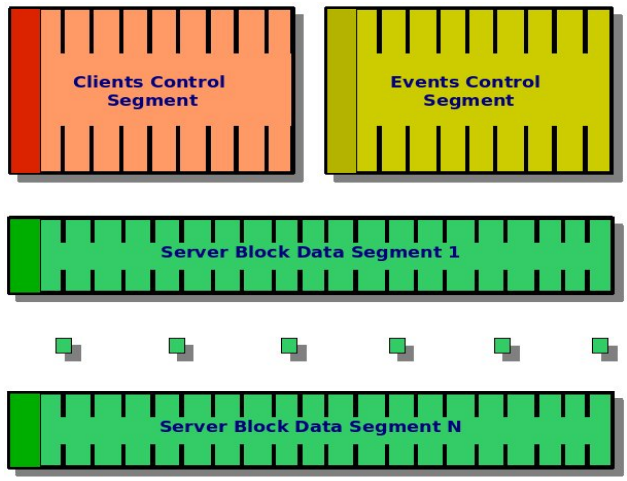

Figure 2: BM architecture

the registration of all processes willing to make use of the BM.

Every process has to register by the BM for every event type and a list of server blocks within the event of interest either as a writer (producer) or a reader (consumer) or both. One client has to use as many slots as required to register all its requests.

The information in the clients slots contains: process ID, last access time, event type, event count, client type (a producer or a consumer), priority, number of server blocks and their names. In addition every producer provides the maximum rate it is capable to write its server blocks. Every consumer in turn also provides information about maximum rate it accepts the server blocks and its control UDP port.

The event control segment also consists of several slots. Every slot is on-fly used by the BM to build up an event. The following fields are filled in the slot: event time stamp, event ID, event type, trigger mask, event status, time stamp of he first server block arrival, time stamp of the last server block arrival, producer process ID, event size, expected number of server blocks, number of filled in server blocks, number of clients interested in the event. The event slot fields are complete with the information about the clients and server blocks. Two fields are filled for every client: slot number in the clients control segment and the client status regarding the event processing. Every server block data containing: server block name, server block data segment ID and its slot number.

Both control segments are created by one of the clients acting as the BM master. The run control is responsible for the BM master assignment.

The server block data segments are created by the BM producers. Every data segment is split into some number

of slots. One data slot is used for one server block written by the producer - the owner of the segment.

In Operation

The BM master creates control segments according to the parameters provided by the run control (number of slots) and makes their initialization. After that all other clients including the master can start the registration of their requests. The RC takes care about the number of slots in the control segments in order to guarantee all request of the clients can be registered. The registration and all other interactions with the buffer manager are done via the BM library API written in C++. The concurrent access to the BM is controlled by means of semaphores located in the memory space of the control and data segments.

The procedure of the fast event building is described bellow. After the DAQ reaches the "run" state, all front-end servers start to send their server blocks for every accelerator shot. On receiving a server block the fast collector writes it to a free slot of its server block data segment. Writing server block to the slot automatically checks if a new event slot in the event control segment is required to start building a new event. If this is the case then a new event slot is reserved. If event building where this server block belongs to is already in progress then only the server block registration in the corresponding event slot is done. In both cases the fields "server block name", "segment ID" and "segment slot" are written to the event slot. On the registration of the new server block in the event slot all registered clients are checked with respect to their request fulfilment. If there are requests that need only the server blocks that have been already collected then the corresponding clients are informed via a UDP messages to the corresponding clients ports given in the client control segment.

On getting such a message the client fulfils its call back routines that perform required operations on the data of the server blocks. Once the routine is over the BM believes that the client has finished with the event. Its status in the event slot is market as "done". As soon as all clients in the event slot have the status "done" the BM consider the event as processed by all clients. After that the client slot and all corresponding data slots are released and can be used for building other events.

The buffer manager master has a watchdog thread that checks out if some event slots are used too long (a parameter in the BM master, usually 10 sec). If the event building time exceeds this time the event slot and all corresponding data segment slots are released (forced clean-up).

At the end of the DAQ run all BM clients remove their registrations. The BM master removes all shared memory segments.

EXPERIENCE

Running the FLASH DAQ since the spring 2005 has proved the correct choice of the system design and shared memory based approach for the event building. The buffer manager was verified reliably running at least at the rate of ~50MB/s. Most of problems leading to forced segments clean-up are traced to the incorrect behaviour of clients.

CONCLUSIONS

The FLASH DAQ has been developed and now is already reliable running more than 3 years. It provides a powerful service of collecting diagnostics and experiment data on the shot-by-shot basis.

Utilization of the shared memory for data collection was proven as a very efficient approach both for the event building and for the data access providing to the middle layer servers that require synchronized data from various sources in the FLASH.

REFERENCES

[1] Rossbach et al., "Generation of GW radiation pulses from a VUV free-electron laser operating in the femtosecond regime". Phys. Rev. Let., vol. 88, p. 104802, 2002.

[2] A.Agababyan et al., "Multi-Processor Based Fast Data Acquisition for a Free electron Laser and Experiments". IEEE Transactions on Nuclear Science, Vol. 55 No. 1, February 2008

[3] M.Ernst, P.Fuhrmann, M.Gasthuber, T.Mkrtchyan, C.Waldman, "dCache, a distributed storage data caching system", CHEP 2001, Beijing, http://www.ihep.ac.cn/~chep01/

[4] G.Grygiel, O.Hensler, K.Rehlich, "DOOCS: a Distributed Object Oriented Control System on PC's and Workstations", PCaPAC96, DESY, Hamburg, October 1996.