Distributed files for SAA

by R. A. Demers

Files are still a major way of storing data in computer systems, and they are a significant part of the information to be handled by the distributed processing networks that are developing. Systems Application Architecture is supporting distributed files. In this paper, the goals, benefits, and problems of providing this support are discussed, along with the role of Distributed Data Management architecture.

Tremendous volumes of data are stored in files. Making this great amount of data available to the applications and users of today's distributed processing networks is a major challenge faced by IBM's Systems Application Architecture (SAA). This paper addresses the goals, benefits, and problems faced by SAA in supporting distributed files, and the role of IBM's Distributed Data Management architecture in meeting this challenge.

The paper begins with a brief overview of the evolution of file systems, emphasizing the continuing trend of file systems to increase user productivity by providing higher levels of services. File systems are then contrasted with databases to clarify what these terms mean. The general concepts and benefits of distributed file systems are presented, followed by a discussion of SAA common programming interfaces for distributed files. The role of distributed files in cooperative processing, local area networks, and wide area networks is then considered. Five levels of distributed file processing are outlined to present a picture of where we have been, where we are, and where we are going with distributed files. The paper concludes with a brief discussion of the performance implications of distributed files.

The evolution of file systems

As operating systems have evolved, responsibilities for using and managing system resources have migrated from application programs to system components designed to simplify the job of programming. This is especially true for data storage and access. In the earliest days of computing, each application program that stored and accessed data on a tape or disk had to have its own logic for requesting services from the device, as shown in Part A of Figure 1. The programmer was required to know about device characteristics and interfaces and to expend considerable effort in managing the allocation of space on the device. Sharing a device with other programs was largely a matter of conventions of use and luck in avoiding conflicts.

To ease these burdens, operating systems provided a variety of *access method* services, as shown in Part B of Figure 1. The tasks of using, managing, and sharing devices became a system responsibility, allowing the application programmer to better concentrate on application requirements. As high-level programming languages became available, the direct use of system access methods was replaced by language statements, as shown in Part C of Figure 1. The result was a still-higher-level view of data management for application programmers that eliminated many of the concerns that remained with the use of

system access methods, such as interface conventions, error handling, and buffer management.

Each high-level language evolved its own file models, but there was considerable cross-fertilization among the languages, and a number of generally useful concepts gradually evolved. Among them are the concepts of records, of operations over a sequence of records, of keyed forms of access, and of file sharing on a record basis. And as new systems were developed that were required to support these languages, their access methods were designed to support these concepts and allow file sharing between programs written in different languages.

With the availability of these enhancements, it quickly became apparent that additional file services were needed by applications. As the number of files became large, a system facility for cataloguing and

Figure 1 Application programs and data management

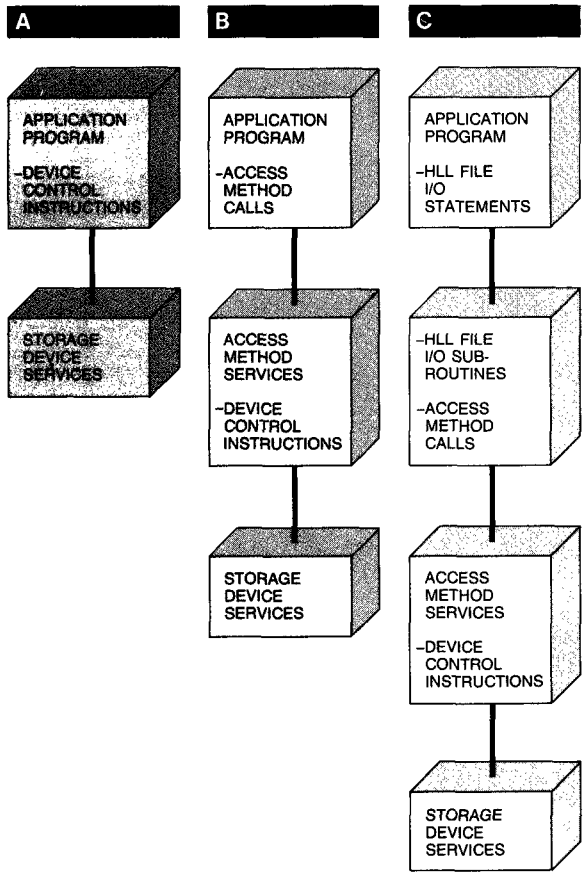
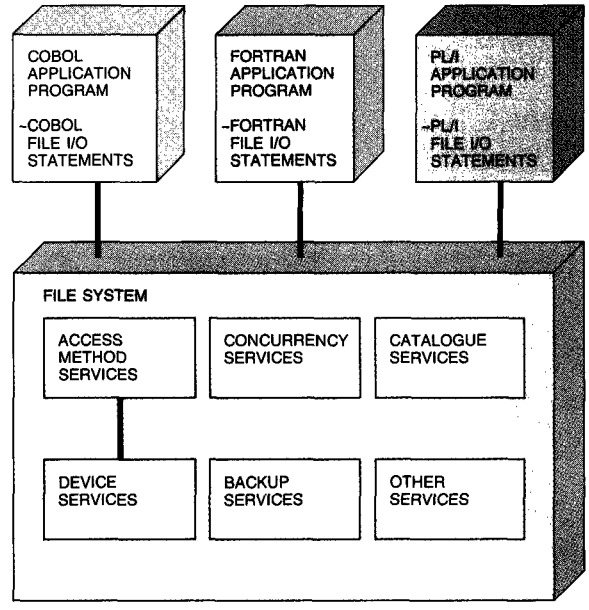


