SAA distributed processing

by A. L. Scherr

Discussed are motivations for distributed versus centralized data processing, the relative advantages of each, and the trade-offs involved as they relate to Systems Application Architecture (SAA). Presented is a taxonomy of the various approaches to designing applications to operate in a distributed manner. SAA support for these modes is described. The management of an enterprise-wide network of systems is discussed.

SAA will provide important new capabilities for *distributed data processing*, which is defined as the implementation of a related set of programs and data across two or more data processing centers or nodes. Generally, each of these data processing nodes is fully capable of storing data, executing application programs, and interfacing with user workstations. As we shall show, distributed processing provides new degrees of freedom for the designer of application programs. This flexibility allows for additional trade-offs to be made to optimize performance and cost as well as to satisfy organizational goals and other data processing requirements. The functionality of SAA and its program and data portability characteristics all contribute to being able to capitalize on these new degrees of freedom.

This paper is organized into four major sections. The first describes the motivations for distributed versus centralized data processing. The relative advantages of each and the trade-offs are described. The second section presents a taxonomy of the various approaches to designing applications to operate in a distributed manner. In the third section, the underlying SAA support for these modes is described. Finally, the facilities needed to manage a widely dispersed enterprise-wide network of systems are discussed. The broad conclusion of this paper is that distributed processing provides valuable additional degrees of freedom for the designer of application systems. On the other hand, it also can add to the complexity of these systems. The intention of distributed processing support in SAA is to maximize the utility of this additional flexibility and minimize the additional complexities seen by the end users, the system and application programmers, the network designers, and the operational and administrative staff.

Distributed versus centralized processing trade-offs

Perhaps the most pervasive motivations for distributed processing derive from geographical considerations. The fact that data processing users are geographically dispersed introduces the need to make trade-offs between communication costs and computing costs. This computing versus communication cost trade-off can occur at two levels: the economic cost for the hardware and software facilities, and the delays incurred for the communication.

In the 1960s, this trade-off was virtually always made in the direction of centralizing computing to achieve economy of scale. Because computers were relatively expensive, the communication costs necessary to provide users access to the centralized equipment were considered to be a reasonable price to pay for the advantages of such centralization.

Although it is difficult to predict relative cost and price trends in these areas, the additional delays associated with communications—both in the communication facilities themselves and in the various data processing elements needed to send and receive messages—are significant factors that can be predicted and dealt with. Thus, for example, placing computing capability close to the end users to avoid communications delays is clearly a way of providing better responsiveness for highly interactive applications.

For simple applications, the price/performance of small machines can be significantly better than that of larger machines.

Another source for distributed processing is the building of bridges between two or more existing applications and their data, where these applications run on geographically or architecturally disjoint equipment. A common example of this might be the result of a merger of two companies where, for instance, a consolidated personnel application is created.

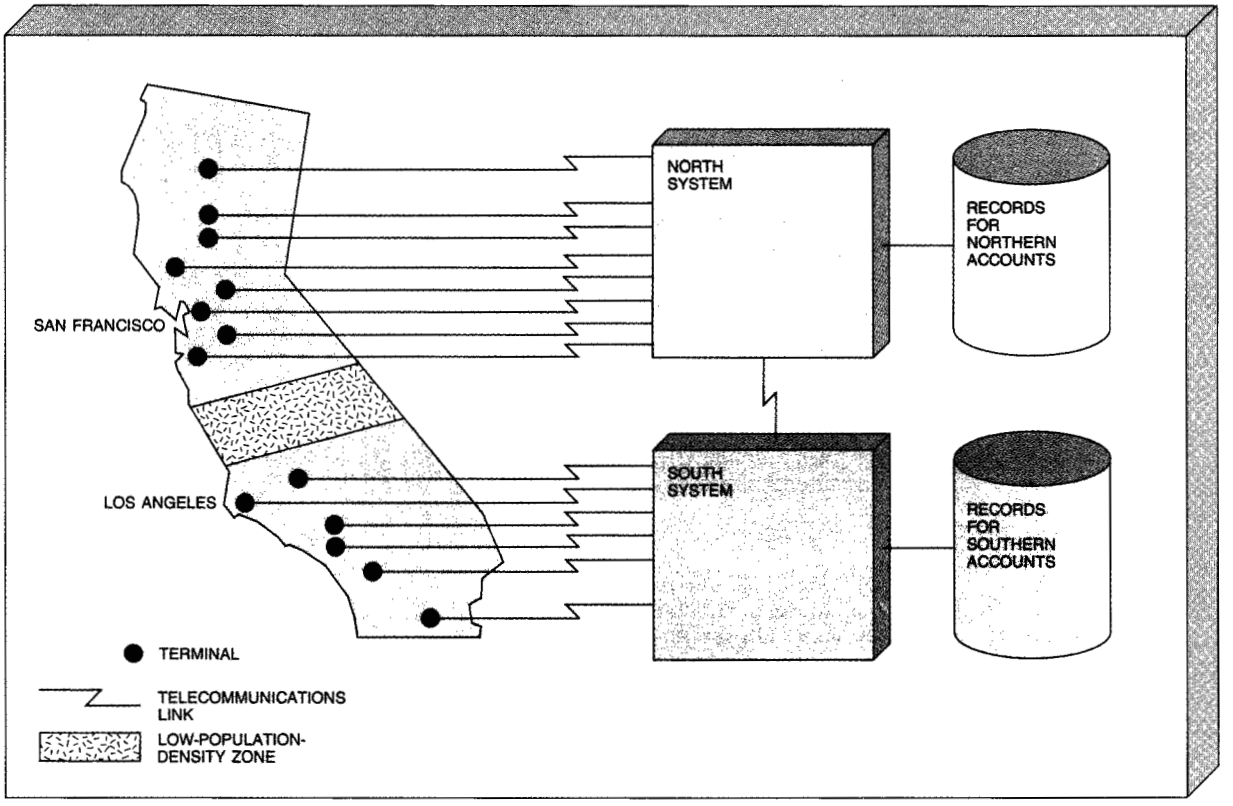
The desire to exploit special characteristics of particular data processing hardware is another motivator. For instance, for relatively simple applications, the price/performance of small machines can be significantly better than that of larger, more complex machines. Therefore, it is possible to see cases where applications that have been implemented across a network of small machines achieve outstanding price/performance, even though the application structure might have been simpler on a large, centralized machine. Another argument often seen in this area is the relative simplicity of smaller machines. Whereas this may be true only in the eye of the beholder and may be a difference that is diminishing, software for smaller machines is usually held to be easier to use and less complex to maintain than that which supports large systems. Finally, there are unique functions associated with different processors—such as special floating-point hardware, array processors, or the highly interactive capability of personal computers—that may be required by particular applications. Therefore, the execution of such applications must occur on particular hardware. This then can result in the kind of multinode program execution that is associated with distributed processing.

