Management/ Data Base Systems
H. Morgan Editor

A Back-end Computer for Data Base Management

R.H. Canaday, R.D. Harrison, E.L. Ivie, J.L. Ryder and L.A. Wehr
Bell Telephone Laboratories, Inc.
Piscataway, New Jersey

It is proposed that the data base management function be placed on a dedicated back-end computer which accepts commands (in a relatively high level language such as the CODASYL Data Base Task Group, April 1971 Report) from a host computer, accesses the data base on secondary storage, and returns results. The advantages of such a configuration are discussed. An experimental implementation, called the eXperimental Data Management System, XDMS, is described and certain conclusions about the back-end approach are drawn from this implementation.

Key Words and Phrases: data base management, information retrieval, computer configurations, computer networks, Data Base Task Group Language, data base protection, data base portability, back-end computer

CR Categories: 3.79, 4.22, 4.33, 4.35

Background

Data base management has, more perhaps than any other subject during the past few years, attracted and held the interest of the business data processing community. This stems in part from an increased awareness of the central role that the data base plays in most business applications. It can also be attributed to the fact that both the size and complexity of the data upon which an application is based have increased to the point that, typically, a sizeable portion of the resources allocated to that application are tied up in accessing and changing the data base. The term, data base management, is defined for the purposes of this paper to include all activity related to the creation, accessing, and maintenance of a large collection of information containing complex interrelationships. Typically, a data base resides in a computer's secondary storage and is used by a number of different applications.

Data base management is distinguished from file management in that file managment provides access to files with simple structures (sequential, indexed sequential, etc.). A data base management system may use a file management system to gain access to the simple physical files which contain the complex data base relationships.

The dichotomy in the computer industry (business versus scientific/academic) has also resulted in a divergence in terminology. The term information retrieval is often used in academic circles. It is sometimes used in the restricted sense of document retrieval, but in its more general context it is equivalent to data base management. We will use the term data base management in this paper in deference to its wider acceptance in the industry.

The problem of data base management has been attacked from a number of directions. Numerous data base management systems (IMS, TOTAL, IDS, etc. [2, 3]) have been developed and are in wide use. Through the efforts of the CODASYL Data Base Task Group, DBTG, a data base management language has been developed which holds promise of providing a more uniform interface between the application program and the data base management system being used. More will be said of this effort later. Numerous proposals have been made on the kinds of positions needed to manage a data base (e.g. data base administrator). A number of related topics such as data base privacy and reliability have also received considerable attention. In this paper the data base management problem is attacked from a still different direction. A new computer configuration for data base management is proposed. In the next section, Back-end Concept, the basic idea behind the configuration will be described. In the following two sections, Advantages and Disadvantages, the relative merits of the approach are discussed. An experimental implementation of the concept is then described, Experimental Data Management System-XDMS. The final two sections, Evaluation and Conclusions, summarize what has been learned to date about the back-end approach.

Back-end Concept

The basic idea behind the back-end concept is shown in Figure 1. In a conventional data base management system, all of the major software components—operating system, data base management system, and application programs—execute on a single machine which

Authors' address: Bell Telephone Laboratories, Inc., P.O Box. 2020, New Brunswick, NJ 08703.

Fig. 1. Back-end concept.

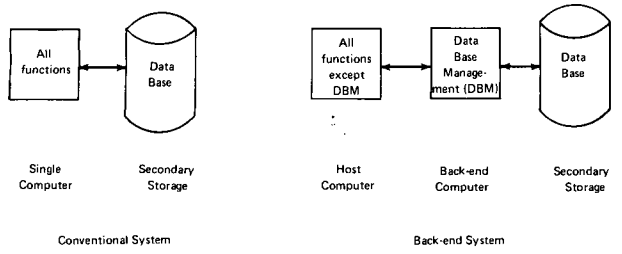
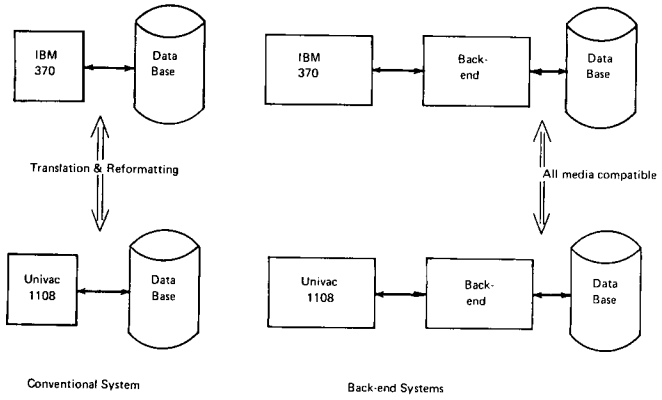


Fig. 2. Data transfer.



has direct access to the data base on secondary storage. In the back-end system, the data base management function is implemented on a separate machine which has exclusive access to the data base.

