THE POLYHEDRA® HISTORIAN
How Enea's Polyhedra database products can handle large volumes of time-series data

Contents

The limitations of an in-memory DBMS

The Polyhedra IMDB is designed for use in embedded systems, where configuration information and data about the current state of the system has to be kept readily accessible, rapidly alterable… and safe. To ensure fast access, the data is kept in-memory, but can be backed up by a variety of mechanisms such as snapshots, transaction journals, and even a hot standby where appropriate.

In 32-bit systems, memory addressability puts a limit on the size of the database. Sometimes, though, there is a need to store large amounts of data, perhaps extending to tens or hundreds of gigabytes, or even terabyte. Even with the 64-bit edition of Polyhedra, the cost of memory (and address-space limitations imposed by motherboards, say) will mean that there are many circumstances where the data cannot all be held in memory.

While Polyhedra cannot offer a generic solution for handling vast amounts of data, it does have a special module for capturing and preserving time-series data, which addresses the most common reason that databases in Polyhedra's target markets can grow very large. This document discusses the capabilities of this module, known as the Polyhedra® Historian, and describes how it can make the data available both for dynamic retrieval for displaying trends and also for more complex offline analysis.

The Polyhedra Historian

In an embedded application where, say, the current values of sensors or shares are kept in-hand in a database, it is often useful or important to know how the values are changing over time. In some industries, for example, it is a legal requirement to keep historic information online for at least 90 days, and offline records for 10 years. In a system monitoring 10's of thousands of analogue sensors whose values are changing by the second, this can rapidly add up to a huge volume of data that can swamp a normal database, let alone an in-memory one!

Fortunately, there are characteristics to this kind of data - and in particular, to the way that it is used - that means we have been able to develop a special module for the Polyhedra IMDB to handle this requirement. The Historian sub-module is part of the database server, and can be configured (by creating and updating records in certain tables) to set up a number of 'logging streams', each monitoring a given table with a chosen sample interval (or 'on change'), and then, for each logging stream, specify:

- which columns are to be logged;
- which column, if any, is to be used to flag the records to be logged;
- which column, if any, holds a timestamp field to be used (instead of 'now') in the logged samples;
- the amount of disk space to be used as a circular buffer containing the raw sample data;
- the amount of disk space to be used to hold time-compressed data at varying intervals; and.
- the names of 'pseudo-tables' that can be queried to retrieve data

It is thus possible, just by using standard SQL 'insert' and 'update' statements, to set up a series of configuration records that say:

- *"Monitor the table called ANALOG, logging away the (floating point) VALUE attribute and (integer) STATUS attribute every 5 seconds, but only for those records where the 'LOG_ME' attribute is true; use a 5-megabyte in-memory buffer to capture the time-series data, and use a gigabyte file (organised as 200 blocks of 5 megabytes each) as a circular FIFO buffer of raw samples. Use another 200 megabytes for time-compressed records at 1-minute granularity, and another 200 megabytes at one hour granularity (based on the 1-minute samples); when producing time-compressed samples, record the min, max and average for the value column, and both the bitwise AND and the bitwise OR of the status field. Raw samples can be retrieved through a table called ASAMPLE, time-compressed samples via a table called ASAMPLE_C."*
- *"Monitor the table called DIGITAL, logging away the (Boolean) VALUE and (integer) STATUS attributes 'on change', using a 200 megabyte file as a circular buffer of raw samples; allow samples to be retrieved through the DSAMPLE table."*

With such a set-up, one would retrieve information about earlier values by querying one of the special tables that had been set up:

```
select * from dsample
where name='D4RS' and timestamp > now()-hours(1)

select * from asample_c
where name in 'J4RU,J4KT'
  and granularity='3600s'
  and timestamp>'05-JAN-2003 09:38:15'
  and timestamp<'19-JAN-2003 09:38:15'
```

The former retrieves the last hours-worth of samples for a given digital point, the latter retrieves 2 weeks-worth of 1-hour time-compressed samples for two analogue points. The Historian is optimised for queries such as these, and maintains its own indexing on the file contents so as to reduce the number of file accesses required when performing queries. Historic queries can take some time if blocks of data need to be retrieved from disk, but this does not block other activity occurring on the database; in particular, Historian queries do not stop other clients from updating records in the tables that the Historian is monitoring.

