A distributed data processing system is composed of a set of nodes that are interdependent yet capable of operating autonomously. This paper describes a procedure for controlling the interdependencies and nodal autonomies with a logical distribution of applications and their data. The procedure is illustrated with data that were obtained from an on-line operations planning and control system at a steel mill.

Logical distribution of applications and data

by C. T. Baker

Among the many reasons for using distributed data processing systems are: economy, convenience, local autonomy, availability, simplicity, manageability, and responsiveness. There are also factors that tend to inhibit the use of distributed systems. These include the costs of conversion from existing centralized systems, the tendency of small systems to become small computing centers, the difficulty of implementing data base systems on small machines, and the difficulty of defining clear boundaries for applications. Moreover, in a distributed environment the applications on the several small systems usually should be coordinated so that the data processing needs of the parent organization are satisfied, which may not be a trivial task. These and many other points connected with the properties of distributed data processing systems are discussed thoroughly in References 1 through 4.

In this paper we are concerned with one aspect of distributed data processing system analysis—how to define the nodes of such a system in terms of its applications and their data. The procedures we describe are generic in that they are not restricted to any particular application environment. The procedures are illustrated with data that were obtained from an on-line operations planning and control system at a steel mill.

Logical distribution

We wish to define a distributed data processing system in terms of its applications and data. We can gain some appreciation of this task by reviewing the definitions of distributed data processing systems that are given in References 1, 2, and 3.

In Reference 1 we find the definition: "Distributed data processing is defined as the implementation of a related set of programs across two or more data processing centers or nodes. The programs are related in that they share or pass data between them. Each node is generally capable of performing data processing applications independently, and thus would normally have data storage and program execution facilities."

The definition stated in Reference 2 is: "A distributed system is one in which there are several autonomous but interacting processors and/or data stores at different geographical locations."

In Reference 3, Item 5 of a seven-item definition is: " 'Cooperative autonomy'—There are interactions among the components of the system: that is, they cooperate on certain tasks while handling other tasks autonomously."

We infer from these definitions that to obtain a clear understanding of our task, we should work with a collection of related applications that have been implemented, and that we must group these applications with their data in a way that permits each group to perform a significant amount of useful work independently, However, since we are working with a collection of related applications, each group will either rely to some extent upon another group or be relied upon to some extent by another group.

Two key phrases must be considered: "permits each group to perform a significant amount of useful work independently," and "each group will rely to some extent upon another group." These notions contain the essence of the distributed data processing system definitions. In this paper we use a concept of intergroup dependency to include both notions. As a group's dependencies are reduced, its autonomy is increased. In essence, this paper is about dependencies—what they are and how they can be measured.

We are concerned with the distribution of applications and their data. For this analysis it is convenient to define the logical distribution, which reflects our immediate interests. (A definition of a distributed data processing system is discussed in the Appendix.)

A *logical distribution* is a partitioning of a collection of related applications and their data into a maximum number of groups that

have a specified low level of interdependence. A logical distribution is composed of at least two groups. Every proper subset of the logical distribution has an interdependency with at least one group not in the subset.

Because of its interdependencies, a logical distribution represents a single application environment; consequently any implementation of these applications, either centralized or distributed, will be a single data processing system. If the implementation is distributed, a central coordination of communication structures, key data structures, and key applications will be required—for without central coordination there can be no assurance that the dependencies among the groups will be supported.

Dependencies

Our distribution procedures are general; they apply to any data processing system, either manual or automated. For convenience we have used information that was obtained from an on-line data base system; hence our terminology is related to this type of system. An application is a set of application programs. An application program belongs to only one application and supports a transaction—for each transaction there is one application program. A transaction is synonymous with its application program.

Application programs frequently require data from two or more data bases. Data bases, in turn, frequently supply data to two or more application programs. It is evident that as the programs and data of a collection of related applications are put into two or more groups some of the programs in a group will probably require some of their data from a data base that is in another group. This is the basis for the dependencies that are discussed in this paper.

transaction dependencies

Dependencies among the groups of a logical distribution are composed of transaction dependencies. A *transaction dependency* is that part of a dependency that is due to a single transaction type. It is characterized by an orientation and two numbers. The orientation, schematically represented by an arrow, shows where the remote data of the transaction are located. The first number, called the active component, gives the frequency of use of the transaction in usages per day. It is called the active component because it represents a part of the activity of the system. The second number, called the passive component, is the number of bytes of remote data that could be used by the transaction. It is called the passive component because it represents data that must be available to the transaction, independent of the activity of the transaction.