Figure 2 Programming languages and file systems



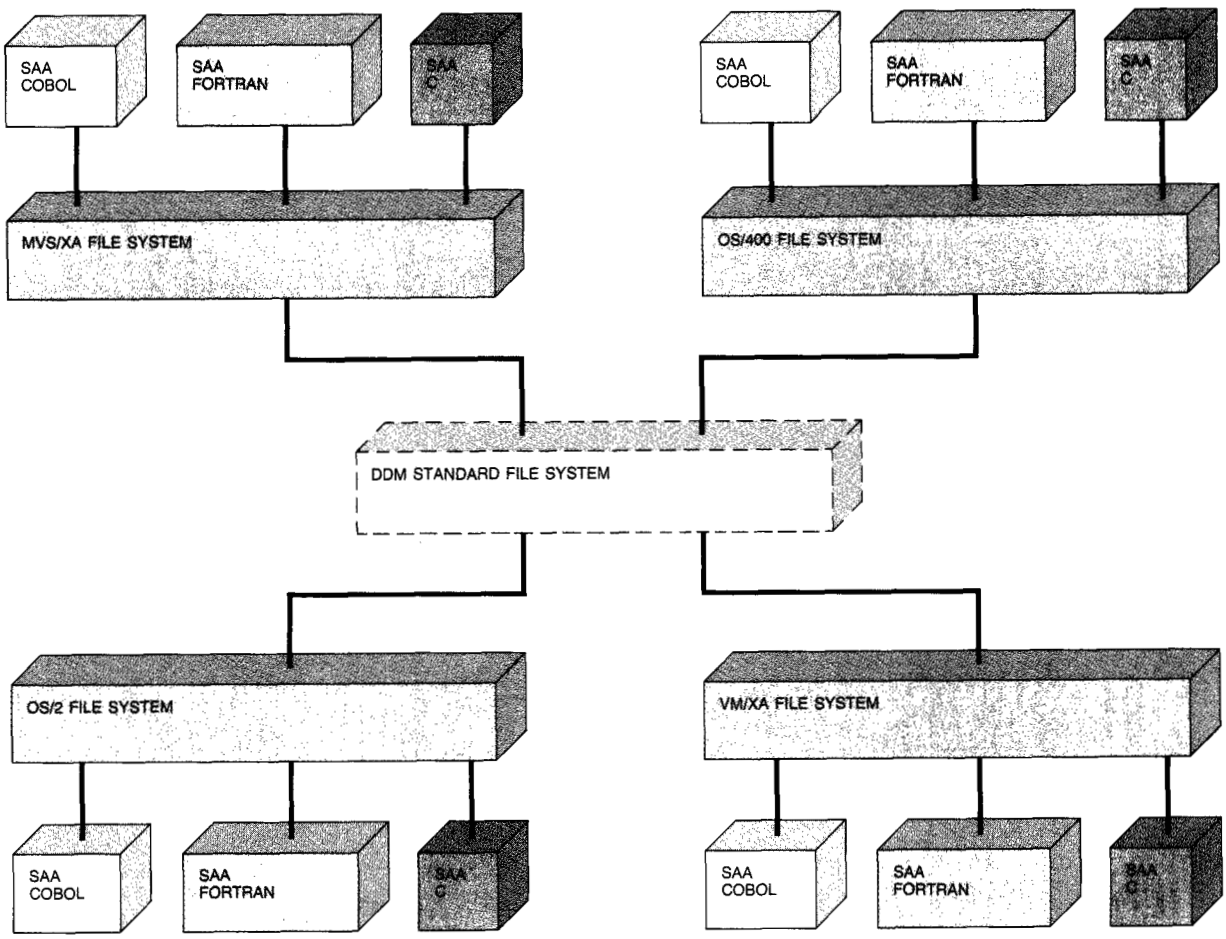
locating files became necessary, along with system facilities for securing them, for backing them up and restoring them for disaster recovery, for describing their records, and for keeping track of the programs that use them. Additional system facilities also became available in the form of generalized system utility programs for entering data into files, for sorting files, for copying files, and for writing reports.

These elaborate sets of system facilities for managing, accessing, and using files eventually were seen as *file systems*, as organized bodies of data managed and controlled by a comprehensive and tightly integrated component of the operating system. To a large extent, these file systems have evolved to meet the application development requirements associated with the use of high-level languages, as shown in Figure 2. The file systems are designed to support the file models of multiple programming languages.

Common programming interfaces for files

To enhance program portability between SAA systems, the file interfaces of each of the designated SAA programming languages (COBOL, FORTRAN, RPG III, and C) are being standardized. That is, all SAA COBOL programs have the same file input/output capabilities, but these are not necessarily the same as the file

Figure 3 SAA Common Programming Interfaces for files



capabilities found in SAA FORTRAN programs or SAA C programs. File models are included in the designs of these languages, and they differ in many ways for a variety of historical reasons.

The file systems of SAA systems must support all of these languages, as illustrated in Figure 3. The file models of each SAA programming language are standardized for program portability. The file systems of each SAA system must support the file models of all SAA languages. In general, the file systems provide one or more underlying access methods used by the language compilers to implement the file models of their language. These access methods can be viewed as lower-level file models, each providing a range of useful functions that the languages can use. The mapping from language file model to system file model is not necessarily a direct one. In some cases, multiple file system requests must be issued to support a single language request. In other cases, file system requests must be carefully parameterized to avoid side effects not acceptable to the language.

In SAA distributed file processing, IBM's Distributed Data Management architecture (DDM) defines a still lower level of file models, which is also illustrated by Figure 3. In order to access a remote file, the file system interfaces of the requesting system are trapped and converted to a sequence of DDM file model commands. At the remote system, the DDM commands are then converted to a sequence of requests to its file system. This process is illustrated in Figure 4. The common programming interfaces to files defined by each SAA language are mapped by their compilers to the local SAA system file interfaces. For access to remote files, these interfaces are then

mapped to the interfaces of the DDM standard file system. The mapping is not necessarily direct, but it does allow for a high degree of local/remote transparency. The logical distance from the language file models to the DDM file models determines the cost of performing interface conversions. For SAA systems this is generally not a problem, because of the standardization that has occurred in the design of both system file models and DDM file models.

File models

File systems support two different kinds of files: stream files and record files. Stream files provide only the minimum amount of support needed to access file data, but thereby allow programmers maximum flexibility in organizing and accessing the data. Examples of stream files are those of UNIX® and the IBM Personal Computer Disk Operating System (PC DOS). When these files are used, strings of bytes can be read or written according to their relative position within the file.

In contrast, record files attempt to provide a broad set of functions that are known to be useful in implementing the file models of high-level languages. Examples of these file systems are the Virtual Storage Access Method (VSAM) of the Multiple Virtual Storage/Extended Architecture (MVS/XA) operating system and the integrated file system of the OS/400™ operating system. These file systems directly support the view that data consist of fields that are organized into records stored in files. The following models of record-oriented files are generally supported as a base for high-level-language file models:

Sequential—A linearly organized set of records. These records can be accessed relative to one another (e.g., the record at the next file position), or randomly by position in the file (e.g., record number 57).

