Highlighted in this technical essay are discussions of the nature of distributed systems, design processes associated with the distribution of processing, and the conditions under which benefits accrue. The essay concentrates on some of the major benefits expected from distributed systems so as to provide a context in which to judge particular designs and their benefits. Among the judgment-informing considerations are the following: centralized management, historical relationships with on-line systems, reliability and fail-soft, security and privacy, system growth and capacity limitations, and fitting the system to the organizational structure.

Distributed processing: An assessment

by H. Lorin

The general notion of a distributed system is that various elements of a data processing system can be partitioned into well-defined units that may be located at various logical sites and linked by agreed-upon protocols. Examples of distributed processing have as many points of dissimilarity as similarity and reflect a diversity of system solutions to system problems. Distributed systems take different views of the distributable system components, of the essential shape of the system, and of the geographical proximity of the nodes. Various choices are made concerning placement of data and programs, and the form and location of systems control.

The diversity of system details suggests that distributed processing systems are not a new class of system. Definitions fail to distinguish precisely among different levels of cooperation, interaction and extent of appearance as a single system (*single systems image*). There are too many variations of too subtle a nature to achieve a comprehensive definition.

Figure 1 suggests an aspect of distribution as a feature of all systems, wherein each circle represents a layer of system function. The inner layer is that of *hardware* and represents a physical processing node. This layer supports a layer of software that is the fundamental operating system, which is often characterized as the *kernel* and is made part of a single structural element of programming. Above the layer of the kernel is a set of extended services that an operating system undertakes in support of a program running in its environment. Such services might be to acquire and release memory space, get and place records in files, etc. Above this layer, Figure 1 shows a *monitor*, which is a layer of a software structure, such as a data base manager. Above the data base manager there is an *application layer*.

Figure 1 Systems layers with multiple terminals

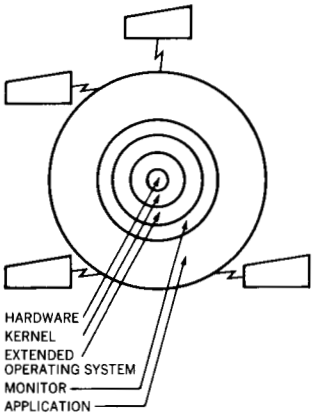


Figure 1 suggests that a system is a number of software layers resting on a hardware layer. Each software layer depends upon a lower software layer for the delivery of certain services and cedes certain aspects of control to a lower layer. The figure is an attempt to unify some thinking about distributed systems. One can think of various kinds of systems as a result of configuration and sharing decisions applied to the layers. Figure 1 shows a dedicated application system. Any transaction entering the system from the terminals talks to the same application. All terminals share application code and application logic.

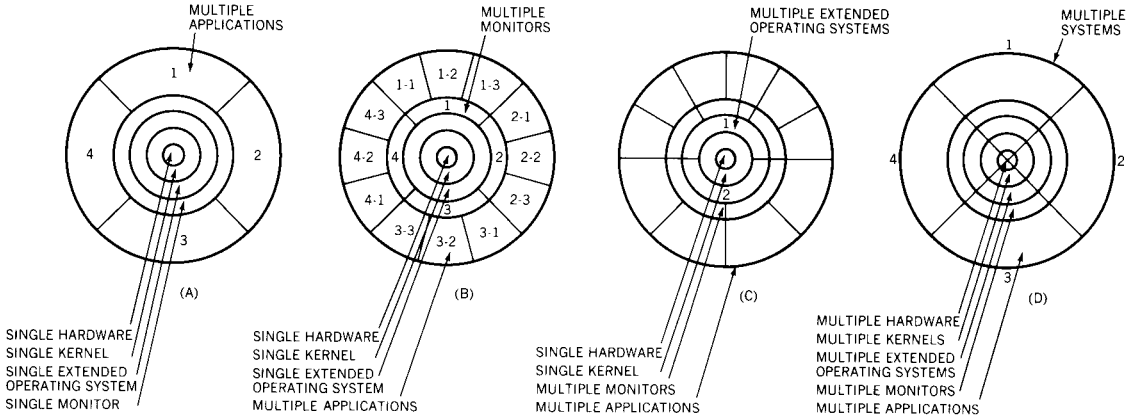
In Figure 2A, lines have been drawn through the application layer, suggesting that more than one application is sharing all levels of this node. A terminal population may talk to different applications that share no application code.

The multiple applications of Figure 2A suggest a uniprocessor that has been multiprogrammed to support independent applications. It is possible that the uniprocessor is a multiprocessor that is running under the control of the same operating system and appears to all applications to be a uniprocessor because of the image presented by the monitor and operating system levels. This system suggests a lower level of sharing in that a specific application partitioning has been undertaken.

Figure 2B shows the line extended through the monitor level, suggesting that terminals have access to different monitor level software as well as to different applications. The system of Figure 2B has multiple subsystems that share an underlying operating system. Each subsystem presents a unique interface and a unique set of services to an upper layer of application and an upper layer of terminal users.