The requirement to achieve high availability for end users often leads to some form of distributed processing. For example, by partitioning the workload across the branch offices of a business and using an individual system in each branch, the scope of any given failure can often be limited to a single branch. Thus the overall availability seen by the end users may be better than a configuration in which every branch is served through a communications network by a single, centralized system. Often this approach is augmented by using both branch office systems and a centralized system accessed through a network. In this way, a failure of the communication lines or the central system can be counteracted by having the front-end system assume some level of the application function. On the other hand, a local system that fails can be bypassed and its functions performed by the central system. Ultimately, the ability to configure front-end/back-end and/or horizontal partitions of the workload in these ways and actually achieve higher availability with reasonable performance is a strong function of the geographical organization of the enterprise, as well as the particular nature of the applications being implemented.

The State of California is an example in which the geography is ideal for this type of split, because the major population centers, San Francisco and Los Angeles, are separated by a large zone with relatively low population density. Figure 1 shows how a credit authorization might be implemented on two nodes. It is relatively easy to balance the load between the two nodes, and the probability of communications between them is quite low. In contrast, in a single-center geographical region such as the New York metropolitan area, this approach would not be practical. This example also demonstrates another characteristic of distributed processing: The feasibility of an application configuration is often a function of special-case situations.

A related reason for distributed processing is to achieve higher levels of physical security. Large enterprises will often set up two or more data centers

Figure 1 Partitioned credit authorization system



as a way of protecting against natural disasters, power failures, sabotage, etc. Whereas the capacity created may be more than is needed for normal operations, the excess usually is not as much as double the base requirement. By identifying critical applications and providing redundant capacity for them alone, savings can be realized. All other applications would then be split between the two or more data centers. Achieving this type of physical security thus requires distributed data processing facilities as a means for exploiting the additional capacity as well as for making backup data available in the event of an emergency.

The next motivation is the real and/or perceived limitations of capacity in the data processing complex of an organization. The capacity needed for many applications is a function of an enterprise's volume of business. In some cases, the upper capacity limit of the largest available centralized data processing system is insufficient. Then, in order to accommodate increased business volumes, the applications must be split so that two or more systems can be used to share the load.

A similar situation occurs when the size of the organization providing data processing facilities within an enterprise is seen to have passed the point of diminishing returns with respect to its efficiency and effectiveness. There have been cases where a new organization with its own data processing installations has been created because management declined to add additional workload to an existing organization. In this case it was management's judgment that the existing organization would be unable to effectively manage the additional complexity and workload. Thus, data processing operations have often become decentralized as a way of creating a more manageable operation.

We have just presented one example of the many possible scenarios leading to the decentralization of

data processing. The management style and philosophy of a corporation in its approach to organizing the relationship of data processing management and line management have sometimes led to having line management directly control the data processing necessary for their operation. In some organizations, departments have acquired their own computing resources outside the normal data processing establishment. In these cases, the arrangement of the various systems in the network usually takes on the form of the organization. Another related organizational consideration is historical; because of events such as mergers and acquisitions, different styles of applications, different operating systems, and different hardware architectures must be interconnected. In all these cases, the need to tie together separate application programs and data leads to distributed processing.

There are considerations to counter each of the points that have just been made; however, these counterarguments can be interpreted only in the context of the perception and judgment of the people making the decisions. In most cases, there really are two sides to the story, and, depending on the exact situation, one factor may be judged more important than the other. Thus, economies of scale in both hardware and organization have been used for years to justify centralized data processing. These arguments are still valid and will remain so until either the hardware capacity or the organization size reaches a point of diminishing returns. To add to the complexity, many of the cost factors may be intangible or may involve long-term versus short-term considerations.

The single point of control that is available in a centralized data processing system for operations, security, systems programming, and applications programming is an advantage that is hard to argue with. Most companies that do distributed processing generally continue to require some degree of centralized capability in these areas.

Another advantage of centralized processing is that it inherently has fewer degrees of freedom and is therefore simpler, because where there are fewer decisions to make, there are fewer decisions to unmake. For instance, it is unnecessary to choose where to place applications and their data. In the distributed case, data and applications are placed in a network as a consequence of the motivations described earlier. The values of the parameters will change over time as a result of adding new applications, changing usage patterns, inaccurate predictions, changes in philosophy, and so on. At this point, the network may need to be restructured and rebalanced, resulting in applications and/or data having to be moved to new nodes. In the past, this has required reprogramming and/or data conversion. With the realization of SAA, the cost of changing the location of data and programs will be significantly less, and tuning of this type will be more practical.

