Data Management Requirements: The Similarity of Memory Management, Database Systems, and Message Processing

Olin H. Bray

Sperry Univac Defense Systems Division, St. Paul, Minnesota

Introduction

Memory management, database management, and message processing have in the past been defined in a relatively narrow way. With memory management the problem was to obtain cost effective use of real memory. Given a multiprogrammed environment, virtual memory systems allowed more effective use of expensive real memory. Memory management has become even more important with the development of very large and complex memory hierarchies. Database management systems were developed to allow the more effective use, sharing, and control of data resources – objectives which operating systems had previously provided for hardware resources. The driving force behind message processing has been the increased use of data communications and computer networks. This paper will consider the basis of the overlap in these areas, their common data management functions. Data management, as defined in this paper, includes the locating, routing, moving, and translating of data resources and the locating, reserving, and releasing of physical resources, i.e., primary and secondary storage.

The analysis performed in this paper is essential because of trends in computer architecture discussed below. Early hardware was designed for general purpose environments with software used to tailor it to specific applications. However, according to Gagliardi[9] future systems will consist of a set of subsystems, including a storage subsystem at the core surrounded by computational, spooling, and communications subsystems. The computational subsystem is the traditional "number cruncher" part of the system. The spooling subsystem provides the I/O interface between the system and the outside world. The communications subsystem links the various subsystems together and provides an interface to the rest of the network if the system is part of a larger distributed system. The storage subsystem consists of all the system's storage resources and their control processes. It controls all levels of the system memory and storage hierarchy. The storage subsystem controls the allocation of the physical storage resources and the movement of the data resources through the system. Depending on how these resources are used, they may be non-conserved or conserved, and if conserved, either serially reusable or sharable. Physical and data resources may be located, and if necessary reserved, independently or jointly.

Gagliardi proposed that these subsystems be implemented using special purpose processors. Performance needs and advanced hardware technology now make it feasible and cost effective to augment general purpose mainframes with such special purpose equipment as front end message processors, back end database machines, and memory management machines. However, a requirements analysis is essential for any special purpose equipment. It makes little sense to develop several special purpose processors if the functional requirements are similar enough to be met by one new type of processor. On the other hand, if the functions are different enough to justify different processor types, requirements analysis is needed to functionally specify the necessary system concepts. This paper considers such an analysis.

First, the paper defines a requirement analysis and explains the part on which it will focus. Then it describes two ways resources can be classified: as either physical or data resource or by usage as non-conserved, serially reusable, or sharable. Classification by usage is the more important since it determines how the resources can be reserved or shared. The next section describes several ways of classifying data structures and implementation methods. Then the paper considers seven activities, a subset of which must be performed in any data management function: (1) the need for a data management function must be recognized; (2) the request for the function must be issued; (3) the source and target resources must be located and if necessary reserved; (4) a physical access path from the source to the target must be determined; (5) the access path must be obtained or reserved for use; (6) the data must be physically moved or transferred; and (7) any necessary data translation must be performed. Then the paper describes how these activities can be combined to provide the four basic data management operations – retrieval, modification, insertion, and deletion. Several query types are identified. The final section identifies several important architectural alternatives and describes how they can be used.

Requirements Analysis

This section sets the stage for the paper by defining a requirements analysis and identifying the steps on which the paper will focus. A requirements analysis, as defined by Thurber,[20,21] includes six steps. First, the problem must be defined. Second, based on the problem definition, the user's needs and wants are identified. Needs are essential capabilities without which the system is not viable. Wants are features which the user would find desirable, but which are not essential. System alternatives are ranked on the basis of wants. Third, system specifications are written which clearly identify the needs and wants. Fourth, alternate system architectures are identified which will meet the system requirements. Fifth, the alternatives are evaluated based on the needs and wants criteria identified in steps 2 and 3. Finally, the "best" system alternative is chosen. Since this paper is a preliminary analysis, the focus is on the problem definition and, at a general level, identification of some architectural alternatives.

The next two sections discuss two important areas which will determine these requirements. First, there is a discussion of how resources can be classified and how they are used and controlled. The second area develops several classifications for data structures and implementation methods.

Resource Types

Before discussing activities, several resource types must be identified since the activities performed differ depending on the resources involved and how they must be controlled.

Resources can be classified in several ways. They can be either physical storage or data resources. With the conventional approach operating systems control the allocation of physical resources, while database management systems control the allocation of data resources. However, there is some confusion in using only this physical and data distinction, since most operating systems have limited data management functions. Also problems and inefficiencies occur when different subsystems must coordinate the allocation of related resources.

Resources may also be classified by their usage as consumable, serial reusable, or sharable. Consumable resources, such as status information or real-time data, are temporary. They are lost either when they decay, become outdated, or are used by a process. Non-consumable resources are permanent. Depending on how they are used, they are either serially reusable or sharable. If they are serially reusable, only one process can use them at a time. However, once a process has finished with them, they can be assigned to another process. These are the resources which a process must explicitly request, reserve, and release. On the other hand, sharable resources, such as read only data, can be used concurrently by multiple users without interference.

