SETUP OF A HISTORY STORAGE ENGINE BASED ON HYPERTABLE AT ELSA

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

Abstract

The electron stretcher accelerator ELSA serves external hadron physics experiments with a beam of unpolarized and polarized electrons of up to 3.2 GeV energy. Its in house developed control system is able to provide real time beam diagnostics as well as steering tasks in one homogeneous environment.

The existing archive engine, a simple application logging parameter changes to a file storage, was unable to cope with the rising amount of parameter updates per second. Therefore a new storage system based on the non-relational database system hypertable has been introduced. It is capable of storing huge amounts of data to distributed storage systems, thus being able to handle the recording of every parameter change at any given time. The data can be read back with low latency to a newly developed graphical data browser using a C++ interface.

This contribution will give details on the setup and performance of the history storage engine on top of hypertable.

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 (Ref. [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.

One third of the 55 applications attached to the control system are so-called *expert engines*. They represent the physical intelligence of the control system, bringing in any physical calculations needed to operate the accelerator. Each expert engine can handle a set of rules which are basically finite state machines. The rule engine is supplied with a consistent database snapshot of all parameters captured at the same time, and itself writes all computed values back to the online database. The other applications are either *kernel applications* (these take care of memory management, process net communication, etc) or userspace applications.

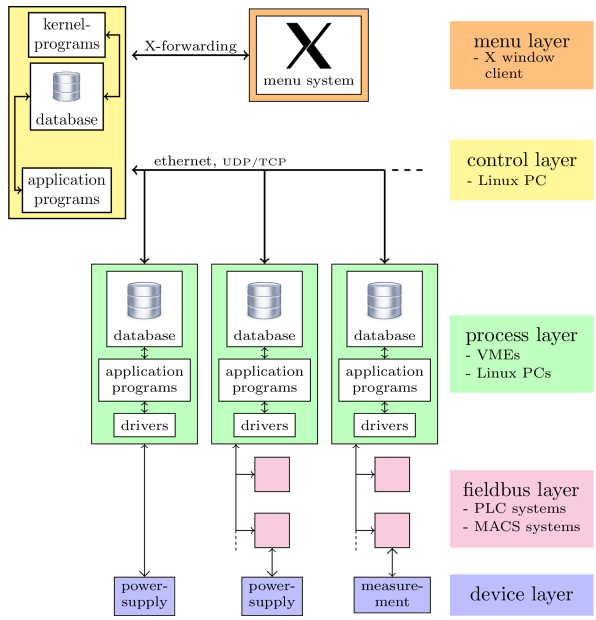

Figure 1: Hard- and software layers of the control system.

There are currently 14 274 parameters defined in the control system. These are grouped into *controlled* (≈ 4000), *measured* (≈ 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.

In total there are on average 675 updates/s. The data rate is roughly 50 kB/s to 100 kB/s[1] resulting in a total volume of ≈ 6.1 GB/d.

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.

* proft@physik.uni-bonn.de

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

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.

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. On the other hand, the performance is quite poor for many-parameter-few-value access patterns.

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, 8 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). Thus, the additional cost of the system was just the investment into two hard drives. Furthermore it is easy extensible by using fallow hard disks on different process hosts, which can be used as additional *RangeServers*.

[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 had to be suffixed by a date based string 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.

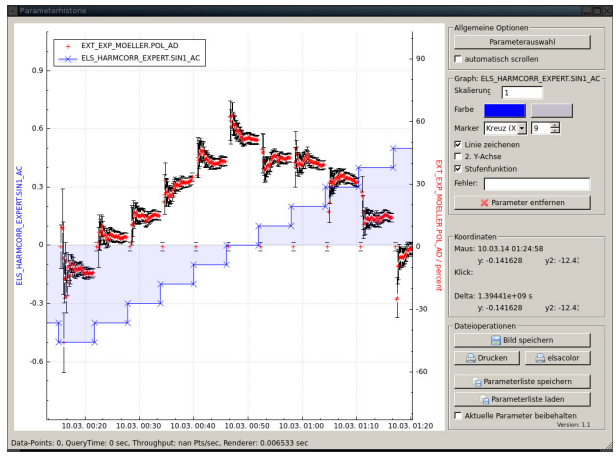

Figure 2: Graphical user interface: parameter history browser.

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.

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.

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

ISBN 978-3-95450-132-8

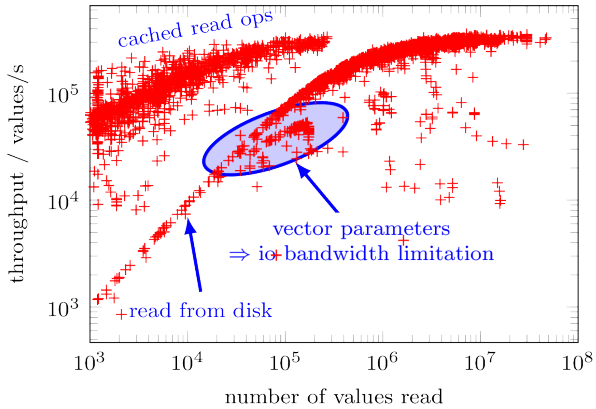

Figure 3: Parameter readout performance vs. number of parameters read.

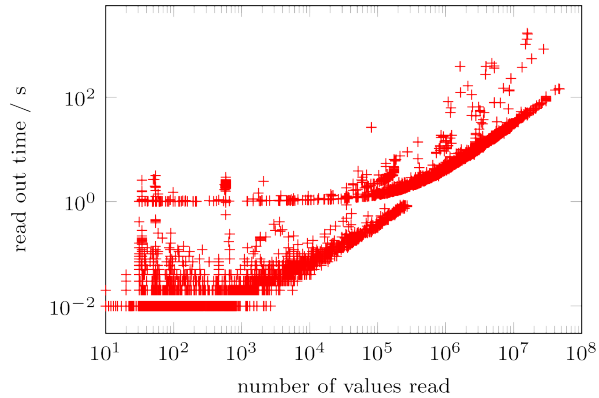

Figure 4: Readout time vs. number of parameters read.

Other implemented tools are:

- *cshexport*: Console application to bulk-export parameter values of given time series to (gnuplot compatible) ASCII files.
- *cshplot*: A console application for plotting history data using gnuplot.
- *cshget*: A console application to lookup single values of parameters at given dates.
- *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.

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 random access memory. 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.

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.

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.