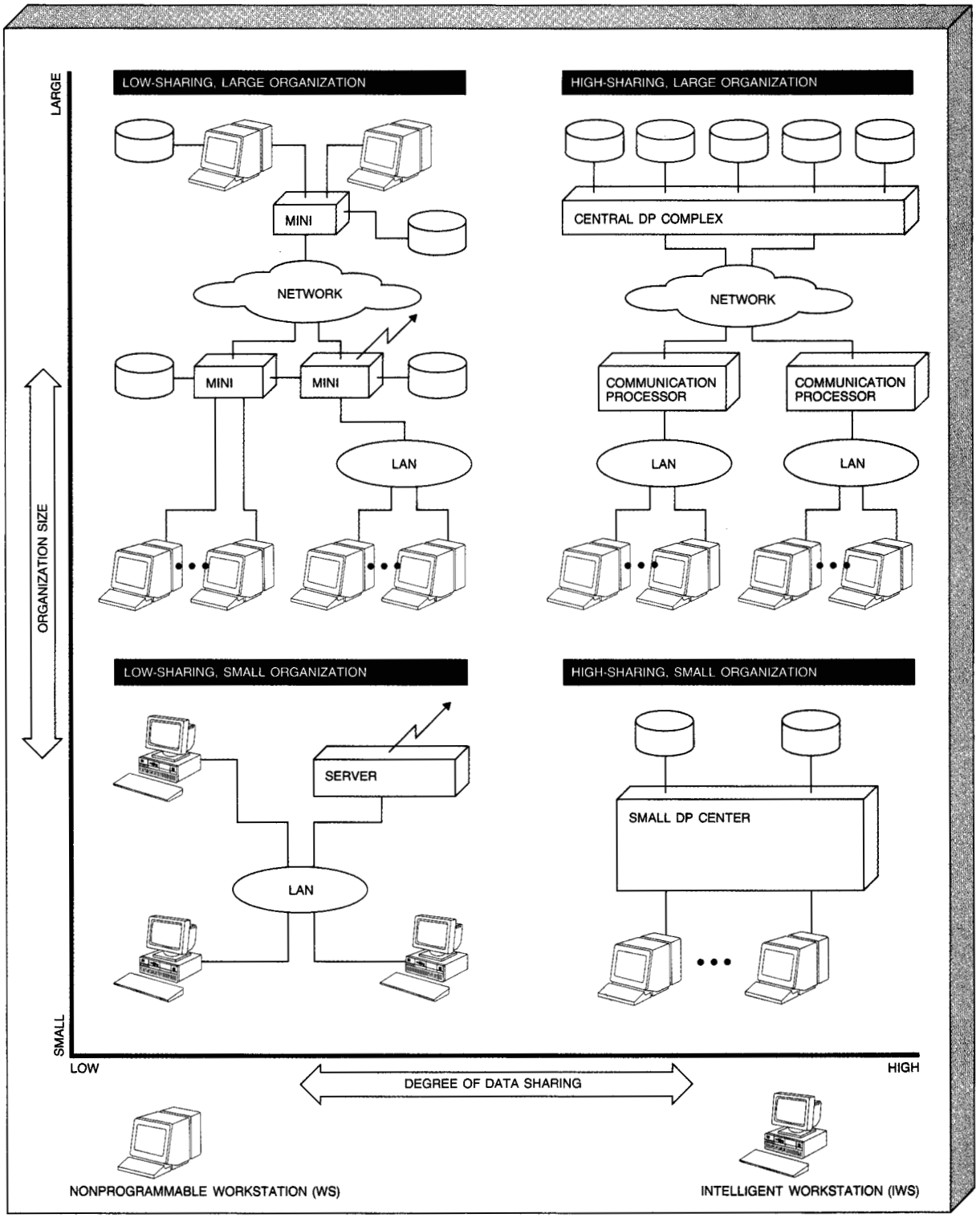
Table 1 Motivations for distributed processing

Geography
Computing versus communications trade-off between costs and delays
Responsiveness of the user interface
Bridges between existing applications
Characteristics of processors
Exploiting price and performance differences
Simplicity of small machines
Unique function
Higher system availability for end users
Scope of failures limited by partitioning the workload
Impact of failures reduced by redundancy in the front end and the back end
Physical security
Protection against natural disasters, power failures, sabotage, etc.
Security provisions
Single-system capacity limitations
Single computer with adequate capacity not available
Operational complexity of large centralized system
Cost of migration to larger system and/or new operating systems
Organizational considerations
Size of centralized data processing organization
Management style and philosophy
Historical growth
Mergers and acquisitions

Table 2 Motivations for centralized processing

Economies of scale
Hardware
Organization
Single point of control
Operations
Security
Systems and applications programming
Simplicity
Fewer decisions to make and unmake
Tuning
Shared-data applications
Only feasible solution when data must be current and frequently updated
Acessibility
Systems, data, and programs all equally accessible from anywhere in the network

Figure 2 Variety of distributed processing configurations



Finally, because centralized systems are equally accessible to all users, the centralized system is often the only feasible way to implement applications that are broadly used. Widespread usage of a shared database with frequent updates where the data must be kept current demands a single centrally available copy of each data element. Multiple copies of the data in cases like this are usually impractical.

