To: Distribution

From: J. A. Weeldreyer

Date: April 5, 1978

Subject: Enhancements to the Multics Data Base Manager

Introduction

With the advent of MR6.0, the Multics Data Base Manager (MDBM) reached a significant milestone. Namely, most of the basic functionality required to make the MDBM a viable database management package has been implemented. (The major exception is the interface to the vfile_ journalization and concurrent usage control features planned for MR7.0.)

However, there are several things which can be done to improve this product. This MTB provides an overview of our perception of the future direction of MDBM development efforts. This perception is based to a large extent on customer feedback and marketing requirements. Our plans for database restructuring, for database security, and for concurrent usage control are described in separate MTB's. Unless otherwise noted, items discussed in this MTB are planned for MR7.0. Please mail comments and suggestions to Weeldreyer.Multics on System M, or call (602) 249-7244 or HVN 341-7244.

The reader is assumed to be familiar with relational and CODASYL data base terminology. The MRDS Reference Manual (Order No. AW53) and the MIDS Reference Manual (Draft) can provide enlightenment where the assumed familiarity is lacking.

New Data Base Architecture

Since the MDBM was initially released in June 1976, several factors have caused a re-evaluation of the current architecture. Among these factors are:

a. enhancements to vfile_, including the select/exclude capability and the forthcoming concurrent usage control features,

---

b. the marketing requirement for a CODASYL database capability with increased performance to better provide a growth path for GCOS IDS users, and

c. recent articles in the literature indicating that significant performance improvements can result for relational databases when implemented using a more traditional, highly structured architecture.

Currently, an MDBM database (MRDS or MIDS) is implemented in a very straight-forward manner using vfile_. In a Multics Relational Data Store (MRDS) database, each relation is represented by a keyed sequential file, and every tuple within the relation is a record within the file. The primary key of the relation is the key of the file. It is possible to specify secondary indexes to a relation, and these are implemented using the more sophisticated vfile_ control orders for index manipulation.

A Multics Integrated Data Store (MIDS) database is a special case of an MRDS database. A record type is implemented as an MRDS relation, and a set type is also implemented as an MRDS relation associating the primary keys of the owner and member record occurrences to form set occurrences.

There are several disadvantages of this current architecture, some of which are listed below.

a. The new vfile_ concurrent usage control mechanism requires that an additional six words (for a total of eight words) of control information be stored with each record in the file. In a typical database, tuples average only 50 to 100 characters in length. The requirement for 32 characters of control information for every 50 to 100 characters of data, strictly for concurrent usage control, is prohibitively expensive for large databases.

b. Recent articles in the literature suggest that the tuple is a finer than optimum level of granularity for concurrent usage control.

c. If the primary key of a tuple is known, an average of from two to three page faults is required to access the tuple, assuming a node height of three with the root node and some second level nodes resident in main memory. It is estimated that this type of access could be performed in an average of about 1.3 page faults using a hashing scheme, based upon information from the developers of a database manager using such an access method.

d. There is no way for a Data Base Administrator to structure a database to "cluster" tuples of different relations, which are frequently referenced together, so as to minimize page

faults. Also, it is not possible to explicitly "link" such tuples, other than through the use of secondary indices. Assuming typically sized relations (node height of three), the secondary index is from two to three times as expensive in terms of I/O operations as a direct link would be. According to our observations, clustering and linking would be applicable in over one-half of all customer queries.

e. The current architecture does not provide a good foundation for large CODASYL databases. Keyed sequential access methods do not provide efficient "calcing" or set implementations.

The preceding discussion leads to the following proposal for a new database architecture.

a. Remove the one-to-one correspondence between a vfile_ keyed sequential file and a relation. Instead, introduce the concept of a database "file" which corresponds to the CODASYL concept of an area. Two different types of files will be allowed, "blocked" and "unblocked". (Note: The use of the terms blocked and unblocked is unfortunate because of the possibility of confusion with vfile_ blocked files. However, they do precisely describe the primary difference between the file types. Any suggestions for less confusing terminology would be appreciated.) Unblocked files will contain exactly one relation much like the current architecture, whereas blocked files may contain multiple relations. Both types of files are implemented as vfile_ keyed sequential files. Hence, a database will consist of one or more blocked and/or unblocked files, each containing one or more relations.