The Historian is not limited to logging numeric values; it can log fields of any type supported by Polyhedra. Thus, to record events, one notes that they are normally associated with objects already being represented in the database; one simply adds an extra string attribute to store the most recent event associated with that object (and perhaps a datetime attribute to record a timestamp), and configure the Historian accordingly:

- *"monitor the latestevent and timeoflatestevent columns of the component table, using a 100 Mbyte file to hold the samples that are recorded on change, and allowing the samples to be retrieved through queries an a table called event."*

Archives and extracts

While the Historian undoubtedly can extend the amount of data stored by a Polyhedra database, there are still limits on the amount of data storage - though this time it is because there are limits on the size of a file. To cope with this, the Historian introduces the idea of archive files and extract files:

- At any time, you can instruct the Historian to produce an archive file containing the oldest <n> blocks of un-archived data from the buffer file. A file of the indicated name is produced, and the blocks in the buffer file are marked as archived. An archive file can be brought back online by creating a configuration record in the database. If a query is made of a special Historian table to retrieve samples, such as the example queries given earlier, then the Historian does not limit itself to using the information in the main buffer file, but it also uses any applicable archive files that contain data for the relevant period.

- As well as archives, it is also possible to generate extract files. When doing so, you specify the time period in which you are interested, and you can also specify the data points in which you are interested, and the Historian will produce a new file with all relevant information currently on-hand. Extracting information in this way does not affect the archival status of information in the main buffer files, but once produced extract files are the same format as archive files, and can be brought back online in the same way.

Archive (and extract) files are in a machine-independent format, and are not tied to any particular database instance; consequently, archive and extract files can be brought back online into any database with a similar Historian configuration. This gives great flexibility in the way the Historian is used: it is possible, for example, to use the Historian on an embedded system as a data logger in a vehicle; when back at base, the archive files for recent journeys could be transferred (via a network, perhaps, or removable media such as flash cards or CD-R) to a workstation and then brought online for detailed analysis.

Automated production of archives

By deriving new tables from the Historian configuration tables and then attaching CL-coded behaviour to the new tables, you can automate the system so that whenever the Historian buffer file gets full - or, more accurately, when most of the data in the buffer file needs to be archived to avoid the risk of old samples being overwritten - new archive files are produced. The Historian uses the table 'logdata' for configuring file buffer sizes, recording information about the proportion not yet archived, and for triggering new archives, so we can simply derive from that table to add the attributes we need:

```
create table auto_archive_logdata
(           persistent
,           derived from    logdata
,           filenamebase    char    default 'archive-'
,           log_num         integer default 0
);
```

The trigger code is also very simple:

```
script auto_archive_logdata
    on  set percentfree
        if percentfree < 50 and not archiving then
           debug "trigger auto-archive" && log_num
           set archbuffercount to buffercount div 2
           set archfilename to filenamebase & "_" & log_num
           add 1 to log_num
           set archiving to true
        end if
    end set percentfree
end script
```

This can be readily extended so that the most recently-generated archive files are automatically brought online as soon as they are created, with older archive files taken offline.

Exporting time-series data

It obviously fairly straightforward to develop a Polyhedra client application that queries the historical data, and then exports it to another database (either directly via its ODBC or JDBC interface, or by producing, say, an XML file which is uploaded by a separate task). However, the Polyhedra Historian comes with two mechanisms that taken together can enable ready exportation of the historical data with low latency and low overhead on the Polyhedra database server.