Transaction dependencies are put into four categories with reference to an active threshold and a passive threshold. In the following illustrative examples, the active threshold is 500 transactions per day (Tx/Day), and the passive threshold is four million bytes.

1. Active component = 1000 Tx/Day
   Passive component = 0.1 million bytes
   Classification = High, Low (HL)
2. Active component = 20 Tx/Day
   Passive component = 10 million bytes
   Classification = Low, High (LH)
3. Active component = 20 Tx/Day
   Passive component = 0.1 million bytes
   Classification = Low, Low (LL)
4. Active component = 1000 Tx/Day
   Passive component = 10 million bytes
   Classification = High, High (HH)

The purpose of these classifications is to measure the transaction dependency. In the HL category the transaction requires little remote data, so the transaction dependency is low. In the LH category the transaction is not used very often, so the transaction dependency is again low. Clearly, the LL dependencies are low. Obviously the HH dependencies are high since they are used frequently and require a large amount of remote data. This leads to one condition for a logical distribution—it may not contain any HH transaction dependencies.

intergroup dependencies

The dependency between two groups of a logical distribution is often due to more than one transaction type. Hence, we consolidate the transaction dependencies. We do this within the transaction dependency categories by adding the active components and by measuring the set theoretic union of the passive components. For example,

Transaction Type A
Active = 600 Tx/Day
Passive = 0.15 million bytes

Transaction Type B
Active = 500 Tx/Day
Passive = 0.1 million bytes

Assume A and B could use 0.05 million bytes of remote data in common. Their consolidated dependency is

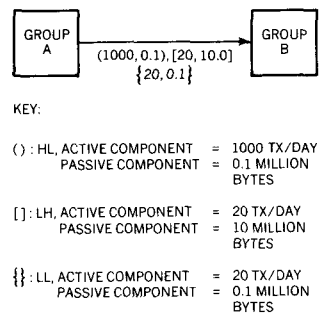
Active = 1100 Tx/Day
Passive = 0.2 million bytes

After the transaction dependencies are consolidated within their respective categories, the consolidated dependencies are classi-

fied using the same thresholds. This consolidation leads to a further condition for a logical distribution—a consolidated dependency cannot be in the HH category.

We can now describe a dependency between two groups of a logical distribution. It consists of an orientation and the three consolidated dependencies, HL, LH, and LL. The schematic representation of an intergroup dependency is shown in Figure 1.

Figure 1 Dependency schematic



The dependency definition reflects some implementation considerations. The HL dependencies do not involve much remote data, hence their transactions might be decoupled—supported locally—through the use of copies of data. The LH dependencies, however, do not have much transaction message traffic; they could be supported by data communications facilities.

It is evident that the values of the thresholds determine the number of groups in a logical distribution, the extent of a group's autonomy, and the strength of its dependencies. The threshold values at present are determined judgmentally, after a thorough familiarity with the applications has been attained. A visual aid for use in the determination of the threshold values is described later. In the future, more formal methods, currently in development, may be used in these analyses.[5,6]

Steel mill data system

We have described some general distributed data processing concepts that can apply to any application environment. For reasons of efficiency and convenience, we have worked with data that were obtained from on-line Information Management System (IMS) applications. We required a system that was composed of a substantial number of related applications. This requirement was satisfied by the selected system: an on-line operations planning and control system at a steel mill. Our subsequent discussion is in terms of this steel mill system.

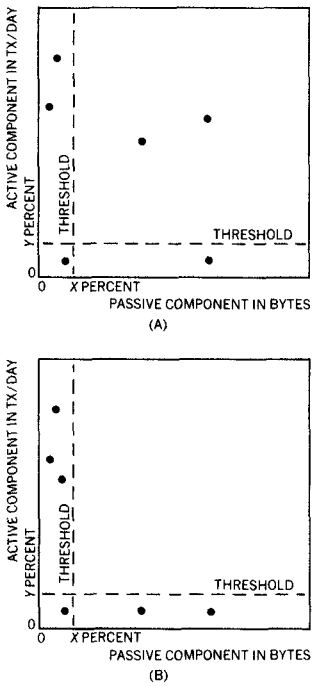
The steel mill system is implemented with IMS/VS, Version 1.1.4. At present two major categories of production activity are supported by the on-line system: primary steel production and steel plate production.

