VLB ARRAY MEMO No. 485

Selection Criteria for the VLBA Database

JONATHAN D. ROMNEY
National Radio Astronomy Observatory
Charlottesville, Virginia

1985 September 9

INTRODUCTION

Selection of a database management system for the VLBA will require a careful survey and evaluation process if we are to realize all the potential benefits of maintaining the Array's operational data in such a system. Broadly considered, the selection procedure must determine an optimal match to our requirements as currently understood without overly constraining the flexibility of reorganization and potential for expansion which will surely be required in the future.

In an earlier memorandum (VLBA Memo 469) I outlined at a fairly abstract level the structure of a VLBA database and its interfaces to the Array's subsystems. One important feature of our application which will be a major determinant in selecting a database management system is the dual requirement for extremely flexible access to a diverse, but "reasonably" sized, body of general information, as well as infrequent, rapid, but fairly straightforward access to a voluminous array of specialized monitor and status data. The latter requirement is also discussed in detail in Clark's VLBA Memos 278 and 396. While these two requirements tend to drive the optimal choice of database management system in rather different directions, nevertheless there are important intersections between the two data sets involved. Because of this I hope it will be possible to accommodate both requirements within one system.

The objective of the present memorandum is to suggest the criteria we should apply in evaluating candidate management systems for the VLBA database. For this purpose it will be necessary first to continue the tutorial on database systems begun in Memo 469, and also to discuss further the possible approaches to supporting the large volume of monitor data returned to the operations center from the remote antenna sites.

DATA MODELS

The previous memorandum on this topic considered the fundamental elements and basic attributes of database management systems in general. A primary characteristic of all systems is the interpolation of an abstract conceptual *data model* between the user and the physical storage constructs. And one of the chief distinctions among these systems lies in the types of conceptual structures and relationships encompassed in the data model. Virtually all database management systems support one of the three fundamental models outlined in the following paragraphs.

In the **relational model**, the stored information is always visualized as residing in one or more two-dimensional tables. In relational terminology, the table represents a *relation*, and consists of a number of *tuples* (conventionally visualized as "rows" of the table), each of which specifies — uniquely — one or more *attributes* ("columns"), whose values are members of a *domain*. Some analogies may be illuminating here: the tuples and attributes would correspond to records and fields in a standard fixed-format file, and to material items and their characteristics in the physical world. Every attribute value in every tuple is atomic, and is potentially identifiable. Information about the relationships among data elements is represented only through the identity of particular attribute values among several tuples, and is *not* explicitly encoded.

Both exceedingly simple and broadly applicable, this basic structure underlies the frequent claim that the relational model provides the most flexible and "user friendly" approach to the database. Beyond these obviously advantageous features, however, the relational concept supports a mathematically complete higher-level *relational algebra* capable of manipulating entire relations as its elements. Indeed, *all* retrievals against a relationally-structured database can be expressed in terms of set-level operations such as *select*, which forms a subset of a relation containing only those tuples which satisfy a given condition; *project*, in which a subset of attributes is formed and redundant tuples eliminated; and, perhaps most fundamentally, *join*, which merges two relations into a new table containing the information in both for those tuples where equal values (or some other condition) occur in specified attributes.

These characteristics make relational systems most suitable for applications which emphasize frequent and possibly complex retrieval queries, expeditious migration into new applications, and ease of reorganization. The close analogy between the relational structure and the tabular organization of spreadsheet data also makes these systems natural candidates for supporting communication between a central database and personal computers. The advantages of the relational model are evidently becoming widely acknowledged for both scientific and business applications, to the extent that this trend can no longer be dismissed as "CSBS". Indeed, "relational" has become a new software buzzword: some systems advertised as relational or "relational-like" are in fact hybrids, which will be discussed further below.

The flexibility and convenience of the relational approach carries a price in computing resouces, of course, although it is certainly possible to optimize the storage architecture toward the most common or critical retrievals. Relational systems appear to be considered rather prodigal in this regard, and this has led to the development of *database machines* which implement the relational model in specialized hard- and software. These machines provide extremely efficient data storage and access, and reduce the burden on the host to formulating and transmitting queries and receiving the results of retrievals.