b. For blocked files, remove the one-to-one correspondence between a vfile_ record and a tuple in a relation. Instead, introduce the concept of a file "block", which contains one or more tuples which may be from different relations. The file block is implemented as the data portion of a vfile_ record. The size of a block must be an integral multiple of the Multics page size (allowing for vfile_ control information) and will begin on a page boundary. The Multics area manager will be used to manage the space within a block. The block corresponds to the IDS-II concept of a "page", and is the level of granularity at which concurrent usage will be controlled for blocked files. Unblocked files, on the other hand, will retain the correspondence between a vfile_ record and a tuple. Concurrent usage for unblocked files will be controlled at the tuple level.

c. Introduce the concept of a "tuple-id", which is a 36 bit identifier unique for each tuple in a database. The tuple-id will consist of a file number concatenated with a record number local to the file. From the record number, it is possible to determine the location of a given tuple. The tuple-id remains constant for the life of a tuple, and

corresponds in concept to the CODASYL "database key".

d. Within blocked files, organize all tuples within each relation into a threaded list, such that each tuple is threaded to the (physically) nearest tuples within the relation. Currently, tuples are ordered physically according to the time they were stored, which does not necessarily correspond to the logical ordering as determined by primary key sort order.

e. Provide a method to explicitly link related tuples in different relations using tuple-ids.

f. For relations which are not link "children" and which reside in blocked files, implement primary keys via a hashing scheme rather than a vfile_ index. For all other relations, primary keys will continue to be implemented as vfile_ indexes.

g. Retain the capability to specify secondary indexes into relations, but utilize the select/exclude vfile_ control orders to reference these indices more efficiently. In addition, place all indexes for a file within the same vfile_ index, as opposed to using separate vfile_ files which is done currently.

Implications of the New Architecture

There are many improvements which will result from the implementation of the new architecture, however there are also several disadvantages. Most of the advantages are in the areas of enhanced retrieval performance and CODASYL compatibility. Increased complexity and the reduced flexibility inherent in blocked files are the primary drawbacks.

The primary advantage is anticipated to be significantly improved performance resulting from the minimization of page faults. This improvement will be most noticeable for large databases. The reduction of page faults results from the use of several features of the new architecture. (Note: the following discussion assumes medium to large sized relations such that under the current architecture, the vfile_ index node height would be three.) For example, the hashing scheme available with blocked files should reduce page faults by a factor of 1.5 - 2.5 when accessing tuples by primary key. The use of links will reduce page faults by a factor of 2 - 3 when clustering is not used, and by a factor of 3 - 4 when clustering is used, in comparison to the current use of secondary indices to provide this capability. Finally, sequential searches through relations residing within blocked files will be somewhat more efficient because tuples will be examined in physical sequential order, eliminating the "skipping" from page to page which currently

occurs when a relation has not been stored in ascending primary key order.

A second advantage of the new architecture is that it provides a far better foundation upon which to build a CODASYL-compliant interface to the MDBM than does the current architecture. The actual implementation of such an interface is not planned tor MR7.0, but will be accomplished in a subsequent release. However, an efficient implementation requires such capabilities as hashing, links, and clustering, none of which are provided by the current architecture.

Another benefit is that the blocked file in the new architecture more efficiently utilizes the vfile_ extensions for concurrent usage control. Although there is actually a higher overhead using blocked files for very large tuples (blocking factor of less than 7), for average-sized tuples (resulting in a blocking factor of more than 30 per one-page block) there is a savings of over 50 percent in storage overhead required for concurrent usage control. Also, there will be additional savings resulting from the fact that concurrency is controlled at a more efficient level of granularity, i.e. the block level rather than the tuple level.

There are also some disadvantages to the new architecture. The internal structure of a database, particularly the blocked file structure, will be considerably more complex than is currently the case. This additional complexity must be managed entirely within the MDBM, possibly resulting in the requirement for additional code. However, for large databases, it should be advantageous to spend some additional processor time in order to significantly reduce paging. It is also possible that database update operations will be slightly slower than they are currently, but this should not be significant.

Secondly, the additional database complexity will cause database restructuring to be somewhat more difficult than it is now. Currently, it is possible for users to utilize other Multics commands to restructure a database without completely recreating it. The additional complexity will necessitate the development of a specialized restructuring utility. However, this is not a severe disadvantage, since customers have already requested such a utility for current databases.