Primary steel production starts with the introduction of molten pig iron, scrap steel, and other raw material into a basic oxygen furnace. After a 45-minute process cycle, the furnace produces a "heat"—300 tons of molten steel—with a specified chemical composition. After several processing steps, the heat is formed into slabs, an intermediate product that weighs roughly 17 tons and is up to eight inches thick, up to six feet wide, and 20 feet long. Slabs are the end product of primary steel production.

Table 1 Data base names and codes

Code	*Item*	*Millions of bytes*	*Code*	*Item*	*Millions of bytes*
DB01	Metallurgical, check analysis	19.1	DB29	Slab inventory, 8	2
DB06	Primary steel orders, 1	10.7	DB30	Primary steel orders, 2	0.5
DB07	Slab inventory, primary steel	10.7	DB31	Chemical analysis	34.2
DB08	Slab orders, 1	5	DB32	Slab inventory, 9	66.2
DB09	Slab orders, 2	1.6	DB33	Plate production sequence	1.7
DB10	Metallurgical grade practice	0.5	DB34	Plate production reports	12.4
DB11	Primary steel terminal	0.5	DB35	Slab incentive, 1	5.2
DB13	Slab orders, 3	0.5	DB36	Slab incentive, 2	11.5
DB14	Metallurgical specifications	5.0	DB37	Plate inventory, 1	11.4
DB15	Plate orders, 1	1.2	DB38	Plate inventory, 2	0.3
DB16	Plate orders, 2	45.8	DB39	Shipping services	14.6
DB18	Physical test, 1	26.2	DB40	Shipping, 1	1.2
DB19	Physical test, 2	7.4	DB41	Shipping, 2	0.7
DB20	Slab inventory, 2	20	DB42	Shipping, 3	0.7
DB21	Slab inventory, 3	2	DB43	Shipping, 4	0.3
DB24	Slab inventory, 4	1.5	DB44	Shipping, 5	1
DB25	Slab inventory, 5	39.1	DB45	Shipping, 6	1
DB26	Slab inventory, 6	2.5	DB46	Shipping, 7	0.3
DB27	Slab pusher	7	DB47	Primary steel production	14.6
DB28	Slab inventory, 7	0.7	DB49	Finished inventory	1.7

Figure 2 Dependency diagrams



Steel plates, which are a finished product of this mill, are produced from slabs by a series of rolling and shearing operations. A steel plate is defined (arbitrarily) by its thickness, which must equal or exceed 0.18 inch. A steel plate can be as much as 160 inches wide and up to 60 feet in length.

The steel mill data system is composed of 16 applications (listed later at the bottom of Table 2). Each application is composed of a set of transaction types. There are 271 on-line transaction types and 284 batch programs in the system. The on-line transactions are used approximately 37 000 times a day.

The 400 million bytes of on-line data in the steel mill system are organized into 40 physical IMS data bases, 18 of which are isolated in that they have no IMS logical relations to other data bases. The remaining 22 data bases are organized into nine groups. Logical relations connect the data bases within each group, but no logical relations connect any group to any other group. The descriptions, codes, and sizes of the data bases are given in Table 1.

Automated analysis

The volume and complexity of the relationships that must be analyzed in the IMS applications are apparent. A detailed analysis of the steel mill system is impractical without data processing support. To this end one of the members of our group, R. M. Gale,

prepared a set of APL programs that provide a comprehensive analysis of the data that are generated by IMS utility programs.[7] Most of the information that is needed for the analysis of dependencies can be obtained with these programs. The functions of these programs are described in the section on distribution procedure.

