Data Archiving in Experimental Physics

Leo R. Dalesio, William Watson III and Matthew Bickley (Jefferson Laboratory),
Matthias Clausen (Deutches Elektronen Synchroton)

LANSCE 8, MS H820
Los Alamos National Laboratory
Los Alamos, New Mexico USA 87544

Abstract

In experimental physics, data is archived from a wide variety of sources and used for a wide variety of purposes. In each of these environments, trade-offs are made between data storage rate, data availability, and retrieval rate. This paper presents archive alternatives in EPICS, the overall archiver design and details on the data collection and retrieval requirements, performance studies, design choices, design alternatives, and measurements made on the beta version of the archiver.

1 Introduction

When designing a new archive facility for data archiving for the Experimental Physics and Industrial Control System (EPICS), a collaborative team was formed from Jefferson Laboratory (JLAB) and Los Alamos National Laboratory (LANL). The requirements were collected, the work was divided among collaborating institutions, and an incremental completion plan was made. As a preliminary step for the data collection and retrieval portion of the project, some performance studies were done to investigate alternative design options.

2 Requirements

Requirements were gathered from JLAB, LANL, Stanford Linear Accelerator Center (SLAC), and Argonne National Laboratory (ANL) for a variety of projects. Archiving at multiple workstations with the ability to archive the same channel in multiple archives. Data must be saved as single channels with varying rates and as sets where many channels are taken at a single point in time. Archive data must be available to distributed data viewers, data management, and data analysis packages, anywhere on the network. These requirements led to the design of a distributed data archiver that collects all data types and arrays (one-dimensional) of all data types from any I/O Controller (IOC) in the control network.

The requirements that affect the design of the archive file structure are data storage rate, data retrieval rate, and latency between archiving and using archive data. For long term storage of system parameters, SLAC requires 1,000 channels every 3 minutes; Jefferson Lab requires 6,000 channels every 10 seconds; and LANSCE requires 15,000 channels every minute. To support analysis of accelerator transient events, we need to save data on change at 5,000 channels per second and have the ability to trigger data buffers with before event and after event data (like a hardware scope). Channel data is wanted for data analysis and trending plots of channels through time. The trending plots of channels through time is the most demanding in that the last day′s worth of data containing up to 1,000 samples are required to be displayed within one second per channel. We are allotting 200 msec of this time for retrieval and 800 msec for display. In addition, 1,000 samples from the last 30 days must be displayed within 4 seconds per channel. We are allotting 3 seconds of this time for retrieval. Requirements were given to view data within a minute of data taking. With these goals in mind, we reviewed the existing archive facilities in EPICS.

3 Alternatives existing in EPICS

Several archiving solutions are already available in the EPICS toolkit. They were developed with different aspects of data archiving in mind. Alternative archiving methods include: ARR from LANL, a relational database solution that was done at Tate Integrated Systems (TIS), an IOC based archiver done by Kinetic Systems Corporation (KSC) and a distributed archiver and viewer done at Deutches Elektronen Synchrotronen (DESY).

The archiver that is distributed with EPICS was designed for archiving commissioning data for a linear accelerator. It handled five thousand event changes per second. The file size is defined at invocation and is not able to expand past this allocation. The user interface has been found to be difficult to use and the lacks feedback during data taking. As a result of the lack of positive feedback on data collection, several commissioning runs were without data although they believed it was being collected.

The KSC solution was designed for short term, high volume data in the industrial data acquisition market. It archives up to fifteen megabytes of data per second. The data is archived onto mass storage in the IOC under vxWorks. The data is only available for viewing and analysis after the experiment is complete.

