Switching the Fermilab Accelerator Control System to a Relational Database

S. Shtirbu (shtirbu@fnal.fnal.gov)
Fermi National Accelerator Laboratory[1]
P. O. Box 500
Batavia, IL 60510

Abstract

The accelerator control system ("ACNET") at Fermilab is using a made-in-house, Assembly language, database. The database holds device information, which is mostly used for finding out how to read/set devices and how to interpret alarms. This is a very efficient implementation, but it lacks the needed flexibility and forces applications to store data in private/shared files. This database is being replaced by an of-the-shelf relational database (Sybase[2]). The major constraints on switching are the necessity to maintain/improve response time and to minimize changes to existing applications. Innovative methods are used to help achieve the required performance, and a layer seven gateway simulates the old database for existing programs. The new database is running on a DEC ALPHA/VMS platform, and provides better performance. The switch is also exposing problems with the data currently stored in the database, and is helping in cleaning up erroneous data. The flexibility of the new relational database is going to facilitate many new applications in the future (e.g. a 3D presentation of device location). The new database is expected to fully replace the old database during this summer's shutdown.

I. INTRODUCTION

The accelerator control system is using a made-in-house, hierarchical, based on indexed files, client-server, database manager (called DBM). The current implementation is in use since 1985 [1&2]. The implementation uses code-based protocol (e.g. in order to read setting information one needs to know the code associated with the setting property). In the past eight years there were very few changes made in the database, the reason being the complexity of providing new functionality since both the client and the server sides have to be changed to support new features, which is an expensive ordeal, especially due to the usage of Assembly language. The lack of flexibility in the central database forces many applications to create their own representation of data stored in the database (by reading the whole database and storing the information in new files, or by analyzing the changes made in the database). A great deal of other device-related information ends up in private/shared files, and even in source code. The inconsistency in interfaces and sources of information creates an unmanageable situation (a device might be deleted from the central database, but information related to the device and dependencies on the device are still present in private/shared files, and hidden inside applications). In addition, there is no trivial mechanism to query the database in ad-hoc fashion, and as a result, there are many inconstancies even in the central database. Even with all these problems, the database provides good service to the control system, since problems are rectified when encountered.

The advantages of a relational database, supporting the ANSI SQL protocol, over the above, are overwhelming: on-line changes in table structure, ad-hoc queries, ability to enforce referential integrity, and many others. But relational databases have one major deficit - they are much slower than a customized, application dependent, Assembly language, implementation. Fortunately, CPU performance is improving fast, and can help relational databases achieve sufficient performance.

Switching from one database to another is not a simple task. The existing data has to be loaded to the new database. Existing query protocols have to be supported. Existing tools which directly accessed the database files, DBM audit files, or DABBEL files (DABBEL was the language used to insert/update DBM) must be converted to use a new interface (or rewritten). A new device entry/manipulation tool, that can support the new fields in, and different organization of, the new database has to be written.

II. RELATIONAL DATABASE PERFORMANCE[3]

We tested several products. The performance tests were done on parameter page queries (which probably are the most common queries in our environment, though not the most complex ones). Conversion programs were written to take DBM files and generate ASCII, ready for load (i.e. first normal form), version of them. This way we could load our real data to the database, and get meaningful benchmarks. There are many benchmarks available in the literature, but the results they report vary widely, and none of them seems to resemble the type of activities in real-time environments. We also realized a major problem regarding I/O. DBM did cache information from the files, and so do most databases in the market. But the caching algorithm used by most relational databases is LRU (least recently used), and is not aware of the logical meaning of the data. Therefore, one might wipe out all the cached information by querying a large, relatively low usage, table (e.g. we have a table holding all the last settings sent to devices). There is only one way to overcome this problem, and that is to provide enough memory to allow the whole database to reside in cache. It is not very hard to do, since our database is small in database terms (less than 30 MB). We realize that the size is going to increase drastically over time (we have over 300MB of shared files), but for the time being, it is feasible to put enough memory in the server machine to provide full caching (in the future, we might move large, non critical, tables to a separate server node).

In our testing, we discovered a major problem with most relational databases - the client library is very slow. The

1 Operated by the Universities Research Association, Inc. under contract with the U.S. Department of Energy.

2 Sybase is a registered trademark of Sybase Inc.

reason the client's performance is very important to us is that the users see only response time - they do not care where the time was spent, and our need for a gateway to the DBM protocol provided us with a unique opportunity to time the performance of both the server and the client.

Our testing proved that using a relational database is feasible for our control system. The product we chose was Sybase, which had the best performance overall (simple and complex queries).

III. DBM-SQL GATEWAY

As mentioned above, we have to support existing applications with the least impact. The most natural way to use relational databases is to use their client libraries (or pre-compilers which produce the calls to the client libraries). This solution can not work for us since we have database clients residing in front-ends, running real-time kernels -- and no vendor has client libraries available for pSOS, MTOS or even VxWorks... One way to resolve this problem with minimal impact is to build a layer seven gateway. This gateway receives (via ACNET, our home grown protocol) requests using the DBM protocol, translates the request to optimized SQL, calls the applicable client library routines to retrieve the data, translates the replies to DBM protocol, and sends the reply back to the requester. As far as existing applications programs and other DBM clients are concerned - nothing has changed, even devices added to the database after the switch to Sybase will be fully accessible by existing applications.

Our gateway also supports SQL requests from applications. The major benefit of doing so is providing a very simple interface for applications to access the database: all an application needs to provide is the applicable SQL statement, and the address of an array of structures where the data retrieved from the database should be put. The application (or other clients) need to call only one procedure. In addition, using the gateway to interact with the database simplifies monitoring and management of database access, and allows us to implement diverse security/priority schemes (e.g. based on client's node id). The applications can also benefit from multiple asynchronous requests and packeting of large requests/replies provided by the client's access routine. Hiding the vendor's client library routines from the user also removes the dependency on a specific vendor's interface, and drastically simplifies switching to another vendor's ANSI SQL server in the future (not that we see a need for that).

