Telemetry Archiving: An application of non-normalised database design or How to optimise storage efficiency, retrieval speed and real-time performance.

Christian Kumpf
Lead Software Engineer, MakaluMedia Aerospace, Darmstadt, Hessen, 64293, Germany

Over the last three years, Makalumedia have developed a high performance telemetry data archive for Eutelsat. This archive initially served as data storage component of the TeleViews web application but was designed from the beginning to support a wider range of applications. It stores change-only telemetry data for all satellites (currently 25) in the Eutelsat fleet on two dedicated servers using standard PC hardware.

In this paper I'll describe in detail the design of this archive, especially how we tailored the storage structure to typical use cases in accessing telemetry data; it supports extremely fast data retrieval speeds along with quasi-live streaming.

The design of the archive is very scalable and it is straightforward to add new satellites or move data between servers without interrupting the availability of the archived data.

The use of standard interfaces and protocols eases the integration of the archive in the distributed structures of modern control systems and organizations. Archived telemetry data is made available to other applications via a simple HTTP interface. Data is currently served in either JSON or CSV formats and extension to other formats is straightforward. The archive also supports several server-side statistical calculations, for example minimum, maximum and average, over any desired time interval and sampling period.

I. Introduction

Satellite telemetry data has a couple of special properties that makes it challenging to archive. Beyond the vast amount of data, the different kinds of diversity are the most challenging as they represent a wide range of real world objects, i.e. temperatures, switches, frequencies, clocks, voltages. Even if we call all of them TM parameters, we have to cope with various data types (floating point numbers, integer numbers, status codes, time stamps) as well as a wide variety of update frequencies. Looking at a typical telecommunication satellite, we find that 85% of the parameters change at most once a day (if at all) whereas about 2% change at least once a minute and contribute to 98% of the total amount of data.

Besides the actual calibrated telemetry value itself we have to store the raw value and its asserted validity as well as its out of limits status.

A. Use Cases

Together with our customer, we have identified three different uses case with respect to the archive solution:

- the most basic query: get the value of telemetry parameter at a specific point in time.
- get all values of a telemetry point during a period (i.e. for plotting charts).
- get the list of all telemetry parameters that are out of limits at a certain point in time.

Clearly, the latter two use cases could be implemented using the first query, but they have higher demands on performance.

B. Constraints

Another challenge is caused by the fact, that we have to deal with Eutelsat's complete fleet of spacecrafts over their complete lifetime. With 25 spacecrafts currently in orbit and an expected lifetime of at least 15 years per spacecraft this means more than 375 years of telemetry. On the average we have almost 8 million value changes per satellite and day. For each parameter value change we record attributes listed in table 1.

Attribute	Size *(octets)*
parameter id	2
point in time*	8
raw value	4†
calibrated value	4
validity	1
OOL status	1
	20 byte/sample

Table 1. Attributes of TM value change

As we have to expect at least $375 \cdot 365 \cdot 8 \cdot 10^6 \approx 10^{12}$ samples, we would have to cope with at least 20 Terabyte of data, not counting any overhead imposed by database indexes, file systems etc. As each sample itself is of comparatively small size, the offset imposed by database indexes and the file system could easily add to a multiple of the minimum 20 bytes required. When using MySQL ISAM tables, the indexes have almost the same size as the table itself for these small records.

Even if disk space itself is rather cheap these days, it becomes more expensive when you hit the border of 2TB as more complex solutions than a single hard disk or RAID-array are required. These more complex solutions (like SANs) not only have a cost effect but typically also a penalty on performance, especially latency. The more complex a storage solution is, the more expensive is also its maintenance and the more demanding are regular tasks like backups.

C. Existing Solutions

During our initial research we evaluated commonly used systems to archive satellite telemetry. All of them use change-only archiving.

MUST had been developed by ESOC and uses a second normal-form database layout to store telemetry. It partitions the TM data by data type, i.e. all parameters of a data type share the same table.

