DATA ARCHIVING IN EPICS

K. Kasemir, L. Dalesio, LANL, Los Alamos, New Mexico, USA

Abstract

The Data Archiver is operational at the Low Energy Demonstrator Accelerator (LEDA) at Los Alamos National Laboratory (LANL). In the original design, several data taking engines were foreseen. In the implementation, the periodic engine and the monitor-based engine were combined. This paper discusses the performance, disk utilization, and use of the Archiver. It will also talk about plans for data management, the integration of the archive viewer from Jefferson Laboratory, and modularization to support other data storage methods.

1 INTRODUCTION

Since the original design, the EPICS Archiver has undertaken several significant changes. Most important has been the rewriting of the archive engine, so that the data retrieval and data collection would share a common set of access methods. This new work has been performed by Kay Kasemir at LANL. The original file structure and functions reported at the ICALEPCS in 1997 have been maintained [1]. This paper now covers the performance, disk utilization, use of the Archiver; the web based archive control and data viewing, the code structure and the current state of archive data management.

2 PERFORMANCE

Several performance tests have been performed on the channel Archiver. The original tests performed were to study the relative delays between seek, file open/close, large buffers and small buffers. The result was reported in the original paper. The later tests are performed on the new archive data taking engine and retrieval routines.

Retrieval tests were done on two data sets using an Ultra 1 from Sun Microsystems. The first set contained over 3.0 Gbytes of data taken over 1 year. A minimum of one file per day was created. Some of the early data had as many as six files per day. The second set of tests was done on a data file that was split from the original and contained only the last 3 months of data with only one file per month. This second file contained 1.3 Gbytes of data. In the first set of tests, data was fetched for the first of every month going back in time. Two measurements are given; one for CPU time used and the other for actual elapsed time. During tests on the large data set, where data was retrieved from the previous month, the CPU usage time ranged between 7 msec and 16 msec. Elapsed time ranged from 1 second to 2 seconds. Testing retrieval for each of the last ten months, the CPU usage time ranged from 120 msec up to 1,160 msec. Elapsed time ranged from 1 second to 98 seconds. The elapsed time became shorter as subsequent tests were run. It is our belief that as the tests were run, more resources were assigned to our task, but we have not studied this in sufficient depth. Another possibility for the disparity in retrieval times may be bad file links in the archive file. The delay of over a minute to retrieve archive data is unacceptable and we will continue to work to understand the issue. The second set of tests was done on the three monthly files that were extracted from the original. The total range for CPU usage accessing any of the data was 40 msec to 280 msec. The elapsed time never exceeded one second. This second set of tests was well within all performance parameters.

Data collection tests were performed on a 450 MHz Pentium II. In both tests, the buffers are being flushed to disk every 15 seconds, which means that we are buffering 150 samples per channel. When archiving 500 channels at 10 Hz, the CPU load was normally at 20% with peaks to 100% for five seconds every 15 seconds. Then a test was done for 1,000 channels at 10 Hz. The CPU usage was at 100% continually. In the last case, we are loosing a little data. Tests will be rerun with circular buffers that can support more than the number of samples required for the specified interval. In the second case, the web-browser was used to access the status of the Archiver while archiving was active. Status data took about 5 seconds to appear, which was much slower than normal. However, the Archiver continued to catch all signals.

Performance requirements for the Archiver stated that we would archive 10K channels per second on a $5,000 computer and access as many as 4 channels from as long ago as 30 days in under 4 seconds. Shortly, all of these requirements will be met. We still have concern about the numbers that we observed for accessing data from more than six months ago, however.

3 DISK UTILIZATION

Modifications since the original work include an incremental buffer assignment and some data compression capability. When data is being taken for a new file, the original buffer is allocated for 64 values. This minimizes the disk usage for a channel that is not changing very rapidly. If this buffer is filled, a new buffer in the same file is allocated for 256 samples. When this buffer is filled, all subsequent buffers are allocated at 1,024 samples. The file split utility also compresses these data files by combining the buffers into chunks up to 4,000 samples in length. In addition to combining the chunks, it also removes any unused buffer space. The data being archived includes the process value, time stamp and alarm condition. If a value does not change, two samples are stored, the first time it was at this value, and the last time is was at this value. Other than this, nothing is currently done to reduce the disk space being used.