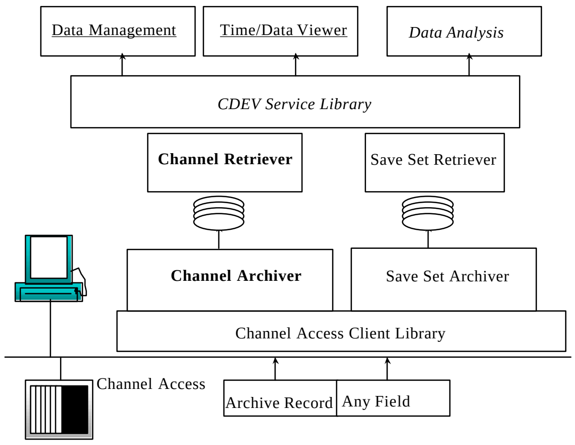
The TIS solution uses relational database technology. There is support for Dbase and ORACLE. The solution was designed for low volume data that will be kept for many years. Access to/from DBASE on a SPARC 5 was measured at 200 records per second, where each record can contain up to 255 values. On ORACLE, the performance on a SPARC 20 was measured at 50 records per second. The latency between data taking and data viewing gets long with large, high rate channel counts.

At DESY, the archiver was designed to keep facility information and make it available in a multi-control system environment using CORBA and IDL. In this design, each channel is stored in a separate file. This makes access time within a channel very fast. Data storage for a high channel count will slow archiving by requiring many files to be opened and closed. The file access issue is mitigated using front end buffering and flushing the buffers when needed.

Each of these solutions has been operational and successful. However, the full ranges of requirements were not met by any of them. Finally, we decided to design an alternative archiving facility.

4 Archiver design

The collaborative team from JLAB and LANL developed a preliminary design to meet the requirements. (Figure 1) The initial release will use the channel access protocol to collect data. There are two data collection engines included: one for collecting sets of data at a given point in time and a channel archiver for collecting individual channels at independent rates. The data retriever is initially connected directly to a set of data management routines, and has a set of routines available for use by data analysis packages and data viewers. There are two data viewers currently under development in UNIX and JAVA.



This paper will primarily focus on the channel-based archiver and retriever.

5 Data storage techniques

Optimization for data storage and retrieval is accomplished using several mechanisms. Data buffering is used to optimize storage rates. When the buffering is done in the front end computers, it optimizes network transmission and client side CPU utilization by minimizing the number of packets that are sent over the network and handled by the client. It also offers a backup storage when the archiving machines or the network are unavailable. The buffering increases the latency for data availability, however.

Using a streaming write on the client side optimizes disk access by removing any need to seek to an area before storing the data. This approach optimizes writing time, but introduces delays on retrieval time by requiring the retrieval routines to follow backward links to pick up a multitude of data chunks. Scrolling forward in time would be most severely impacted using this method. Creating and maintaining data areas for each channel, limits the number of file operations for data retrieval, but puts a burden on the data storage side to incrementally fill these areas by seeking and writing the next partial set of samples. It becomes clear that there is a large range of possibilities for optimizing either storage or retrieval. The optimization of one – tends to affect the performance of the other. To decide on an approach, the requirements were reviewed and some performance tests were made.

6 Performance of file operations

A study was done changing the amount of data, the buffer size, the number of writes, the number of seeks and the number of file opens/closes. (Table 1)

Channels	Disk Writes	Bytes	Seeks	Opens	Seconds	Bytes/Sec
1,000	100	1K	100K	1	198.1	505K
1,000	100	100	100K	1	19.6	510K
1,000	10	100	10K	1	1.2	833K
100	100	100	10K	1	1.2	833K
10	1,000	100	10K	1	1.3	769K
10	1,000	100	10K	1K	56.5	17K
10	1,000	100	100K	10K	454	2K

The table shows that opening and closing file descriptors on each operation results in a 90% reduction in performance. File open/close needs to be minimized. File seeks reduce the performance by about 40%. If we seek during storage, to group data into one area in the file then we reduce the seeks required to fetch it. If we stream data to disk then we require more seeks when the data is retrieved.