The APL programs are essential for efficient analysis. Earlier, in a manual analysis of IMS applications in a manufacturing plant, it was necessary to use a sampling method and roughly five times as much human effort to obtain results that are much less detailed than the results described here. Moreover, in the manufacturing application the relations among the applications and the data bases are much simpler, logically, than the comparable relations in the steel mill.[8]

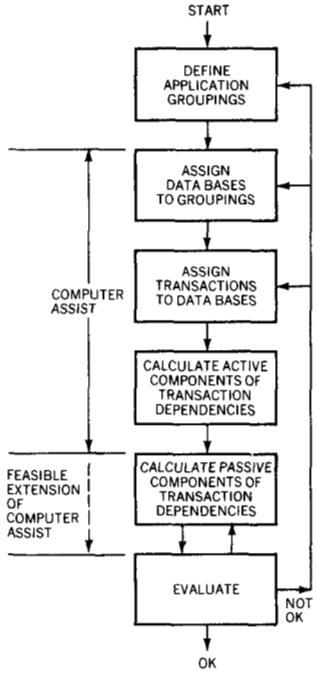
determination of the threshold values

The threshold values that were used to classify the dependencies were determined judgmentally, after a thorough familiarity with the characteristics of the applications had been attained and while several early iterations of the distribution procedure were made.

A graph of all of the dependencies of a distribution is a useful visual aid for these analyses. An example is in Figure 2, which shows six dependencies for two variations of a logical distribution. The variations usually involve the relocation of a few transactions. The percentages shown in Figure 2 are percentages of the total on-line bytes and the total number of on-line transactions per day.

Comparison of Figure 2A with 2B shows that in 2B the points are closer to the horizontal and the vertical axes. It is the closeness of the points to the axes that characterizes the better distribution and leads to the judgmental determination of the threshold values. In the steel mill system analysis a one percent threshold was used. This figure put the active threshold at 370 transactions per day and the passive threshold at four million bytes.

Figure 3 Distribution procedure



Distribution procedure

A pragmatic, iterative procedure was used to distribute the steel mill system—all decisions were made with human judgment. The analysis is based upon a complete record of the IMS system operation for three nonconsecutive days during a single week of June 1978. No further sampling methods were used.

The distribution procedure is shown in Figure 3. Many iterations of this procedure were required before the final distribution was obtained. We describe each step of the procedure but do not describe individual iterations.

Table 2 Update activity table (updates per day)

Data bases	*Applications**										
	A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*
DB01	0	0	312	0	0	0	0	0	0	0	0
DB06	0	122	0	0	0	0	0	0	0	0	0
DB07	0	122	0	0	0	0	0	0	0	0	0
DB08	0	127	0	0	0	0	0	0	0	0	0
DB09	0	44	0	0	0	0	0	0	0	0	0
DB10	0	0	0	0	0	0	0	0	0	0	0
DB11	0	257	0	0	0	0	0	0	0	0	0
DB12	0	0	0	0	0	0	0	0	0	0	0
DB13	0	0	0	0	0	0	0	0	0	0	0
DB14	0	0	0	0	0	390	0	0	0	0	0
DB15	0	0	0	0	5	0	97	239	0	0	0
DB16	0	0	0	54	245	14	143	239	0	0	0
DB18	0	0	0	0	0	2358	0	0	0	0	0
DB19	0	0	0	0	0	2358	0	0	0	0	0
DB20	0	0	0	0	0	0	0	0	0	0	0
DB21	0	0	0	0	0	0	0	0	0	0	0
DB24	0	0	0	0	0	0	0	0	0	0	0
DB25	0	0	0	968	0	1	0	0	0	0	21
DB26	0	0	0	0	0	0	0	0	0	0	0
DB27	0	0	0	0	0	0	0	0	0	0	0
DB28	0	0	0	968	0	0	0	0	0	0	21
DB29	0	0	0	0	0	0	0	0	0	0	0
DB30	0	390	0	0	0	0	0	0	0	0	0
DB31	0	0	341	0	0	0	0	0	0	0	0
DB32	0	0	0	2263	0	416	0	0	572	17	45
DB33	0	0	0	192	0	0	0	0	0	0	0
DB34	0	0	0	1719	0	0	0	0	452	0	66
DB35	0	0	0	0	0	0	0	0	0	0	0
DB36	0	0	0	0	0	0	0	0	0	0	0
DB37	0	0	0	1747	0	0	97	0	500	17	24
DB38	0	0	0	833	0	0	0	0	513	0	24
DB39	0	0	0	0	1453	7	0	0	0	0	0
DB40	0	0	0	0	0	0	0	0	252	0	0
DB41	0	0	0	0	0	0	0	0	442	0	0
DB42	0	0	0	0	0	0	0	0	252	0	0
DB43	0	0	0	0	0	0	0	0	442	0	0
DB44	0	0	0	0	68	403	0	0	252	0	0
DB45	0	0	0	0	68	403	0	0	252	0	0
DB46	0	0	0	0	0	403	0	0	252	0	0
DB47	44	0	0	0	0	0	0	0	0	0	0
DB49	0	0	0	0	0	0	0	0	0	0	0