4 ARCHIVER USE

The Archiver has been in use at LEDA for just over a year. It is also being tested at the Synchrotron Light Source (SLS) at the Paul Sherrer Institute (PSI) and ISAC at TRIUMF. At LEDA, we are archiving around 1,800 signals every 30 seconds. This is only 60 samples per second. In the first year, we collected over 3 gigabytes of data. Since August of 1999, we have been archiving approximately 1.2 Gbytes per month, which compresses down to 400 Mbytes per month. In addition, several archives have been set up at LEDA to find controller responses or study problem areas. These studies were archiving every change with a maximum rate of 10 Hz.

5 USER INTERFACE FOR OPERATION AND DATA VIEWING

The goal of the user interface for both operation and data viewing is to make status, control, and data retrieval available on any workstation with reasonable ease and acceptable performance. The original data archive engine had the user interface as an integral part of the code. This has two major problems. First, the Archiver could not be started without a windowing environment and therefore could not be started when the workstation rebooted. Second, the status of the archive engine was only available at the workstation where the Archiver was run. A data viewer from Jefferson Laboratory [R] that allows users access to this data provides a great tool for studying the data in the UNIX environment. At LEDA, we also need access to this data from PCs, so an additional tool was made.

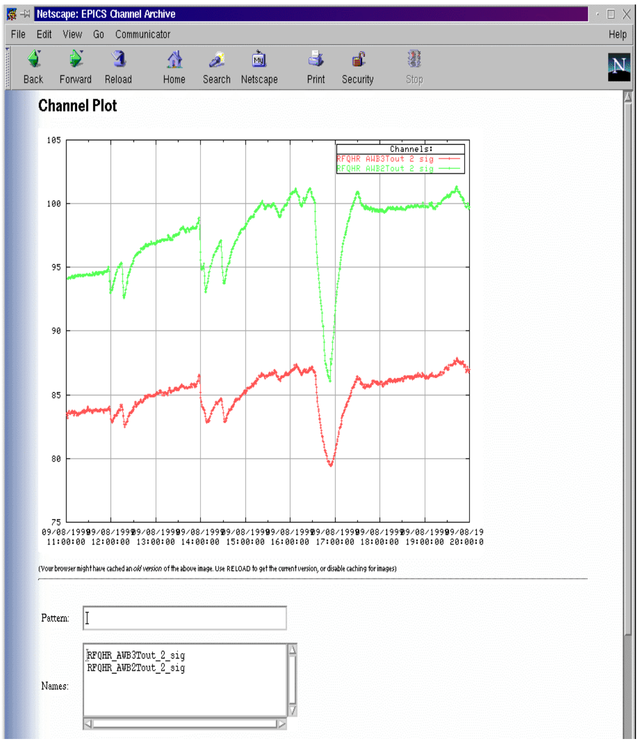
To determine the best approach to solving these shortcomings in an expedient way, alternatives were considered. This study of alternatives is presented in Table 1.

The availability of client programs for all platforms of interest (Win32, Unix) was considered the most important factor. A defined protocol is preferred over a newly designed protocol since it provides better chances for finding general tools, having a lower development cost, and ease shared development. Therefore, HHTP was chosen because it provided the bests results for little effort. The only problem with this approach is the handling of dynamic status updates. In this approach, there is no way of telling the browser about new information, since HTTP does not define a common server push mechanism. Using the client pull mechanism, the browser will simply update periodically, regardless of actual changes. As for any Epics addition, Channel Access (CA) was initially considered a natural choice. The engine would then serve pseudo channels like X_is_archived for each channel X.