DARC had been developed by Logica with the same approach as MUST for the data storage. The major difference to MUST of the prototype we've evaluated had been the usage of InnoDB as storage engine.

ARCHIVA by GMV introduced a couple of fresh ideas. It partitions the tables for telemetry data by time, i.e. having a separate table for each month of the calendar. Also it uses a different approach to distinguish between data types. Instead of having different tables for each type, it just has two different tables. One for data width size 32 bit or less and one table for data with size 33 to 64 bits.

To achieve reasonable speeds when extracting data over longer periods of times (e.g. for plots), GMV also introduced statistical levels that contain hourly, daily averages and extremes. [1]

When we run our initial tests on these systems, they all performed reasonably well for a couple of days when retrieving values for TM parameters at arbitrary points in time. But for all of them performance dropped drastically after these few days by several orders of magnitude. After approximately the same time, the performance dropped again considerably.

What happened? Each of these solutions started to suffer from increased I/O activity as soon as the size of the database hit the amount of RAM available for I/O buffering to the Linux kernel. The second drop came when the database indexes exceeded the amount of RAM available to the underlying database server. For MUST and DARC retrieval of time series became too slow for a satisfactory user experience.

DARC had been affected somewhat earlier than MUST by the underlying problem as the InnoDB engine has a larger per-record storage offset than MyISAM.

ARCHIVA had the advantage of the precalculated statistical values as long as query interval had a matching table in the ARCHIVA schema. For arbitrary intervals it performed not better than MUST or DARC. Also the statistical tables nearly doubled the storage demands.

A programming language independent API hadn't been available to us for any of those solutions.

II. Telemetry Parameter Archive

Given the experiences with the existing systems we derived the goals for our archive solution:

- minimize the number of I/O operations required to load a time series of a parameter.
- minimize the storage requirements of the data.

- decouple the archive access from the underlying storage algorithms.

- it is not feasible to expect human intervention in order to tune the archive for individual TM parameters as the sheer number of telemetry points ($> 30,000$ per satellite) forbids this.

A. Exposed Interface

As we designed the archive around the queries we wanted or needed to support, I'll take the API as a starting point and describe the architecture later on.

A simple, language independent interface had been very important to us. Therefore the API of the Telemetry Parameter Archive (TPA) supports queries using HTTP[2] GET requests. We have a dedicated archive server instance for each satellite. Responses are in JSON[3] format. HTTP libraries and JSON parsers are comparetively lightweight, widely available and easy to implement for new domains.

We currently support the following queries

parameter set values what is the value of a given set of parameters now or at a certain point in time? (and)

parameter data series I what are the value changes of a TM parameter during a given period of time? (changes)

parameter data series II what are the average/minimum/maxmimum values of a TM parameter for all intervals of length Δ during a given period of time? (statistics)

out of limit which parameters are out of limit at a given point in time? (OOL)

B. Architecture

As we have based our architecture on the supported queries, I'll describe the individual storage components by the queries they are meant to support.

1. Time Series

Whereas we found it acceptable to have a small penalty on retrieval of a single TM parameter we wanted to improve speed on extraction of time series.

Therefore we especially focused on minimizing the number of I/O operations on extraction of large data series, i.e. changes and statistics queries.

We assume, that TM data is stored in the archive the very moment it arrives in the control center (or a short period afterwards needed for decommuting and processing). With the normalized approaches above, data series queries suffer largely from database fragmentation as consecutive values of the same TM parameter are interspersed with TM values of other parameters. In the worst case (which is reached after a couple of days) we have to support multiple disk accesses to read the index as well as for the value itself for each sampling point.

We mitigate this effect by accumulating consecutive values for the same parameter and store them in packed records (see C. below). For each of these packed records we store a row in the database holding meta information about this record, namely begin time, end time and parameter id. This approach has three beneficial effects to our query:

- the table containing the packed record meta data is several order of magnitudes smaller than a corresponding normalized table would be. Therefore its associated database indexes are smaller as well in fit into RAM almost all times, thus avoiding I/O accesses during locating the data.