Different procedures are required for controlling the three types of resources. Consumable resources need only be assigned or given to the proper user. Since they are destroyed, there is no need to explicitly release them. Serially reusable resources must be requested and reserved for a user and explicitly released when they are no longer needed. When to request and release them may be determined by the user or in some cases by the system. Semaphores and monitors are software implementations of ways to reserve and release serially reusable resources.[25] Sharable resources do not need to be reserved for a user since there is no possibility of interference in their use.

Consumable resources are relatively rare and are normally only data. However, serially reusable and sharable resources may be either physical or data resources. Tape drives, printers, and terminals are serially reusable. Almost all file systems and some database systems are controlled in this way. For example, a database may be serially reusable for updating, while for retrieval it is sharable. In some cases it is the nature of the resource but in other cases it is how the resource is used that determines how it must be controlled. Other examples of sharable resources include channels, large mass storage devices, and data used for retrieval only. Programs and data can be any of the three types of resources. Reentrant programs and data used for retrieval only are sharable. Programs which modify themselves are serially reusable if they have an initialization section, otherwise they are non-conserved.

To avoid confusion the point must be emphasized that classification as serially reusable or sharable is made with respect to the entire process, not simply a single use of the resource. Once a user has acquired a serially reusable resource, e.g., a tape drive, no one else can use it until he is through with it. Semaphores can control this type of resource allocation. However, with a sharable resource the user is never really assigned the resource, he is simply allowed to use it whenever he needs it. But at the time he is using it no one else can use it. The important point is that he uses it only for relatively brief periods during his processing time. In some cases the resource is even assigned to a separate process, such as a part of the operating system which receives and processes the requests for its use. This involves the use of a monitor.

To clarify the point, let's consider two examples, an I/O channel and real memory. Although a user may have reserved an I/O device for his own dedicated use, he is not assigned a specific path to it. When he needs to do an I/O, a path is dynamically created to his device, the data is transferred, and the path ceases to exist. Parts of the path can then be used by other users to obtain access to their own dedicated resources. From the user perspective, the path of an I/O channel is a sharable rather than a serially reusable resource.

As more sophistication develops in the use of a resource, it may shift from being a serially reusable resource to being a sharable resource. This has happened with memory. In the earlier systems it was a serially reusable resource. A user was assigned a block of memory and kept it until his job was completed. Then he released it, so it would be available for the next user. However, with virtual storage systems real memory has become a sharable resource, at least from the user's perspective. Multiple users are swapped in and out of real memory during their execution, creating an impression of sharing. Actual sharing does occur when multiple users use the same copy of a reentrant routine.

Data Structures and Implementation

This section identifies several different data structures and implementation methods. Data structures reflect the real world structure which they describe, whereas the implementation methods are determined on the basis of efficiency. This section describes ways to classify data structures and their implementation methods and indicates some of the implementation trade-offs. The next sections discuss data management activities and operations. The last section describes several architectural alternatives and considers their relative effectiveness with the various data structures and implementations.

A data structure can be classified using (1) the number of entry points and (2) the number of connection sequences.[17] An entry point simply identifies the first record in a file or set of records. A connection sequence specifies the ordering or sequence of the records, i.e., how to move from one record to the next. Figure 1 summarizes the combination of entry points and connection sequence alternatives.

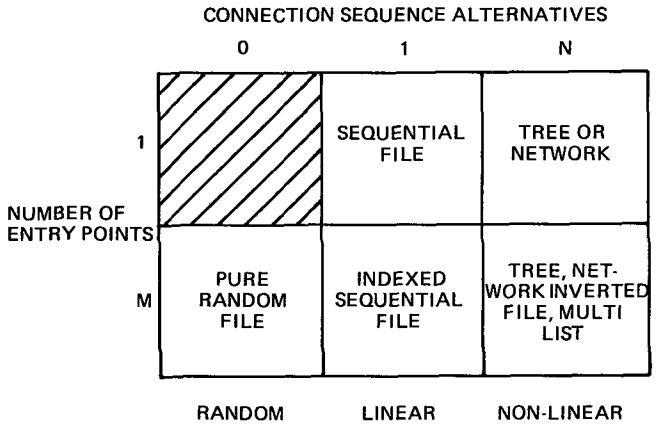

Figure 1. File Organizations

The number of entry points and connection sequences determine whether the file is random or sequential. Every set of records must have at least one entry point, but depending on how it is organized, it may have multiple entry points. Independent of the number of entry points, a file may or may not have a connection sequence. However, if it has no connection sequence, then there must be an entry point for each record. This is a random file organization. The traditional sequential file has only one connection sequence. The file may or may not be ordered, but if it is, then it must be ordered on the basis of the key that determines the connection sequence. An indexed sequential file is simply an extension of the concept with multiple entry points. Indexed sequential files are ordered with entry points for each block of records. In an extreme case with an inverted file there is an entry point for each record.

With a single connection sequence the data structure is linear: given one record, the next record in the sequence can be uniquely determined. Non-linear data structures (hierarchies and networks) have more than one connection sequence. In other words, given a record, there are several records which can be logically considered as the next record. Since the system has no way of knowing which one to use, the user must navigate his way through the data structure.