	AS IS	*CA SERVER*	*HTTP*	*CORBA*	*BUILD IT*
CLIENTS EXIST	X11	EPICS CLIENTS	WEB TOOL	NONE	NONE
PROTOCOL DEFINED	N/A	YES	YES	YES	NO
1 CLEINT PER ENGINE	YES	YES	YES	YES	YES
> 1 CLIENT PER ENGINE	NO	YES	YES	YES	YES
CHANNEL SEARCH	NO	YES	NO	?	YES
>1 ENGINE PER CHANNEL	YES	NO	YES	?	YES
MONITOR >1 ENGINE PER CLIENT	NO	YES	No	?	YES
DYNAMIC STATUS UPDATE	YES	YES	PUSH	?	YES

Table 1: Alternatives for providing a user interface to the Archiver for data collection

Unfortunately, only few non-UNIX display tools are available for CA, and none of them fit the Archiver requirements well. CA is currently limited to a single CA server per computer that serves the same channel only once per sub-net. The custom TCP/IP protocol approach allows the implementation of all the desired features. However, the necessary development for protocol definition would definitely postpone the first release of a new archive engine. CORBA was not investigated thoroughly because of time constraints. It might still be the second best choice to HTTP for someone interested in writing sophisticated clients that need deeper access to the engine. The channel access server approach may be more attractive when the current limitation of one server per workstation is removed. The HTTP server was chosen and made operational in several weeks. Figure 1 shows a screen that is available for viewing archived data.



(Your browser might have cached an *old version* of the above image. Use RELOAD to get the current version, or disable caching for images)

Pattern:

Names: RFQHR_AWB3Tout_2_sig
RFQHR_AWB2Tout_2_sig

Figure 1: Data Retrieval

6 CODE STRUCTURE

A set of database-style C++ classes is under development for reading and writing Channel Archiver data files. The top-level *Archive* class provides channel information via *ChannelIterators*, which in turn can generate *ValueIterators* that hold a *Value* for a specific point in time. They are based on the current standard C++ library as described in [2] with few exceptions:

1) (FILE *)-I/O is used to access the Channel Archiver files because tests yielded poor results for seeks using the standard C++ fiostream classes
2) The standard exception class is not used as a base class because the present egcs C++ compiler (used on Linux, Solaris, HPUX) does not support them.

The *Value* class allows access to a value's time stamp, status information and numeric value in both numeric and string format, independent of the underlying *dbr_time_xxx* type which is actually found in the archive. An example of this code is shown in figure 1. The code is currently running on Win32 with Microsoft Visual C++ 6.0 as well as many UNIX systems, namely Linux, HPUX, and Solaris with egcs-2.91.66.

Whenever possible, these iterator-type helper classes should be used. Not only do they allow easy sequential access; they are also highly generic thus portable to a matching set of classes that could operate on a commercial relational database.

```
// Given a Channel Archiver directory file name
// and regular expression channel name pattern,
// list values stamped from start to end time
// for all matching channels:
void listValues (const string &directory, const
string pattern,
const osiTime &start,
const osiTime &end)
{
  Archive archive (directory);
  ChannelIterator channel =
    archive.findChannelByPattern (pattern);

  while (channel){
    cout << "Channel: " << channel->getName()
      << endl;
    ValueIterator value;
    for (value =
      channel.getValueAfterTime(start);
      value && value->getTime() < end;
      ++value){
           cout << *value << endl;
      }
    ++channel;
  }
}
```

Figure 2. Typical Use of the iterator class to report archived data

7 DATA MANAGEMENT

There are many ways to provide data compression. At present the only one supported will split a file into pieces and remove unused buffer space. This allows older data to be removed to secondary storage as a means of data management. It also allows data to be removed to other machines to be studied. In the future, a compressive tool to provide data management is planned.

8 CONCLUSION

The tool for providing long term archiving has been running at LEDA for one year now. A replacement that is written in C++ and produces the same file format is in beta test and offers a network interface to runtime archive information and archived data through a web browser. The new code also enables the replacement of the iterator class talking to an in-house file format, with data storage to a relational database or some other data store. In the future, a server like CORBA or Channel Access may make the data available though a client interface, but LANL is not planning such an effort. Both data taking engines are near meeting the requirement of saving 10K samples per second. They are able to retrieve data from the last eight hours in less than 1 second and retrieve the last 30 days in less than 4 seconds. Further work is required to provide an extensive set of data management functionality.

REFERENCES

[1] L.R. Dalesio, et.al. "Data Archiving in Experimental Physics," ICALEPCS'97, Beijing, Nov. 1997. http://www.aps.anl.gov/icalepcs97/paper97/p218.pdf
[2] B. Stroustrup, The C++ Programming Language, 3rd edition, July 1997, Addison-Wesley