The **hierarchical model** adopts a very different visualization: that of a tree structure. In this organization a fundamental entity, the *root*, is logically connected to one or more *dependents*, which in turn are *superior* to sub-dependents, *etc.*, so that the structure is explicitly asymmetric. These relationships are encoded by index pointers or *links* embedded in the database. A sequential-file analogy would require indexed, variable-length, self-identifying records — a considerable complication over the relational case.

While quite rigidly limited, this structure facilitates very efficient retrievals along the allowable paths. (In some cases this may approach the most efficient of all retrieval systems: the hard-coded stand-alone program.) Thus, the hierarchical model is most appropriate for applications involving repetitive, highly structured access to data organized in standard formats. However, introduction of new relationships, unforseen when the database was created, can be accomplished only as a major programming task. And although it is possible to pose queries in a higher-level relational algebra against a hierarchical database, this does not arise naturally from the underlying structure, and can be supported only with additional complexity.

Not surprisingly, heirarchical database systems excel in applications where the model is a good representation of the relationships inherent in the data. When the relationships are well established from the beginning, and flexibility is not a major requirement, the dividend in efficiency would recommend this choice. There seems to be general agreement, though, that application of a hierarchical system where the model is inapplicable, or where new relationships may need to be supported, confronts the user with difficulties attributable more to the model than to the operations supported.

Finally, the **network model** is best viewed as a generalization of the hierarchical structure. The asymmetry of the hierarchy is relieved by allowing multiple links between "superiors" and dependents; again the links encode explicitly the relationships among data elements. This form of database management system is the subject of some thoroughly developed standards issued by the CODASYL DBTG.

The network supports more naturally the many-to-many structure typical of a complex application, but at the cost of another increment of complexity beyond the hierarchical model. Efficient retrieval paths can be provided for a wider variety of queries with advance planning, but the task of reprogramming or restructuring to accommodate new applications is correspondingly more difficult.

Some Structural Details

An important concept applicable to all three models is that of the **data dictionary**, effectively an auxiliary database containing the conceptual and physical schemata which in turn describe the database entities and their mapping to physical storage. Every database has such a dictionary, if only as scribblings on paper, but a "dictionary-driven" system integrates the dictionary into the database itself, and takes advantage of this self-describing feature to facilitate access, as well as validation and restructuring operations.

The popularity of the relational model, in particular the power of the relational algebra, has provoked the development of hybrid systems — often termed **"born-again" relational** implementations — which basically provide a hierarchical data structure with a relational front end. As noted earlier, there is a complexity penalty in supporting relationally-phrased queries against a hierarchy, and since the relational operations do not arise naturally from the underlying structure each must be implemented as a special case. Thus, relational purists tend to find the algebra incomplete. On the other side, this hybrid approach makes available to the user the efficient, record-oriented, lower-level hierarchical data access facility, which in specialized circumstances can be used to advantage.

Another important distinction among relational systems concerns the index structures which are built for efficient retrieval, which in **inverted-list** systems are available to the user. This departure from the "pure" relational concept makes possible some enhancements in efficiency; many queries can be answered directly from the indexes without any reference to the database itself.

The Extended Database

In some contexts it is useful to broaden the scope of the conceptual database to include ordinary "flat files" in the host processor supporting the database management system. While external to the database itself, these files can be indexed or otherwise described from within the database to facilitate relatively efficient access to a small subset of a large data volume. This "extended database" concept violates some of the fundamental tenets presented in Memo 469 — chiefly data independence and integrity — but may represent a worthwhile compromise when efficiency is at a premium.

Two specific features available in some database systems provide substantial support for this approach. The first is a **record management interface** which makes the facilities of the host operating system's record-oriented I/O facility available to the database programmer. The database itself can then serve as an extensive index to a much larger volume of data in the extended database; operating on the index alone, complex retrieval operations can specify only those data actually required for access through the record management interface.

Secondly, some relational systems have a **fast load/unload** feature which implements rapid transfer between the database and plain fixed-format sequential files. This operation eliminates the repetitive processing of commands and most of the indexing overhead involved in individual insertions or retrievals of the same tuples. A sort of "virtual database" approach is then possible in the case of large data structures where accesses are infrequent and confined to well-defined blocks.