The term back-end was selected to describe this configuration because of the rather obvious analogy to front-end computers [4]. A front-end computer serves as the interface between host computer and its external inputs (from terminals, computer networks, etc.). The back-end computer serves as the interface between the host computer and its data base.

Front-end systems range from special-purpose machines, which tend to be less expensive, to general-purpose computers, which offer greater flexibility. Analogously, one could visualize special-purpose back-ends which might be considered the next step up in the continuing sophistication of the hardware used to control secondary storage devices. Or one could consider the use of a general-purpose computer as a back-end, which is the approach taken in the experimental system to be described later in this paper.

With either approach, the host computer will require some software to interface with the back-end. This interface will be responsible for collecting the data base management requests from the application programs and transmitting them to the back-end. In turn it will accept results and status from the back-end and distribute them to the application programs.

Potential Advantages

Having outlined what is meant by a back-end system, we will now discuss some of the possible advantages and disadvantages of such an approach. In a later section, we will discuss the extent to which these advantages have been proved or disproved through our experimental implementation.

Advantage 1. Economy through specialization. The first potential advantage of the back-end approach that will be examined is the fact that the hardware and software used for the back-end can be specialized to handle just the data base management function. For the software this means that a large general purpose operating system is not required. The back-end operating system can now be tailored to serve just the data base management function. This allows a greatly simplified interface between the operating system and the data management function and much more flexibility in the distribution of work between these two components. Economies should thereby accrue in at least the following areas: (a) smaller on-line system requiring less core; (b) simpler programs requiring less processing time; (c) smaller development costs; and (d) shorter development cycle.

In the hardware area a machine can be selected which is particularly suited to data base management. A back-end computer should, for example, have good byte manipulation facilities and have high input/output thruput. However, it does not need floating point instructions, fast multiply and divide circuitry, a large word size for high precision, a wide variety of peripherals, etc.

Advantage 2. Shared data. A second possible advantage is an enhanced ability to share data between computer systems. The simplest type of data sharing between two computers is the transfer of files so that each system has its own copy of the information. This transfer can be accomplished by physically moving a tape or disk pack, switching a secondary storage device from one system to another, or by the transmitting of the files over some type of computer network. If the transfer takes place between two different types of computers or between two computers of the same type but with different file structure conventions, then considerable reformatting and character translation are required in one or both machines. With the back-end approach, however, data can be transferred between even very different host machines without reformatting and translation if the back-ends are the same (see Figure 2).

A more challenging type of data sharing is the simultaneous access of the same data base by two or more computer systems which may be physically separated and which may differ in make or model. A multiprocessor system does allow for data sharing between processor units, but such processor units are not really different computer systems and they are not normally physically separated. Another approach to real time data sharing is the secondary storage device (say a disk) that

accepts requests from two computer systems concurrently. Such systems are finding increasingly wide use and currently exist even between different types of computers. There are however, three problems that restrict the utility of such systems. First, adequate locking techniques have not been developed to allow one or both computers to update the data base dynamically. Second, the reformatting and translation problem still exists between different types of computers and also between different data management systems on the same type of computer. Third, these secondary storage devices have in general not been designed for access from a distance.

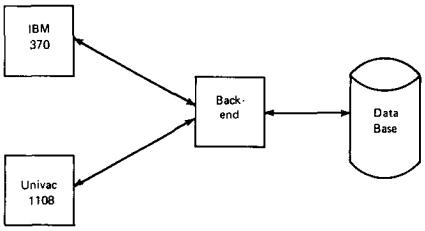
A back-end system which serves two or more host machines (see Figure 3) does, however, provide an answer to these problems. Since the data management function is centralized in the back-end, control and coordination of update requests from several hosts are possible. Also, physical separation of host and back-end is feasible. This allows the host machines to be located either remotely or locally and still share the data base.