Figure 2C shows an extension of the partitioning line through the extended operating system layer, which suggests virtual machines. Not only do application programs interface with unique

Figure 2 Partition levels: (A) Multiple applications; (B) Multiple monitors; (C) Multiple operating systems; (D) Multiple systems



subsystem layers, but also the subsystem layers interface with a set of unique extended operating systems. The monitor levels might be the data base manager (IMS), the time-sharing monitor (TSO), or the transaction manager/file manager (CICS). The extended operating systems might be MVS or DOS. The underlying kernel would be VM/370. Actually, in implementation detail, VM/370 is a level beneath the entire DOS or MVS operating systems, not just a substitution for their lowest levels. In Figure 2C a decision has been made to partition work so that multiple operating systems can be run on the same hardware.

Figure 2D shows the final level of partitioning. The partitioning line extends through the hardware circle and gives a family of dedicated nodes. The decision has been made to dedicate hardware to an operating system kernel.

The circle is complete, having started with an application-dedicated system followed by a series of partitioning decisions through multiple applications, multiple monitors, and multiple operating systems to a family of dedicated hardware nodes. The sequence of Figure 2 shows a relationship among systems of various types along a continuum of sharing and partitioning choices. It implies that it might be possible, in the future, to configure large systems that are software partitioned and systems that are hardware partitioned at various levels in a manner not too unlike the way we currently choose the number of channels or storage devices we want for a configuration. In future systems, it might also be possible to move a set of applications from systems of the type shown in Figure 2B to those like Figure 2D with ease, or at least with acceptable levels of effort.

A great deal of software and hardware packaging thought must be undertaken before Figure 2 is a real picture. However, the view of distributed processing as a point in a set of alternative configurations is useful as an aid to understanding the relationships among distribution, virtual machines, multisubsystem nodes, and multiprogrammed nodes. There is no conceptual limit to the process of partitioning. Starting with any of the slices of Figure 2D, more partitions can be undertaken so that each slice decomposes into a number of smaller slices. The real limit to the process is the available hardware, interconnection mechanisms, and understanding of the way application and system software structures can be layered and decomposed.

The nature of the distribution process

Distribution as discussed in this section has strong top-down overtones. Although much applies to interconnecting autonomous systems, the major thrust is toward concepts and activities used to design an application across multiple nodes, where the single systems image is high.

Distribution is a result of a system design process in which it is necessary to:

- *Define partitions of work.* Systems and application activities should be discovered that cluster together into well-defined, separate units of program.
- *Define partitions of data.* One should seek to discover natural segments or extractions, and to determine where these partitions are used, where they are changed, and to whom they must be made available. One should also determine what access paths to data must be defined, where partitioning is useful, where replication is useful, and identify synchronization requirements.
- *Define relationships between data and work.* This is truly a reflection of the need to define work and data partitions somewhat iteratively and jointly. Data reference patterns are part of recognizing separable programs. Program reference patterns are part of recognizing separable data.
- *Define relationships among partitions.* Determine the degrees of autonomy among programs and the intensity of their interaction and interdependency. This will partially indicate how geographically distant the partitions may be. In addition, the method of interaction must be defined. This involves determination of whether the partitions will be synchronous or asynchronous, talk to each other on a message basis or on a batched queue basis.
- *Determine a set of possible work structures.* Given degrees of freedom for the previous determinations, the work structure

variations are expected to differ in the specific clusters, execution speeds, access times, and interaction characteristics at a program level.

- *Define a set of potential hardware bases.* For the possible logical work structures, a choice must be made between large systems surrounded by trivial work stations, large virtual networks, geographically distributed small systems, multiprocessors, etc.
- *Choose hardware.* This choice requires estimates of the ability of the nodes to meet capacity requicements for each partition, and the ability of the system to meet interaction requirements across interconnect facilities. Also required is the ability to meet reliability goals. Software available with hardware must be sufficient to minimize the risk and cost of applications development. The hardware and software must be available in reasonable time from a reliable source or set of sources. Finally, there must be an acceptable cost balance involving cost tradeoffs with respect to processor/memory, storage, device population, communications, operations, programming, installation, maintenance, and end user convenience.

If the above sequence of activities seems to suggest that undertaking a definition of a distributed system may not be less work than undertaking the design of a centralized system, the point is well taken. The design of multinode systems is not simpler than the design of complex single-node systems. Certain aspects of systems use may certainly be better with a distributed system, and certain kinds of complexity may well disappear. But the image of simple distributed systems as an antidote for the complexity of large single-node systems cannot be expected to hold up in general. As we discuss further in this paper, it is not usually clear under exactly what circumstances the design and operation of distributed systems is simpler, more stable, and more attractive than centralized alternatives. This is an arena of equally astonishing counterexample.

Discussed in the following sections are reasons commonly given as motivations for distribution. Each motivation is discussed from the point of view of whether the goal desired is a natural attribute of any distributed system or whether it can be achieved only in certain design contexts.