7 Adopted file structure

The file structure chosen attempts to balance the requirements for retrieving data against the storage rate. (Figure 2) To minimize the search time for a given channel, a directory is kept containing all channels that have been archived. A hash table is used to minimize the seek time. The entries contain pointers to the most recent and oldest data collected. Each data file contains a single area for each channel for the time period encompassed by the file. If each file contains 4 hours of time, then there is one area on the file for 4 hours of data for each channel included in the file. Each of these data areas contains a header with the first time, last time, number of samples, and pointers to the next and previous time periods. For channels that do not change in a given time period, no file space will be used. Using one file per time period minimizes the number of file opens/closes. Keeping a single file descriptor for a new file when the next time period occurs incurs only one additional file open. Fetching the data from the most recent time requires only two file opens, one for the directory and one for the data file. Seeking for an arbitrary point in time will require one file open per time period, with one file seek and one read of the data header.

8 Configuration parameters provided

Configuration parameters are provided to optimize

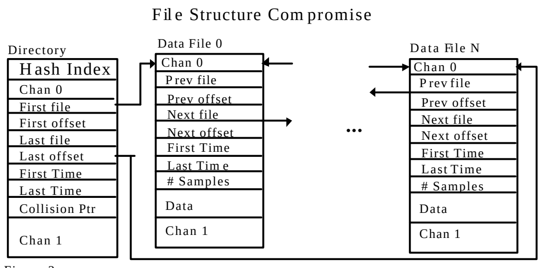


Figure 2

conditions for different requirements. The configuration parameters include data buffer rate, data flush to disk rate, time period per file, and channel access monitor threshold.

The data buffer rate is the maximum frequency that a channel is saved. The data is placed into a circular buffer in the client side at this rate. The rate is specified as a default rate in floating point seconds and supports a different rate on each channel.

The rate that the data is flushed to disk is an archiver wide parameter. It is the rate that the channel archiver will write data to disk. The buffers are allocated to hold twice as much data as is collected in the write frequency. This rate determines the latency between data taking and the use of data by the archive data viewers or analysis packages.

The time period per file determines the size of each file. The longer the time period per file, the larger the file. The more files that are created, the more file operations that are required to retrieve data.

The channel access monitor threshold is used to determine when the archiver will use channel access “monitors” and when it will use channel access “gets”. If a channel is changing at a much higher rate than the data buffer rate, there is a waste of network bandwidth and CPU cycles on the client side for processing events that are overwritten before they are archived. Using channel access “gets” to archive channels at a higher rate than they change, wastes resources as above by fetching and storing data that is not different. It is best to use channel access “gets” when a channel is changing at a higher rate than it is archived. It is most efficient to use “monitors” when a value changes at a rate that is less than or equal to the archive rate. The channel access monitor threshold is currently specified for an instance of the archiver.

9 future plans

In the near future, a standard set of retrieval routines is to be defined. With this standard library, we should be able to have access to all archive data with any viewers or data analysis packages. In addition, the use of the DESY archive buffering techniques will be integrated to allow data backup when an archiver is not active and higher bandwidth through front end filtering and buffering. More performance tests need to be done to characterize the bottlenecks and degraded modes of operation. Further optimization is expected.

10 Conclusions

The channel archiver is currently running at four collaborating laboratories. On a SPARC 10 we have measured 1,000 channels scanned in the IOC and changing at 2 Hz, sending 2,000 monitors per second to the archive client. Each entry includes an 8-byte time stamp and 4 bytes of status. The values are buffered and written to disk every 30 seconds. This was run for 3 hours archiving 432-megabytes of data, timestamps, and status with no data loss. CPU utilization on the SPARC 10 was measured at an average of 20%. Retrieval of 1,000 points from a single file was completed in under 200 msec. Retrieval of 1,000 readings from 30 files was done in under 800 msec. Further test are required to determine if the 5,000 channel per second archive rate will be met on a SPARC 10 class machine. A beta release is available for all code except that in Italics.

Acknowledgements:

This work is part of an extensive effort being put forward by Jefferson Laboratory and Los Alamos National Laboratory. Requirements and initial design work was done by a group from Jefferson Lab, SLAC, ANL, LANL and DESY. Stephanie Allison, Janet Anderson and Marty Kraimer made significant contributions during the initial phase of this project.