Figure 4 depicts a further extension of the back-end concept wherein a network of back-end machines serves a number of different host machines. Such a back-end network offers the very interesting additional challenge of how to appropriately partition the data base between the back-end machines. A second interesting aspect of such a configuration is the choice of the physical placement of the host and back-end machines in remote and/or local locations to optimize system effectiveness. It should be noted that our experimental implementation consists of a single back-end and does not address the multiple back-end problem.

Advantage 3. Data base protection. A third possible benefit of the back-end approach relates to system reliability and security. These two problems are major obstacles in the development of adequate data base management systems.

The reliability of a computer system is threatened when a hardware or software failure occurs. While such failures are inevitable, measures can be taken to limit their effect and to allow the system to recover once they occur. A back-end system offers some unique advantages in limiting the extent to which a failure can propagate and in facilitating the recovery from such a failure. In a back-end system the only way that data can be accessed is over the communication link between the host machine and the back-end machine. Messages received over this link can be scrutinized for consistency, formatting, etc., to determine if a failure has occurred on the other end. If a failure occurs in the host, then the back-end can "rollback" any changes that are being made by active transactions and coast to an inactive state. This rollback is accomplished through an audit trial of data base changes kept by the back-end. On the other hand, if the host detects that the back-end has failed, then it can cease requesting service and can notify the operator. If the failure has corrupted the data base,

Fig. 3. Data base sharing.



then it should be restored to an earlier point in time from a dump tape (or disk pack). The host can then reissue the commands that were sent to the back-end subsequent to the dump. This will be possible if the host has kept an audit trail of those commands.

The basic point to be made here is that two machines should be able to detect an error situation and to contain its effect better than a (faulty) single machine attempting to do self-analysis. The dual audits also provide insurance that the audit trail being used for recovery or rollback has not been corrupted by a faulty machine. These advantages should more than offset the slight decrease in overall system hardware reliability due to additional back-end equipment.

The ability of a back-end system to provide adequate security against accidental or malicious access is also enhanced by the single link between the two machines. There can be no sneak paths through a separate file management system or through some type of breach of the memory protection system which may allow for unauthorized access to the data.

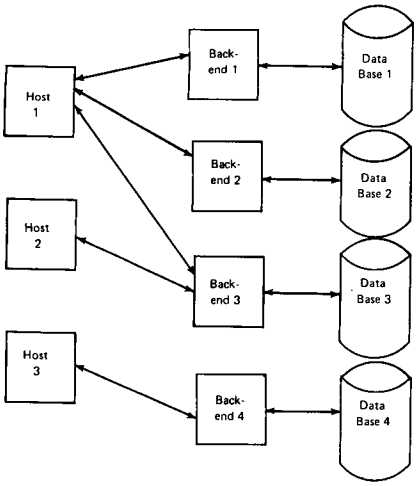
Advange 4. Data base management for new machines. Once a back-end system has been developed for a given host machine (e.g. UNIVAC 1108), the job of writing the interface for a new host (e.g. IBM 370) becomes a much smaller effort than developing the total data base management function on that machine. Also a data base management capability can be provided on a much smaller host machine (e.g. a minicomputer) because of the modest core and cpu requirements for a back-end interface.

Potential Disadvantages

On the other side of the ledger are three potential disadvantages of the back-end approach.

Disadvantage 1. Cost of a second machine. The first issue to consider is the cost of the back-end machine. In examining this issue one must balance the cost of the back-end against possible savings in the cost of the host machine. Since the back-end handles most of the data base management functions for the host, a reduction in the size of the host configuration may be possible. Alternatively, the host may be made available to process additional work. Although the trade-offs here are highly dependent on the particular application being considered,

Fig. 4. Multi host/multi back-end network.



one can observe that in a given situation one can generally realize a savings in the host through a smaller configuration or through accepting additional work. The relative value of this savings in comparison to the cost of a back-end machine depends, of course, on the particular machines under consideration. Of perhaps even greater significance is the fact that the cost of the cpu is becoming a very small part of the total cost of a computer installation. Thus having two or more cpu's (host plus back-end) in a configuration should not have a significant impact on the total cost of a system.

Aside from the question of actual purchase or rental cost, there may be other disadvantages to a second (back-end) machine. For example, it is quite likely that the back-end machine will be manufactured by a vendor different from the host. This may duplicate all the problems associated with contracts, maintenance procedures, operator training, systems programming support, etc. The extent to which this could be a problem depends in part on the extent to which the back-end becomes an integral part of the hardware/software system offered by the host vendor. If it is a part of the total package and if it results in a simpler data base management function, then maintenance and system support may actually be simpler.