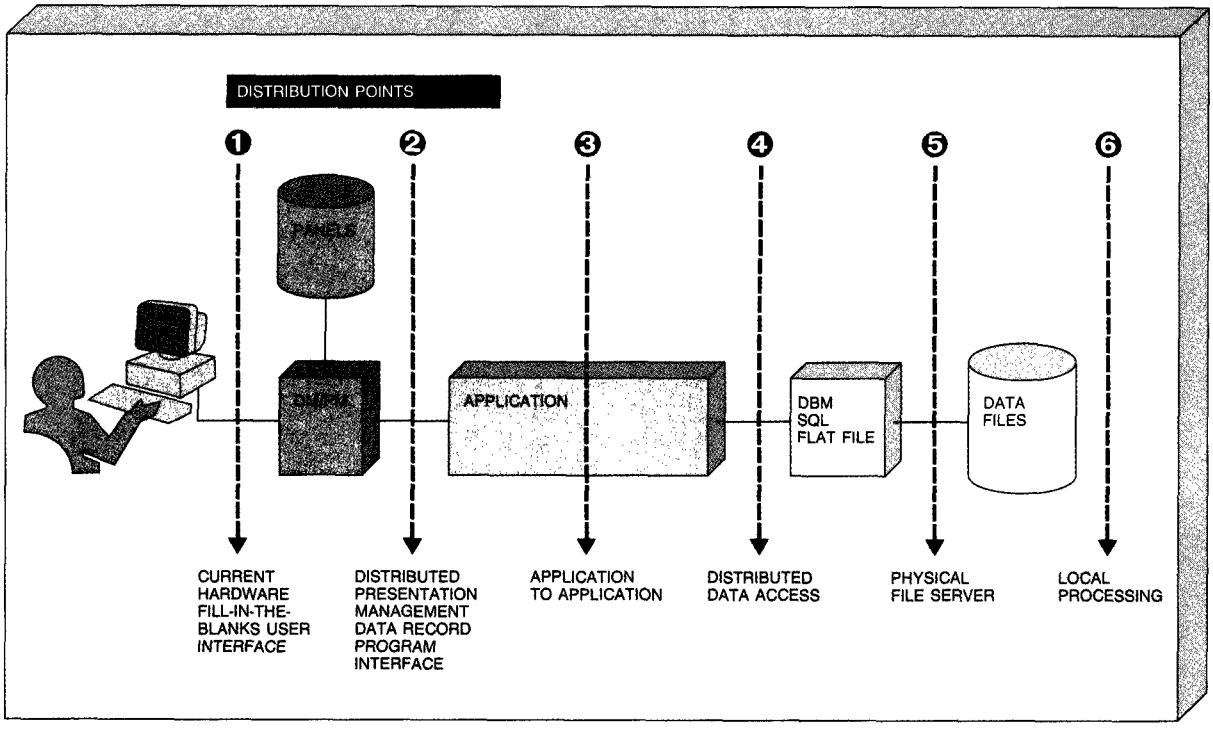
Tables 1 and 2 summarize the factors described. In real situations, these motivations are traded off against one another and it is rare indeed to see a large enterprise doing its data processing in a single organization and in a single complex. The most typical configurations seen involve the interconnection of not only peer systems but also small, intermediate, and large systems in the same complex. Figure 2 shows four possible variations of the essentially infinite combinations available as a function of the degrees to which data sharing is done and the size of the organization. Where data sharing is less prevalent, smaller, more dispersed systems are practical. Thus, for small organizations with little data sharing, a local area network connecting personal computer workstations is sufficient.

Distributed application design

In order to fully explore the facilities of SAA for Distributed Processing, this section discusses the general question of application design in a distributed environment. Perhaps the simplest way to look at the question is to consider the fact that in most cases the network of systems already exists; the question can then be asked: Where should the data and the associated application programs be placed in this network? In other cases, the data are already present, and the question becomes one of where to put the new application program. In either case, the simplified program-data structure shown in Figure 3 contains adequate information to allow most of the points and trade-offs to be discussed.

Figure 3 shows simplified application flow starting with the user at a workstation going through a Dialog Manager/Presentation Manager (DM/PM) facility that has access to stored descriptions of display panels, menus, help information, input forms with field definitions, and so forth. Next is the application program itself, which in turn goes to the database manager, SQL and/or flat file support, which finally

Figure 3 Distributed processing



connects to physical data storage media. For the purposes of simplicity, assume that all I/O devices are either for data storage (e.g., archival tape) and appear on the right, or are "source/sink" devices (e.g., printers) and appear on the left.

There are several points along this flow at which a network could be inserted. The question then becomes: Where in this flow can a communications network be inserted, and what are the consequences?

A full-screen, fill-in-the blanks user interface has been successful for a vast number of applications.

All the modes of distributed processing will be derived this way. The first point of distribution (Figure 3, distribution point 1), between the user's workstation (e.g., keyboard and display) and the DM/PM layer, is the classic distribution point for mainframe-interactive (MFI) display terminals. This interface, as embodied in the IBM 3270 and 5250 display products, has been engineered so that the required communication bandwidth and interrupt rates are as low as possible. Intelligence in either the display head or the controller immediately behind it provides for the ability to do simple editing of input, cursor movement, field definition, and so forth, so that it is possible for the user to enter a full screen of data before an interrupt and transmission to the upstream DM/PM and application program is necessary. This style of interface is characterized as a full-screen, fill-in-the blanks user interface, and it has been successful for a vast number of applications. This interface is less suitable for operations that require subsecond interactions with the user, such as graphic design and what-you-see-is-what-you-get (WYSIWYG) text editors. Although putting a network into this point in the flow is normally not considered to be distributed processing, it is included as an example because it is familiar and illustrates the trade-off consideration.

Adding a network at distribution point 2, between the application and the DM/PM facility, has a similar result. The advantage of distributing at this point is that the application program is unaffected and the end user can be served with improved responsiveness and usability because the DM/PM support is closer to the workstation. The usual implementation of this approach would be to locate the DM/PM programming in an intelligent workstation. In a typical situation, the application program requests that the user make a choice from a menu. The menu and its associated help screens are contained on a disk at the workstation, and the interactions leading to the selection are performed in the workstation. Finally, when the user makes a choice, it is transmitted across the network and back to the remote application. In this way, the user working with a remote host application sees a degree of interactivity approaching that of a local intelligent workstation application. As in the previous distribution point, this distribution is practical if the bandwidth and rate of interaction with the application are relatively low. On the other hand, if the application were a graphics design tool, the interrupt rate and bandwidth required would make this distribution point impractical. It must be emphasized that these trade-offs are based on the application interface, not the user interface. The user interface to fill in a data form may be highly interactive at the keystroke level, whereas the application program simply sees a record with data fields returned after these interactions with the DM/PM component are completed.