Exporting archive files

The internal structure of an archive file is proprietary, and subject to change (though, in keeping with Polyhedra's general compatibility principles, if any changes are made then the new version of Polyhedra will be able to handle both the old and new formats); the details of the structure is not published. Instead, a tool is provided that inspects an archive file and produces an XML description of the contents; this XML can then be used to transfer the information into a standard database system for offline analysis. The conversion tool does not need to be able to access the database server that generated the archive file, and does not have to be running on the same machine; indeed, as the archive files are platform independent, the conversion tool does not need to be running on the same type of computer as the one on which the archive file was produced.

For completeness, while the database service is not running, the conversion tool can be used to analyse the files that the Historian uses in a circular fashion to hold the online data.

Live streaming of historical data

To supplement the file conversion tool mentioned above, the Historian can be configured to provide a special TCP/IP port, which can be used to access the live data that has not yet been archived. A special C-callable library is provided to handle the connection to this port, along with an example application (in both executable and source code formats) that illustrates how the library can be used to launch a connection, request a live stream of data, and to extract the individual samples from the stream of data. The library allows a start time to be specified, to allow the retrieval of existing samples (including those that have been written out to the circular buffer file as well of those that were in the in-memory buffer at the time the connection was established).

Other Historian tools

Archive file merging

By their nature, archive files are smaller than the circular buffer file from which they were generated – and often there are good reasons to make them considerably smaller: they can be generate more quickly, they can be copied backed up and retrieved from archive storage more quickly, and one does not have to wait for the main circular buffer file to become almost full of unarchived material before generating the archive file. However, if storing data that has been produced over a number of years, this can result in a very large number of archive files. To allow the number of archive files that are preserved to be reduced, the Historian has a special tool that allows a number of archive files to be merged into a single file. Thus, if the circular buffer file could hold three days-worth of samples but there was a requirement to keep 90 days-worth of data online for quick access, then you could have periodic batch jobs that made sure that the following was done:

- Every one or two hours, the Historian is told to produce archive files containing all unarchived data, which can be taken offline as a short-term safety backup. These files will be relatively small, and can be generated (and backed up) quickly. However, over the course of a year or more the number of such files would get large!
- At midnight, all archive files generated the previous day are merged into a single archive file; the resultant file is then registered with the Historian as being online, and the one from 90 days ago deregistered and deleted. These archive files will typically be a third of the size of the main circular buffer file, and the number of files online at any time is manageable. (It does not matter that there is an overlap between the online archive files and the data that is in the circular buffer file, as the Historian will avoid getting duplicate information.) Once the daily archive file has been produced (and backed up), there is no need to keep the archive files from which it was made.
- Every week, the daily archives files from the last 7 days are merged together, and the resultant file stored for long-term backup. If information about the weekly archive file is stored in the database, it would then be easy for the operator can flag it for retrieval, and a helper application that monitored that table can then get the file from backup storage and tell the Historian when it is available.

While the details will needed to be adapted to cope with particular circumstances (where the rate of change is very high, for example, or if there are different requirements on the length of time the historic data has to be kept available), a scheme such as this one minimises the number of files that have to be kept long-term, and also minimises data loss if the file system is damaged – especially if the circular buffer file is on a fault-tolerant RAID disk array.

Summary

In embedded systems, the main reason that databases have to cope with huge volumes of data is the need to keep time-series information, to allow trends to be displayed and analysed. The Polyhedra Historian component is specifically designed to capture and preserve such data, complete with an archive mechanism that allows old data to be taken offline when not needed, and with the ability to set up a data port to allow time-stamped samples to be streamed out to an external application (which can batch them up and store them in a disk-based database, say), for offline analysis. The power and flexibility of the Historian allows the Polyhedra IMDB to be used in applications that need both the capability of storing large volumes of write-once data plus the performance of an in-memory DBMS.

for more details of the Polyhedra® products, please visit www.polyhedra.com or www.enea.com/polyhedra, or email us at info@enea.com