D44Arkv1: Prototype on Archiving and Retrieving tool for the high frequency D44.

P. Lebrun
SDA memo

September 24, 2002

Abstract

A prototype of a java-based archiver and data retriever for the high frequency D44 data has been designed, mostly written and will be shortly running on the small Linux cluster at CPD. High frequency data as defined as 1Hz, or lower, dictated only by what the Front-End should deliver, and not by limitation of the datalogger maximum capacity and/or the archival system. A few Java threads are extracting data from the logger nodes and writing ASCII files containing the D44 data. Data compression is implemented. A retrieval class has also been written. This notes describes the motivation for such a product, the prototype, its current performance and limitations. A plan of action is suggested.

1 Motivation

Although the existing SDA data has been quite useful, some detailed and crucial analysis of the Tevatron performance requires more accuracy. This can be obtained from the high frequency data from the D44 dataloggers. A few critical devices, such as the FBI (Fast Bunch Integrator), the SBD (longitudinal emittance) have quite a bit of white noise, so that one must average over a large number of data points, while rejecting the obviously wrong points (glitches). "Blind avering "" at the front-end can be misleading, or totaly wrong when one studies the loss counters, where these glitches are of interest. Most important, the D44 data is not triggered, unlike FTP, SnapShots or SDA data. That is, we have a parallel Data Acquisition system, allowing for critical redudancy.

Using either the old D44 Vax page, the java-based D44 webpage or the OSDA-D44 interface, a few analysis have been performed on "recent" stores:

- Proton and Pbar loss through the Tevatron ramp, study of DC beam.
- Analysis of the losses at 980 GeV.
- Luminosity lifetime analysis.
- Pbar lifetime analysis at 150 GeV, correlation with Emittances

These analysis have been well received.

The D44 data is evidently used extensively in the MCR. Archiving the relevant D44 data, at 1Hz if available at the front-end has not been considered all that critical by a few advocate of quick, mostly correct, analysis done in MCR, after a few hours after the data has been taken. Yet, not all the plots can be produced on the spot, and some not so obvious correlations are missed. Quite a few of us acts like pathologits on seemingly old bodies (of data), looking for deeper clues. We really want to keep this data, such that such analysis can be reproduced by others.

Such a project was not thought to be necessary at the onset of the RunII program. However, (i) the Tevatron complex must operate at much higher performance compared to RunII, (ii) the cost of archiving has come down tremendously! Consider the cost of saving tens of GigaByte of data on spinning disks, compare to one day of RunII operation!

Evidently, we plan to save only the relevant data! We are not proposing to systematically save all the tens of thousands of ACNET device values, at 1 Hz, indiscrimininately. A few relevant Tevatron devices are listed in the result section along with an estimate of the total data load per year.

2 Design of the D44Arkv1 Product

2.1 The extreme simplicity of the D44 Data Model

Writing GigaBytes of data on disk is easy. Organizing it such that it can be archived on tape, and retrieved with a convenient interface might not be simple. Remember the "Data Access" problem during RunI, and the difficult start of the CDF centralized Data Analysis system, with respect to getting at the tapes! However, we have a much easier problem: the inherent data model is much simpler, and the data load is order of magnitude less than what's required for HEP RunII. While the latter advantage is documented down below, let me address the former one.

The atom of data is simply an ACNET device value, which can be either a scalar or an array (typically of a length of the number of bunch circulating in the machine, 4, 36,..). We record this data at a prescribed frequency. We retrieve the data, ACNET device, by ACNET device, for a a given period. That's it! The only access method we are asking for is

$$myVar.fetch(Date\,startTime, Date\,StopTime,$$
$$String\,ACNETDevice, double\,Frequency)$$

where:

- myVar is an istance of a D44 Archivable Variable (very similar to the exsisting OSDA D44Variable class)
- StartTime and StopTime specify the range of time for which we want the value of the ACNET device of interest.
- This String ACNETDevice has either the usual format of an ACNET device *"m:xxxxxx"* where m refers to the machine and xxxx the device of interest, or, for array devices is simply *"m:xxxxx[u:v]* where u and u refers to the start and end indices (inclusive range, folllowing FORTRAN tradition rather than C).
- The last argument might be optional: we may want to ask for the average, set by a frequency (asking for 30 sec average while the D44 data came from a 1Hz datalogger node).

