Astronomical Data Analysis Software and Systems VII
ASP Conference Series, Vol. 145, 1998
R. Albrecht, R. N. Hook and H. A. Bushouse, eds.

A Queriable Repository for HST Telemetry Data, a Case Study in using Data Warehousing for Science and Engineering

Joseph A. Pollizzi, III and Karen Lezon

Space Telescope Science Institute, 3700 San Martin Drive, Baltimore, MD 21218, Email: pollizzi@stsci.edu

Abstract. The Hubble Space Telescope (HST) generates on the order of 7,000 telemetry values, many of which are sampled at 1Hz, and with several hundred parameters being sampled at 40Hz. Such data volumes would quickly tax even the largest of processing facilities. Yet the ability to access the telemetry data in a variety of ways, and in particular, using ad hoc (i.e., no a priori fixed) queries, is essential to assuring the long term viability and usefulness of this instrument. As part of the recent NASA initiative to re-engineer HST's ground control systems, a concept arose to apply newly available data warehousing technologies to this problem. The Space Telescope Science Institute was engaged to develop a pilot to investigate the technology and to create a proof-of-concept testbed that could be demonstrated and evaluated for operational use. This paper describes this effort and its results.

1. HST as a Telemetry Source

The Hubble Space Telescope (HST) is well a known source of significant and substantial amounts of Astronomy data. Less known, however, is that the HST is also one of the most highly instrumented non-manned platforms ever launched. Over 6,000 telemetry points are monitored on the HST. These "monitors" cover practically every aspect of the platform and of the various instrument environmental and state conditions.

In addition to routine study and problem analysis, we use telemetry to look for long term trends in how the platform is behaving. By carefully studying such trends, we hope to uncover potential problems before they arise. In this way, we plan to extend the scientific utility of this unique instrument as far as possible through its planned lifetime (now scheduled through to 2010).

1.1. Complications in Querying Telemetry Values

Since telemetry values are sampled at widely different rates, looking for a "cause-effect" relationship between monitors can rarely be found by identifying time matches between records. Rather, the queries tend to look for overlapping time windows when the monitors acquire some state. We have coined the term, "Fuzzy Query" to describe this kind of query. Using a stylized SQL, a fuzzy query typically appears as:

Select $ParamA \ldots ParamN$ where $Param1 > somelimit$ AND $Param2 > somelimit$ AND $\underline{Time(Param2) \leq Time(Param1) + someDelta}$

The underlined portion in the above query highlights this 'fuzzy' aspect. An SQL designer will recognize the complexity such a query requires.

1.2. Dealing with the Telemetry Volume

Then there's the shear volume of the telemetry data. At its nominal format and rate, the HST generates over 3,000 monitored samples per second. Tracking each sample as a separate record would generate over 95 giga-records/year, or assuming a 16 year Life-of-Mission (LOM), 1.5 tera-records/LOM. Assuming a minimal 20 byte record per transaction yields 1.9 terabytes/year or 30 terabytes/LOM. Such volumes are supported by only the most exotic and expensive custom database systems made.

By careful study of the data, we discovered two properties that could significantly reduce this volume. First, instead of capturing each telemetry measurement, by only capturing when the measurement changed value - we could reduce the volume by almost 3-to-1. Second, we recognized that roughly 100 parameters changed most often (i.e., high frequency parameters) and caused the largest volume of the "change" records. By averaging these parameters over some time period, we could still achieve the necessary engineering accuracy while again reducing the volume of records. In total, we reduced the volume of data down to a reasonable 250 records/sec or approximately 2.5 terabytes/LOM.

2. Data Warehousing as a Potential Solution

The complexities and expected volumes in dealing with telemetry data naturally lead us to consider Data Warehousing. Beyond its data handling abilities, Data Warehouses are designed to be balanced in their approach to the data. That is, they expect to handle the ad-hoc type queries with little or no pre-knowledge of what will be of interest.

2.1. What is Data Warehousing

From a User's perspective, a data warehouse looks very similar to a typical relational database management system (RDBMS). A user will find a query language very similar to (if not the same as) SQL. Typically, the warehouse will support one or more programming interfaces, such as ODBC, which allows the warehouse to be accessed by familiar reporting or analysis tools (such as Microsoft's Access, IDL, PV-Wave,)

To the database designer, a different perspective is shown. There is no transaction or update facility for the warehouse. The warehouse operates either with many readers or a single writer, and the warehouse is "loaded" as opposed to being updated. In laying out the warehouse, the designer quickly learns that their logical definition (i.e., the layout of the tables and attributes of the warehouse) is more intimately tied to the physical definition (how the warehouse is laid-out on the physical i/o subsystem). Choices in one will often significantly affect the other.