The choice of a sequential or random file organization is determined by how the data is used. Sequential files are efficient if most of the records are processed each time the file is processed. The file is entered through the one entry point, or with an indexed sequential file one of a few entry points, and the records are processed in the single connection sequence. The connection is implemented in a way that allows rapid movement to the next record. However, if only a few records are needed, a random organization, i.e., an entry point for each record and no connection sequence, is better. While it takes longer to get any individual record, a specific set of records can be found more quickly since the entire file does not have to be processed.

Entry points can be implemented with pointers or by hashing. With sequential files the single entry points are implemented using pointers. With an indexed sequential file there is a key and a pointer to each segment of the file. The key indicates the value at one end of the segment, while the pointer points to the other end. With random access files multiple entry points are implemented using either inversion or hashing. With inversion there is a set of pointers for each key value – one pointer for each record with that key value. With hashing a key value is transformed in some way to provide an address. The purpose of hashing is to map the key values uniformly over the storage space. Although there are many hashing algorithms, division provides very good overall performance as long as the divisor is not divisible by anything less than 20.

Figure 2 summarizes the ways of implementing access methods. The vertical axis defines how the connection sequence is established. This can be done using either data or pointer sequential methods.

With a data sequential method the records are stored in a physically continguous fashion. A file can be data sequential with respect to only one key. With the pointer sequential method there is a pointer to the next record. There may be multiple pointers for a given sequence or ordering, e.g., a forward and backward pointer for linear structures or with hierarchies pointers to each child or sibling node. With the pointer sequential method records can be ordered with respect to several keys by providing a pointer or pointers for each ordering. The horizontal

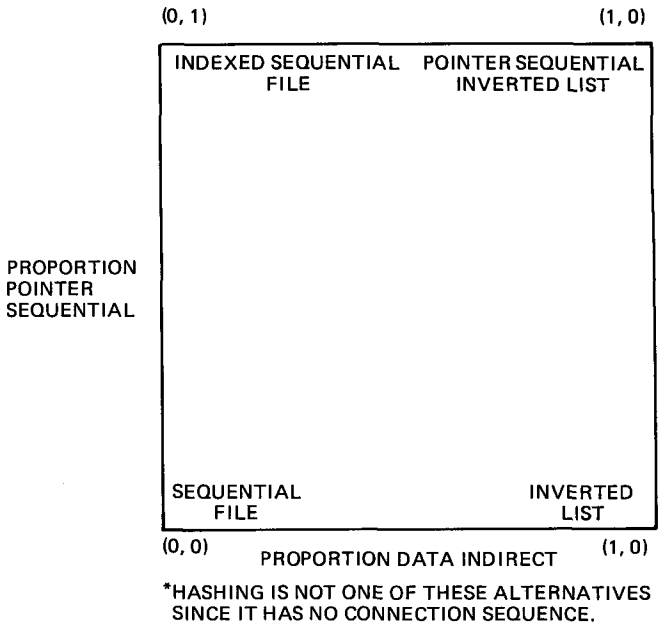

Figure 2. Connection Sequence Implementation Alternatives*

axis indicates what is being pointed to. Data direct pointers point to the data itself. Data indirect pointers point to other pointers rather than the data. Symbolic pointers are an example of the data indirect approach.

Data Management Activities

This section focuses on the seven basic data management activities from which the four data management operations are built. For some of the operations all of the activities are necessary, while for others a few of the activities are not applicable. For example, in memory management there is no data translation activity. This section will discuss each of the activities. The next section will see how each of the four operations are built up from the seven activities in each of the three application areas. Two important issues are covered in this section: first, to what extent an operation can be anticipated and what the consequences of this anticipation are, and second, how various types of resources are located and how they are controlled.

1. *Recognize the Need for a Data Management Operation* First, the need for an operation must be recognized. Depending on the function and the area, the user or the system may recognize the need. If the system can recognize the need and initiate the request, then the function can be transparent to the user. Over time more functions are being recognized and performed by the system rather then the user. Memory management has eliminated the need for the programmer to partition and overlay his program. Database systems are becoming more sophisticated and relieving the user of many functions, e.g., backup and recovery, concurrency control, and much of the integrity testing and maintenance. In some cases the need cannot be recognized until the actual request is made, e.g., demand paging. Then the requesting process must wait until the function has been performed before it can continue. However, if the need can be anticipated, then the request can be started ahead of time to reduce the delay. When trying to anticipate requests, any of the four situations shown in figure 3 can occur. On one dimension, the function may or may not be requested in anticipation of the need. On the other, the request may or may not ultimately be needed. In cases I and IV the correct decision was made. Either the request was issued and later needed, or the request was not issued and the need never