- The number of I/O operations can be reduced considerably as we can now read up to several thousand parameter values in a single operation w/o suffering from fragmentation.

- the total amount of data to be read is smaller due to the smaller requirements of the packed blocks.

The accumulation of consecutive parameters is performed by storing the in-stream of TM data in a "conventional" database (mid term archive) until the number of stored samples reaches a given threshold. Then these samples are deleted from the mid term archive and are stored in a packed record of the long term archive.

These scheme adopts more or less automatically to different parameter update frequencies as the majority of TM parameter representing status codes change rather seldom and therefore stay completely in the mid term archive whereas parameters that represent physical values (and hence needed to be plotted or analyzed of periods of time) change frequently and end up in the long term archive (that supports fast extraction over time) sooner or later.

2. Statistical Time Series

Extraction of statistical time series uses the same data structures as the extraction of changes as the statistical values are computed on the fly. This has the advantage the the sub-interval length Δ can be chosen freely. Performance is sufficient as we achieve currently extraction speeds of $100,000$ samples/second on rapidly changing parameters.

3. Monitoring

When monitoring a satellite, one typically displays a few dozen or even hundred parameters (so called ANDs) in "live" mode. As this is the most common case among the displays opened on our system, we support this special case with a dedicated database table that holds the most recent value of all TM points. This dedicated table considerably lowers the load on our archive servers and improves user experience as these pages load instantly. Retrieving historical ANDs might require a couple of database and file system accesses; but as long as the response time is below one or two seconds, this acceptable for the users.

4. OOL Monitoring/Retrieval

A very special case is the retrieval of OOLs at a certain point in time. A database query over $> 30,000$ parameters to retrieve their OOL status might easily take several seconds in any of the analyzed systems. The question for the most recent OOL change before or earliest OOL change after a point in time becomes infeasible.

In order to support the queries, we take advantage of the fact that on a well configured satellite the number of parameters, that are out of limits, is comparatively small. We use the table holding parameter's current values introduced for monitoring in section 3. to regularly create snapshots of the current overall OOL status. We also create samples of a pseudo TM parameter OOLCHG whenever a TM parameter changes its OOL status. This way we can use the time series of that parameter to monitor the OOL changes (and step forward and backward between them). To get the overall OOL status at a certain point in time, we just have to load the most recent OOL snapshot before that time and apply all changes recorded in OOLCHG to it.

C. Long Term Archive Packing

In order to minimize the storage requirements of the LTA, we've designed a packed format for the samples as listed in table 2.

Attribute	**Size** *(bits)*
combined validity/OOL	2
time offset(ms)	30
raw value	8-32
calibrated value	0-32
	5-12 byte/sample

Table 2. Packed TM sample record

As we are not interested in the OOL status of invalid samples we can combine validity and OOL status into two bits with the values in table 3.

Code	**Meaning**
0	data invalid, OOL status not associated
1	data valid, within limits
2	data valid, outside of soft limits, inside of hard limits
3	data valid, outside of hard limits

Table 3. Combined Validity/OOL Status

The packed format supports several sub-formats depending on the actual range of values used. When packing the data, the packer examines the complete set of values to be stored in that package for leading zero bytes. So if e.g. all values fit in the range 0-65535 only two bytes are used for storage. This check is performed independently for raw and calibrated values.

Also we compare the raw and the calibrated values. If they are equal all the time - a rather common case -, we store only the raw value.

Time is stored as millisecond offset to the first sample time of the package. Given the range of 30 bits, this allows for a little more than a 12 days period in the same package.

Whenever a package required more than one disk block[‡] we also applied *deflate*[4] compression.

Using these packing and compression strategies we achieved and average sample size of 5.44 bytes/sample[§].

Within a packed record, samples are stored in order of increasing timestamps so we can use binary search to locate individual samples.