In giving up the flexibility in transactions and by having the closer linkage between the physical and logical views, the warehouse provides a number of new features particularly in supporting efficient indices for very large data volumes.

2.2. The Star Index

One of the most common indexing methods that characterizes a data warehouse is called the Star. In using a Star index, the designer starts by designing a few number of *fact* tables.

Generally, a fact table can be viewed as a flat file version of an entire collection of typical database tables. The goal here is not to normalize data. Instead, a fact table attempts to bring together as many related attributes as can be expressed in a single table with (effectively) an unlimited number of columns.

While a fact table holds the records of interest to be searched, *dimension* tables provide the meta-data that describes aspects of a fact table and supports the rapid indexing of data. A dimension table can be formed for any column within a fact table, when all possible values for that column can be taken from a pre-defined set of values. For example, a column holding a person's age can be reasonably limited to the set of whole integers from $\{1 \ldots 150\}$; sex from { "Male", "Female" }. Even an arbitrary set of values is appropriate. Consider the social security numbers for all the employees of a company. While the individual numbers themselves may be an arbitrary string of digits, all the numbers are known and can be listed within a dimension table.

The star index then relates a fact table to one or more corresponding dimension tables.

3. Resulting Fact Tables

Applying this strategy to the HST telemetry problem produced the following Fact Tables:

- Telemetry Facts with the columns: Mnemonic, Format-type, Flags, Raw Value, Start Time, Stop Time, Start Millsec, Discrete Value (NULL if the telemetry parameter is not a discrete), and Engineering Units (EU) Value (NULL if the telemetry parameter is a discrete).

- Averaged Facts with the columns: Mnemonic, Format-type, Flags, Start Time, Stop Time, Start Millsec, Averaged, Maximum, Minimum EU Values, and Number of Samples in period.

and with the dimension tables:

- Mnemonics - enumerated all Mnemonics, and other meta-data

- Time - enumerated as years, months, days, hours, minutes, seconds

- Milliseconds - enumerated as 0 - 999

- Format - enumerated as the defined HST telemetry format codes

- Discretes - enumerated as the list of all possible discrete text strings (i.e., 'on', 'off', 'set clear', 'set on', 'state level 1', 'state level 2', . . .)

4. Benchmarking

The apparent simplicity of the above tables belies the subtlety and the iterations necessary to converge to these tables. Key to this refinement was the use of benchmarking against a reasonable set data. The importance of benchmarking against a sufficiently sized dataset cannot be understated. With the traversal power of data warehouse engines, even the most poorly defined warehouse structure will be quickly searched when the datasets are of typical RDBMS testing sizes. For the nuances of a warehouse design to come out, the dataset must reach a reasonable fraction of the total expected size of the holding. In our case, we pushed our test warehouse up to 100 gigabytes in testing for a planned size of 1 to 2 terabytes.

In doing the benchmarks, we constructed a family of queries, each meant to push some aspect typical of the queries we expected the warehouse to handle. The queries were specifically meant to validate the efficiency of the indexing as it related to the size of the warehouse. In terms of the telemetry data, we ran the query suite against the warehouse with 12 through 36 weeks of data.

It was only through this process that we were able to understand the implications of our design choices, and then refine the warehouse scheme to that shown above.

5. Lessons Learned

This prototype effort demonstrated both the capabilities and limitations of warehousing technology. On the plus side, warehousing technology shows a lot of promise. Once the design was settled, we were impressed with the performance and shear data handling capability of the warehouse product. It is clear that this technology can have significant benefit for those working with reams of discrete information.

As a weakness, this technology is still quite young. The products are only beginning to stabilize and one must be prepared for a number of false starts. For the scientific/engineering user, it is important to realize that warehouse technology is being driven by the commercial sector. There is little experience on the part of the vendors in scientific data issues, and in many cases the warehouse product might have a rich set of functions and primitives for commercial or financial use - but be missing rudimentary scientific ones. In particular, be aware of the use of time, high precision real numbers, scientific notation and functions. Most importantly, it must be remembered that a data warehouse is not designed as one would design a relational database. The warehouse designer has new indexing tools that tend to drive the design more, and the tight linkage between the logical and physical designs must also be reckoned with.

Finally, the critical lesson is the absolute need for benchmarking a sizable dataset. It was only through the actual trial-and-error of manipulating the design, and pushing the technology against a fairly sized dataset, that the real power and limitations of the warehouse are exposed.