Maintain advantages of centralized management

Centralized management is cited first to emphasize the distinction between distribution and decentralization. The position taken by many enterprises is that they wish to maintain an enterprise level of control over the development and operation of data processing applications and equipment. Although computing power is be-

coming rapidly less expensive and, in some versions, need no longer be thought of as a capital investment subject to classical return on investment justification, it is still true that a company benefits from enterprise-wide direction and planning.

There are situations in which it is not necessary to apply enterprise-level direction and standards. Computing devices that affect only the work of a small unit of the business, that require no professional systems support, that involve no expensive programming effort or operational staff, that operate as a departmental tool, may be allowed with a minimum of higher-level control. It is important, however, to define the situation in which the installation of such equipment is permitted, in order to avoid unforeseen complications. Even where independent installation of computing equipment is feasible, it is useful to provide some central technological guidance as to qualified vendors, contract negotiation, and application feasibility. The intent is not to discourage or constrain the installation of equipment, but to ensure that the degree of the autonomy of the unit is well specified, and to limit the risk of failure.

Centralized services may assume various forms. These services range from a computer in the data processing department (in support of remote operation, maintenance, and software service) to on-site personnel employed by the data processing department rather than the using organization. Once again the intent is not to limit or constrain the use of local computer units, but to provide various services that ease the burden on the using business unit or location.

The maintenance of a centralized systems management function is essential when the distributed computer structure supports cross-departmental functions that are planned as a single system although they are applied to multiple computer nodes. In this situation there is need for professional systems planning and high levels of systems assurance.

The desire to maintain centralized management even for dispersed computer nodes is based upon three underlying needs: better control, personnel, and avoiding the costs of incoherence.

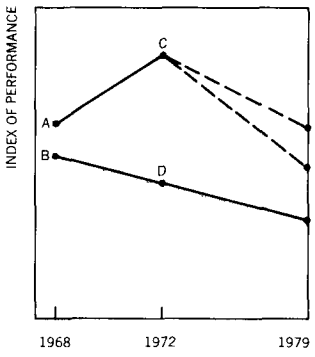
Better control implies a site for the generalization of standards for programming, equipment, operation, and interconnection. The function is to set a policy to ensure the orderly acquisition and use of computing and avoid bad surprises in vendor selection, equipment quality and availability, programming difficulty, etc. The actual amount of decision-making power possessed by a central policy and competence organization may vary widely from that of advisor to that of enforcer, depending upon the particular style of policy enforcement.

As for personnel, a data processing staff provides the ability to attract good professional data processing management and operational skills. No matter how hardware is dispersed and interconnected, a minimum set of skills is required to define and ensure systems and to avoid expensive mistakes. In order to maintain these skills, there must be a company-wide career path. Technology change and the importance of data processing equipment to the competitive and profit position of the whole enterprise suggest the continuing need for skilled computer planning and analysis.

An incoherent system is one that springs up and develops as an afterthought. Such an unplanned system may be costly. Nodes that start autonomously may grow toward one another as profitable instances of intercommunication are discovered. A framework must be provided for achieving post-installation communication at reasonable cost and effort. Otherwise, the organization that has allowed uncontrolled growth of computer nodes may find major problems in causing these nodes to communicate with one another.

On-line systems

Figure 3 Performance comparison for on-line and small systems



In the early 1970s, on-line systems were sometimes associated with small systems. Whether this is valid or not, the facts analyzed in Figure 3 suggest that there is no longer clear preference between large and small systems for on-line applications. Figure 3 depicts a large-system problem at a time of transition when interest in small-system solutions was becoming widespread.

Consider first a large system of 1968 as represented by point A on Figure 3. This system had an associated price/performance ratio determined by its hardware architecture and the software that it supported. The vertical axis of Figure 3 shows some notion of price per transaction (index of performance) for on-line use of this large system. The achievement of a particular price/performance ratio was determined in part by the interrupt structure of the hardware and by the software structure of the operating system and data manager system. The structure of both hardware and software was heavily influenced by its orientation toward a batch environment. On-line use of large systems was just beginning to receive serious attention in the late sixties, and very few hardware or software designs were oriented toward on-line commercial use.

At the same time, as represented by point B on Figure 3, there already existed a class of small machines of important computational power and attractive prices. These small machines were frequently generalizations of architectures aimed at good per-

formance in process control applications. They had interrupt structures that were characteristically more sophisticated than large batch commercial processors, and they had other architectural features that suggested good performance in on-line environments similar in requirement to process control. At one time in their early history, however, they lacked mature or complete operating and programming development systems.

Figure 3 represents, at point C, the evolution of a late 1960s large system to an early 1970s large system. The astonishing slope of the line suggests that the price/performance ratio of the 1970s large system, in terms of cost per transaction, was seen in some cases to have degraded when compared to the 1968 system. This degradation occurred in selected software environments despite an increase in raw system capacity and a general lowering of prices across components of the system. The phenomenon may be explained by the fact that the functional richness of new software placed a disproportionately heavy load on the system in an on-line transaction processing situation. The batch-like hardware and software architectural features remained in place and new software function reduced the ability of the system to process transactions at an acceptable rate. Very large systems had very disappointing rates in transactions per second when running standard operating systems and data managers. There was an unacceptable burden of systems software on the large system and a number of basic activities associated with terminal support that a large system did not do well.

