SETUP OF A HISTORY STORAGE ENGINE BASED ON A NON-RELATIONAL DATABASE AT ELSA

D. Proft*, F. Frommberger, W. Hillert, ELSA, Bonn, Germany

Abstract

The electron stretcher facility ELSA provides a beam of unpolarized and polarized electrons of up to 3.2 GeV energy to external hadron physics experiments. Its in house developed distributed computer control system is able to provide real time beam diagnostics as well as steering tasks in one homogeneous environment. Recently it was ported from HP-UX running on three HP workstations to a single Linux personal computer.

This upgrade to powerful PC hardware opened up the way for the development of a new archive engine with a noSQL database backend based on Hyptertable. The system is capable of recording every parameter change at any given time. Beside the visualization in a newly developed graphical history data browser, the data can be exported to several programs - for example a diff-like tool to compare and recall settings of the accelerator.

This contribution will give details on recent improvements of the control system and the setup of the history storage engine.

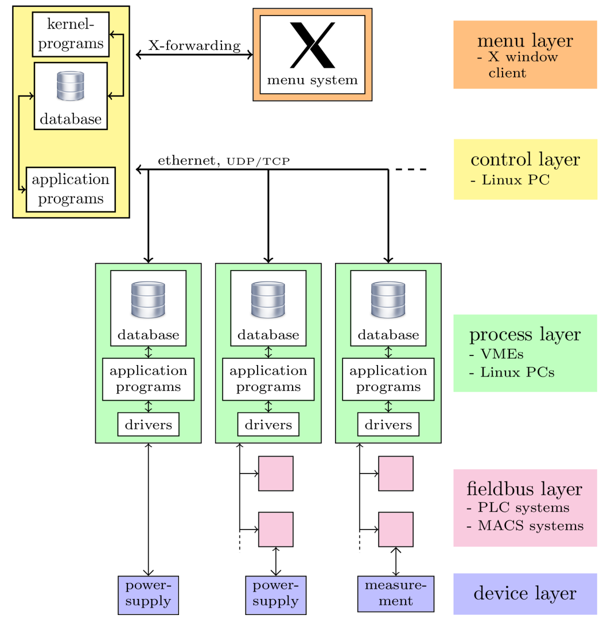

Figure 1: Hard- and software layers of the control system.

INTRODUCTION

The main features of the ELSA accelerator control system [1, 2] include a completely event based data handling model and a separation of the core functionality (database and event handling by the *kernel*) from userspace applications. It combines steering tasks and real time beam diagnostics in one homogeneous environment. A transparent design allows access to the X windows-based graphical user interface from any computer. An overview of the hard- and software layers of the whole system is given in Figure 1, [3].

A key component of the control system is a kernel managing a central shared memory database. The database is separated into several parts, i.e. the *resource base* containing structural information about parameters like limits, max. number of vector elements and the quantity's physical unit. The structural information is complemented by the online database filled with actual parameter values, which are updated continuously at runtime.

There are currently 14 827 parameters defined in the control system. These are grouped into *controlled* ($\approx$ 4000), *measured* ($\approx$ 9000) and other parameters. Each group consists of four different data types: analog values (represented by floating point numbers), digital values (mostly switching values or integers), strings (character sequences) and arbitrary byte sequences.

The update of controlled parameters occurs rather rarely, and is mainly invoked by user interaction or automatic measurement processes. On the other hand most measured parameters are updated on a regular basis, either cycle-synchronous (typically every 5 s) or with arbitrary rates up to 10 Hz. Parameters with higher rates are accumulated in vectors and transfered (also on a regular basis) with a slower rate.

The data rate produced by 675 updates/s (on average) is roughly 50 kB/s to 100 kB/s[1]. This results in a total volume of $\approx$ 6.1 GB per day.

Primary goal of the newly developed archive engine is, of course, to archive all these changes together with a timestamp, regardless of the type or source of the values. Second goal is to keep the investment cost as low as possible. Therefore the archive database should run on a regular desktop computer with no special hardware needs. Here, a bottleneck could be the access time, in which the data can be returned back from the database. For best user experience access times in the magnitude of few seconds are required.

DATABASE BACKEND

Hypertable is a non relational database with Google's *Bigtable* design which was chosen as the database backend. It runs on top of several file systems, including distributed ones (e.g. *HDFS*) and storage in the local file system. The instances of the main server, called *RangeServer*, can be distributed among different machines with one *Master* process for administration.