Disadvantage 2. Unbalanced resources. By dedicating a fixed part of one's computational power to a given function, one loses some flexibility in being able to balance the load as the functional requirements change. For example, the cpu of the back-end machine may turn out to be highly overloaded, while the host is mostly idle, or vice versa. This may be a permanent mismatch, or it could occur only at certain times during the day. With the two separate and different machines, there is no obvious way to balance the processor load as would automatically occur if a single or perhaps multiprocessor machine handled all of the functions.

Here again, as the cost of cpu's drops, the economic impact of an unbalanced system becomes less significant. Thus, if the back-end was underworked, it would not mean much of a dollar loss. If it was overloaded, upgrading to a faster (or a second) cpu would not be a significant part of the total budget.

While the back-end processor may be only a small part of the total installation cost, the secondary storage equipment (disk controllers, drivers, packs) attached to the back-end system will probably be a significant part of the total cost. A large imbalance in this area could thus be very expensive. One possible solution to such an imbalance would be to have the capability of switching (or sharing) some of the drives between the two systems, but this option may not be available. Alternatively, one could rent some extra drives for the overloaded machine and return some of the unused drives on the other machine. Since secondary storage requirements are normally one of the more predictable resources and since these requirements do not generally fluctuate rapidly with time, balancing by rental and return is probably adequate for most installations.

Disadvantage 3. Response time overhead. Satisfying a particular data base request in the conventional single machine configuration may require none, one, or a number of accesses to secondary storage. The desired data is then moved from a buffer to a working area where the application program can process it. With the back-end system, the request must first be transmitted to the small machine, and the secondary storage accesses must be executed, and then the data must be transferred back to the host machine. If one makes the assumption that the secondary storage activity (number of disk accesses and associated cpu time) will be about the same whether done in a single machine or with a back-end configuration, then the back-end approach is left with a response time overhead consisting of: (a) transmission time of the command to the back-end; (b) transmission time of the results back to the host; (c) task queueing delays related to these transmissions; and (d) possible conversion overheads associated with incompatible work lengths, character sets, data formats, etc.

The delay in response time due to items (a) and (b) can be made arbitrarily small if the band-width between the two machines is made sufficiently large. On the other hand, a broad-band link increases the cost of the back-end system. The question here is whether a suitable compromise can be struck which does not significantly increase response time and also does not add appreciably to the system cost.

Similar questions can be raised relating to items (c) and (d) which are difficult to address properly except in the context of a specific system.

Experimental Data Management System (XDMS)

Implementation Objectives

An analysis of the potential advantages and disadvantages of the back-end concept led to the conclusion

that the approach held promise. But were there unforeseen technical problems that would make such an approach economically unattractive? Could such a system be implemented in a reasonable time and with a reasonable expenditure of resources? What would be the thruput per dollar of such an approach? All of these questions were impossible to answer adequately without the availability of a working system. In November 1970, therefore, development of a prototype back-end system was started. This prototype system is called the eXperimental Data Management System, XDMS.

In addition to providing a tool for investigating the back-end concept, XDMS served a second important objective. One member of the development group was also chairman of the Data Base Task Group, DBTG, committee of the COference on Data System Languages, CODASYL. This committee had been established in 1966 to develop a language for data base management. In October 1969, the DBTG published its first proposal, which was subsequently revised and republished in April 1971. At the time the XDMS project was started, there was considerable skepticism about whether the DBTG language could be implemented efficiently. There was also a need for implementation feedback to the DBTG committee. Thus, the selection of the DBTG language for the XDMS project had the fortunate result of not only providing the back-end system with an advanced data management language but also providing the DBTG effort with valuable implementation feedback.