Point D represents the maturity of small systems in the early 1970s. Small systems had continued to demonstrate very good price/performance ratios, had improved in raw capacity, and had acquired a reasonable set of software for node control and programming. The operating systems tended to be event-driven in structure and to demonstrate some properties that made them attractive for on-line applications.

The effect of software burden and complexity of use of large systems in on-line environments, coupled with maturing capacity and software support for small and intermediate systems led a number of designers to the inspection of single or interconnected small systems as possible alternatives to single-node large systems.

Two basic alternatives emerged. The possibility of augmenting a configuration by adding small processors to undertake on-line application functions led to systems shaped as hierarchic trees. The possibility of building a system completely out of aggregated small processing nodes led to cooperating peer structures.

The trend toward on-line applications has had an effect on software for large systems. In the interval between the early 1970s and the present time, important results have been achieved in reducing burden on a large processor and significant improvements in large-system efficiency have been achieved.

As a result of hardware and software trends it is no longer clear whether small machines have a natural advantage over large machines for on-line operation. Setting aside considerations of reliability and availability, which will be discussed later in this paper, large systems may now be effective in more instances than a decade ago because they have experienced impressive capacity increases in the mid-1970s as a result of technology and software improvements.

One very important aspect when considering large versus small systems, or a large system versus a small-system aggregate, is the nature of the load to be placed on small systems. Small processors of a given capacity may be very effective for simple transactions that require low levels of computational service and have simple data reference patterns. Thus sets of simple enquiries or basic data entry activities may be very effective on small nodes. Transactions requiring massive computation or involving complex patterns of data reference may exceed the capacity of a small node or perform poorly on a small node and be better supported by a large system of greater computational power and more flexible data subsystem interconnections.

Communications costs

A frequently stated motivation for distributed processing is the desire to reduce the cost of a set of unintelligent terminals communicating at geographically significant distances with a data center.

Although it is possible to achieve a reduction in communications costs, it is by no means clear under what circumstances this reduction will occur. Communications costs may be a function of the specific offerings of communications carriers, sensitivity or insensitivity to geographical distances between points of data or query entry and data manipulation, required speeds, applied loads, complexities of network definition, etc. Experience has shown that distribution of data processing capability has caused communications costs to rise or fall, depending on many factors related to each particular system. Instances where these costs rise are by no means failures if other costs are reduced or if some value is added to offset the rise in communications costs.

Typically a system in which communications costs are lowered is one that has intelligent terminal-processors at a using location. Traffic on communications lines is reduced because the local processor reduces the volume of traffic to the data center. It does this by sending summary data rather than raw transactions, by eliminating the need for reference to the data center for a class of transactions, or by batching and timing transmissions to take advantage of special features of certain tariffs or economies of scale in transmission bandwidths being offered. If a set of processing nodes are geographically dispersed to sites where there was no previous computational power, and if high-speed, mesh-like interaction between nodes is required, it is clear that communications costs may rise.

Reliability and fail-soft

The view that distributed processing can provide greater reliability or availability is based upon the economics of replication and the granularity of configurability that interconnected smaller systems may provide. The duplexing or triplexing of small processors into multiple-processor logical nodes is quite common. The practice is attractive because small processor/memory units are inexpensive and additional units give disproportionate reliability increments, while adding modestly to the total system cost. Thus various partitioning and replication designs can provide scope of error containment, equivalent performance backup and fail-soft levels when multiple machines are used. The same approach is not equally well applied to large processing nodes because of the larger prices and the incremental jump in total system cost when a large unit is replicated in a system.

This approach, however, is constrained by a number of considerations. The processing/memory units are the most reliable components of a data processing system, and it is not clear, in general, how replication of the most reliable units addresses system-wide issues of reliability. The replication of data storage devices is limited by their relative costliness, in terms of cost per byte stored, when compared to larger units and by the logic of an application. The replication of storage units implies design conventions about how data are to be spread across units, transferred, synchronized, and accessed. Problems of data integrity emerge in systems that can be partially operational, which do not occur in systems that are either up or down. Increased reliability comes not from the replication *per se* of hardware units but from systems designs that provide quick recovery while guaranteeing integrity.

If fail-soft levels and scope of error containment points can be defined, it is probably cheaper to replicate critical points than to

replicate larger, more monolithic systems. It is important, however, that the replication be selective. If it is necessary to replicate the entire system of collected small nodes to achieve a desired reliability, it is not clear that the replication of a single large node may not be equally effective.

The operability of the system is the joint probability of the operability of all nodes. The probability that some part of a system will be down is very high. On the other hand, the probability of all nodes being down is equally small. Thus, if a system depends upon all nodes being up, it is not a reliable system. If proper backup and fail-soft levels can be defined so that the system is meaningfully operational with inoperative nodes, a reliable system may be designed from collections of smaller units. In the end, reliability expectation is an expression of the definition of fail-soft and backup levels.