* proft@physik.uni-bonn.de

[1] 50 kB/s during maintenance, 100 kB/s during usual operation

ISBN 978-3-95450-146-5

Hypertable uses a *key-value* based data storage model. The key itself is made up by a row key string, a high resolution nanosecond timestamp, a column *family:qualifier*-pair and control flags. The timestamp can be understood in two ways: First as a simple timestamp either assigned automatically upon creation or given by the user and second as a revision of the *key-value* row. The column family[2] represents the *column name* in relational databases. These fields are assigned to the archive engine fields as shown in Table 1.[3]

Table 1: Hypertable ⇔ Archive Engine Field Assignment

hypertable	archive engine
row key	parameter name
column	fixed column family name "data"
timestamp	recorded parameter change date
value	parameter value

The *key-value* pairs are sorted by their key and stored inside the memory in *CellCaches* or they are written to compressed *CellStores* residing on disk. The data on disk is supplemented by a block index, to increase search performance.

This type of data storage directly implies the optimal way of data readout: Because the data is sorted by the key (i.e. parameter and timestamp) it is most efficient to read out a big time frame for a single parameter. This is exactly what most of the history-tools (and especially the history-browser application) require, so it matches the requirements for the database backend.

Currently the hypertable database (one *Master* and one *RangeServer*) is running on the same machine as the control system. It is equipped with an Intel i7 CPU with six physical cores, 40 GB RAM[4] and two desktop harddrives with each 3000 GB capacity configured as a raid1 (no distributed file system is used at the moment).

INTEGRATION INTO THE EVENT SYSTEM

The interface to the database backend is set up on the control host. The shared memory database running here has a consistent view of all parameters and their current values. Upon each parameter update, the event system is triggered to inform other applications of the value change. At this point a new hook was installed to communicate with the history database.

For the implementation, emphasis was put on the strict separation of the control system's core and the database communication. Therefore a new shared memory database was introduced to act as an intermediate database. Whenever a parameter gets updated, a nanosecond timestamp with the current time is created and stored in the shared memory database along with the parameter's name and value. Numerical values (integers and floating point values) are stored in their binary representation with 32 bit size[5] and strings as zero terminated character arrays.

The isolation from the control system core is achieved by using only one application with access to both systems. Its purpose is to flush the contents of the intermediate database every three seconds and insert the appropriate records into the hypertable database. Each new record is filtered by a regular expression during the insert to filter away unneeded parameters by name to save storage size.

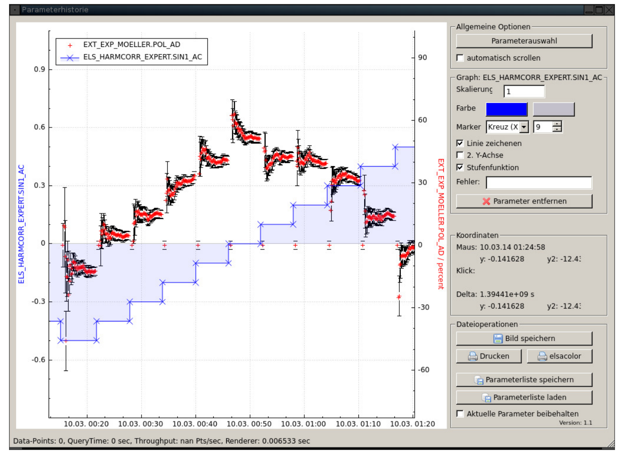

Figure 2: Graphical user interface: parameter history browser showing the correlation between set point value and corresponding measurement value.

TOOLS

For interaction with the history data a couple of tools have been developed. The most important one is a graphical user interface, which can be directly invoked from the accelerators menu system (see Figure 2). Within the GUI, one can ask for values of multiple parameters and have them plotted versus time. The application is based on *QCustomPlot*, a Qt plotting widget with integrated support for easy panning and zooming by mouse.

Beside simple tools for extraction of data to ASCII files and plotting in gnuplot there are two important applications:

cshdiff

This application creates two snapshots of the values of all parameters at two given dates and afterwards reports any differences between them. The number of parameters can be filtered by type (e.g. controlled or measured parameters) and by name (regular expression). That way, a comparison of two different machine states is easily achieved. This information may then be used to restore the accelerator to a previous state.

[2] Because only one column is used for the historic data, this feature is effectively unused.

[3] The parameter name used as the row key is suffixed by a date based string for faster indexed searches and due to a maximum revision count in hypertable.

[4] Before a recent upgrade of the control system to 64 bit the usable RAM of the database was limited to 2 GB. All further performance analysis has been performed with this limitation present.

[5] Accordingly, vectorial parameters are stored as $n \times 32$ bit values.