Direct—A linearly organized set of records in which there is an application-defined relationship between the contents of a record and its position in the file. These records can be accessed relative to one another (e.g., the record at the next file position), or randomly by position in the file (e.g., record number 57).

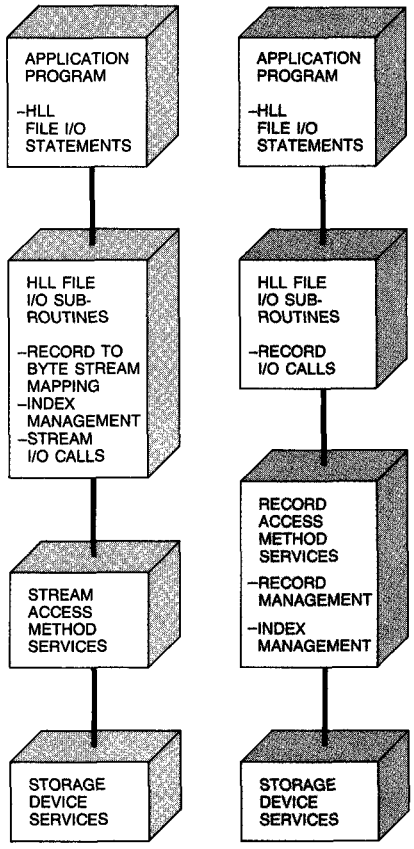
Keyed—A linearly organized set of records together with an index that supports efficient access to records by the values of their key fields. These records can be accessed

- Relative to one another by record position (e.g., the record at the next file position)
- Randomly by position in the file (e.g., record number 57)

Figure 4 Mapping SAA programming interfaces

MVS/XA JCL	MVS/XA FILE SYSTEM	DDM STANDARD FILE SYSTEM	OS/400 FILE SYSTEM
// DISP = (MOD)	—ALLOCATE THE FILE	DCLFIL	—ASSOCIATE A FILE HANDLE WITH A FILE NAME
		CRTSEQF	—CREATE THE FILE IF IT DOES NOT EXIST
		LCKFIL MODNONLK	—ALLOW NO OTHER USERS CONCURRENT ACCESS
SAA COBOL PROGRAM			
OPEN OUTPUT	—OPEN A QSAM FILE FOR OUTPUT	OPEN RELRNBAM	—OPEN THE FILE FOR SEQUENTIAL ACCESS
WRITE	—PUT A RECORD AT THE END OF THE FILE	INSRECEF	—WRITE A RECORD TO THE END OF THE FILE
CLOSE	—CLOSE THE FILE	CLOSE	—CLOSE THE FILE
		DELDCL	—DELETE THE FILE HANDLE
	—DEALLOCATE THE FILE	ULKFIL	—UNLOCK THE FILE

Figure 5 Stream versus record file models



- Relative to one another by key value sequence (e.g., the record with the next highest key value)
- Randomly by key value of the file (e.g., the record with key "DUCK")

Alternate Index—An index over a base sequential, direct, or keyed file that supports efficient access to records by the values of alternate key fields. These records can be accessed in the same ways as those of keyed files.

Stream files achieve maximum flexibility in storing, accessing, and managing data by supporting only the most primitive of functions. But programming costs are not avoided; they are only moved from the system domain to the application domain (as shown in Figure 5), where there is far less opportunity for standardization and system support. For high-level-language programs, the compilers must each take on the burden of implementing their file models on top of stream files.

From this discussion, it should not be construed that either the stream-oriented or the record-oriented approach is necessarily superior. Trade-offs are made between flexibility and system services in designing file systems, but each model is better suited for certain types of applications. The stream model is more appropriate for complexly structured data (such as documents and executable programs), whereas the record-oriented models are more appropriate for traditional business data processing.

Because file systems have generally not supported both file models, this distinction has been lost and is a source of problems today. Application and system programs mask the differences between the file models that are available and the file models they require. As we progress into a discussion of distributed file systems, this problem becomes even more significant, since programs developed for stream access may be required to access data stored in record files, or vice versa.

File systems versus databases

The term *database* is often loosely used to describe advanced file systems, but this terminology is inaccurate and causes much confusion. To understand why SAA includes support for both files and databases, we must first distinguish between them.

As file systems evolved and became more elaborate, the general concepts of managing data for integrity, recovery, security, and access flexibility became understood, along with the technology for achieving these objectives. The resulting data management systems were seen to be something qualitatively different and better than existing file systems.

File systems can be enhanced to support these requirements, as seen in the integrated file/database support of the OS/400, which evolved from the file system of the IBM System/38. But in most systems separate database support was added. For example, the MVS/XA operating system provides extensive file services with its Virtual Storage Access Method, but the designers of Database 2 (DB2) chose to start fresh to obtain the performance and functional characteristics their customers requested. Similar considerations led to the development of Structured Query Language/Data System (SQL/DS) for the Virtual Machine/Extended Architecture (VM/XA) systems and for the Operating System/2™ Extended Edition (OS/2™ EE) database.

Database technology provides many features of great value to some applications. Many of IBM's largest customers base their most critical applications on Information Management System (IMS), a hierarchical database system, and increasingly on DB2, a relational database system. But most businesses also contain significant amounts of data for which database capabilities are simply not needed. Examples of such data are programs (both source and object), private note files, documents, images, on-line conference forums, and other forms of data lacking the regularity of structure that underlies most database designs. Simpler file services are sufficient for their storage and access.

Perhaps most importantly, enormous numbers of applications exist that were designed to use file system interfaces. Migrating programs to use database interfaces and moving their data from files to a database can be expensive. Customers have adopted a rational approach based on perceived value, using database systems for key enterprise data, writing new applications that use database interfaces, and migrating existing applications and data when it makes sense to do so.

In short, file systems are not being superseded by databases. They remain useful and economical for many applications and will be supported by operating systems for the foreseeable future. As operating systems continue to evolve toward more comprehensive services, their file systems must evolve with them. Today, this evolution is in the direction of services distributed over the systems of telecommunications networks.