*Key to applications:
A is primary steel production reporting; *B* is primary steel production planning, 1; *C* is chemical analysis; *D* is work in process reporting; *E* is shipping data services; *G* is order receipt; *H* is order position; *I* is in-process inventory maintenance; *J* is finished inventory maintenance; *K* is plate production reporting; *L* is shipping; *M* is slab inventory maintenance; *N* is slab requirements planning; *O* is slab scheduling; *P* is primary steel production planning, 2; *Q* is labor incentive accounting, primary steel; *R* is primary steel production planning, 3.

define application groupings

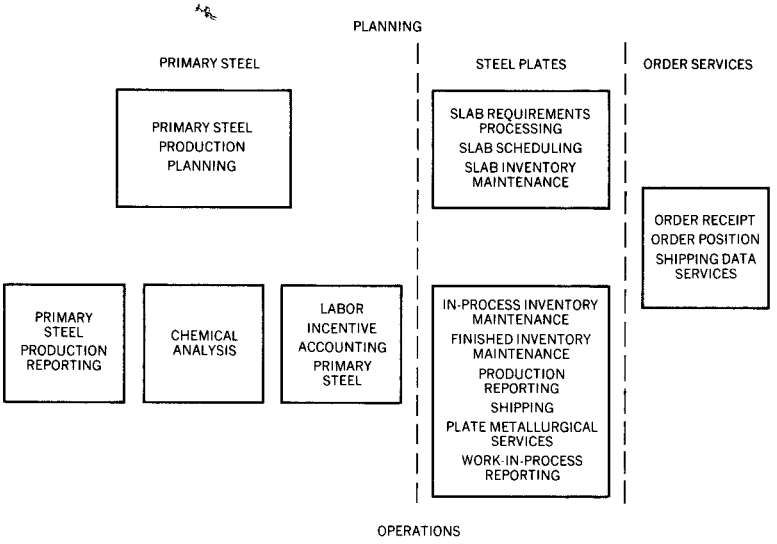
For the first step of the procedure it is necessary to form groups of applications. This requirement involved several considerations. The first and most obvious consideration is that the data system supports two major categories of production activities—primary steel and steel plates. This consideration gives one major partitioning. A second major partitioning is based upon type of work activity—planning and operations. The distinction between planning and operations is based on the time horizon of the activity. Slab scheduling, for example, prepares the sequence of slab

L	*M*	*N*	*O*	*P*	*Q*	*R*
0	0	0	0	0	0	0
0	0	0	0	1055	0	886
0	0	0	0	453	0	1267
0	0	1581	2537	1910	0	886
0	0	0	0	1161	0	886
0	0	1369	0	207	0	0
0	0	0	0	3019	0	0
0	0	0	0	0	0	0
0	0	1465	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	187	0	0	0	0	0
0	187	0	0	0	0	0
0	187	0	0	0	0	0
0	2579	1465	2885	0	0	0
0	1419	0	0	0	0	0
0	1108	1539	2537	0	0	0
0	1193	1413	569	0	0	0
0	1683	1413	2070	0	0	0
0	0	0	0	0	0	1
0	0	0	0	0	0	0
1439	0	0	299	0	0	0
0	0	0	237	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	153	0
0	0	0	0	0	153	0
929	0	0	299	0	0	0
221	0	0	299	0	0	0
22	0	0	0	0	0	0
1742	0	0	0	0	0	0
874	0	0	0	0	0	0
883	0	0	0	0	0	0
864	0	0	0	0	0	0
1156	0	0	0	0	0	0
1434	0	0	0	0	0	0
910	0	0	0	0	0	0
0	0	0	0	185	0	0
0	0	0	0	0	0	0

consumption several days in advance. It is a planning activity. Plate production, however, is reported within an eight-hour interval. It is an operations activity. The classification of the applications by work activity required extensive discussion with persons familiar with steel production and finally a review with members of the data processing staff that developed the system.