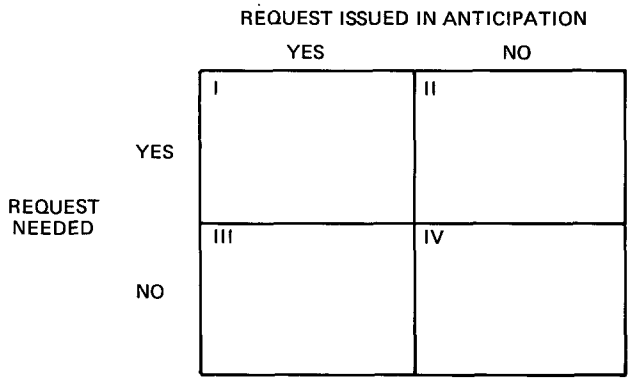

Figure 3. Request Decisions

materialized. However, situations II and III result in problems. In case II the request was not issued, but the need did develop; therefore the process was delayed. A delay could also occur in case I if the request was not issued early enough. In case III the request was issued, but the need never materialized. This results in some unnecessary processing. The breakpoint at which a request is issued depends on the relative costs of each type of error, delay, or unnecessary processing, and the probability of making it.

2. *Make the Request for a Data Management Operation* The second activity, actually issuing the request, may be done by either the user or the system, depending on who recognizes the need. If the request cannot be anticipated, then it is issued when the need occurs. However, frequently there can be some anticipation of a request, so a decision must be made when to issue it. There is usually an ideal interval during which the request should be issued. If it is not issued soon enough after the need is recognized, it will not be completed by the deadline and the requesting process will be delayed. On the other hand, if the request is issued too early, then it may be completed and preempted before the results are used.

For example, consider I/O buffering. When processing a random file, the current record provides no information about the next record that will be needed. Therefore, anticipation is not possible. However, when processing a sequential file, the next record can usually be identified. In a non-virtual memory system, when this occurs the next record is read into a buffer in anticipation of that need. In this case the ideal interval has only one limit – the read should be completed before the data is needed. Response time is the sole constraint. However, in a virtual memory system there is another constraint on the request interval. If the data is requested and read too early, it may be paged out before it can be used. This would require still another request, although probably a much faster one, before the data could be used.

3. *Locate Resources* The third activity involves locating, and if necessary reserving, resources – either data or physical storage resources. Although all data management functions require locating resources, there may be three distinct types of locates. A request may involve locating only physical resources, only data resources, or both physical and data resources. Further, a data management function may require two separate locates, one locate for the source of the data and one for the target or destination of the data.

First, let's consider how physical and data resources can be specified or located. Physical resources may be located by type or by identification of a specific unit. However, the request for a specific unit usually is done in conjunction with a request for data resources. Data resources can be located by either address or content. In conventional systems the location is ultimately by position, as either an absolute address or a relative position in a data structure. However, associative memory systems allow the user to directly locate data by its content, rather than by position or address. Although many conventional database systems appear to provide the user with this content addressing capability, ultimately the location is by position since the system makes the appropriate conversion from a content to a position reference.

Depending on whether physical or data resources are involved, there are three types of locates, as shown in figure 4. In case I only physical resources are requested. Since data is not also being requested, any resource of the specified type can satisfy the request. The data will be loaded into the storage block later. When a block of storage is requested in this way, something is going to be written into it. Therefore, it must be reserved for the requesting process. Allocating buffers or a block of pages for a process working set are examples of a type I request.

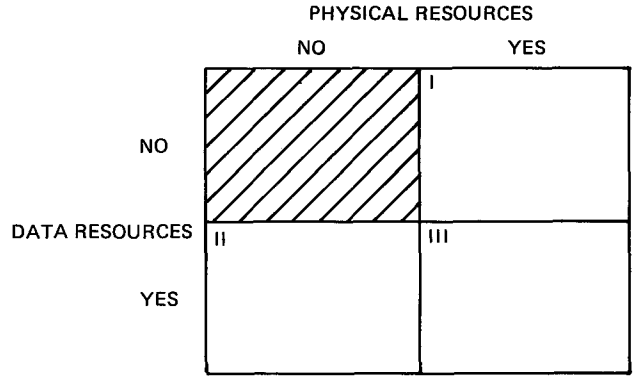

Figure 4. Types of Locates

In case II only data resources are needed. The requesting process already has the physical resources in which to load the data. Type II requests may be used in a sequence with type I. A process may issue a type I request once to obtain a block of storage. Then it does a series of type II requests, loading different data into that block. This reduces the overhead and improves the process's response time since the physical resource only has to be located and reserved once. However, since the storage requested in this way becomes serially reusable and must be reserved for the process, its sharability is decreased.

A type III request is in effect a combination of types I and II: a process concurrently requests both a physical and data resource. In this case the request is for a specific physical resource, the one containing the required data. The request may be for either the data or the physical resource. If the data is to be modified, then both resources are serially reusable and must be reserved. However, if the data is only being retrieved, then neither resource has to be reserved since they are both sharable.