Distributed files

Telecommunications facilities have been used for many years to transmit data between computer systems. These data have generally consisted of display data streams, electronic mail, documents, whole files, and the transactions of distributed applications. More recently, support for distributed files has also added to telecommunications traffic.

But why are distributed file services of value in application design? This section examines some of the benefits of distributed files.

Availability. Data that cannot be accessed are of no value to a user. Networks of connected systems have grown in size and complexity while data and users have become distributed throughout the network. But users have had difficulties in accessing the data they need. The costs of developing specialized communications programs to read or update files stored in remote systems is simply too high for most applications and prohibitive for infrequent or casual queries. Distributed file systems reduce these costs essentially to the costs of using communications facilities.

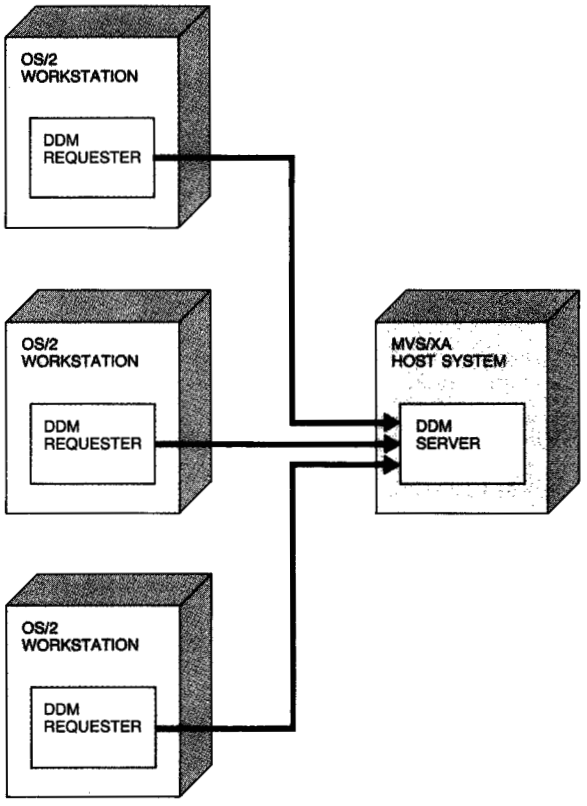
Timeliness. Obtaining current data has often been a problem for applications. Copies, or snapshots, of whole files have only a limited "shelf life," and the use of expired data can produce erroneous results. The use of distributed file systems can eliminate these problems. The data available to anyone in the network are as current as the applications that update the data can provide.

Sharing. A major component of availability is file sharing. Files locked or allocated to single users for long periods of time are not available to other users. A well-designed file system attempts to maximize concurrency of access to its files by its users. This concept must be extended to distributed file systems and applied to both local and remote users of the file.

Security. In opposition to requirements for availability is the simple fact that not everyone should be able to access all of the data available to distributed file systems. Two components make up network data security—user authentication and user authorization. Systems Network Architecture (SNA) Logical Unit 6.2 (LU 6.2) provides a high degree of assurance that the user of a file system is who he/she is supposed to be. Each file system must then provide assurances that the users of a particular file are authorized to access it in the ways they are attempting. This assurance is a responsibility of well-designed file systems, for both local and remote users. Making data available through a distributed file system allows the owner of the data to control who can access the data. In contrast, when copies of files are manually or electronically distributed, it is the owner of the copy who controls its accessibility. In this case, control shared is usually control lost.

Storage management. With the development of personal computers came small direct-access storage devices. The availability of 10 or 20 megabytes of on-line storage greatly enhanced the operability and convenience of personal computers, but some significant problems were introduced at the same time. On the one hand, some storage capacity is wasted, at least for a while. It takes time for most users to fill

Figure 6 DDM file servers in cooperative processing



up that much storage. But, on the other hand, 10 or 20 megabytes is not very much storage for some applications. The result is that some users have insufficient capacity, whereas others have unused, and unsharable, capacity. Distributed file systems alleviate these problems by providing enormous storage capacity relative to the needs of any one user, along with the ability to share that capacity among many users.

A distributed file system can also provide some important additional services to its users. One of these is performing periodic backups to other media. No system or device is 100 percent reliable over an indefinite period of time. Prudent users make backups of their data for quick recovery in case of failure, but it is a nuisance to do so, and most users prefer to have it done for them.

Another service consists of automatically staging data onto the storage devices with the lowest costs relative to frequency of use. For example, the Hierarchical Storage Manager (HSM) component of MVS/XA automatically moves data between different levels of storage with different costs. When MVS/XA is used as a distributed file server by the IBM Enhanced Connectivity Facility Virtual File component (ECF VFILE), the HSM support is provided without the remote VFILE users being aware of it.

Program portability. One of the objectives of SAA is to allow programs written in one of the SAA-designated languages to be easily moved from one SAA system to another SAA system. But it is not enough to move just the programs. Either the files they access must be moved with the programs, or it must be possible to access the files where they currently reside. The distributed file support of SAA systems makes it possible to move either the files or the programs.

Distributed processing environments

SAA support for distributed files is recognition of the advances that have been made in this area. This section briefly discusses three distributed processing environments and discusses where they lead us.

Cooperative processing. The concept of cooperative processing arose because personal computer users wanted to use the resources of the host systems to which they were attached. A key requirement is access to host data storage and to host files. IBM developed the ECF product for cooperative processing with MVS/XA and VM/XA hosts, and the AS/400™ PC Support product for cooperative processing with OS/400 hosts.

The ECF product supports access to host files through its VFILE component. This facility emulates the PC DOS file system on the host system with good transparency to personal computer applications. When personal computer data is stored on the host, VFILE simply maps the personal computer stream file onto a set of fixed-length host file records. Files can be transferred between the personal computer and the host, or bytes can be read from or written to specified positions of the mapped stream file. VFILE also allows personal computer programs to access existing record-oriented files as stream files. In this case, however, VFILE must convert data between personal computer and host system representations on the basis of user-supplied descriptions of the host data.