Evidently, we could also provide convenience functions for averaging, rejecting bad points. However, that single *fetch* methods describes the simplicity of the Data Model.

Table 1: Tentative Classification for Tevatron devices

Class	Content
TeVIntensities	FBI (proton, pbar, wide and narrow gates), IBEAMx
TeVLoss	Proton and Pbar loss counter at D0 and CDF
TeVLuminosity	CDF and D0 Luminosity
TeVMagnetCurrent	Circuits, DFG's,..
TeVMisc	Others, no specifics yet
TeVSBD	Sample Bunch Display (T:SBDxxx)
TeVSyncLight	Synchrotron Light Devices (T:SLxxxx)
TevMagnetPos	Low Beta Quads Position sensors,..

2.2 Data Model

.

First the ACNET device can be split into a Machine and a Machine/Beam property Next, this property can easily be *classified* by what's measured: Magnet currents or Optics, beam intensities, Emittances,.. Such a classification is shown on table 1 for the Tevatron devices.

Within a given class, we define a "directory" for each ACNET device. Finaly, in each such directory, we create a permanent repository, a list of file paths, in fact, one line corresponding to each file corresponding to time period. Such time period are simply dictated by practical consideration of storage, speed at which the information gets written out, and, finaly access speed. Each file simply contains a flat table, with 2 colums for scalar device (time with respect to the startTime on the file, and the device value), for scalar ACNET devices. For array devices, the number of columns is two times the inclusive index range. Note that the existing datalogger treats each ACNET array device element separately, implying that they could be measured at slightly different time.

In doing so, we split the "Meta-Data" from the bulk of the data. The Meta-Data consists of the hierarchical structure *machine-deviceType/device* and, for each device, a directory listing keyed by times stamps. The data itself is in flat ASCII files. Data compression occurs in two distinct ways:

- Numbers are written out in ASCII form, using the java.text.NumberFormat class. We can set the number of decimal points. Most of the devices are not more accurate than $\approx 1\%$ (except the Toroids, but these are scalar devices!). So we do not typically need 8 bytes (a double) per number. In addition, this class truncates meaningless strings or zero values. Likewise, by only writing a time stamp, in seconds, with respect to the start time of the file, as an int, we again do not need 8 bytes per time stamp. Note that we still need this time stamp, because the dataloggers do not read data rigorously synchroneously with a master clock.

- Such ASCII files can be easily zipped.

2.3 Archiving

We propose to use Enstore as the D44 data archiver, with its disk caching system. Note that the archiving engine use only the Meta-data. Since this Meta-data is basically organized hierarchically (as oppose to a big Relational Database), it is easy to map this hierarchy to what Enstore excepts (i.e., a plain and simple UNIX File system).

3 Status of the prototype: the osdaarkv package

This OSDA-D44 java based package contains 5 classes, one for simply defining a D44 Variable that can be archived, one for defining the classification, one for the Writer, one for the Reader and finaly one Master Class to run the package for about 20 to 30 different ACNET devices.

It is straightforward java, except for:

- The zipping is done via a RunTime command. It is therefore system depedent. Since we plan to run this on SUN or on Linux, this is not a problem, as the "/usr/bin/zip" command is largely POSIX standard.

- The retrieval from the archive (Enstore) will also done via a *encp* command (see Enstore User's manual). These commands will evidently be "non-standard" and will also require a RunTime command.

Table 2: File size growth rate for a limited number of Tevatron devices. Compression factors and daily rates are estimated. These measurements were taken during normal opration of the Tevatron.

Device	Frequency (Hz)	ASCII rate (MgB/day)	Zip Compression factor
T:FBIPNG[1:36]	0.56	19.9	5.1
C:B0ILUM[1:12]	$\approx$ .03	0.30	4.8
T:SLPSV[1:36]	.08	3.04	4.2
T:SBDPSS[1:36]	.3	10.5	3.95

This last item has not been yet implemented. The package runs on Windows or on Linux. Most of the code consists or exception handling and, for the Writer (the biggest and most complex class), having the ability to stop and restart the system, picking the stream of data where it was left off.

Multiple instances java threads (processes) can run concurrently to build this new D44 data repository. Based on current performance of the OSDAD44 servlets and interfaces, one thread can save all Tevatron Intensity devices, and emittances, and (probably), the luminosity, at 1Hz. We plan to keep running this prototype on the patmu3.fnal.gov Linux node to study such performance issues, for a few weeks.