4. *Determine the Physical Access Path* The fourth activity is determining the path to be used to move the data from the source to the target. First, the path selection or routing algorithm determines the alternative paths. With memory management and database management in a non-distributed environment there are only a few relatively simple paths to identify and evaluate. In this case the algorithm will be very simple. However, in a distributed system with a large communications network it will be necessary to identify and evaluate many complex paths, each of which may

contain several subpaths. The complexity of the problem is sometimes reduced by using a static rather then a dynamic path selection algorithm. In the static case a single fixed path (or a few specific paths) are determined once and used for all data transfers between a given source and target pair. However, over time, problems can develop with these paths. They may become overloaded and busy, develop a high error rate, or even fail completely. These problems can be avoided by using dynamic path selection, which determines the best path each time data is transferred. While this results in a better path being selected for each message, it does create additional overhead since the path has to be determined many times rather than once, as was the case with the static selection. Path selection can also be simplified by using dedicated rather than shared paths. However, the disadvantage is that dedicated paths are much more expensive, especially when long distances are involved.

5. *Reserve the Selected Path* The fifth activity is acquiring or reserving the selected path. With a dedicated path this is no problem. However, with shared paths, such as buses, this reservation or arbitration can be difficult. During a transmission the path must be reserved to prevent interference. From this perspective paths are serially reusable. When data is being passed over a path, the next request, even if it is from the same user, must wait until the transmission has been completed. However, from another perspective a path is a sharable resource. Users do not reserve paths continuously for long periods. With shared paths the user does actually reserve the path for a short time, although this is done at the hardware rather than the software level. In other cases the path is permanently assigned to a specific process or monitor which receives and processes the user's requests. Since buses are normally only effective over short distances, the latter approach is usually used in large communications networks.

6. *Transfer the Data* Once the source and the target have been determined and the path selected and reserved, the actual data transfer is relatively simple. However, the data may be transferred as a single block or it may be divided into many smaller blocks. Dedicated paths can transfer large blocks of data, although for some applications even these transfers are broken down into a series of smaller blocks. Shared paths usually transfer small blocks of data. Since a shared path is serially reusable during a single block transfer, but sharable between blocks, using smaller blocks allows the path to be shared by more processes during a period of time. However, when the data transfer is broken into a series of smaller blocks, additional overhead is incurred by each block. At one extreme each block may be processed independently. This means that to transfer a large block there would be multiple locates of the source and target, path determination, and reservation. At the other extreme there would be multiple reservations of a path which had been determined only once. After the transfer is completed, the path is released.

7. *Translate the Data* Finally, there may be a data translation activity. This may be done because of data encoding specified by the database user or data compression or encryption done by the system through either hardware or software. Data translation will often change the data volume. This change complicates the locate activity, especially when the final volume cannot be predicted until the translation is completed.

The next section describes how these seven activities are combined to provide the four data management operations – retrieval, modification, insertion, and deletion.

Data Management Operations

Everest[7] reviews nine of the data management functional taxonomies proposed since the late 1950's. Most of them include file definitions, sometimes at the detailed data and storage structure level; creation; update; and interrogation, retrieval, or report generation. However, for this paper a more basic set of operations are used. These four common data management operations are retrieval, modification, insertion, and deletion. Both data management and memory management perform all of these operations. The two basic differences involve the data structures and whether the system or the user recognizes the need and issues the requests.

A memory management system using a single address space can be implemented with a simple data structure – a single connection sequence ordered by address and multiple entry points, in other words, an indexed sequential data structure. More complex data structures are needed to implement multiple virtual address spaces and allow sharing of procedures and data. Database management is still more complex since it must provide for a variety of data structures where different users may view the same data differently. Ideally the database management system, but sometimes the users, must know these structures and how to convert from one to the other.

The first operation is retrieval, in which data is simply located and read. In database management the retrieval need is recognized and the request is explicitly issued by the user, whereas in memory management the user process is executing and if that part of the address space it needs is not in residence, then the system recognizes the need and issues the request. These two approaches converge with virtual I/O where the user defines his data file as part of his address space and then references it by simply addressing it rather than by explicitly issuing I/O commands. With retrieval, both the source and the target must be located. With databases, locating the source may be very difficult because of the variety of query types, but the target is usually specified by the user as a particular buffer or data area. On the other hand, with memory management the target may be more difficult to locate than the source both because of the complex page reclaim algorithms and the fact that there is only a single simple query type. Determining and acquiring the physical access path and transferring the data are comparable for both database and memory management, although there are usually more devices and paths with the database. However, in both cases the paths are of similar complexity, except with a distributed database. Data translation is exclusively a database activity in which the data must be converted from its stored form described in the schema to the user specified form defined in the subschema.

There are two types of modification, the second operation. Value dependent modifications take the current value and operate on it in some way to produce the new value. Value independent modifications simply replace the old value – its previous value is irrelevant. The interference problem with concurrent updates occurs with value dependent modifications where the previous value may be incorrect if the data is not locked. With a value dependent modification there are two operations: a retrieval with a lock and then the actual modify or write. The user must recognize the need and issue the value dependent modification. There is no way for the system to recognize this need. For the system, a retrieval as a complete operation is indistinguishable from a retrieval as the first step in a value dependent modification. With value independent modifications there is no need for a prior retrieval

and lock. Either the user or the system can recognize a need and issue a request for this type of operation. Data undergoing a value dependent modification is a serially reusable resource, whereas data being retrieved or undergoing a value independent modification is sharable. Neither the retrieval nor the modification require the allocation of additional space. All of the other activities for the modification are similar to those for the retrieval.