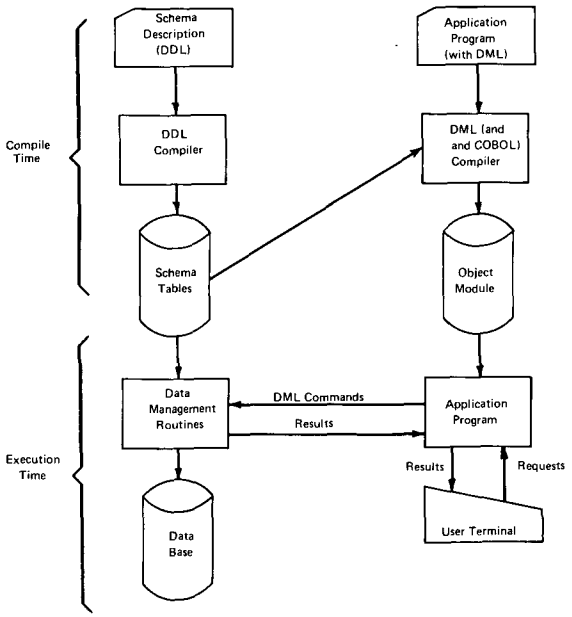
System Configuration

Selection of the DBTG language for the project led to a second important decision: the selection of the UNIVAC 1108 as the host computer. To understand why this occurred, one must look a little more closely at what is needed to implement a DBTG data management system. The approach taken by the DBTG was to separate the description of a data base from the programs which manipulate the data base and also from the actual data residing in the data base. Such a description of a data base is called a schema. The language used to describe a schema is called the Data Description Language, DDL. The DDL describes what fields are contained in a record, how records are interrelated, etc.

Actual access to the data base is provided through a second language, the Data Manipulation Language, DML. The version of DML appearing in the April 1971 Report consists of a set of 15 commands which, when added to COBOL, gives that language an extensive data base management capability. Other DML's can be designated for other general purpose languages such as PL/I, FORTRAN, etc. Examples of DML commands are FIND a record, MODIFY a record, ORDER a set of records, etc.

Thus, a typical DBTG system would have compilers for the DDL and DML languages in addition to the actual data management routines called by the application programs at execution time. Figure 5 shows how these components might interact. The DDL compiler accepts the schema description and produces the schema tables

Fig. 5. A conventional DBTG data management system configuration.



that are used by the data management routines at execution time to access the data base. The DML (and COBOL) compiler accepts application programs containing DML commands and produces object modules. Note that the schema tables are used by the DML compiler in addition to being used by the data management routines. Execution of a DML command in an application program results in a call to data management routines. These routines access the data base and return the results to the user work area. An application program might typically be associated with an on-line user terminal where the requests originated and where the results are returned.

The reason UNIVAC 1108 was selected as the host computer can now be described. UNIVAC, in parallel with the effort described here, was developing a DBTG system along the lines of the system shown in Figure 5 for its 1100 series machines [5]. Use of the UNIVAC DDL and DML compilers allowed the implementors of the XDMS project to apply all of their efforts to the development of the back-end system and to the investigation of the data base management techniques.

The choice of the Digital Scientific META-4 for the back-end was based on an entirely different set of criteria. Here an inexpensive machine with high input/output capacity was needed. It was also felt that a microprogrammable machine would allow development of an instruction set tailored to data base management. The META-4 is microprogrammable, and it has interleaved memory with I/O cycle-stealing, which allows the rapid transfer of large amounts of data to and from core memory. Another advantage is the availability on the META-4 of IBM 1130 software through emulation.

The configuration finally adopted is shown in Figure 6. The compile time facilities are basically those provided with the UNIVAC DBTG system. (The schema tables are augmented to include some additional DBTG features not yet handled by the UNIVAC DDL compiler.) The UNIVAC execution time system has been replaced by the XDMS interface, which controls the communication between the UNIVAC 1108 and META-4. Since schema tables information is needed in both the host and back-end, these tables are installed on the back-end also.

The chain of events that might occur in response to a data base request typed at a user terminal will now be traced. A request generated by a batch run is essentially the same. The request would first be passed to the appropriate application program for syntactic and semantic analysis. Any number of user request languages could be developed for the system by design of the suitable application programs. In particular, an interactive data base management language called DATABASIC, which allows on-line manipulation of a data base, was developed as part of XDMS.

Having interpreted the request, the application program would then issue the necessary DML command(s) to the XDMS interface program. The command would then be encoded and transmitted to the back-end system over the data link. The XDMS system in the back-end would interpret the DML command and would access the data base using schema table information. None, one, or perhaps a large number of calls to the data base in secondary storage might be necessary to satisfy the DML command. Having executed the command, the back-end would transmit the results back to the interface program in the host. These results would then be passed on to the application program for display at the user terminal as appropriate.

Fig. 6. XDMS data management system components.

