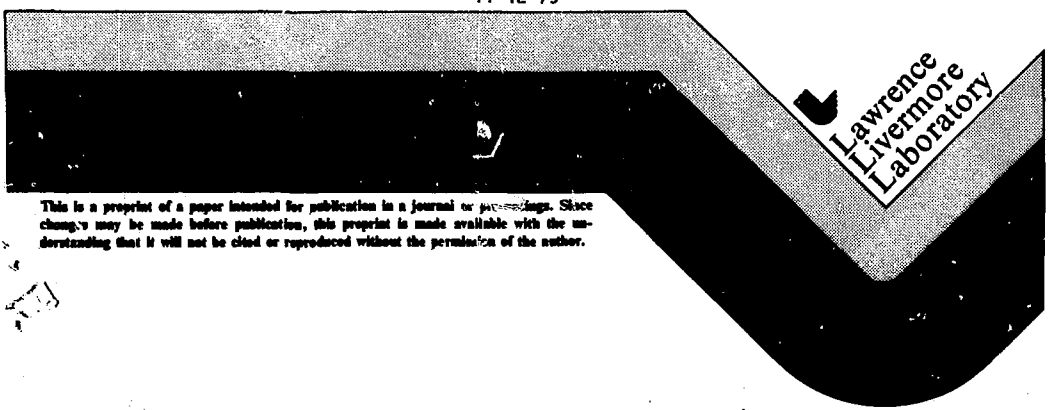
UCRL- 82963
PREPRINT
CONF-791102--116

A DATA BASE MANAGEMENT SYSTEM FOR THE MFTF

JOSEPH H. CHOY
JOHN A. WADE

This paper was prepared for submittal to the
8th SYMPOSIUM ON ENGINEERING PROBLEMS OF
FUSION RESEARCH; IEEE; SHERATON HOTEL,
SAN FRANCISCO, CA., NOVEMBER 13-16, 1979

11-12-79



A DATA BASE MANAGEMENT SYSTEM FOR THE MFTF*

Joseph H. Choy and John A. Wade
Lawrence Livermore Laboratory, University of California
Livermore, California 94550

Summary

The data base management system (DBMS) for the Mirror Fusion Test Facility (MFTF) is described as relational in nature and distributed across the nine computers of the supervisory control and diagnostics system. This paper deals with a reentrant runtime package of routines that are used to access data items, the data structures to support the runtime package, and some of the utilities in support of the DBMS.

Introduction

The supervisory control and diagnostics system (SCDS) for MFTF is a distributed system of nine computers that are connected by a bank of shared memory in a star configuration (see Fig. 1). The workload is divided among the computers by function, which reduces the amount of computer-computer interactions. Each computer's logical functions are backed up by one of the other computers in the SCDS. All communication between tasks on different computers is through the shared memory. (For a further description of the distributed SCDS, see McGoldrick in these proceedings.[4])

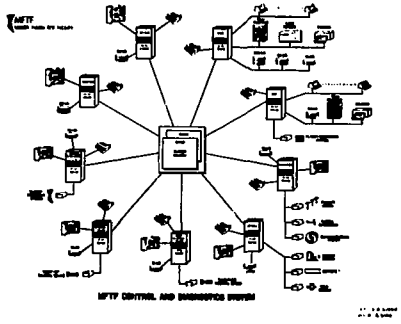


Fig. 1. MFTF Control and Diagnostics System

An MFTF shot will produce several megabytes of diagnostic data that must be recorded and analyzed at a shot rate of one every five minutes. In addition, about thirty kilobytes of mostly scalar element and a few small vectors control the MFTF. On the other hand, diagnostic data are comprised chiefly of several large vectors. The DBMS must be able to handle both scalars and large vectors efficiently and quickly.

*Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore Laboratory under contract number W-7405-ENG-48.

The DBMS is relational in nature: i.e., it is fundamentally a set of tables. The user retrieves data from a table by specifying the row-and-column location of the desired data. The DBMS also permits the user to search tables for specified conditions and to operate upon selected rows. (For a further description of relational data base management systems, see Chamberlain[1] and Kim.[3])

Program Level Interface

The DBMS permits a user to access the data base through his program via the Program Level Interface (PLI) or through an interactive terminal via the Query Level Interface (QLI), which is built using the PLI facilities. The PLI consists of two parts: a precompiler and a reentrant runtime library (see Fig. 2). (For details of the PLI user interface, see Wade and Choy in these proceedings.[5])