Query Languages

My discussion of database management system features in Memo 469 slighted somewhat the important area of the system's interfaces to application programs and interactive users. Besides the relatively straightforward subprogram-call linkage, most vendors offer a non-procedural, "English-like", "fourth-generation language". While the final phrase can be dismissed as commercial puffery, and English-likeness is probably a dubious virtue, non-procedural specification of a retrieval or other operation is an important concept, closely associated with the relational algebra discussed above. In a non-procedural program, the user specifies globally what result is to be achieved, without having to describe the procedure by which this is accomplished. These "4GL" facilities are generally available to interactive users as well. In both cases, this approach is oriented to non-technical users, and/or to applications where manipulations of the database predominate.

Similar advanced tools may also be available to the database administrator for specifying the data dictionary. In this case the source version from which the dictionary is compiled can provide a valuable documentation tool.

Basic Selection Criteria

The range of available database management systems is so broad that I have had to adopt two basic requirements in order to make a market survey a tractable proposition. One arises directly from our decision some time ago to select VAX processors operating under VMS as the hosts for both the array and correlator control systems. Since the VLBA database will have major interfaces to both these systems (and will presumably be resident in one or the other of these computers) a clear requirement follows that the database management system be supported under VMS on VAX processors. This restriction does eliminate many candidates, but a surprisingly large number remain.

Perhaps more controversially, I have also considered only relational systems. I am convinced that the range of applications suggested in Memo 469 requires the flexibility and adaptability of a relational model, and that selecting a hierarchical system as a more efficient alternative would lock us out of numerous attractive applications and vitiate much of the saving we might achieve in programming effort. In fact, my early investigations included a number of "born-again" relational systems which I had not previously realized were actually hybrids, and this may offer a valuable option for our particular case. Also included as relational were database machines (actually only one is VMS-compatible at present).

Evaluation Criteria

More than a dozen well-established database management systems meet the two basic requirements proposed in the previous section. I have not yet completed a survey of this field. The next stage, however, should be an evaluation of suitable candidates from the field to arrive at a short list for further consideration, possibly including tests under a trial license. I am soliciting input here from all areas of the VLBA project on criteria and weights for this evaluation. The following commentary proposes some possible criteria (drawn from several evaluations by other organizations and from my own investigations) which I would like to see discussed. Comments or additional suggestions are welcome.

Trial license. Some vendors offer a trial license, typically for 60 days. (Others only grudgingly allow a "15-day acceptance", which I would regard as next to useless.) This *might* be an attractive option for evaluation of the short list, but only if we are prepared to devote a significant level of programming resources over the next half-year to creating and testing a realistic simulation of VLBA operations.

Support under VMS 4.0 and subsequent versions. This is an obvious necessity, but also an obvious informal claim by all candidates. Should we try to require it as a condition of purchase? We will be a relatively minor customer, and may have to accept the vendors' conditions.

Network communications. The VLBA database will have to support frequent transactions to and from both the array- and correlator-control subsystems; (at least) one of these systems will run in a remote host, so that the database management system will have to support host-to-host networking efficiently, including remote *updates* as

well as retrievals. Further, some maintenance operations will require access via the (non-VMS) nodes at the individual VLBA stations.

Subroutine-call interface to C and Fortran. One such program-level interface is essential, of course, and should be convenient since it will form a primary point of contact with the database for both the array- and correlator-control software. Interfaces to both our chosen languages would be a convenience.

Conflict resolution and concurrency control. A satisfactory database management system must provide sufficient lockouts to prevent concurrent updates to database elements, and must avoid the "deadly embrace" where two or more processes are waiting for each other to complete. Lockouts at several levels will be necessary, ranging from the entire database to individual tuples or records.

Security and integrity features. The VLBA will be, unavoidably, heavily dependent on computer storage of its operational data. The database management system should contribute to minimizing the vulnerability to hacking and crashes of all types by providing user access controls, backup/restore and journaling facilities, and blocking of commands with a deferred commit to modification of the database.