The third and fourth operations, insertion and deletion, are different from retrieval and modification since they require the location and reservation or release of physical resources instead of just data resources. With both operations the user of the database must recognize the need and issue the request, whereas with memory management the system does this, usually at job initiation and termination. With insertion, the source is the data which is specified and the target is the additional physical resources which must be located and reserved. The physical access path must be selected and reserved and the data transferred to the newly allocated storage.

With deletion there is no target resource. The source data is located and the corresponding physical resources are released. Access paths and data transfers are not needed, except with secure data which must be destroyed or overwritten rather than simply released.

For all of these operations the complexity of the query type is one of the significant differences between database and memory management. Therefore, the next section will describe the various query types. The last section will then describe several possible data management machine architectures and show how they would process the various query types.

Query Types

One of the major differences between database and memory management is in the range of query types. With memory management there is a single query type – given a specific address find the pages or segments of the address space on which it occurs. The problem is further simplified since the space is ordered on a single variable or connection sequence, the address.

In a database the selection criteria in a query can take many forms. Some examples are shown below:

1. X = value
   LIST EMPLOYEE WHERE (SKILL = ENG).

2. X = range (value 1 $\leqslant$ X $\leqslant$ value 2)
   LIST EMPLOYEE WHERE (SALARY BETWEEN 15000 AND 25000).

3. X = extreme (either maximum or minimum)
   LIST EMPLOYEE WHERE (SALARY = MAXIMUM).

4. X = Y
   LIST EMPLOYEE WHERE (JOB CODE = SKILL).

5. X = value 1 "boolean operator" Y = value 2
   LIST EMPLOYEE WHERE (SKILL = ENG .AND. DEPT = PROD).

6. X = Y "boolean operator" Z = W

The two basic characteristics in the query types are (1) the search is for an X equal to a specific value or equal to another variable and (2) there is a single search argument or several arguments are connected by boolean operators.

Architectural Alternatives

This section discusses two of the more promising database processor architectures – the associative memory processor and the associative array processor. First, it shows where these two architectures fit into a general classification. Second, it identifies the components of an associative processor and describes their operation for the different query types. Third, it describes several ways in which these elements can be linked to obtain an associative array processor. Finally, it describes how an associative array processor could process the various types of queries described in the previous section.

Flynn[8] has proposed a four-way system classification. There may be a single or multiple instruction stream. Each type can be further subdivided into a single or a multiple data stream. This yields four types of systems – SISD, SIMD, MISD, and MIMD. Higbie[10] has further subdivided the SIMD (Single Instruction Multiple Data) classification into four types. His categories are: (1) an array processor, which operates on parallel data streams addressed by position; (2) an associative memory processor, which operates on multiple data words addressed by content; (3) an associative array processor, which operates on parallel blocks of data addressed by content; and (4) an orthogonal processor, which operates on multiple data streams addressed by either content or position, i.e., bit and word slice memory organization. Since database operations are primarily based on content addressing, the two most promising types are the associative memory processor and the associative array processor.

Both associative processors have the same five basic elements, shown in figure 5. The data register contains the value for the search. The mask register indicates which part of the data register is to be used in the search. The data array contains the data words being searched. The word select register indicates which words in the array are to be checked. In the search results register a bit is set corresponding to every word for which there is a match. When there is more than one match, the MMR or multiple match resolver points to the first word in the set.

Now, let's consider how these elements can be used to process the various query types. The two basic query forms, X = "value" and X = Y are simple. The data register is loaded with either the value or the variable, the data array is loaded with the data to be checked, the mask register and the word select register are set, and the test for equality is issued. The results are obtained from the search results register and the MMR. Depending on the extent of the instruction set, the test could be for equality, inequality, less than, less than or equal, greater than, or greater than or equal. It may also be possible to identify the maximum or minimum value in the data array, in which case only the mask, not the data register, would to be loaded. Compound queries are formed by boolean combinations of simple queries, e.g., X = value 1 .and. Y = value 2. Range is another example of a command query, e.g., X $\geqslant$ value 1 .and. X $\leqslant$ value 2 (value 1 $\leqslant$ X $\leqslant$ value 2). For a range check, the lower limit is set in the data register; the data array, mask register, and word select register are loaded; and the greater than or equal test is issued. Then the upper limit is set in the data register, the search results register is moved to the word select register, and a less than or equal test is issued. A series of .and.s can be implemented by shifting the search results register to the word select register and resetting the search results register to zero after each test. A series of