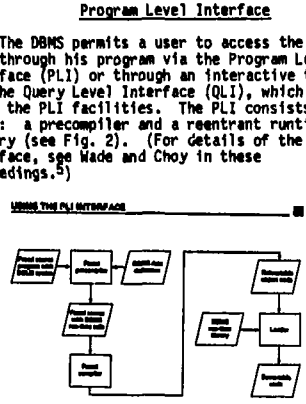


Fig. 2. Using the PLI Interface

The Precompiler

The precompiler accepts as input a Pascal[2] source program that contains data base declarations and references. A dot notation used for data-base references is similar to Pascal's record data structure reference.[5] The main objective of the precompiler is to eliminate as much processing as possible, and thus, minimize the runtime to access data. The precompiler parses the source program, analyzes the data-base declarations and access statements, and transforms the access statements to the appropriate runtime function and procedure references. During the data-base transformations, the location of data items being referenced is resolved as offsets from given points in a table. Since the physical location of a data item is partially resolved prior to runtime, the DBMS is not totally independent of the storage structure characteristics of a given table. A change in the structure of a table usually requires the programs referencing that table to be run through the precompiler, compiler, and load sequence depicted in Fig. 2. The resultant output of the precompiler is a Pascal source program with DBMS runtime function and procedure calls that is then passed on to the Pascal compiler.

The compiler generates a relocatable object code that is then loaded and externals are resolved. DBMS externals are resolved against the DBMS reentrant runtime library.

A Distributed DBMS

Each table in the DBMS is kept on one of the nine computers in the network. In essence, only one original copy of a table--not several copies--are referenced on different computers. A backup copy may be kept if loss of the original would be consequential. A user's program can make reference to a DBMS table any of the computers in the network. Fig. 3 shows how a user's task can access a table that is located on the same or a different computer.

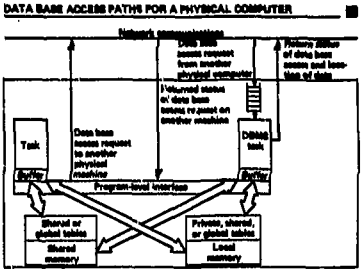


Fig. 3. Data Base Access Paths for a Physical Computer

The files containing a table are stored on the disks of and "belong" to a specific computer. A table opened by a user is either brought entirely into memory or paged in portions on a demand basis. Whether a table is memory-contained or paged depends on the data-base usage pattern (frequency of references, speed of response required for the table, and complexity of data base searches) and size of the table.

When a table is brought into memory, it can be placed into the memory of the local computer or into the memory shared by all nine computers. It is desirable to place into shared memory those tables that are accessed frequently by more than one computer. It is also preferable to place only memory-contained tables into shared memory because of the difficulty and overhead requirements of paging data into shared memory. Paging a table into shared memory is an available option; it is controlled by the computer that "owns" the files of the table being accessed. Semaphores and their associated P and V operators are used to control access to the paging data structures in shared memory.

As shown in Fig. 3, a task--through the use of the PLI--can access tables in the local memory of the computer it is running on, or tables in the shared memory. If the requested table is owned by another computer and is not contained in shared memory, a data-base access request is communicated through the network to the DBMS task on the computer owning the files of the desired table. The DBMS task processes the request, places the desired data in shared memory, and notifies the requesting task of the data's location.

The choice of storage structures for the tables depends on the internal characteristics of the data (such as the size of the table and the distribution of values) and on data-base usage patterns (frequency of references and complexity of searches).

PLI Software

The reentrant runtime library is written in Pascal and contains routines that permit a user to create, delete, open, close, read, and write a DBMS table. In addition, routines are available to lock, unlock, and checkpoint a data base. A hierarchical view of the I/O package is given in Fig. 4.

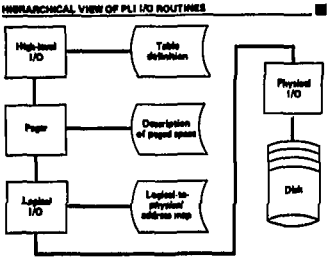


Fig. 4. Hierarchical View of PLI I/O Routines

The high-level I/O routines check for the storage structure and logical address of the table and validate other parameters that describe the table. The high-level routines resolve to a logical byte address offset the location of the data being referenced. All I/O is made through the DBMS pager, using logical addresses. If the desired logical address needs to be read or written, the pager calls the appropriate logical I/O routine, which transforms the logical address to a physical file and address. Finally, the physical I/O routines are called to perform the actual I/O to the paged space from the disks, or vice versa. Once the data are in the paged space, the high-level I/O routines move the data to or from the user's data space. References to a table that require network communications are intercepted by the high-level I/O routines, which route the request to the appropriate computer and then wait for the data to be placed in the shared-memory page space. At this point, the requested data are moved to/from the user's data space.

If the task includes reference to data from a previous shot, the table name is qualified with a shot identifier. The current shot is always the default. If the request for historic data is not on the disk, the high-level I/O routines request the computer operator to mount the appropriate magnetic tape(s) and reads the data onto disk.