ISBN 978-3-95450-146-5

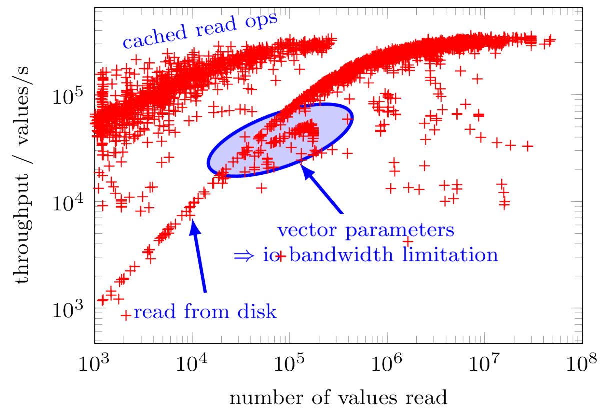

Figure 3: Parameter readout performance vs. number of parameters read.

csplayback

This application is designed to be executed on a development system running the control system. Instead of connecting the control system to process hosts, only a single application (beside the crucial kernel applications) is running. It downloads all parameter values from the database system for a given time range and feeds them back to the development system. This allows a real time playback of all accelerator parameters of any time range in the past[6]. From the control systems point of view there is no difference in data handling in comparison to normal operation with attached process hosts, because the same event- and notification system is used.

The most important scenario where this feature can be used is software development: On the development system new software, for example measurement applications, can be tested offline and no dedicated beam time is required.

PERFORMANCE

The most used feature of the archive engine is the history database browser. For maximum user experience a fast readout and display of the data is required.

Figure 3 shows the basic readout performance of the database system after 8 month of operation. Every data point represents the throughput during readout of all values stored in the database of one accelerator parameter. Dependent on the update rate of each specific parameter, the total number of values per parameter varies among 5 orders of magnitude. The data was collected in a random order during usual system load, especially the collection of new data was not interrupted.

Most of the parameters with only few data points (less than 1×10^5) can be hold in cache, thus being accessible directly from the RAM. These queries can be executed at high throughput and are located in the upper left region of figure 3. If the data is not cached, the readout of small amounts of data takes significantly longer due to an additional overhead by I/O latency of the hard drives and on-the-fly decompression of the data. The throughput increases with bigger amounts of data being read, because the time needed for preparation of the data is constant. On the other hand, parameter values which are vectors (n-tuples of scalar values instead of single scalar values) can only be read out by a lower rate due to I/O bandwidth limitations.

[6] since begin of the recording of course

Figure 4 shows the total time required for export versus the number of values queried from the database. The readout again was performed in a random order and takes less than 1.5 s for the readout of up to 10 000 values. Above that point the throughput is dominated by the delay given by I/O operations for reading the *CellStores* from disk and the corresponding decompression.

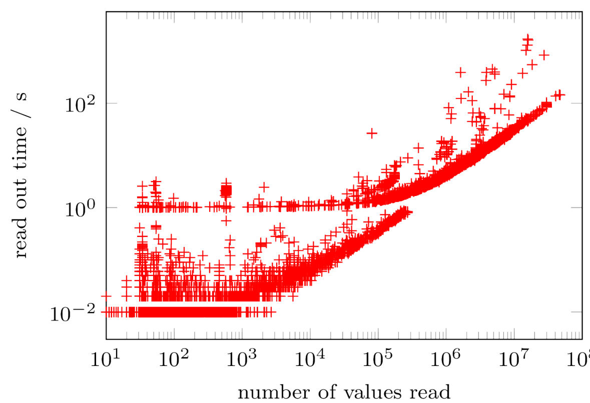

Figure 4: Readout time vs. number of parameters read.

CONCLUSION

The possible uses of the archive engine overshoot the simple recording and display of data: Now post-mortem analysis of component failures are possible. One can find correlations between different parameters - either controlled or measured ones - and watch their evolution over time. For that, the most important improvement introduced is the graphical history browser application. It quickly became an integral and vital part of the control system.

REFERENCES

[1] T. Götz, “Entwicklung und Inbetriebnahme eines verteilten Rechnerkontrollsystems zur Steuerung der Elektronen-Stretcher-Anlage ELSA, unter besonderer Berücksichtigung der Anforderungen des Nachbeschleunigungsbetriebes bis 3.5 GeV”, PhD theses, University of Bonn, 1995

[2] M. Picard, “Entwurf, Entwicklung und Inbetriebnahme eines verteilten Rechnerkontrollsystems für die Elektronen-Stretcher-Anlage ELSA, unter besonderer Berücksichtigung der Extraktion im Nachbeschleunigungsbetrieb bis 3.5 GeV”, PhD theses, University of Bonn, 1995

[3] D. Proft, “The accelerator control system at ELSA”, IPAC2013, Shanghai, May 2013, THPEA002, p. 3149.

ISBN 978-3-95450-146-5