AS/400 PC Support provides fully transparent personal computer file system emulation on AS/400 hosts, including hierarchical directories and file sharing, as defined by PC DOS. It is not possible to access data

stored in host files with AS/400 PC Support, but a companion product called DDM/PC provides that function. A key difference between ECF VFILE and DDM/PC is that DDM/PC introduces a record-oriented programming interface as an extension to the PC DOS stream file interface and eliminates the need for mapping stream interfaces to record-oriented interfaces. AS/400 PC Support implements the stream file and directory subsets of DDM architecture for communications with AS/400 hosts. DDM/PC implements the record-oriented file models of DDM architecture for communications with all SAA systems.

An environment consisting of SAA OS/2 workstations and an SAA host system, such as an AS/400 system, a VM/XA system, or an MVS/XA system, is illustrated in Figure 6. In this environment, requests for data flow from the workstation requester to the host system server, and data are returned.

Local area networks. Local area networks (LANs) provide high-speed data communications between a limited number of systems within a small geographical area such as a single building or a college campus. They allow a fundamentally different approach to system design. Instead of a central host system controlling and providing services for workstations, a LAN allows a collection of workstations to interact with one another as peers. They can request services from one another and share resources such as processor time, printers, and on-line storage. When one of these peers is specialized to provide services for a single resource, it is referred to as a "server" for that resource.

If all workstations attached to a LAN are of the same type, it is not too difficult to design messages and protocols for server access. Examples of IBM homogeneous LAN servers are the PC Network and the RT PC™ Distributed File System.[2] Increasingly, however, it is desirable to attach different types of systems to a LAN. One example of a heterogeneous LAN is the Andrew[3] system developed by Carnegie Mellon University. To obtain the desired network transparency for end users, requests for LAN services are converted to and from a message format designed by the Andrew architects. For files, this was possible because only a single model of files was supported on the LAN, the stream model popularized by UNIX and PC DOS.

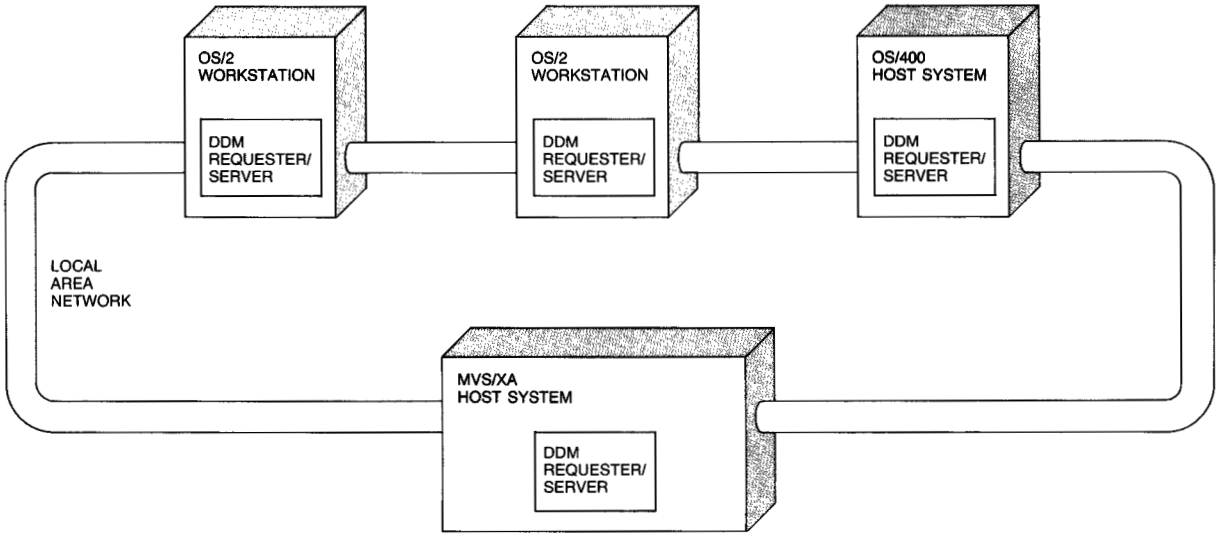
In this discussion, all LAN systems have been referred to as *workstations*, but actually host systems are also being attached to LANs, especially to act as servers. The files distributed among the workstations of a LAN may not be available when needed, and proper backup and recovery processing can be awkward. Attaching host systems, such as the AS/400 system or an IBM 9370 to LANs allows them to provide services in a more disciplined way. It also makes their enormous volumes of data available to the workstation users. But then we again run into the problems of different file models and different data representations.

Any system can be a requester and any system can be a provider of file services.

The SAA solution to these problems is through the use of the SAA cross-systems architectures with DDM for file services. As an example of what becomes possible, IBM OS/2 systems, attached as workstations to a LAN, can access files stored in AS/400, MVS/XA, or OS/2 systems as file servers, as shown in Figure 7. In this environment, any system can request file services from any other system supporting DDM file services attached to the LAN.

Wide area networks. The preceding discussions of cooperative processing and local area networks focused on the attachment of workstations to file servers, but, in a larger sense, any system can be a requester and any system can be a provider of file services. In Figure 8, the accounting department and the shipping department may each have their own system for local applications, but they also need to interact with each other, with their workstations, and with headquarters systems. Whether these systems are geographically distributed or in the same machine room makes no difference to the applications being run. Note that multiple DDM file servers can exist in the same system and that connectivity with systems outside of the enterprise may also be required.

Figure 7 DDM requesters and servers attached to a local area network



Levels of file distribution

In this section, five levels of file distribution are discussed. The discussion parallels similar levels of database distribution discussed by Reinsch.[1]