4 Results from the prototype

4.1 Data Compressions and Mass Media requirements

Because the code was still in a debugging phase while such measurements took place, the prototype ran for about only 10 hours, with numerous stops and starts. However, files on which such measurements were made were complete, in the sense that there were no blank periods. From table 2, we derive table 3, coming up to yearly rate of 91 Gb for the Tevatron devices, un-zipped (decompressed) and $\approx$ 20 Gb when compressed.

Evidently, many devices were omitted. Even if this rate is underestimated by one order of magnitude, it is still ridiculously small compared to what a modest fraction of what Enstore can deliver. This is duable!

Table 3: Tentative estimate of data rate

Class	Equation	Ascii Rate (MgB/day)	Zipped Rate (MgB/day)
TeVIntensities	4*20 + 5	$\approx$ 100	20
TeVLoss	2*20 + 20	60	13
TeVLuminosity	2*10	20	5
TeVSBD	4 * 11	44	11
TeVSyncLight	8 * 3.1	25	6
TevMagnetPos	?	?	?
TeVMagnetCurrent	?	?	?
TeVMisc	?	?	?
Total (tentative)	-	250	55

4.2 Learned lessons

1. Robustness and Error recovery. This is 90 % of the code. Extensive use of Exceptions, not much choice.. Because the File Repository contain all the information, it is possible to "catch up" while the OSDA Servlets must be restarted. This has been implemented, further debugging and optimization is required. However, restarting the D44Arkv threads is currently done manually. This might not be feasible, if the OSDA-D44 servlets needs to be restarted often. We have more work to do to make this system more robust.

2. The datalogger do not really achieve their advertised rate. In addition, the data does not necessarily shows up in sequential order, and some data duplication occurs as well. Thus, a cleanup utility to re-order the delivered data will have to be implemented in the D44ArkvReader. Such a function already exists in the OSDA-D44 package, it can be easily re-implemented.

3. Data Organization: This simple data organization has a drawback: the length of these ASCII files must be optimized carefully. During write and archiving, one can and must make these files relatively large (hundreds of MgB). During read, one must avoid reading these tens of MgB unecessarily, while reaching for the time period of intenrest. Fast read access means therefore a lot of little files, which is far from

optimum when archiving, or simply listing files. This deficiency can be addressed by pushing the bulk data one directory down, organizing the time periods into "macro-periods" rather than having all the data for one device in one directory. The entire macro-periods are archived as a tar file, rather than individual files. This has not been implemented in the prototype.

4. No formal databases have been used. We entirely rely on the Unix file base system, and on flat list of files in the Reposity listings. Evidently, such listing could and probably should be transfered to a Relational database, for ease of use and maintainability.

5 Outlook, Plan of Action

This feasibility study has been completed after about 1 man-week worth of effort. Making it a real product should be less than 2 to 3 man-months, if based on the same model. Note that a model where the OSDA-D44 layer is entirely removed could be discussed: we could re-write the BD Control datalogger DAE to write directly the type of files presented here, on a local repository connected to Enstore. However, one must still keep the current dataloggers to support the existing data browsers and applications. In addition, some buffering of the data on the DAE engines must be done, to streamline the creation of our D44Arkv files.

Thus, the above architecture will remain in place. We have to work on robustness and i/o optimization of the OSDA-D44 servlets. For instance, we could exchange binary data instead of XML files. And we have to implement the missing parts of the package (archiving and retrieval). Therefore, the plan is:

1. Review of this document.

2. Design.

3. Consolidation of the OSDA-D44 and programmatic improvements to the existing Datalogger (move to Unix, higher performance and reliability).

4. Implementation, Collaboration between BD/Control and CD/CPD, CD/OD and CD/ISD. Only one programmer is needed, with consulting support from members of these Department.

6 Credits

Jim Kowalkowski, Jean Slaughter and myself met and discussed this project. The idea of cleanly separating the Meta data from the bulk of the data and the use of simple ASCII files for our problem emerged from these discussions. Don Petravic strongly encouraged this project and suggested to use Enstore. The OSDA-D44 system was written by Timofey Bolshakov and Suzanne Panacek. Finally, let me thank Jim Patrick and Bryan Hendricks for their support.