If an application is highly interactive, and at the same time requires accessing of data that are remote from the workstation, a straightforward way to achieve good performance is to split the application at some point so that the interactive part of the program runs near the user and the data access part of the application runs in the node with the data. It obviously requires sophistication to structure the application program in this way. Some luck is also required, because not every program can be so split. Usually, a "groove" in the application can be found and the application broken at this point. The characteristics of a groove are that once the program is split at this point, the bandwidth and rate of communication are low compared to those found in other places considered for the split. Obviously, the absolute rates must be low enough to allow for acceptable response time and communications cost.

A simpler solution to this situation is to run the application program in the node closest to the user to achieve the interactivity required, but to access the data record by record across the network. This

approach is practical only if the frequency of remote data use is low. If a significant number of records are accessed, the fact that each record accessed costs a substantial number of additional computing cycles and encounters significantly higher delays for queuing on resources and for communication can make the responsiveness of the application unacceptable. Cases like this may require copying the entire data file, copying the only portion that is required for the application, or possibly an alternative application design.

For SAA, distribution points 2, 3, and 4 of Figure 3 represent the key choices available. Figure 4 shows a different way to characterize each of the modes of distributed data processing. Four modes are shown:

In a well-designed distributed system, the local data processing mode is usually prevalent.

Local data processing, distributed dialog manager/presentation manager, distributed program-to-program, and distributed data access. The first mode of distributed processing is *local processing*, where the application and data are in the same node, which is either the user's intelligent workstation itself or the node closest to the user's workstation. In a well-designed distributed system, the local data processing mode is usually prevalent, because it offers the best performance and lowest cost. Another reason for including it is to make the point that all of these modes typically are intermixed in any realistic environment. SAA provides for this breadth of operation.

In the second mode, called *distributed DM/PM*, the application program and data are in a node remote from the user, and the dialog manager/presentation manager (DM/PM) is in the intelligent workstation. In addition to the functions previously discussed, facilities in the workstation may be acting to route communications on a long-term basis (i.e., a session from log-on to log-off), or the duration may be for the length of a transaction, where each transaction is routed to a remote application or to a local transaction as a function of the transaction code or one of its parameters.

The third mode is called *distributed program-to-program*, and is used when either the data required for the application are spread across two or more nodes, or the application program is both highly interactive and heavily uses remote data.

In the fourth mode, called *distributed data access*, the application is remote from its data. Typically this mode is used when most of the application data are in the same node as the program and a smaller amount of the data are remote. Distributed data access may be the only solution when the highly interactive application heavily uses remote data and cannot be split as in the previous mode.

In any given application, the above modes could be used in combination to handle operations in more than two nodes. Figure 5 summarizes the decision process for determining the structure of an application program. The underlying assumption is that the data are in one place. It is a straightforward extension of the logic to handle multicite data.

General considerations for data and application placement

SAA provides facilities to support each of the modes just described. Choosing the appropriate approach requires an analysis of the usage and traffic in the proposed system. Most systems grow and apply function and usage changes. SAA's inherent portability structures for data and programs allow for more feasible reconfiguration of these elements so as to facilitate reoptimizing traffic patterns and responsiveness. As an example, consider a three-stage application growth scenario in which the third stage establishes a relationship between the first two. The first stage is a horizontally distributed system to support on-line bank tellers. The second stage involves the use of a separate system to do credit card authorization. The third stage is the connection of the preceding two systems to accomplish electronic funds transfer.

Figure 6 shows the configuration for the on-line bank teller application. The approach selected for this implementation is a horizontal distribution, where the terminals for branches are connected to a partic-