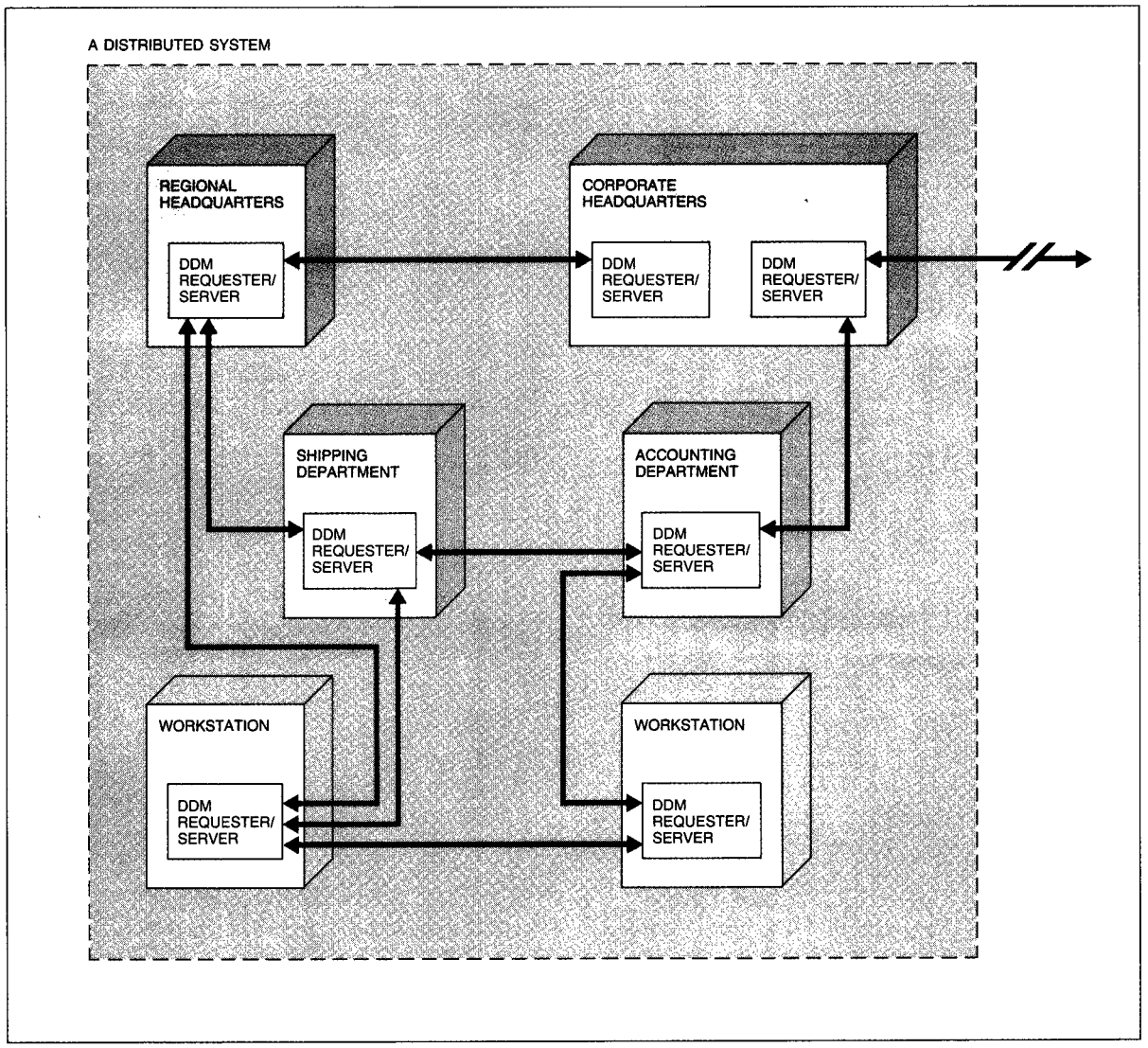
User-assisted file distribution. From the very beginnings of the computer industry, it has been necessary to transfer files from one system to another. When tapes were the primary medium for storing files, it was a simple matter to copy a tape and manually transport it to wherever the file was needed. But even then, there were problems with reading files written to tape by other types of systems. Aside from formatting and labeling differences, it took specially written application programs to read and convert the data from the representations of the originating system to those of the reading system. Early tape files contained virtually no meta-data, that is, data that described the contents of the file or provided information about the file. At best, this information accompanied the tape on paper, but more typically it was embedded in the programs written to process the file. Thus, data conversions could not be handled by file system utilities.

Atomic requests. An atomic request is one that is performed completely before any information is returned to the requester, with no intermediate state information remaining in the file system to be used by subsequent requests. Examples of atomic requests are operations on whole files, such as requests to rename a file, to list its attributes, to make a copy of a file, or to lock a file.

Requests to transfer files are also atomic. Transferring files between systems was one of the most obvious, and therefore earliest, uses of data communications. Numerous *file transfer* programs have been marketed, with varying capabilities and varying abilities to transfer files between different types of systems. Among the currently available IBM offerings are the File Transfer Program and the Bulk Data Transfer program for MVS/XA and VM/XA, the File Transfer Subroutines of System/36, the Object Distribution Facility of System/38, and numerous programs for the IBM PC and Personal System/2® (PS/2®) systems. Although these programs work well in transferring files between like (or similar) systems, they were not designed to work with one another. As a result, transferring files between a System/36 and MVS/XA, for example, is not possible with these programs.

DDM architecture supports file transfers between SAA systems over SNA LU 6.2 communications. As with other aspects of DDM, SAA systems attempt to provide file transfer as a transparent capability of their existing utilities for copying files within the system. For

Figure 8 DDM file servers in a wide area network



example, the OS/400 Copy File command can be used to copy a file to or from a remote system with the convenience of copying a file from one library to another library on the same system.

Remote file processing. In the Remote File level of file distribution, applications executing in one system can access the contents of a remote file. At this point, however, major difficulties arise, leading to the following questions:

- Is the file system of the requester functionally compatible with that of the file server?
- Do they support the same file models?
- Can their interfaces be mapped to and from a common message format?

When the requester and the file server are of the same system type and operating system type, the answer is clearly *yes*, and distributed file services are easily implemented. A good example of homogene-

ous distributed file services is the Distributed File Services (DFS) component of the Advanced Interactive Executive (AIX™) operating system of the RT PC. In this case, an RT PC with large storage capacity is used as a file server for other, smaller RT PCs on a LAN. Any message format convenient to the developers of DFS was acceptable.

But in the case of mixed system types and mixed operating systems—the heterogeneous systems case—the picture is quite different. The file systems of an IBM PC running PC DOS and an IBM System/370 running MVS/XA bear little resemblance to each other, yet it was possible for the Enhanced Connectivity Facility (ECF) product to bridge these differences and make MVS/XA data available to PC DOS applications. It should be noted, however, that components of the ECF product were added to both PC DOS and MVS/XA. That is, the designers of ECF were responsible for both ends of the communications line and were able to design messages and protocols that were tailored to that pair of system types.