III. Maintenance Aspects

A. Database Tuning

Database engines are not well tuned for data that gets frequently deleted. After a lot of deletions[¶] they suffer from "holes" in the table spaces that are hard for the to regain. We apply a trick to mitigate this effect: we partition the table of the mid-term archive by parameter id based on their update frequency. Once a week, we calculate a partitioning scheme for the table to put the ≈ 300 most frequent changing parameters into separate partitions and combine the remaining ones into partitions of approximately equal size. This procedure has two effects:

- The table space and the associated indexes are completely rebuilt. This regains all "holes".
- Separation of the most frequent parameters into separate tables enables the database to speed up insert/deletions of these parameters.

B. Backup Strategies

After a couple of week the database size for each satellite becomes more or less constant as any incoming data will be compensated by data moved to the long term archive.

The major part of the data belongs to the long term archive. Any data stored there will never change. As we are storing the LTA data as files, a backup strategy is straight forward. The LTA will be backed up incrementally plus a snapshot of the managing tables and the short/mid term archive can be used to restore the system.

The ability of restoring the LTA content independently from the database helps to minimize the required downtime when a recovery/restore is needed.

C. Scalability

The overall processing of the TPA is I/O bound. Frankly spoken, the more independent disk sub-systems and RAM a server has, the more satellites it can host. However, with increasing number of satellites and increasing size of complexity of them, there will always be the case that the I/O bandwidth of a single server will be exhausted.

As we assign different port-numbers to each server instance, an HTTP proxy (we use Apache at Eutelsat) can be used to make distribution of the fleet's archive among multiple servers transparent to client applications.

Storing the LTA data in files also eases the migration from one server to another as it can be copied to the new server in advance w/o affecting current operations.

IV. Results

At Eutelsat we are currently archiving telemetry for 25 spacecrafts on two servers equipped with 24 GByte of RAM and 2 TByte of disk space each. We now (January, 2011) have a total of 12,500 days (34.3 years) of telemetry archived. The total size of the mid-term archive and all management tables is about 50 GByte. The accumulated size of the long term archive is 500 GByte including any file system overhead.

We store now $98 \cdot 10^9$ TM value changes on these two servers and have capacity for eight times the current amount/archiving period. Given any of the other examined storage architectures we would have exceeded our capacities over a year ago.

The TPA currently servers as archive to the TeleViews application providing direct TM data to all space engineers at Eutelsat. Besides interactive Web access, the engineers also build a couple of analysis applications that connect directly from Excel to the archive.

After a couple of code optimisations Eutelsat's engineers are satisfied also with extraction speed of statistical values.

[‡] 4kByte in our reference implementation

[§] 5.56 bytes/sample when we add storage of the database index tables

[¶] and we are deleting over 90% of the data

Appendix

Acronyms

AND	Alphanumeric Display
API	Application Programming Interface
CSV	Comma Separated Values
HTTP	HyperText Transfer Protocol
I/O	Input/Output
JSON	JavaScript Object Notation
LTA	Long Term Archive
OOL	Out Of Limit
RAID	Redundant Array of Inexpensive Disks
RAM	Random Access Memory
SAN	Storage Area Network
TB	TeraByte, 2^{40} byte
TM	TeleMetry
TPA	Telemetry Parameter Archive

Acknowledgments

We'd like to thank Eutelsat for the opportunity to develop this parameter archive and their patience during the *operational validation phase*.

References

[1]Thomas Morel, Gonzalo Garcia, M. P. and Gil, J. C., "High Performance Telemetry Archiving and Trending for Satellite Control Centers," *Proceedings of SpaceOps 2010 Conference*, AIAA, Huntsville, AL, 2010.

[2]Berners-Lee, T., Fielding, R., and Frystyk, H., "Hypertext Transfer Protocol – HTTP/1.0," RFC 1945 (Informational), May 1996.

[3]Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)," RFC 4627 (Informational), July 2006.

[4]Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3," RFC 1951 (Informational), May 1996.