User interfaces

Better user interfaces are frequently cited as an advantage of distributed processing. The nature of such interfaces is not well understood in the data processing industry at this time. Nevertheless, there seems to be converging opinion that a good interface has some of the following characteristics:

- It provides system response times appropriate to the activity; it does not introduce a perception of instability, unpredictability, or lengthiness that disturbs a user so as to make him less effective.
- The semantics of a good interface is consistent with the semantics of the work being done.
- The syntax of the interface is as natural as possible and appears intuitively obvious to a user.
- The syntax is uniform and consistent within the context of the work. Accomplishing the same function by multiple variant forms is minimized, and the use of variant forms for similar functions is eliminated.
- The system allows selectable levels of aid and guidance for users of different degrees of expertise. Friendly software need not be chatty software, and experts should not be burdened with conventions for aiding trainees.

Clearly, the placing of a processing unit at a geographical site does not in itself provide good end user interfaces. It may be true, however, that good end user interfaces are more affordable in the context of some distributed designs. Elements of good user interface relating to systems response times may be achieved by the dedication of processing units to application activities so as to reduce instances of resource contention which occur on intensively shared systems.

If the load on a node is more predictable because the workload is more homogeneous and the system is simpler to analyze, responsiveness may increase. Similarly, if end user actions are completely contained within the node they may be faster than if they must be serviced by a remote node running a complex workload. The requirement for fast or consistent responsiveness cannot be met, of course, if the local system is overloaded. It is also true that small systems tend to become unresponsive at lower levels of utilization than large systems and that they are inherently slower. Consequently, better responsiveness is achieved at the cost of maintaining consistently lower loads on the smaller local nodes in a hierarchic system. In a peer system, stable responsiveness may imply increased partitioning across a set of nodes and very careful attention to cross-node referencing. The responsiveness of a node should not be perturbed because it is waiting for interaction with other nodes or because other nodes are inflating its local workload.

Those aspects of end user interface that are concerned with the quality of dialogue may be improved by distribution because distribution may make good dialogue more affordable. Good dialogue characteristics involve increased potential interaction with a processing node and potentially more significant displays of data and format. A user operating in tutorial mode, for example, may require many more transmissions between his terminal and a processor than an expert. Similarly the replacement of terse codes by descriptive phrases increases data flow from system to tube.

Large computationally effective processing units may have a disproportionate burden placed upon them when they do formatting and display organization. This increase in load discourages designers of interfaces from rich support of dialogue and encourages cryptic and sometimes artificially terse message formats. This tendency is enhanced by a desire to minimize data flow into and out of the system. In view of this, it is reasonable to provide for a node in the system that can effectively improve dialogue without a serious increase in load on the computational engine. Very terse and compressed messages may flow between the dialogue support node and the computationally oriented processor. The dialogue support node expands the messages into a form convenient for users, supports tutorial phases, and compresses user-entered syntactical structures without burdening the computational or data-manipulating element of the system.

The partitioning of dialogue support function suggests both that the dialogue processor is very good at these functions and that its load is sufficiently low as to maintain good responsiveness. It is not clear, except for some improvement in interrupt logic, that smaller processors are more efficient format and character han-

dlers than large processors; thus very careful definition of load and activity on the dialogue processor must be undertaken. If major application logic is also resident in the dialogue support processor, response characteristics may become undesirable.

Once the dialogue support function has been isolated, a decision must yet be made about where it is to be placed relative to using terminals and the computational node. It is probably true that the dialogue support processor is best placed local to using terminals to achieve lower transmission volumes across teleprocessing lines. However, it is possible to conceive of situations where the dialogue processor should be local to the computational processor.

Although it seems natural for screen quality and message control to be local, it is not always true that the data managers or operating systems of a vendor are able to permit such partitioning, so that some degree of duplication of function may be necessary to achieve the result.

The point is sometimes made that cost to a business unit to use data processing power is reduced with distributed processing. The argument shows confusion between the issue of distributed versus centralized processing and that of batch versus on-line processing. In discussing the advantages gained by going to a distributed system, much of the literature points out how much less expensive and how much more convenient computer use has become because of the availability of on-line terminals that replace awkward batch submission interfaces. It is clear that the increase in convenience and reduction in costs come from changing the mode of access and not from the dispersion of processing nodes. Most users agree that terminals are an effective input/output medium for computers, but whether the presence of local processing and storage contributes more than the presence of a terminal is not generally clear. The confusion comes about because many batch systems that were centralized have been replaced by some form of on-line distributed system. The virtues of going on-line are mistaken for the virtues of going distributed.

In a similar but complementary vein, distributed systems are sometimes said to be more complex than centralized systems. Frequently this point of view arises out of a movement from a batch to an on-line environment. Many aspects of design and planning for on-line use are more complex than designing and planning for batch use. The complexity is a function of being on-line, not of being distributed. The real issue is whether the relative increase in complexity by being on-line over being batch is less or greater when one goes on-line with a centralized or a distributed system.