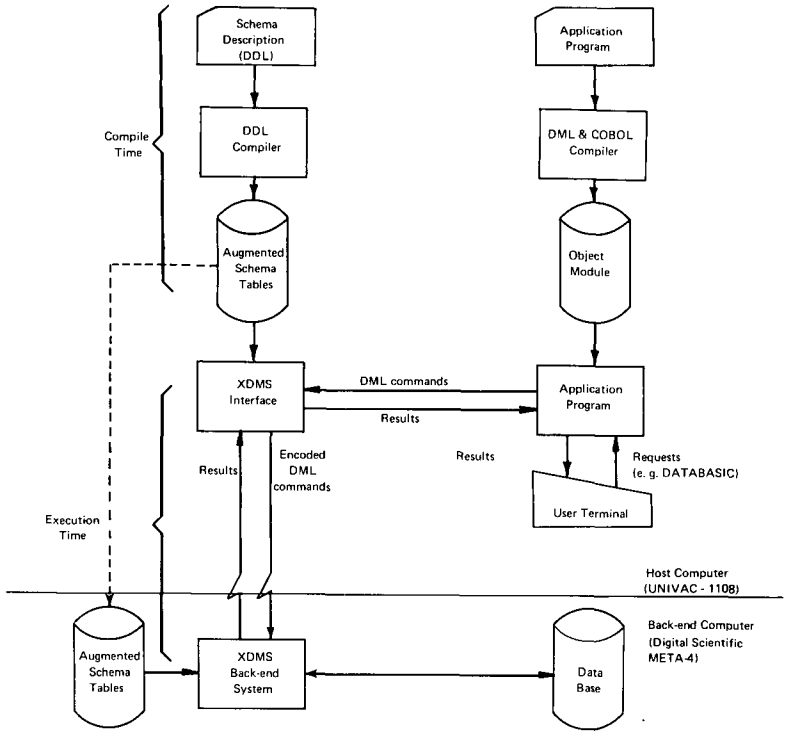
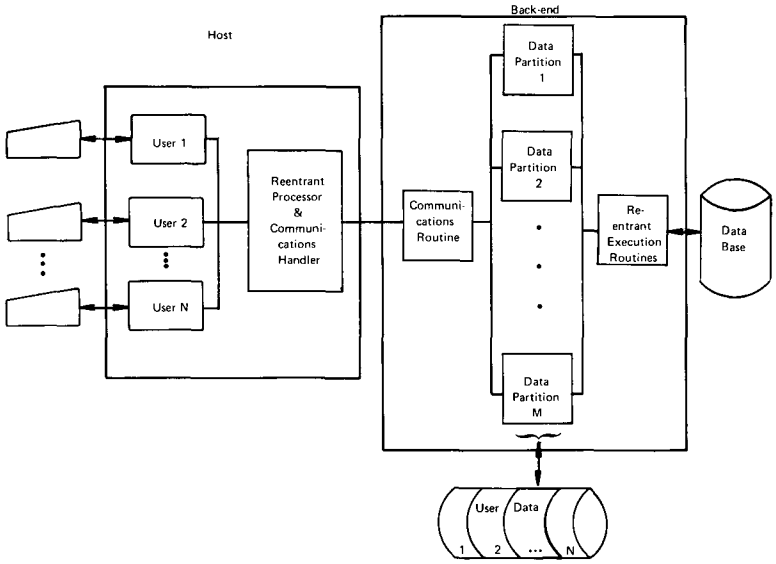


Fig. 7. XDMS multiuser capability.



System Capabilities

It is not the purpose of this paper to fully describe the XDMS system. Instead, three of the more salient features of the system will be discussed, in order to establish the fact that XDMS is more than an experimental toy. It is a working prototype which has the basic capabilities which one would expect in a full production system.

Capability 1. Multi-user system. XDMS was designed to handle many users simultaneously. In the host computer, this was accomplished by designing the XDMS interface so that commands can be collected from, and results distributed to, a number of application programs. This design took the form of a re-entrant processor and a communications handler as shown in Figure 7. In the back-end, a data partition area is provided in core for each active DML command and a user data area is provided on secondary storage containing status information about the user. The maximum number of users (N) and the maximum number of possible active DML commands (M) can be varied. They are currently set at 10 and 4 respectively.