Also, the Data Base Administrator will be faced with many more options when defining a database than is currently the case. Thus, there will be more opportunity for error. It also may be more difficult for individuals to define and maintain private databases, although this problem is alleviated to a large extent by providing intelligent defaults and by the unblocked file capability.

Finally, the hashing algorithm will require that blocked files be pre-allocated and pre-formatted when the database is created. This will require a database designer to estimate the maximum size of those relations residing in blocked files at database definition time. Such a requirement is particularly distasteful in the Multics environment where file sizes have traditionally been dynamic. Also, retrieval performance for frequently updated blocked files will degrade as the amount of contained data approaches the pre-allocated size of the file, necessitating file reorganization. However, the capability to define unblocked files will allow the database designer to avoid these problems for applications not well served by the highly structured blocked files.

Hence, although there are numerous disadvantages, it is felt that the potential performance improvement together with the better CODASYL interface foundation justifies the implementation of the new database architecture.

User Interface Changes for the New Architecture

Databases of the new architecture will be assigned a new version number. The MDBM will continue to function with old-version databases with no changes in the user interface. There are some differences in the user interface for new-version databases. All changes, with the exception of one, are upward compatible. The incompatible change is that the maximum (indexed) primary key and secondary index lengths will be reduced from 256 characters to 252 characters. This reduction is necessitated by the placement of all indexes for a file in the same vfile_ index, requiring that identification information be carried within the keys to differentiate among logically separate indexes. Since it is highly unlikely that any customers currently have any 256-character indexes, this incompatibility should have no impact. Other changes include additional MRDS data sublanguage subroutines, applicable only to new-version databases, which are discussed in MTB-361, MDBM Recovery and Concurrency Control. There are also some additional capabilities which can be specified in the create_mrds_db source segment, and these are discussed below.

There will be an optional -max_tuples argument which may be specified with the definition of a relation residing in a blocked file. This allows the Data Base Administrator to specify the maximum number of tuples to be allowed in the relation. If not specified, the default value will be 1000.

An optional file statement will be provided to allow the Data Base Administrator to explicitly specify the associations among the various relations and files within the database. Also, certain file characteristics may optionally be specified. If no

file statement is present, each relation is assumed to reside in an identically named unblocked file. The file statement is as follows:

```
file: <file_spec>[, <file_spec>, ...];

<file_spec> ::= <file_name> (<rel_name> [<rel_name> ...])
                [-blocked [<n>] [<h>/<b>]] [-unblocked]
```

where <file_name> is the name of the file being defined, <rel_name> is the name of a relation residing in the file. If -blocked is specified, <n> is the number of Multics pages per block, and <h> is the number of hash bucket headers per block <b>. It is possible to specify multiple headers per block (<h> > 1, <b> = 1) or one header for several blocks (<h> = 1, <b> > 1). The default is <h> = <b> = 1. The default for block_size is <n> = 1. If -unblocked is specified, then the file is defined to be an unblocked file. In this case, only one <rel_name> may be designated. The -blocked and -unblocked arguments are mutually exclusive. If neither is specified, the default file type is determined as follows. If multiple <rel_name>s are present, the file will be blocked with the default values for <n>, <h>, and <b>. If only one <rel_name> is present, the file will be unblocked.

Finally, an optional file statement within the create_mrds_dsm source segment will be provided to allow the user to specify database files. The syntax is:

```
file: <dsm_file_name> [= <dm_file_name>][, <dsm_file_name>
      [= <dm_file_name>], ...];
```

where <dsm_file_name> is the data submodel name of the file and <dm_file_name> is the name of the file within the data model. If a <dsm_file_name> is specified without a corresponding <dm_file_name>, it is assumed that the data model file name is the same as the data submodel file name. If the file statement is omitted, an identically named file is assumed for every relation specified in the data submodel.

Other Enhancements

In addition to the new database architecture, there are several other enhancements planned for the MDBM. Several of these, namely database restructuring, attribute-level security, and concurrent usage control are discussed in separate MTB's. There are other, more minor, changes which are briefly discussed below.