Figure 4 Ways of characterizing the modes of distributed data processing

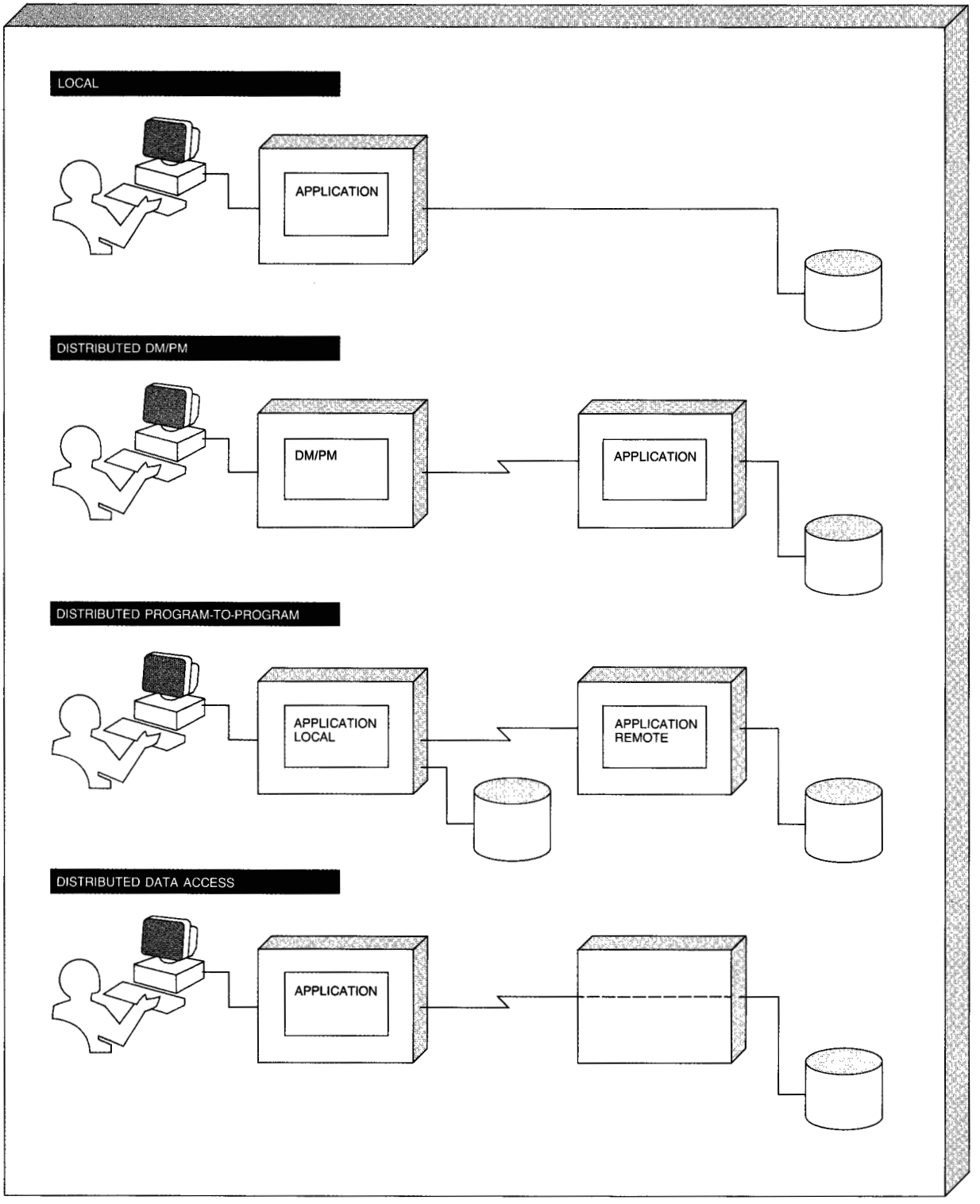


Figure 5 Roadmap of the decision process for determining the structure of an application program

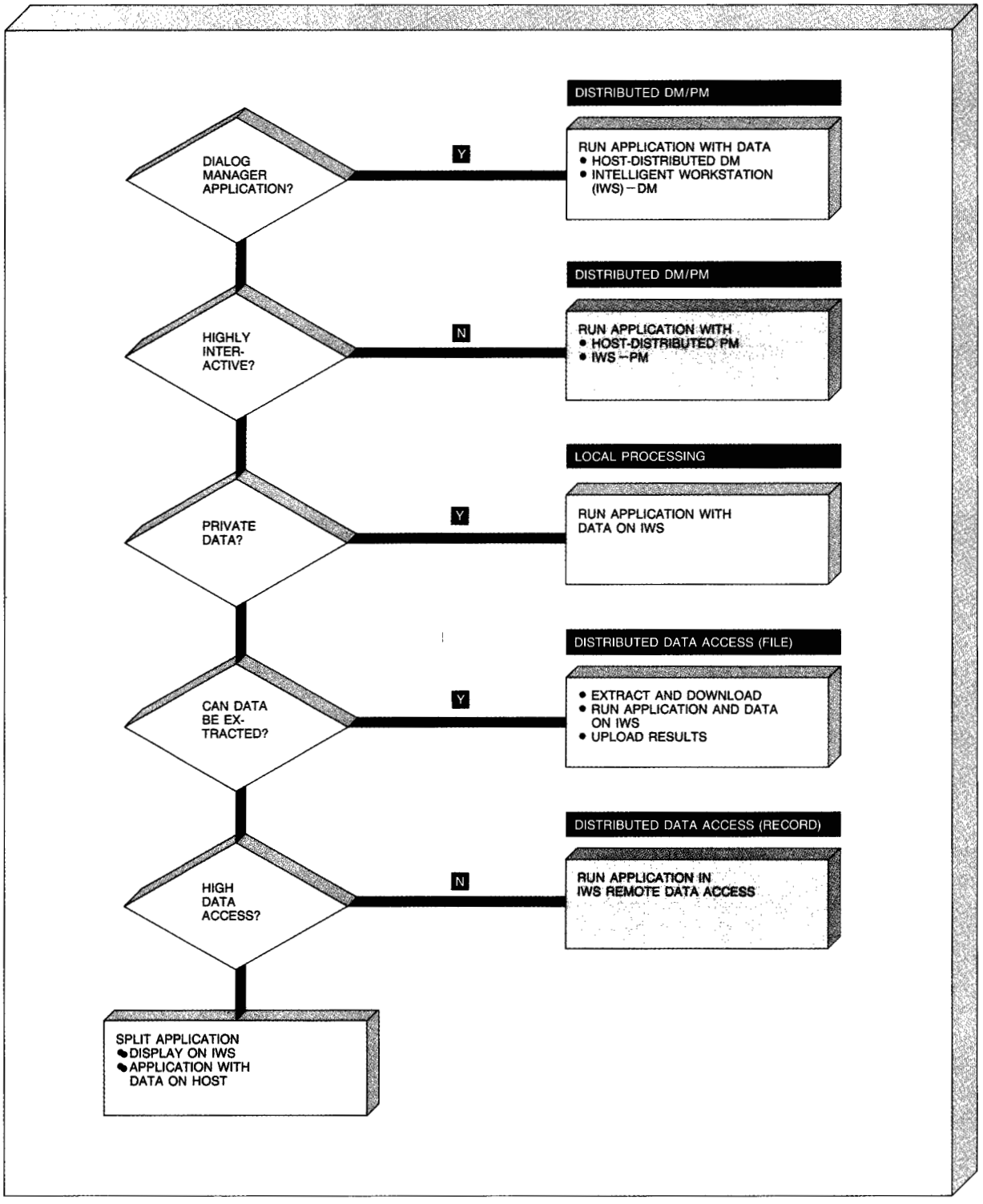
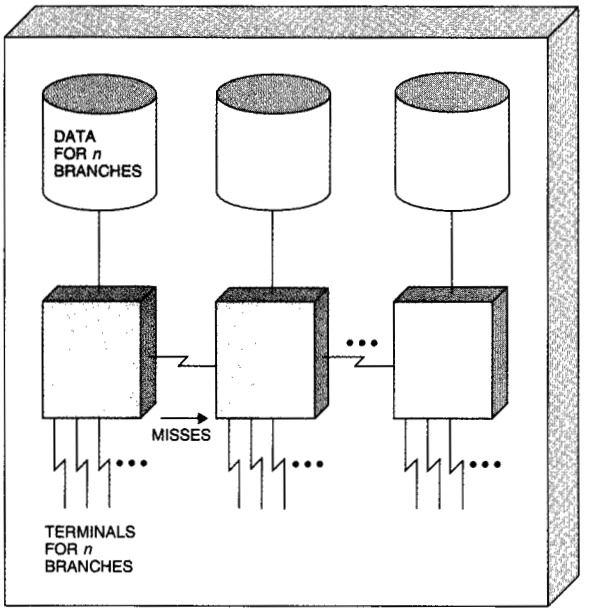


Figure 6 Configuration for the on-line bank teller system



ular machine that holds the data associated with the accounts in those branches.