Security and privacy

The use of hardware-isolated nodes is often justified by a desire for increased security of data and for increased privacy. This is also an area of astonishing counterexample and differing judgments for a number of reasons.

Many users of large systems are disturbed by the fact that they have no essential control over data. Because of various backup and archiving procedures used at a data center and because of data center operational prerogatives, it is virtually impossible for a user to control access to his files. Encryption techniques that address this issue are beginning to come into maturity, but they are installed only at the discretion of the data center.

Similarly, there are varying degrees of confidence in both the user and software procedures for ensuring against intrusion by other users. Although the subversion of software structures is not a general skill, it causes concern in many places, particularly where sensitive information is involved. The concern is sufficiently great that the federal government, vendors, and universities are undertaking studies to determine exactly what the structural and functional characteristics of a secure software system really are.

With this as a background, the idea of private data on physically inaccessible data media in rooms where access can be controlled by the owning business unit or mission unit becomes very attractive. Whether increased security and privacy are achieved by using central professional security staffs or by using private safes is largely a judgmental issue. But the increase of these characteristics in a centralized or distributed system must follow the same lines of argument.

An important aspect of the discussion is the source of potential violation. If there is reason to suspect that the source of violation lies in persons or agencies unknown, one may prefer privately imposed security. If there is reason to suspect that the source of violations is within the mission or department, there is reason to prefer professional security at a central place.

Professional security at a central place occurs in the form of large software packages that impose security within a well-disciplined set of staff members following well-planned security procedures. Private security must rely more on physical control of media and access, since smaller systems may not be able to sustain the software loads of high-level access control software packages.

Another dimension to this issue is that distributed systems are distinct from stand-alone systems and by their nature imply some amount of physical access from one node to the data of another.

A secure system must provide software that protects against remote violations of privacy. It is not absolutely clear just what software designs will ensure privacy and security in multinode systems and be less subject to subversion than software in large systems. The problem is compounded, of course, in systems consisting of heterogeneous nodes.

Economics of dedication

The fundamental assumption of the idea that it is better to distribute activity across a family of processing nodes is that economy of scale in systems pricing is no longer an important aspect of the computer marketplace. Small processing units may display price/performance ratios equal to or even better than those of large-scale processors. To illuminate the issues surrounding this point, consider an example involving a review of current prices for different processor classes and a division of these numbers by a rated processor speed. (Very strong arguments against this kind of exercise can be mustered, but it does provide some instructive results.)

The total hardware cost of a system is determined less and less by the price of processors and more and more by communications, storage and peripheral equipment. Therefore, despite the loss of economy of scale across processors, a collection of small systems with power equivalent to that of a large system may cost more in hardware than the large system with more price-effective storage and peripheral units. In addition, there is evidence from queuing theory in support of the idea that a single system of a given power can deliver more service than a system composed of a set of smaller units of equivalent nominal power. Even considering our confidence in queuing theory, it is, nevertheless, by no means clear how much more power is required for small systems to match a large system, nor under what exact conditions of workload and software load characteristics the superiority of the large single system obtains.

In any event, the economic feasibility, if not the preferability, of building large systems from interconnected processing nodes is a reality. There seems to be a potential economics of dedication that is replacing the economics of sharing based upon economy of scale. Thus it is at least doubtful whether a consolidated workload machine that attempts to support a large number of unrelated and disparate users is in general a more efficient instrument than a machine dedicated to the work of each using business unit or location.

There are degrees in the notions of dedication. Some distributed architectures built from single-board computers dedicate very

specific and small units of work to each processor to form a system of highly specialized activity nodes. Such designs may be found in aircraft or submarine monitoring systems. At the other extreme, the entire set of applications relevant to an entire department or business unit may be put on a single system node. The departmental application set may contain a number of unrelated applications sharing the machine on a multiprogrammed or time-shared basis. Between these extremes there are many points on a continuum of sharing of equipment between units of work.

There are a number of factors that determine the congruency of the activity/equipment mapping. One is the extent to which units of work are decomposable and isolatable. Work units that tend to access the same data or talk to each other intensively may be left on the same system. Another factor is the availability of various kinds of interconnection. It may be possible, for example, to decompose work into thirty-four areas of major activity and dedicate a processing unit to each area. It may not, however, be possible to achieve an interconnection between them that displays desirable characteristics of speed and performance. Another detail is that traces of economy of scale may remain, in that very small versions of an architecture may be less attractively priced than somewhat larger versions. Although price/performance ratio advantages may, for practical purposes, disappear in comparing machines from the middle performance range to the upper performance range, it is possible to rediscover them in comparing machines at the low performance range to those tending toward the middle.

The rapid performance improvements in middle-range processors has shifted the balance somewhat toward the use of multifunction rather than single-function small systems. Nevertheless, the heterogeniety of work and extensiveness of sharing across a population of such systems is certainly less than with a large single processing node.