Several other applications were put into additional groups that were determined with the aid of a table of the update activity of

Figure 4 Application groupings



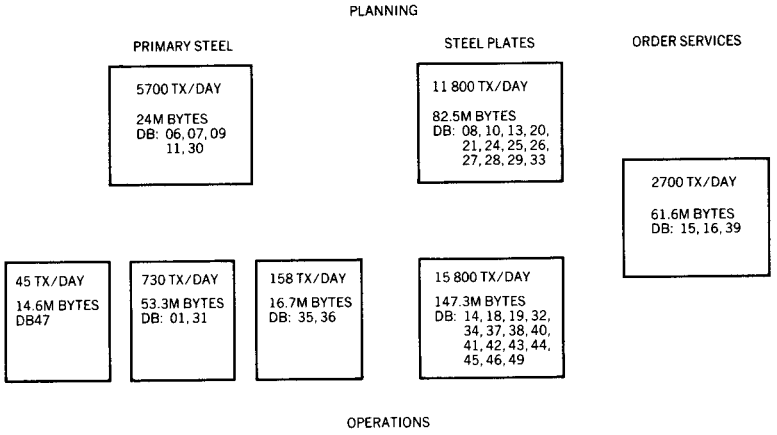
the applications as shown in Table 2. The table was generated by the APL programs. Consider first the chemical analysis application. The table shows that it updates only DB01 and DB31. Moreover these data bases are not updated by any of the other applications. Clearly, the chemical analysis application can be put into a group by itself.

A similar situation exists with respect to the primary steel labor incentive accounting application and DB35 and DB36. Further examination of the table shows that this is also very nearly true for the primary steel production reporting application with respect to DB47. At this point we apply our knowledge of the applications and recognize that these three applications comprise the entire primary steel operations part of the distribution. Hence, for consistency we also put the primary steel production reporting application in a separate group. Finally, the order receipt, order position, and shipping data services applications were identified as a separate group through the use of an experimental clustering algorithm. The final structure of the groupings is shown in Figure 4.

assign data bases

The next step of the distribution procedure, which is the first of the APL processing steps, is to assign the data bases to the application groups according to their update activity. A data base is assigned to the application group that performs the most updates upon it. The assignment of the data bases is shown in Figure 5. The assignment of the data bases to application groups was constrained so that no IMS logical relations link any groups. Thus, logical relations are wholly contained within their own group.

Figure 5 Transaction and data volumes and data base groupings



This step of the procedure produces the basic structure of the logical distribution. It is for this reason that the data base assignments are based upon update activity, which tends to minimize the number of dependencies that involve updates. This becomes significant when implementations are considered, since one of the implementation options involves the use of copies of data to provide local (within the group) support for dependencies. Obviously it is desirable to minimize the need for data copies that must be updated, since this can involve some difficult control problems.

assign transactions

At this point we have a grouping of applications and data bases. In the next step of the APL processing of the procedure the initial groups of applications are ignored, and the transaction types are assigned to data bases according to their total activity—read-only and update. This assignment yields a grouping of transactions that strongly resembles the initial application groups but differs in that a few transactions are in different groups. The number of daily transaction occurrences and the amounts of data contained are shown for each group of the final logical distribution in Figure 5.

calculate active components

The last of the APL processing steps calculates the active components of the transaction dependencies. The results within each dependency are listed by transaction type code and show the frequency of occurrence daily, remote data segments accessed, and the type of access—read-only or update. A sample list is shown in Table 3. A summary table for all active components of the dependencies is shown in Table 4.

The distribution that is attained at this stage of the procedure is based upon the activity of the transactions only. It is determined

Table 3 List of active components of transaction dependencies

Tx Code	*Tx/Day*	*Segments accessed*				
SP1C80	61	DB081 R	DB082 R	DB083 RU		
SP1C82	5	DB081 R	DB082 R	DB083 RU	DB084 RU	
SP1C84	14	DB081 R	DB082 R	DB083 RU	DB084 RU	
SP1C90	14	DB101 R				
SP1T26	5	DB081 R	DB082 R	DB083 RU	DB084 RU	
SP2P65	268	DB081 RU	DB082 RU	DB083 RU	DB084 RU	DB101 RU
SP2P74	17	DB101 R				
SP2P80	810	DB081 R	DB082 R	DB083 RU		
SP2P82	47	DB081 R	DB082 R	DB083 RU	DB084 RU	
SP2P86	207	DB081 RU	DB082 RU	DB083 RU	DB084 RU	DB101 RU
SP3T22	838	DB081 R	DB082 R	DB083 RU	DB084 RU	
SP3T29	48	DB081 R	DB082 R	DB083 RU	DB084 RU	
	2334	= Active component of dependency				

Dependency: From Primary Steel Planning Group to Steel Plate Planning Group.
Data usage key: R = read-only; RU = read or update.
Key to DB code: DB08① ← Segment 1.