Capability 2. Concurrent update. Allowing more than one user to simultaneously read from a data base is a fairly straightforward problem. Consider, however, the difficulties that arise as soon as more than one user can change the data base simultaneously. Will the data

base look consistent to a user who is reading and/or updating the same part of the data base that is undergoing change by another user? Even worse, will the data base be kept in a rational condition if two users simultaneously update the same section?

Various "locking" techniques have been proposed which allow users to update a data base in a multi-user system. The simplest locking technique is to suspend the access rights of all other users while a given user is changing the data base. This can be done for the whole data base or for some large portion of the data base (perhaps a DBTG area). The problem with this solution is that the response time for users that are locked out is degraded. In fact, if the updating load reaches a certain level, it is possible that some users may never get serviced.

To avoid this congestive situation, locking must be done on as small a portion of the data base as possible. XDMS locks on a physical record within a page. This adds some complexity to XDMS but helps to minimize congestion. The only lower level of locking that might be considered is the locking of fields within a record. Actually in XDMS a logical DBTG record is separated into two physical records (pointers and data) so locking on entities within a logical record is provided, which is considered to be the lowest level of locking that is reasonable.

Associated with the locking problem is the deadlock problem: user A has record 1 locked and is waiting for record 2 while user B has record 2 locked and is waiting for record 1. The XDMS system currently provides the capability for one user to back off once deadlock is detected.

Capability 3. Rollback and recovery. The dual audit capability described in the section on back-end advantages was implemented on XDMS. Actually rollback exists at two levels: command and transaction. Command rollback allows the effects of a given DML command to be erased if a problem is encountered in the back-end. Status is then returned to the host indicating that the command was not executed. Transaction rollback allows the effects of a series of DML commands (a transaction) to be erased. This might be initiated by a user at a terminal or by the host machine if problems are encountered.

Status

The XDMS System first achieved a limited operational capability in June 1972. Since that time there has been extensive upgrading and debugging. In general, XDMS has reached a point where (provided the link and disk capacity was improved) it could be used for a live application.

The cost of implementing XDMS has been about six man-years and about $60K in equipment. The back-end portion of the XDMS system occupies about 15K of 16-bit words, with the other 17K being used for input/output buffers. The XDMS interface in the host is about 4.5K of 36-bit words and about 1K of buffer space.

The capabilities of the XDMS system can best be described in terms of the DBTG language. Every significant feature of that language was considered in the design of the system. An extensive and useful subset of the language is currently operational. The only major DBTG capabilities which are not operational in the XDMS system are: (1) SUBSCHEMAS; (2) ORDER DML command; (3) ON conditions; (4) PRIVACY facilities; (5) special data items; and (6) MODE IS CHAIN **evaluation.**

Having briefly described the XDMS system, we will now focus our attention on what was learned from developing the system. In particular, we will analyze—based on XDMS experience—the projected advantages and disadvantages of the back-end approach that were covered in the first part of this paper.

Advantage 1. Economy through specialization. The core size of XDMS and the effort and money required to develop it have already been noted. Intuitively these numbers appear to be very good. Comparatively they also look good. A conventional DBTG system in about the same stage of development has been examined which implements the same language, and which has roughly equivalent capabilities. Its current core size is about twice the size of XDMS; it has been in development somewhat longer; and it appears to have taken more in manpower and computer costs to develop.

Development costs are only one economic factor to be considered. For a long life system or for one which will be used in a number of installations, the operational costs (machine rental, etc.) may far outweigh development and maintenance costs. In order to attack this problem, a trial data base (a Bell Labs internal personnel and organization file of about 1500 records) was loaded into XDMS and into the conventional system. A battery of eight different DML command streams representing various types of possible applications was also generated. The eight DML command streams were then run against both systems measuring core residency, cpu time, I/O time, etc. Next, costs were assigned to each of these resources, and an overall cost to perform the eight DML command streams on the conventional system and on the back-end XDMS system was determined. For simple commands (e.g. FIND NEXT) the costs were more or less equivalent, but for complex commands (e.g. STORE a record that is a member of several sorted sets) the back-end system was less expensive by one or two orders of magnitude.

Advantage 2. Shared data base. The sharing of data by transfer between XDMS systems has not been attempted, but it appears obvious at this point that there would be no difficulty in doing this. The simultaneous sharing of a data base by two different types of computers is the more interesting experiment. An XDMS interface for the IBM 370 was investigated and has been partially implemented. No real obstacles have been uncovered in providing access to the same data base concurrently from both the UNIVAC 1108 and IBM 370. However, final verification of this advantage is not complete.