The DDM file system architecture consists of a set of standardized file models within a standardized file system.

The situation becomes even more complex when the commitment is made to provide distributed file services among any of N different system types, as IBM has done with its SAA systems. At this point, we enter the realm of cross-systems architecture. The cost factor for developing and testing system-pair solutions is N^2, and is prohibitive for even small values of N. The alternative is to connect each system type to a standard file system and reduce the cost factor to $2N$.

This approach was selected for DDM, as shown in Figure 9. The DDM file system architecture consists of a set of standardized file models that exist within a standardized file system. They are *standardized* in the sense that representatives from a wide variety of IBM systems participated in their design, seeking out the file models and features that would best match the needs of the high-level languages available on all of their systems. In the figure, it is shown that data management requests for remote file services are trapped below local data management file system interfaces and converted into messages acceptable to the standard DDM file models.

The File Transfer, Access, and Management (FTAM) protocol of the Open Systems Interconnection (OSI) architecture[4] also defines a standard file model, but there is a major design difference between DDM and FTAM. DDM defines a set of file models that are each tailored to the specific requirements of a well-understood high-level-language file model. FTAM defines a single hierarchical file model that can be constrained in various ways to meet specific access requirements, but not with transparent access for existing applications. IBM has committed to supporting FTAM for connectivity with other OSI vendors, but it has adopted DDM into the framework of SAA because of its emphasis on cross-systems transparency. Since DDM is itself a published architecture, DDM requesters and servers can also be implemented by other vendors.

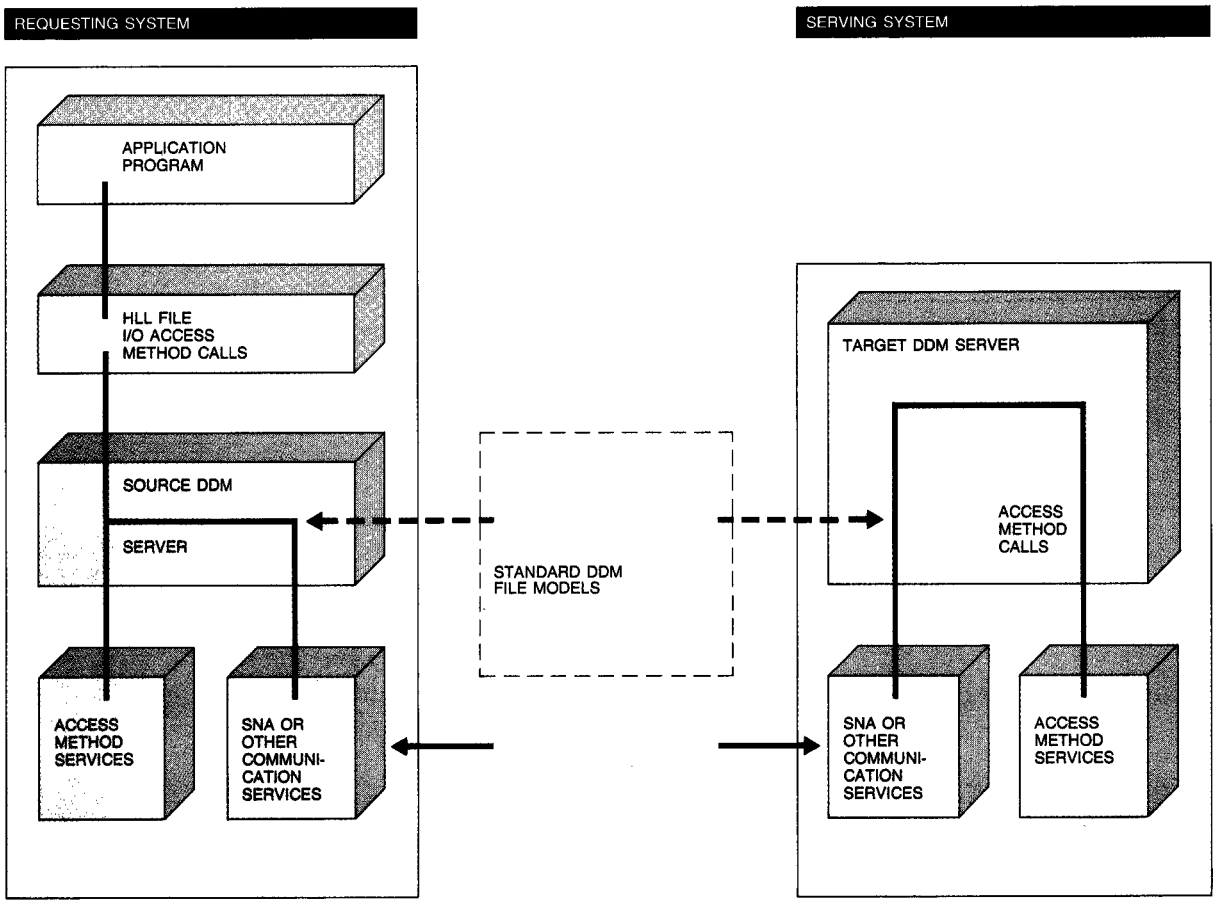
To date, support for distributed file services based on DDM architecture has been provided by the following IBM systems:

- System/36 (requester and server)
- System/38 (requester and server)
- MVS/XA Customer Information Control System (CICS) (server only)
- PC DOS (requester only)
- NetView/PC™ (file upload only)
- OS/400 (requester and server)

As with other SAA cross-systems architectures, support for DDM is planned for all SAA systems, including OS/2, MVS/XA, and VM/XA.

Distributed file transactions. At the Remote File Processing level, changes to the contents of files occur when an individual request completes. No consideration is given to coordinated changes to multiple files or to recovery should there be a failure. This level of support is adequate for applications that perform file updates individually or that have only elementary recovery requirements. It is generally not adequate for *transaction processing* applications,

Figure 9 DDM processing overview



where coordinated sequences of updates must be performed on one or more resources. These applications are often interactive and require prompt and automatic recovery from failures.