Table 4 Active components of dependencies

		TO						
		1	2	3	4	5	6	7
FROM	1	0	2334	0	0	0	0	185
	2	2938	0	3079	470	353	0	0
	3	0	0	0	279	260	0	0
	4	0	496	0	0	0	0	0
	5	0	951	2223	805	0	0	0
	6	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	0

Key:
1 = Primary Steel Planning
2 = Steel Plate Planning
3 = Order Services
4 = Chemical Analysis
5 = Plate Operations
6 = Primary Steel Labor Incentive Accounting
7 = Primary Steel Production Reporting

initially by update activity and then modified by the total activity, read-only plus updates.

calculate passive components

The next step in the procedure is to calculate the passive components of the transaction dependencies. The data required include a description of the data bases—number of child segment occurrences per parent, field descriptions by segment type, field sizes, and hierarchical structure. Also required is a list of field usage by transaction type.

The passive components are calculated manually, although APL programs could also be written for this step. The starting point is

the list of active components, as in Table 3. These transactions are put into two groups—those whose occurrences exceed the active threshold and those whose occurrences do not. The passive components are first calculated for the group with occurrences that exceed the active threshold. If a passive dependency exceeds the passive threshold, another iteration is required. If all passive dependencies in the group fall below the passive threshold, the group is consolidated. If the passive component of the consolidated dependency exceeds the passive threshold, another iteration is required. Otherwise we continue with the remaining transaction dependencies.

The passive components of the remaining transaction dependencies are now calculated. These are either LL or LH dependencies, since their active components are all less than the active threshold. Those in the LL category are consolidated. If the resulting consolidated dependency is HH, another iteration is required. Otherwise we continue with the remaining LH transaction dependencies, which are consolidated. If the resulting consolidated dependency is HH, another iteration is required. Otherwise the logical distribution is finished.

The actual calculations of the passive components first are made with a single data base record. This calculation yields the maximum number of bytes that a transaction type could require and is expressed as a percentage of the data base record size. This percentage is applied to the size of the entire data base to give the size of the passive component of the transaction dependency for that data base. The procedure is repeated for each data base that is accessed by that transaction type. The sum of the individual data base components gives the total passive component of the transaction dependency.

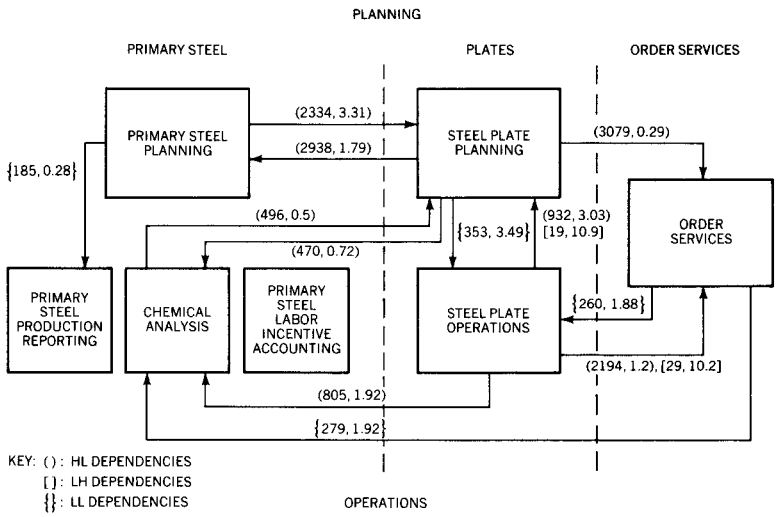
The final logical distribution was produced after more than ten iterations of the procedure. The changes in successive iterations included the redefinition of application groups, movement of individual transaction types from one group to another, and the movement of individual data bases from one group to another.