Generally, branch banking can be managed so that branches are roughly the same size and have roughly the same traffic. Therefore, all terminals can be made to have very similar traffic patterns. This homogeneity, plus the fact that people tend to bank where they have an account or in a branch that is nearby, allows the communication overhead to be low enough for the system to be viable. The probability that a transaction requires access to data in another node is the key parameter in the design. This probability is referred to later as the *miss ratio*. One of the techniques used to minimize the miss ratio is to place the data and terminals for a set of branches in adjacent geographical areas in the same machine.

The second application to be implemented is the authorization of credit. Assume that the same banking establishment requires a system with terminals in retail establishments to perform the credit authorization for its bank credit card. This application has little homogeneity. There is a wide variance in traffic among terminals, and there is a skewed distribution of terminals on a geographical basis. Since these characteristics do not lend themselves to a distributed implementation, assume that this application is placed on a single-node system.

The third stage of growth of the example on-line banking system is electronic funds transfer. When customers make purchases in a retail store using the second application system, the charges are reflected against the customer's checking account balance in the first application system. This new application obviously will place a significant additional load on the combined system.

Probably the most attractive solution to the problem of handling this additional load is to add more nodes to the first application system and redistribute the terminals and data so that each node handles fewer branches than before. In this way, additional capacity is made available to handle the new application. Because each node has fewer branches, however, the miss ratio increases, with a corresponding increase in communication overhead. This increased overhead adds to the load of the system, and still more capacity is required. If more nodes were added to solve this problem, the resulting increase in communication overhead might totally offset the increased capacity. In this way, the situation might degenerate to the point where no number of nodes would be able to handle the required load.

Another simple solution to the problem is to use machines that have greater processing capacity in each node. This alternative is sometimes not attractive because of the economics of replacing existing equipment.

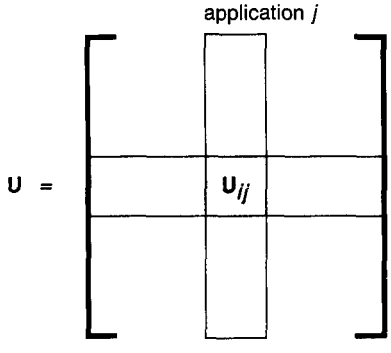
If neither faster equipment nor the simple extension of the distribution techniques already used yields viable configurations, the only other approach is to redesign the system. This generally involves redistribution of data and programs between nodes, including the node for credit authorization, along with the addition of equipment to augment capacity. In many application-growth situations, this approach is the only one that can be used.

The lesson to be learned from this example is simply that growth of applications must be carefully planned and managed. New applications typically create new relationships among existing data and application programs. As a result, the balance of the system design, which is based upon having particular values for certain interaction probabilities, may be upset, and the system may have to be rebalanced.

General considerations for data and program placement

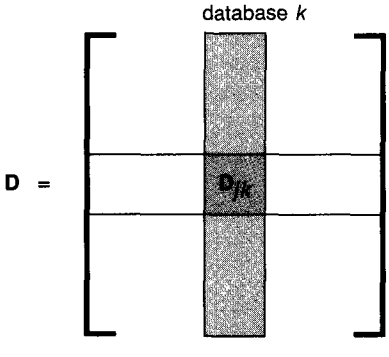
This section is a theoretical discussion of techniques for determining the placement of data and application programs in a network. It attempts to start with first principles and to derive methods for achieving optimization. Although this is not an optimization process *per se*, it does illustrate the major steps that must be taken to perform an optimization.

The first step is to determine the relationship between users and the applications programs they use. Consider a matrix **U** [Equation (1)] with one row for



(1)

each user and one column for each application such that an entry, U_{ij}, shows the messages per second as a consequence of user i's usage of application j. The entries in this matrix are the number of messages going back and forth between applications and users to accomplish the transactions. Next, a second matrix **D**, as shown in Equation (2), is constructed, in



(2)

which each row corresponds to an application program and each column represents an individual data base. The entries in **D** are derived from matrix **U** and from knowledge of each program's usage of data. The entry D_{jk} gives the number of messages or accesses per unit time from application program j to database k. For completeness, a third matrix showing message traffic between application programs would be added. It is omitted in this analysis for the sake of brevity, but its addition is straightforward.

The next step is to introduce geographical considerations into the analysis. Each user, database, and application program is assigned a location, and the rows and columns of **U** and **D** are sorted by location. In some cases the data and/or applications are already located in the network, so that the assignment is a given. In other cases, the assignment of location for applications and data is a variable, and, for the purposes of the rest of this discussion, we assume that an arbitrary assignment is made. The next step is to generate two more matrices, $\mathbf{U}_L$ and $\mathbf{D}_L$, that show the message traffic by location from the users and to the applications and from the applications to the data. The consolidated location-to-location traffic for both programs is given the sum of the two: $\mathbf{T}_L = \mathbf{U}_L + \mathbf{D}_L$.