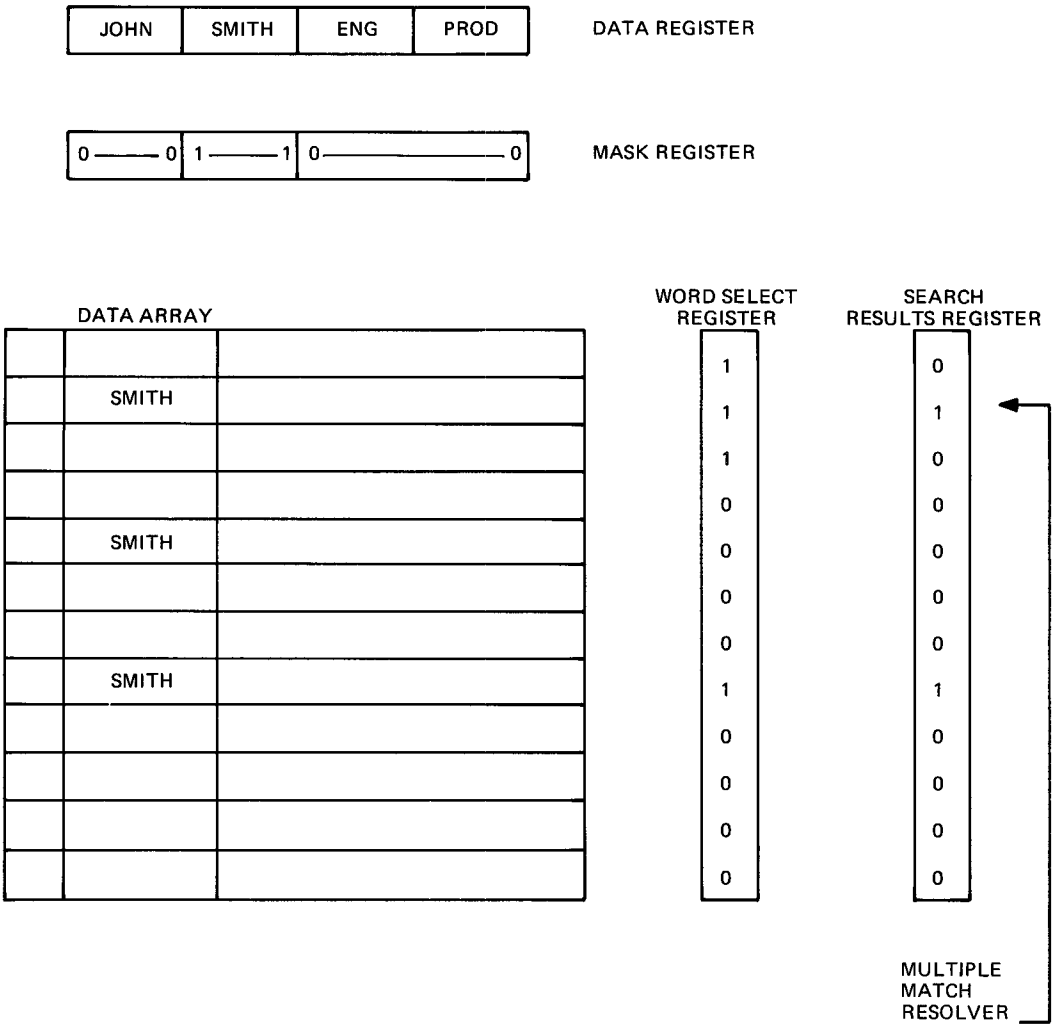

Figure 5. Associative Processor Elements

.or.s can be implemented simply by not resetting the search results register after each test. The final search results register bit is set if the corresponding word met any of the conditions.

Almost all virtual memory systems use a small, special purpose associative memory processor for their implementation. Typically, page and segment numbers are used as the arguments for an associative search. When there is a match, the corresponding base address is added to a displacement and the real address is generated. If a match is not found on the associative memory, the search continues using a conventional search procedure on an indexed sequential address structure.

There are several ways in which associative memory processors can be connected to provide an associative array processor. These approaches are determined by whether there is a single or multiple copies of each of the five elements. The first array processor type involves multiple data arrays, search results registers, and MMRs, but a single data, mask, and word select register. This simply allows a single associative processor to operate on larger blocks of data. The second type, of which STARAN is an example, is an extension of the first type created by providing a separate word select register for each array. This would allow different words in each data array to be searched for the same search argument. Types three, four, and five are created by adding a separate data register, mask register, or both for each data array. By providing separate data and mask registers each array can be searched in parallel for different arguments. Greater parallelism can be obtained so the searches can proceed asynchronously on each data array. With this configuration, multiple independent search requests, for example, requests from different users or various parts of a single compound request, could be processed concurrently.

There is an additional feature which would allow faster processing. Very high data rates are used to load the multiple data arrays. Assume multiple search arguments are needed, each applicable over a large block of data. An effective much higher data rate can be obtained if the entire block can be loaded into the data arrays only for all of the searches. This could be done if it were possible to shift the data and mask registers assigned to each array to the next array. However, this would require much larger word select and search results registers, since each one would have to span all of the data arrays rather than only one.

Summary

The goal of this paper was to lay out some of the functional requirements for a data management machine. It started with two basic premises. First, although functionally distributed database machines are being implemented on general purpose minicomputers, hardware technology now makes it feasible to develop a special purpose database processor. Second, there is enough

overlap in three areas – memory management, database management, and message processing – to enable a single design to meet all their requirements. This paper was the beginning of an analysis of those requirements.