PLI Data Structures

The data structures to support the PLI exist in shared memory or local memory of a computer, depending on the function and use of the structure.

A list of the "tables currently open" by some task in the SCDS is kept in shared memory. The purpose of this list is to ensure that a table is opened only once. The list also decreases the time to process an open if the table is currently open for another task, since the runtime entry (described below) need not be constructed. For each table name in the list, there is a pointer to the runtime entry that describes the table's attributes and a count of the number of tasks that currently have the table open.

When a task requests the DBMS to delete a table, the table's definition is flagged as deleted, and the name of the table is placed on a list of "tables to be deleted," which is pointed to from shared memory. This gives the task a faster response for delete operations that can be time-consuming in returning data file space and removing the table definition from the index. The creation of temporary tables that are frequently used is also faster because the create function checks to see if the table being created exists but is only marked as deleted.

There are separate paging mechanisms for the local memory of each of the computers. Each local memory contains the data structures to describe the page space and its associated "last recently used" queue. Shared memory has one set of the data structures that describe the shared-memory page space and one "last recently used" queue. Access to the shared-memory data structures for paging is controlled by semaphores in shared memory. The semaphores are implemented with the aid of an indivisible hardware test-and-set instruction.

Whenever a table is open, there exists in shared memory a runtime entry that is pointed to from the list of "tables currently open" (described above). This runtime entry is a subset of the complete definition of a table. This subset describes only those few parameters needed during the runtime. In addition, the runtime entry points to a list of task names that have this table open and collects usage statistics for the table.

As mentioned above, a table exists in only one place and belongs to a specific computer. Each computer has in its local memory an index to the tables for which it has the files (see Fig. 5). A simple hash table is used to locate a table definition, with collisions handled with a linked list. When a task opens a table, the list of "tables currently open" is checked first. If the table is found to be already open, the pointer to the runtime entry is followed and the task's name added to the list of tasks associated with the open table.

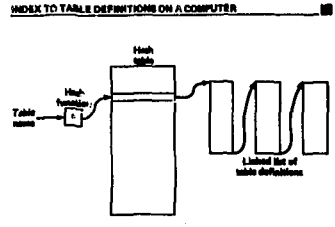


Fig. 5. Index to Table Definitions on a Computer

If the table is not open, the local index to table definitions is checked. If the table is local, a runtime entry is built from the table definition, and the table name is added to the list of tables open. If the table is not local, the Data Base Manager computer is interrogated for the location of the table, and the appropriate computer is requested to open the table.

The table definitions mentioned above are essentially static and can only be modified by the Data Base Administrator. A table definition contains information about the table's storage structure, creation date and time, the table's logical address, whether it is memory-contained or paged, in shared memory or local memory, the number of columns to the table, the data type and size of each column, and a list of tasks permitted access to the table, along with the type of access allowed.

For those tables whose information is critical and must not be lost, a duplicate copy is kept on a different physical computer. When an update is made to the original, the same update is queued to the DBMS task of the computer where the copy exists. If a computer with a table that has a duplicate copy becomes non functional and the SCDS can continue to operate in a degraded mode, the backup copy is made the primary copy and another backup copy is made elsewhere on another computer.

Utilities

Typical utilities create tables (table definition initialized and space allocated) and delete tables on the list of "tables to be deleted." Other routines gather and print usage statistics and check the pointers and integrity of the allocated space. There are also utilities to copy tables to magnetic tape for offline storage and to retrieve the tables from tape when users need to reference historic data. Because of the volume of diagnostic data to be collected with the time constraints of one shot, specialized routines designed with speed in mind format the raw diagnostic data into the DBMS table structure.

The ability of the SCDS to function without a computer in a possibly degraded mode requires that the DBMS have utilities to reconfigure the location of tables from one computer to another, make copies of a table on another computer, and modify the table definition to reflect the changes made.

Conclusions

The MFTF DBMS is designed to be fast, easy to use, and to handle a distributed data base. It handles simple scalar set-point values as well as several megabytes of vector-oriented diagnostic data quickly. The DBMS utilizes a reentrant runtime library to eliminate several copies of the same DBMS routine in memory.

References

1. Chamberlain, D. D. "Relational Data Base Management Systems," Computing Surveys 8, 1 (March, 1976), 43-66.

2. Jensen, K. and Wirth N. PASCAL: User Manual and Report, Springer-Verlag, New York, 1975.

3. Kim, W. "Relational Database Systems," Computer Surveys 11, 3 (September, 1979), 185-211.

4. McGoldrick, P. R. "SCDS Distributed System," Proceedings of the Eighth Symposium on Engineering Problems of Fusion Research (IEEE), 1979.

5. Wade, J. A. and Choy, J. H. "Control and Diagnostic Data Structures for the MFTF," Proceedings Engineering Problems of Fusion Research (IEEE), 1979.