RMS interface & Fast load/unload. These facilities were mentioned above, under "extended database" (RMS refers to the VMS-specific Record Management Services). Either or both will provide valuable support for the high-volume aspects of our application.

Data dictionary. A dictionary-driven structure, described under "structural details" above, is likely to be a more efficient system.

PC links and commonality. A number of database systems support data transfers between PCs and a supermini like the VAX, and/or provide the same user interface on both. This may be an attractive way to allow for growth in interactive use without loading either of the planned VAX processors.

Views. A "view" is a logically-defined relation which can be referenced for efficient retrieval, but is not actually stored as such. This technique will be of particular value in implementing repeated complex retrievals at the program-call level — *e.g.*, as will be necessary to configure the correlator.

Flexible reconfiguration. It should be possible to create relations representing a new conceptual structure of stored data, and then copy information into this from existing relations, without having to update program-level calls or stored queries which access unchanged information. Some implementations, evidently, require that the entire database be unloaded and then reloaded into the new structure.

Efficient implementation of floating-point values. Much of the VLBA's data will be in this form, which should be supported using VAX floating-point types.

"Fourth-generation" language. Such a feature seems to be of marginal value in both our program-interface and interactive applications. The array- and correlator-control programs will be complex real-time systems, which interact with the database in a

fairly peripheral way; our chosen languages are certainly more appropriate for this case. And I presume that interactive users within NRAO will feel as much at home with mathematically-oriented set-level manipulations as with their English translations.

Compilation facility for the query language. The query language is usually oriented toward interactive applications, and *interprets* input queries. Some languages provide as well a compilation facility which stores the interpreted queries for future routine use. This is likely to be an important benefit for our application, where we must support program-level retrievals as well as routine and ad-hoc interactive queries.

Screen forms for entry and retrieval. A "screen" is essentially a displayed representation of a tuple in some relation of the database. Screen facilities provide an extremely flexible and readable vehicle for sequential display of retrieved tuples, and a convenient tool for ad-hoc entries or updates of limited volumes of data.

Graphics facilities. Almost all vendors advertise "graphics", but frequently mean by this little more than pie charts and perhaps histograms. Simple line plots of one variable against another — an easier programming task, one would think — are often not supported. Insistence on graphics suitable to our needs will significantly restrict the range of choices available.

STRATEGIES FOR MONITOR DATA (continued)

The difficulties and requirements posed by the large volume of station monitor data were considered in some detail in Memo 469. I proposed there that we refrain from entering the bulk of these values into the database, but instead sort the data as received, entering only those required for later analysis *or* for current near-real-time diagnosis. The remainder would be buffered in large sequential files from which long-term averages would be extracted periodically; those specific values — if any — required for post-real-time study would have to be re-sorted and entered before use. Some points raised in more recent discussion of this proposal should be considered further here.

First, the record-management interface and fast load/unload facilities described above under "extended database" might almost have been intended to support the proposed scheme. These features reduce the difficulties of post-real-time filling to a single additional programming task which finds and loads the required data.

A second topic is the suggestion by Ewing that some economies might be realized by combining several monitor values (a time sequence of fixed length, say) into a single item. The per-transaction overhead would be reduced by this approach, although the total data volume and I/O bandwidth would probably not be affected significantly. After some thought, however, I believe the gain in efficiency using such a scheme is unlikely to make possible the wholesale entry of the entire volume of monitor data. The volume itself is a major problem; most database systems incorporate limits on the number and/or length of records (or tuples) which are supported, and at a minimum become significantly less efficient as these dimensions are strained. Further, the proposed packing would make the individual values inaccessible for such likely operations as limit testing.

Finally we should reconsider the entire question of graphics. A good integrated graphics facility was one of the advantages we initially hoped to exploit by maintaining the VLBA's monitor data in a commercial database management system. While this is clearly still an important priority, it appears that the sort of graphics we had in mind are not widely available. We should explore as an alternative the use of a separate specialized graphics package, either a commercial version or one in the public domain. A database system supporting the fast load/unload, combined with such a graphics element, could offer us more flexibility and a much wider range of choice.