The gateway is multi-threaded, to optimize SQL server utilization. Currently, the gateway runs on the SQL server's machine, but that is not required. The gateway is going to support TCP/IP and UDP access to the database in the future (in addition to ACNET).

IV. UPDATING DEVICES

DBM uses a special device update utility (called DABBEL). This utility parses text files containing keyed values. The utility does not support interactive changes. Updates are done directly on the physical files.

The new database is adding many fields and tables (DBM has tables that store unstructured information that have to be spread between multiple structured tables). It is clear that the effort needed to change DABBEL to support the changes in the underlining database is not worth the effort.

Instead, an interactive device entry application is on the works (Lee Chapman is responsible for this effort). The interactive utility is a regular application. This application uses multiple windows and some graphical displays to assist in inserting or updating a device. Thanks to its interactive nature, this application eliminates the mystery from updating devices, and allows one to make changes to the information stored for a device and run an application on another window to see the impact the changes had. The user can easily compare the characteristics of two devices, and copy any property from one device to the other. In the future, this utility will be enhanced to provide logical assistance in entering devices (e.g. it will become aware of correct device entry requirements for different types of devices).

V. AUDIT

DBM has a basic audit facility. The audit trail is based on the DABBEL tokens used for updating a device, and can keep track of whom updated what property, and when. The information is stored in a separate file, and a special interface is used to access it. The new database is expected to provide a better audit trail.

The new audit trail keeps track of changes at the field level, and stores the **previous value** of each modified field. Since the audit information is available from the device update application, recovery from erroneous changes becomes trivial.

VI. BACKUP & RECOVERY

Relational databases tend to be very good when recovery is concerned. They support rollback of transactions, and can tell which transactions were complete when the machine losses power, and roll them forward.

We intend to keep a backup database server, used mainly for development and complex reports. The backup server is going to be at most 10 seconds off the operational database, using a Sybase tool called Replication Server. The backup database will maintain multiple copies of the database, to allow logical recovery of information (existing databases do not support recovery at the table level from a backup file).

VII. SUPPORTED TOOLS

Following are examples of issues that need to be addressed when one is switching databases, in order not to lose existing functionality:

DBM has a simplistic report generating tool, mainly for CAMAC devices. The tool was written years ago, and was never updated to support emerging driver types. This tool has a temporary replacement, which replicates the functionality, supports new device types, and is two orders of magnitude faster. The intent is to make this tool part of an application page, in which the user can even choose the data fields desired in the report.

There is a special application used by operations to decide which properties of which devices are important for machine operations, to be able to tell if a restore went smoothly for all the important properties. To generate the data for this application, once a week, DBM files are scanned to learn about changes. The support for this application is now available in the database, and the information is automatically updated when changes are made.

There is a special utility to load changes in devices from the linac control system (written by William Marsh). The linac control system has its own device database, which is considered the source of information about linac devices. The utility produces DABBEL files, and has been converted to generated the needed SQL to update the relational database.

VIII. DEPLOYMENT

Unfortunately, DBM and the new database can not co-exist. Loading all DBM files into the relational database takes close to a weekend, and is done infrequently. We are almost ready for the switch (only the device entry application is not finished yet). A great deal of testing took place, with actual applications, and the results were satisfactory (the performance tests where done on VAX station 4000/60 which is 7 to 10 times slower then the AXP platform). The actual switch is expected to take place before the end of the accelerator shutdown, this summer.

There is a somewhat surprising side-benefit we are already enjoying, even though we have not switched databases yet. Since the first time data was loaded from DBM files to the Sybase database, the relational database has been used to find erroneous device entries, provide ad-hoc reports, and even find bugs in DBM (which went unnoticed for years). The availability of a good ad-hoc access to the data (using the SQL interface provided by Sybase) also is also substituting the need for code development.

IX. EXPANSION

What is the future of using the new database in the Fermilab control system? We hope to move as much as possible private and shared data used by applications into the database. The major justification for that is the simplification of maintenance of both applications and of the data used by them. Other benefits are: improved performance, improved integrity, better control over content and size, better security mechanisms, simplified system management.

Many new applications, taking advantage of the dynamic nature of the new implementation, are also expected. Questions that could not have been answered before without writing complex programs, are now a SQL query away. New tables are designed and are waiting for data (e.g. geographical location of devices, to be used for simplified alarm analysis and other graphical representations). Many of the improvements in the Fermilab control system in the next few years are expected to be related to, and benefit from, the new database.

X. CONCLUSION

Switching real-time control systems to relational databases is possible and very beneficial. The benefits include: simplification of application maintenance, and improved consistency and integrity of the data stored. The same benefits business applications enjoyed for the past two decades, are now readily available to real-time applications.

XI. REFERENCES

[1] A. Waller, "The Fermilab Accelerator Control System Database", in *Proceedings of the Second International Workshop on Accelerator Control Systems*, Los Alamos, NM, October, 1985, pp. 251-258.

[2] S. Sommers, at al, "An Editing and Reporting System for Fermilab's Accelerator Controls System Database", in *Proceedings of the Second International Workshop on Accelerator Control Systems*, Los Alamos, NM, October, 1985, pp. 259-263.

[3] S. Shtirbu, "Using a Relational Database in a Real-Time Environment", in *Proceedings of the fifth Sybase User Meeting & Training Conference* , San Jose, CA, April, 1993.