The next step of the analysis is to generate cost data showing the additional overhead in terms of response time, cycles executed on the various processors, or dollars for communication facilities for each of the communication paths required to implement the message traffic shown by $\mathbf{T}_L$. Once this cost matrix, **C**, is generated to show the location-to-location communication costs, the specific communication cost for this placement of programs and data can be computed.

As this is done, several facts emerge. One is that communication within a data processing node is significantly more efficient and therefore cheaper than communication from node to node. Second, the traffic between applications and data is significantly higher than the traffic between users and applications. Third, the cost of communication between two processors in the same building or campus, because of the ability to use local-area networking facilities, is significantly cheaper than communication between buildings or campuses requiring wide-area networking facilities. All of these facts lead to the following conclusions and rules of thumb regarding placement of programs and data in a network (discussed more fully in Reference 1). As a general rule, users and the applications and data they use should, whenever possible, be placed in the same data processing node. This maximizes the amount of communication on the main diagonal of the matrix $\mathbf{T}_L$. If some form of distribution is required,

generally keeping the programs in the same node as the data they use results in the best performance.

Distributed application processing support

Clearly, each of the modes described has important uses in various application situations.[2] SAA will provide direct support for programming in each of the modes described in this paper. Distributed DM/PM will allow programs written to the SAA Dialog Management Interface (DI) and Presentation Management Interface (PI) to run anywhere in the network, with the user seeing an interface and usability that are independent of the actual placement of the application program. Obviously, applications that operate in this way must be able to function within the restrictions of the dialog manager capabilities or use the PI in such a way so as to keep the node-to-node communication frequency to a practical level.

An associated function mentioned earlier is the ability to invoke application programs on remote systems from the intelligent workstation. Such a facility is called "application management." Application-to-application communications is provided directly by the SAA communications common programming interface. Additional extensions would include a local/remote transparent procedure CALL that would enable one program to CALL another without knowing its location. The CALL would be location-transparent and appropriately efficient in the local case. This facility would be extended for both synchronous connections (i.e., CALL/RETURN) and asynchronous connections.

Finally, SAA will provide distributed data access to flat files and relational data on a record-by-record basis as well as a file transfer capability between nodes that supports both full file transfer and the extraction of data from larger files.

Underlying these facilities must be a structure for implementing security across the network and in the workstation, a naming convention and directory structure to locate data and programs in the network, data transforms to handle the differing data representations of various high-level languages and systems, and so forth.

Enterprise system management

So far, this paper has addressed questions involving the structure and location of application programs and data in the network. The use of multiple data processing systems within an enterprise creates the need for system management facilities designed for a distributed environment. Although each node in the network could be managed as a separate system, most enterprises will want to manage the network of systems as a single entity. There are several aspects, each of which could be centralized or distributed individually according to the requirements of the enterprise. These are the following:

1. Administration of data, databases, security, user authorization, accounting, etc.
2. Operation of individual systems and the network itself
3. Systems programming, including the control and distribution of fixes and new versions
4. Application programming, including the control and distribution of fixes and new versions
5. Problem determination
6. Service of hardware, software, and communications

The underlying support for the above options are the facilities already described for implementing distributed applications. Certain general tools to support centralization of system management are necessary. The following are examples:

- A data distribution and collection facility that would schedule and track the distribution of data files to any or all nodes in a network. This tool must also be capable of collecting data files from the network.
- A common repository for configuration users in profile, security, and addressing information that provides uniform access to data on a network-wide basis.
- A tool that allows operational personnel to monitor and control multiple systems without having to see the individual screens from each. Summarized displays of the status of many systems are a key part of this tool.
- "Programmed operator" support, wherein the need for human intervention is avoided by pre-programming decisions. The purpose is to achieve unattended operation.
- Problem determination facilities allowing a remote expert to monitor the usage of an end user who is experiencing difficulty or to work directly with a failing system.

Concluding remarks

The functions of SAA, and in particular the inherent enablement of data and program portability, provide

a new level of support for distributed data processing. In the past, lower levels of functional support required substantially more application programming to achieve the various modes for distribution. Moreover, when new applications appeared or usage patterns changed, retuning the systems was a major undertaking. SAA solves these problems, thereby making the exploitation of distributed processing techniques far easier and more practical.

Cited references

1. A. L. Scherr, "Structures for networks of systems," *IBM Systems Journal* **26**, No. 1, 4–12 (1987).
2. A. L. Scherr, "Distributed data processing," *IBM Systems Journal* **17**, No. 4, 324–343 (1978).

General reference

A. L. Scherr, "Distributed data processing," *Proceedings of COMPCON '82*, IEEE Computer Society, Los Angeles, CA, 1982, pp. 15–23.

Allan L. Scherr *IBM Application Systems Division, 472 Wheelers Farms Road, Milford, Connecticut 06460.*
 Dr. Scherr is an IBM Fellow and ASD Vice President of Development and Integration responsible for leading the architecture for the application layer of SAA. In addition, his organization produces products in the artificial intelligence field and a series of application programs integrated into complete packages. Earlier in his career, Dr. Scherr had directed several development organizations, including a new midrange system, SNA communications programming, and the distributed processing software for the IBM 8100 system. Dr. Scherr was the originator of TSO and the overall project manager for the first release of MVS, in 1974. He has also held key staff positions, e.g., as a director on the Corporate Engineering, Programming, and Technology staff and as a member of the Corporate Technical Committee. Dr. Scherr joined IBM as a staff engineer in Poughkeepsie, New York, after receiving his Ph.D. at the Massachusetts Institute of Technology in 1965. He received the B.S. and M.S. degrees in electrical engineering in 1962, also from M.I.T. While at M.I.T., Dr. Scherr did research on time-sharing systems performance and user characteristics. His Ph.D. thesis, comparing performance measurements and results from both simulation and analytic models for time-sharing systems, was the first such study ever made and earned him the ACM Grace Murray Hopper Award in 1975.

Reprint Order No. G321-5332.