Although there is still much to be done, this paper provides a start. First, it has identified several types of resources which a data management machine, or in Gagliardi's terms "the storage subsystem," must be able to locate and allocate. Usage seems more important for resource control and allocation than the type of resources, physical or data. Since semaphores and monitors are applicable methods for such resource control, a data management machine should facilitate the definition and use of these methods.

Second, memory management is frequently simply a very limited version of a database – a sequential file with a single entry point, one connection sequence, and a single simple type of retrieval. However, all systems with virtual memory provide some additional hardware, usually a limited associative memory capability, to improve response times and performance. Given that associative hardware can improve the performance of such a limited database, a similar facility could provide drastic improvements in many database applications, especially those based on a relational approach. Associative memory and associative array processors provide the best capability for such a system. They can improve the responses for all of the database query types and can reduce some of the overhead required to maintain the data structures. For example, when the data array is being searched, connection sequences and ordering are irrelevant; therefore, the overhead required to maintain them can be reduced.

Third, data translation for compression and encryption for data security are common requirements for both database and message processing, although not for memory management systems. Schema and subschema mapping in database applications provides an even greater translation requirement. Therefore, any specially designed data management processor should have extensive data translation capabilities.

Finally, a message processor should have features which facilitate the processing of complex routing algorithms. However, memory management and database systems don't seem to need such a capability. Therefore, unless similar algorithms would have other uses, for example, determining logical access paths in a database, these features would be of secondary concern for a data management machine.

There are several additional steps that are needed to further the analysis begun in this paper. First, there should be a survey of specific algorithms for resource management, processing data structures, data compression and encryption, and secondarily for routing or path determination. Then hardware features should be identified which would result in significant performance improvements for these algorithms. Finally, simulation or emulation studies could be done for each of the three types of applications to determine how much of a difference special purpose hardware would make in performance or costs. On the basis of these more complete studies, detailed requirements analyses can be done for each of the areas.

Bib .ography

1. Abrams, M., Blanc, R. P., and Cotton, Ira W., *Computer Networks: Text and References for a Tutorial*, IEEE Computer Society, 1976.
2. Anderson, G. A., and Jensen, E. D., "Computer Interconnection: Taxonomy, Characteristics, and Examples," *Computing Surveys*, December, 1975.
3. Aschim, Frode, "Database Networks – An Overview," *Management Informatics* (3:1), 1974.
4. Canaday, R. W., et.al., "The Back-end Computer for Database Management," *CACM* (17:10), October, 1974.
5. Denning, Peter J., "Virtual Memory," *Computing Surveys*, September, 1970.
6. Dyke, Ruth, "Backend Database Machine for the U.S. Civil Service Commission," *Proceedings: Texas Computer Conference*, 1976.
7. Everest, Gordon C., *Managing Corporate Data Resources: Objectives and a Conceptual Model of Database Management Systems*, Ph. D. Dissertation, University of Pennsylvania, 1974.
8. Flynn, M. J., "Some Computer Organizations and Their Effectiveness," *IEEE Transactions on Computers*, September, 1972.
9. Gagliardi, U. O., "Trends in Computing-System Architecture," *Proceedings of the IEEE* (63:6), June, 1975.
10. Higbie, L. C., "Supercomputer Architecture," *Computer*, December, 1973.
11. Infotech State of the Art Report on *Virtual Storage*, 1976.
12. Infotech State of the Art Report on *Database Systems*, 1975.
13. Infotech State of the Art Report on *Distributed Systems*, 1976.
14. Infotech State of the Art Report on *Network Systems and Software*, 1975.
15. Kimbleton, S., and Schneider, G. Michael, "Computer Communications Networks: Approaches, Objectives, and Performance Considerations," *Computing Surveys*, September, 1975.
16. Lowenthal, Eugene I., "The Distributed Data Management Function" *Proceedings: National Computer Conference*, 1974.
17. Severance, D. G., "A Parametric Model of Alternate File Structures." Department of Operations Research, Cornell University, Technical Report No. 204, October, 1973.
18. Stutzmann, B. W., "Data Communications Control Procedures," *Computing Surveys*, December, 1972
19. Sunshine, C. A., *Interprocess Communication Protocols for Computer Networks*, Ph. D. Dissertation, Stanford University, 1976.
20. Thurber, K. J., "Requirements Oriented Design: An Emerging Design Strategy," *COMPCON*, Spring, 1977.
21. Thurber, K. J., "Techniques for Requirements Oriented Design," 1977 NCC.

22. Thurber, K. J., *Large Scale Computer Architecture: Parallel and Associative Processors*, Rochelle Park, N.J.: Hayden, 1976.

23. Thurber, K. J., and Patton, P. C., *Data Structures and Computer Architecture: Design Issues at the Hardware/Software Interface*, Lexington, Mass.: Heath, 1977.

24. Thurber, K. J., and Wald, L. D., "Associative and Parallel Processors," *Computing Surveys*, (7:4), December, 1975.

25. Tsichritzis, D. C., and Bernstein, P. A., *Operating Systems*, New York: Academic Press, 1974.

26. Yau, S. S., and Fung, H. S., "Associative Processor Architecture – A Survey," *Computing Surveys*, (9:1), March, 1977.