Advantage 3. Data base protection. It has already been noted that we have implemented the double-audit approach as well as the command and transaction roll-back in the XDMS system. These capabilities have been exercised, and appear to offer a substantial contribution to maintaining data base integrity. The real proof of this advantage will come when a sizeable data base has been maintained in a live environment over a sustained period of time.

No facilities have as yet been implemented which take advantage of the back-end configuration to deny access to unauthorized users.

Advantage 4. Data base management for new machines. The first claim in this area is that data base management capability can be provided on a new machine much more quickly when using an existing back-end. The XDMS interface for the UNIVAC-1108 was designed and implemented by one member of the group in a few months. To this, we would have to add the cost of developing suitable DML and DDL compilers. This latter activity might be simplified by use of cross-compilers.

The second claim was that data base management could be provided on much smaller machines through the back-end approach. This contention is supported by the relatively small size of the UNIVAC 1108 XDMS interface (~6K words).

Disadvantage 1. Cost of a second machine. It was pointed out under Advantage 1 that the cost savings of using a back-end approach appear to be substantial for the cases tested. In these studies the back-end cost included both the XDMS interface in the host and in the back-end system. Thus, adding the second machine is a cost-effective step if the host can be applied to other productive work or if the host can be reconfigured into a smaller system with a corresponding savings which exceeds the back-end costs.

Disadvantage 2. Unbalanced resources. The XDMS implementation demonstrates that the cost of the back-end processor can be kept to a small fraction of the total installation cost. Thus an imbalance in the processor load on the two machines should not cause a serious economic loss. The problem of a disk capacity imbalance has not yet been attacked in XDMS.

Disadvantage 3. Response time overhead. The response time for a back-end system has four additional components not present in conventional system: (a) transmission time to send a command to the back-end; (b) transmission time to send the results to the host; (c) task queueing delays associated with the transmissions; and (d) conversion times due to incompatibilities (e.g. word sizes, character sets).

A typical DBTG command might require a total of 250 bytes to be transmitted to send the command over and to get the results back. For our 2000 BAUD link this takes about 1 sec. To this one must add another second for line turnarounds.

Use of a 50K BAUD link would cut the transmission time to .04 sec with about that much again for turnarounds. The additional delay of less than a tenth of a second would not be noticeable to a user waiting for the results at a terminal. It could, however, have some impact of core residency requirements in the host. This would of course depend on how core was allocated and how users were swapped.

The third item which will tend to increase the response time through a back-end system is task queueing for those tasks associated with the transmission. We have not developed meaningful measurements in this area.

The fourth component of back-end response time, conversion and formatting, amounts to approximately 5 msec in the host and another 5 msec in back-end. This is insignificant in comparison with the total response time, and can be ignored.

Conclusions

Final verification of the utility of the back-end concept must await its use in a production environment. Experimental results to date, however, support the conclusion that the back-end approach is an economically attractive alternative for data base management. Not only is there an apparent advantage in throughput per dollar, but there are also a number of new capabilities that such a configuration offers such as the simultaneous sharing of a common data base by different computers and increased security of the data base.

As a side benefit, the XDMS system has demonstrated that a data management system implementing the CODASYL DBTG language can be developed on a small machine (32K 16-bit words) in a relatively short time (18 months) with a relatively small expenditure of manpower (six man-years).

Acknowledgment. In addition to the authors, three former members of the group, Earl Jones, Tax Metaxides, and T.S. Shao, have also made substantial contributions to the project.

Received March 1974

References

1. COnference of DAta SYstems Languages (CODASYL) Data Base Task Group Report, ACM, New York, Oct. 1969 and Apr. 1971.
2. COnference of DAta SYstems Languages (CODASYL) Systems Committee, Feature analysis of generalized data base management systems, ACM, New York, May 1971.
3. Dodd, George G. Elements of data management systems, *Computing Surveys 1* (June 1969), 117–133.
4. Feinroth, Y., Franceschini, Y., and Goldstein, M., Telecommunications using a front-end minicomputer, *Comm. ACM 16*, 3 (Mar. 1973), 53–160
5. UNIVAC DMS 1100 Data Manipulation Language UP-7908
6. UNIVAC DMS 1100 Schema Definition UP-7907 Rev. 1
7. UNIVAC DMS 1100 System Support Functions UP-7909 Rev. 1