Associated with the idea that collections of small systems may be economically feasible is the idea that they may be less expensive to operate. A number of very important systems expenses are associated with trying to share a large machine at significant levels of utilization. There is a constant tuning and performance effort to achieve acceptable response times and maintain required utilization levels. This contrasts with populations of small machines that can run at lower utilization levels and provide good performance at much lower systems tuning costs.

There is an argument that money invested in performance tuning might be better spent on more hardware. Here we immediately run into a problem with the idea that, in general, families of small machines generate fewer operational expenses than a single large

consolidated workload system. Depending upon the software interface characteristics of a system, the cost to operate and adjust tends to vary. Thus a number of geographically distributed units may require local systems programmer or operator staffs at each site. Although the effort of tuning and adjustment may go down at each site, the total cost may go up. The total cost of operating the equipment becomes the sum for all sites and may exceed the cost of operators on a large system at a single center. The additional dimension here is that different systems generate different operational and software support expenses.

There is widespread interest amongst vendors of computing products in reducing the operational costs of remote processing nodes. This reduction is brought about by providing a facility whereby certain operational activities can be provided from a processing node in one place. Thus a collection of remote nodes can be serviced by a single operator. Similarly, mechanisms for remote software support and remote performance analysis are beginning to receive attention. It may be some time, however, before heterogeneous systems can profit from remote operator support.

An alternative approach to reducing operator expenses in geographically remote systems is to improve the operational interfaces presented by those systems. Thus, instead of remote-operator services there would be programmed-operator services whereby each node would sufficiently automate its operational interfaces that only a very reduced operational staff of very reduced skill levels would be locally required.

In general, from both an equipment and an operational point of view, it is no longer absurd to consider various levels of dedication that would have been *prima facie* infeasible in an industry with the pricing structure of the 1960s.

Incremental growth and flexibility

Since distributed processing systems contain a number of small nodes, it should be easy to achieve growth by adding additional nodes to the system as load increases or as new function is added. So goes an argument that finds support in the hardware base for distributed systems. The major hardware limitation seems to lie in the limits of interconnect mechanisms. Although incremental node addition allows an orderly increase in power with small granules of system cost, it is not clear that this can be achieved across all interconnect designs. A single-bus interconnect design is limited by the load it can carry. Considerable reanalysis of load patterns and cross-node loads may be necessary to successfully repartition work. Another limitation in multibus, crossbar, and

switch designs is the capacity of the system to add more members because of physical constraints. In general, it is preferred to have an idea of how the system is expected to grow in order to ensure that growth steps are nondisruptive.

The addition of more nodes to support new applications may be a simpler task than adding new nodes to support increased load or increased function within a single application. Unless care is taken in structuring program modules and attention paid to defining the mechanisms that repartition data, the hardware potential for growth may be denied because of expenses associated with software and data restructuring.

Capacity limitation

Closely connected with the idea of incremental growth is the idea of overcoming the capacity limitations of a system by putting new function on additional nodes. The classic scenario lies in the idea of extending the life of a central large system by offloading function onto associated peripheral processing nodes. In general this is probably a workable notion. It is not clear, however, exactly how effective an offloading strategy may be. The support of a set of small nodes may create a new kind of load for the large system. Many installation managers believe that distributed systems should be controlled from a single point. Software and operator functions to establish systems control, recovery, and remote operation, and to permit remote program development and testing also add to the load at the central site. Whether this load trivially, importantly, or unacceptably counterbalances the offloaded activities is an assessment that must be made for each system. It depends in part on the activity that can be moved out into the smaller nodes and the degrees of control and central function that are vested in the large system.

Increased installation simplicity

The idea that distributed systems allow applications to be brought up more quickly than large systems is often heard expressed, but it is flawed for a number of reasons. It is essentially a carryover from an image of the use of stand-alone autonomous processors surrounded by business unit programming staff. An image of quick installability of applications accrues to small systems because no negotiations are required to get programming staff and computer resource, because the systems are easy to use, and because the applications are often small.

It is not demonstrable that the planning and design of multinode systems represent more or less effort than the planning and design

of single large systems, and it is not clear that the stylistics of the autonomous use of small business systems applies to the definition and implementation of distributed processing.

It is not yet known to what degree the aspects of simplicity which associate with small machines will associate with distributed systems.

A more stable software environment

The idea that operating systems and subsystems environments are more stable for small systems than for large systems is founded on the current stylistics of software offerings in the marketplace. Large systems vendors have tended to announce software products on an evolutionary cycle, making improvements and enhancements from version to version. Thus a user of a large system who depends on an operating system and a collection of access methods and subsystems experiences a constant churning. Each major software component has an independent version release cycle that keeps an installation in perturbation to stay in the mainstream.

By contrast, software for small systems has tended to be offered on a purchase basis at different levels of function and to remain stable through the lifetime of the hardware it supports.