In recognition of this, transaction-processing features have been implemented on several IBM file systems, including the Customer Information Control System, the System/38, the OS/400, and the 8100 Distributed Processing Program Executive (DPPX). In these systems, file processing takes place within a Unit of Work (UOW). A UOW begins automatically at the beginning of a job or at the end of the previous UOW. A UOW ends at an explicit request to *commit* or *rollback* the UOW, or at the end of a job. During a UOW, all requests to update recoverable resources are performed but are not permanent. They become permanent only at the completion of a COMMIT request. If there is any failure, they are automatically backed out. They can also be rolled back by issuing an explicit ROLLBACK request. It is important to note that a UOW covers all recoverable resources that are affected by a transaction. If an application is updating both recoverable files and a database, commit and rollback operations affect both of them.

When access to recoverable files is distributed, UOW processing must also be distributed. For SAA systems, this distribution will be accomplished through the use of SNA LU 6.2 synchronization point services, which provides a two-phase protocol for committing transactions.

Distributed file request processing. A possible direction for the functional growth of file systems can be seen in OS/400. Two capabilities of this file system allow application programs to interact with multiple files as if a single file were being accessed:

- An OS/400 Logical File provides an alternate, *logical* view of the data in an existing base file. Logical files, however, can be constructed over multiple base files. A request to access the records of a logical file causes the file system to access records from the underlying base files. These records are presented to the application one at a time, as if it were accessing a single file with records of varying length and format.

  The order in which records are presented depends on how the logical file was designed. One way is to specify that all of the records, from all of the base files, are to be in ascending or descending sequence in terms of common key fields in all of the records. In this way, a hierarchical relationship between the records of different base files can be established, with *master* records preceding *detail* records, for example. Another way is to specify that all of the records from one base file are to precede all of the records from the next base file. In this way, a large set of records can be partitioned into a number of smaller files.

- An OS/400 Join File is like a logical file in that it is defined over multiple base files. In join files, however, the application sees only records of a single format consisting of one copy of the key fields and the unique fields from each base record.

A challenge for distributed processing is to extend these concepts across the full range of SAA file systems, allowing logical and join files to be defined over base files residing on remote systems, with full local/remote transparency. When this occurs, the file system will have to access files on multiple remote systems in order to acquire or update records. In other words, the processing of the data management request will be distributed to multiple systems.

Beyond these specific examples, support for files replicated on multiple systems or partitioned across multiple systems will also require the distribution of the processing of requests for file services.

Performance

The performance of a distributed file system is a difficult issue because of the number of variables over which the developers of the file system have no control. Among them are the following:

- The performance costs associated with the translation of interfaces to and from the message formats of a distributed file system
- The performance of the communications facilities used
- The performance of the remote file system
- The performance of the storage devices used for a given file

These factors can have either a positive or a negative impact on the performance perceived by an application, but of equal importance is how the distributed file system is used by applications in specific network environments. The performance of an application that queries or updates occasional records in a remote file accessible only over a slow (e.g., 2400 baud) communications line can be acceptable. In contrast, the performance of an application that sorts large volumes of long records in a remote file accessible over a high-speed (e.g., 1.5 megabytes per second) local area network can be unacceptable.

This situation is really no different from what one sees in the use of locally attached direct-access storage devices (DASD) for data access. Data stored in memory can always be accessed faster, but with proper buffering and caching, and with data transmitted to or from the DASD in reasonably large blocks, performance can be acceptable for many applications. Also, where possible, it is desirable to offload complex functions to the DASD or its controller such that only final results are returned to the requester.

In the design of a distributed file system, the same concepts apply. Although specific network environments and applications cannot be known to the developers of a distributed file system, the same basic principles apply: buffer the data, keep the communications overhead to a minimum by using simple messages and protocols, off-load complex functions to the remote file system, and depend on application designers and users to use file capabilities in a reasonable way.

Under the conditions often found with personal computers, including relatively slow local DASD, high-speed LAN communications, and fast remote processors and DASD, a distributed file server can even offer performance advantages for some applications.

Conclusion

SAA allows users, programs, and data to be distributed throughout a network of interconnected IBM systems. Providing distributed access to files is a key

element of this strategy. DDM is the SAA heterogeneous architecture that provides a universal solution to distributed file processing across the range of SAA systems.

Cited references

1. R. Reinsch, "Distributed database for SAA," *IBM Systems Journal* **27**, No. 3, 362–369 (1988, this issue).
2. C. Sauer, D. Johnson, L. Loucks, A. Shaheen-Gouda, and T. Smith, "RT PC distributed services," *Operating Systems Review* **21**, No. 3, 18–29 (July 1987).
3. J. Morris, M. Satyanarayanan, M. Conner, J. Howard, D. Rosenthal, and F. Smith, *Andrew: A Distributed Personal Computing Environment*, Information Technology Center, Carnegie Mellon University, Pittsburgh (November 1985).
4. *Information Processing Systems—Open Systems Interconnection—File Transfer Access and Management—Part 1—General Introduction*, International Standards Organization document DIS 8571/1, Geneva (1986).

General references

IBM Distributed Data Management Architecture: General Information, GC21-9527, IBM Corporation (1986); available through IBM branch offices.

IBM Distributed Data Management Architecture: Implementation Planner's Guide, GC21-9528, IBM Corporation (1986); available through IBM branch offices.

IBM Distributed Data Management Architecture: Implementation Programmer's Guide, SC21-9529, IBM Corporation (1986); available through IBM branch offices.

IBM Distributed Data Management Architecture: Reference, SC21-9526, IBM Corporation (1986); available through IBM branch offices.

Richard A. Demers *IBM Application Business Systems, Highway 53 & NW 37th Street, Rochester, Minnesota 55901.* Mr. Demers is an advisory programmer working in Distributed Data Management (DDM) architecture. He received a B.A. in philosophy from Canisius College, Buffalo, New York, in 1968, and joined IBM as an applications programmer in White Plains, New York. In 1972, he moved to Endicott, New York, where he worked on OS/VS1, OS/VS2, and DOS/VS support for the IBM 3895 Optical Check Reader. Mr. Demers moved to the Rochester laboratory in 1975 to design the message handling and service components of the Control Program Facilities of the IBM System/38. Since 1982 he has been working on the development of DDM architecture, for which he received an IBM Outstanding Innovation Award in 1987. Mr. Demers is a member of the Association for Computing Machinery. His professional interests include operating systems, distributed processing, data management, general systems theory, programming languages, and object-oriented programming.

Reprint Order No. G321-5330.