The capability for an application program to use pre-translated selection expressions will be provided. This

feature will provide a slight increase in performance by allowing the user to translate an MRDS selection expression and save the resulting tables in a segment. The pre-translated selection expression may then be invoked in a dsl_ entry call by specifying a selection expression consisting of:

-path <seg_path>

where <seg_path> is the pathname of the segment containing the pre-translated selection expression tables. If the database has been restructured since the selection expression was translated, it will be automatically re-translated. A command to initially translate a source selection expression will be provided.

The Data Base Administrator will be given the capability to specify various types of integrity checking within the data model. It will be possible to specify foreign key relationships among relations, and to specify various types of value integrity checking for domains within the database.

The foreign key relationships will be implemented as links, and are specified via a foreign_key statement in the create_mrds_db source segment. The syntax of the foreign_key statement is:

```
foreign_key: <fk_spec>[, <fk_spec>, ...];

<fk_spec> ::= <prel_name> (<pattr_name> [<pattr_name> ...])
              <crel_name> (<cattr_name> [<cattr_name> ...])
              [-cluster]
```

where <prel_name> is the name of the parent relation, <crel_name> is the name of the child relation, <pattr_name> is the name of an attribute within a candidate key of the parent relation, and <cattr_name> is the name of an attribute within the child relation which is to be matched with the corresponding <pattr_name>. The <cattr_name>s must correspond in order and quantity with the <pattr_name>s, and corresponding <cattr_name>s and <pattr_name>s must range over the same domain. The <pattr_name>s must comprise a candidate key of (must uniquely determine tuples within) the parent relation. The -cluster argument specifies that the child tuples are to be clustered as closely as possible to their parent tuples. This is applicable only if both parent and child are contained in the same blocked file and is in error otherwise. Also, two relations can be clustered on the basis of only one foreign key definition.

The integrity constraint provided by the foreign key concept is that a parent tuple may not be deleted if it has dependent children. Conversely, a child tuple cannot be added if there is not a corresponding parent already in existence. Hence, this facility provides a method to enforce inter-relation dependencies.

Domain integrity is specified via added arguments within the domain statement of the create_mrds_db source segment. The

-check <boolean_expression>

argument specifies a boolean expression to be satisfied whenever a new value is assigned to an attribute which ranges over the corresponding domain. The

-check_proc <path> <entry>
-encode_proc <path> <entry>
-decode_proc <path> <entry>

arguments specify procedures which are to be invoked whenever a new value is assigned to an attribute which ranges over the corresponding domain. The -check_proc specifies a value integrity checking procedure, the -encode_proc specifies a procedure to encode data values stored into the domain, and the -decode_proc specifies a procedure to decode data values retrieved from the domain.

Finally, there will be a change in the manner in which variable length string attributes are stored within a database. Currently, varying strings are stored exactly as defined in the MPM Reference Guide, i.e. the maximum length is reserved and the string is preceded by a word containing the current length. In the future, varying strings will occupy only that space actually required to contain the current value, plus a word to indicate the current length. This change will drastically reduce the storage requirements for variable length text attributes.

Future Trends

As has been previously mentioned, there is a marketing requirement for a better performing, more complete CODASYL-compliant interface to the MDBM. This requirement played a large part in determining the necessity for the new database architecture. Currently, this capability is planned for MR8.0. However, this will not be a mere "fleshing-out" of MIDS. Instead, a MIDS-II will be developed to interface to the database in parallel with MRDS. The old MIDS will be unchanged, and probably will die from lack of use. The MIDS-II interface will be designed to be efficiently callable from a host language (e.g. COBOL, FORTRAN) runtime package, as opposed to the MIDS interface which was designed to be directly callable by user application programs.

The parallel existence of MRDS and MIDS-II will provide a highly desirable capability with some very interesting byproducts. Namely, a given database (whether originally defined via MRDS or MIDS-II) will be accessable via both interfaces.

Hence, LINUS will serve as an end user facility for both MRDS and MIDS-II databases, as will ROBOT if it is implemented. (ROBOT is a natural language database query facility developed by the Artificial Intelligence Corporation. The possibility of interfacing ROBOT to MRDS is currently being investigated and looks very promising.) Also, any end user facility designed for MIDS-II databases could reference MRDS databases. Some possibilities for such a facility would be MDQS (to provide GCOS compatibility) or the CODASYL end user facility (currently being specified by the CODASYL EUF Committee). These features would make the MDBM one of the most powerful and flexible database management packages available.