While this has been the historical perception, a clear picture has yet to emerge about the preferability of the two approaches. There seems to be a trend toward new features of large software packages announced as optional purchase units for the more stable underlying program. (An example is the MVS operating system and its optional features.) It may be preferable for some users to pace an evolutionary cycle rather than commit to stable software environments that must be radically revised or replaced at certain intervals of time. If the large systems software evolution can be made to be somewhat less disruptive, draw off less resource for installation, and intervene less in application development cycles, the perception of churning can be ameliorated. Large-system software suppliers seem to be sensitive to the need for less installation effort and more mature software systems. It seems today that many of the stylistics of large-system software marketing that have previously been unattractive are being addressed.

The system and the organization

The granular structure of distributed systems suggests the possibility of mapping the system onto the organizational structure.

Regardless of the amount of control exercised over each node, the structure of computing fits the structure of the company. Two caveats apply. Company organizations are not stable, and organizational reforms must not be hampered by computer structures. Although there are many businesses that have achieved a mature organizational structure, there are those that are continuing to discover their proper organizational attitudes, and many that find it useful to change for the sake of change. Experience of successful organizations shows that the computer structure should fit the organization and not that the organizational structure should fit the computer structure.

It is also probably true that it is less burdensome to modify a centralized data base system to represent new business units than it is to move data and computer hardware from one business location to another. It is critical, therefore, when mapping systems to organizational charts, that this be done only when there is confidence that the chart will at least endure for the payback period of the system, or that provision for reasonable variations be included in the initial design.

An additional problem with fitting a system to the organization is that organizations are rarely the neat hierarchic trees drawn on the organization charts. The true organization is a network of which only some of the connections are known. Thus, unless it is very clear who needs specific data, who needs various reports, who needs various system activities, it is risky to undertake hardware partitioning along formal organizational lines. As regards distribution for organizational reasons, the message must be to undertake distribution for this motive only if the organization is stable and really understood.

Concluding remarks

The intent of this paper has been to introduce even-handed critical thought to distributed processing, which is such an amorphous concept.

In view of the many unclears, caveats, cautions, and counterexamples, when is it reasonable to undertake the effort of distributed systems? Insofar as there is an answer to this, it seems to be when the following are clear and well understood:

- Relationships among business organizations and data
- Relationships among organizations and applications
- Relationships among applications and data
- Loads placed upon the system at various points as well as the capacity of nodes present at those points
- Cross-node loads coming from planned internode interactions

If these are the elements of a desirable environment for distribution it is necessary to determine whether the organization is willing to undertake necessary action to clarify its own shape and form.

Certainly all systems design depends upon stability and clarity. However, distributed designs may be more sensitive and require better definitions of applications characteristics. This may be a strength of the distributed approach.

From the above list of desirables one can infer a list of uncertainties, according to which distribution should be looked at very cautiously:

- Communication skills in an enterprise
- Interconnectability of various nodes because of hardware and software capabilities at each node
- Data reference patterns and sources of load, combined with uncertainty about node performance
- The direction of evolution of applications

We do not know, in general, whether complexity will increase or decrease in distributed processing systems, nor how operational costs will evolve. We are just discovering an art.

Despite these factors, distributed processing is a data processing design alternative that is real for any set of applications. We have tried here to highlight the considerations necessary to make considered decisions in order to achieve the potential advantages and avoid the potential disappointments.

GENERAL REFERENCES

G. M. Booth, "Distributed information systems," *AFIPS Conference Proceedings* **45** (1976 National Computer Conference, June 7–10, 1976, New York City), 789–794 (1976).

O. H. Bray, "Distributed data base design considerations," *Trends and Applications: Computer Networks*, IEEE Computer Society, Long Beach, CA (1976), pp. 162–169.

G. S. Champine, "Six approaches to distributed data bases," *Datamation* **23**, No. 5, 69–72 (May 1977) (ITIRC INF0037142).

J. Hannan and L. Fried, "Should you decentralize?" *Computer Decisions* **9**, No. 2, 40–42 (February 1977).

M. Hofri and C. J. Jenny, *On the Allocation of Processes in Distributed Computing Systems*, Research Report RZ905. May be obtained from the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (ITIRC 78A003934).

N. Knottek, "Selecting a distributed processing system," *Computer Decisions* **8**, No. 6, 42–44 (June 1976).

D. L. Mills, *Dynamic File Access in a Distributed Computer Network*, Technical Report TR-415, University of Maryland, College Park, MD (October 1975) (ITIRC 76B000485).

J. F. Rockart, C. V. Bullen, and J. S. Leventer, *Centralization vs. Decentralization of Information Systems: A Preliminary Model for Decision Making*, Center for Information Systems Research, Sloan School, Massachusetts Institute of Technology, Cambridge, MA.

A. L. Scherr, "Distributed data processing," *IBM Systems Journal* **17**, No. 4, 324–343 (1978).

Stanford Research Institute, *The Promise of Distributed Processing*, SRI Business Intelligence Program Guidelines, Report No. 10, Stanford Research Institute, Palo Alto, CA (1976) (ITIRC 77B000339).

The author is located at the IBM Systems Research Institute, 205 East 42nd Street, New York, NY 10017.

Reprint Order No. G321-5111.