The logical distribution is shown in Figure 6. The distribution has 14 consolidated dependencies, eight HL, four LL, and two HL. Figure 7 is a plot of these dependencies. Note that one group, primary steel labor incentive accounting, has no on-line dependencies. It is included in the logical distribution because of its batch processing dependencies, which are not part of this analysis.

use of the logical distribution

A logical distribution can be used in several ways. It can provide a basis for a physically distributed system, either in the original application environment or in similar environments. Some of the

Figure 6 Final logical distribution



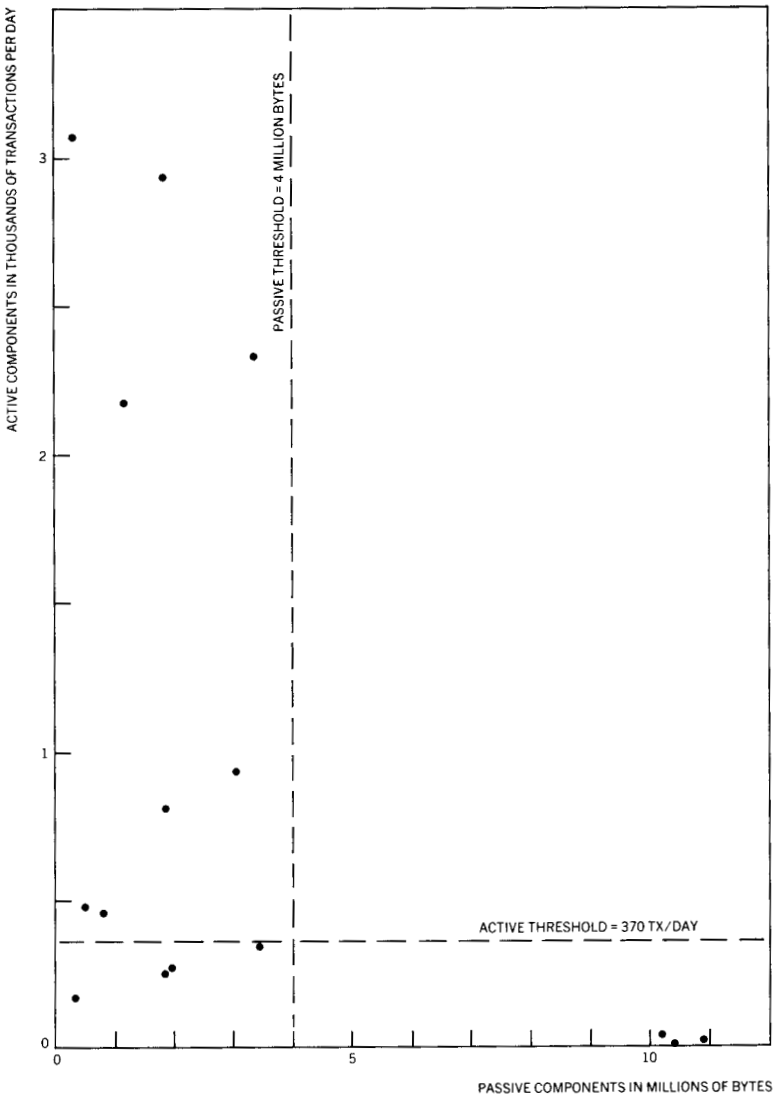
groups of the logical distribution would probably be combined for a physically distributed implementation. Such combinations, which we call *derived distributions*, can be made in many ways. Some considerations that affect the choice of combination are load balance, total amount of data copied, availability, and administrative convenience, where local user groups have their own processors. Other uses of the logical distribution include the organization of application development activities and application maintenance.

We illustrate the evolution of a derived distribution from our logical distribution with a hypothetical example. Suppose the steel mill management wanted their system divided into two systems: one for primary steel and the other for plates and order services. The obvious division on this basis is shown in Figure 8A. Note, however, that this derived distribution requires the support of six dependencies across the interface between the two systems. However, four of these dependencies are related to one small grouping in the primary steel system—the chemical analysis grouping. If this grouping were moved out of the primary steel system and into the plate/order services system, it would then be necessary only to support two, rather than six, dependencies across the interface. The new derived distribution is shown in Figure 8B. It could provide a reasonable basis for a distributed implementation.

Dependency support

Once a derived distribution has been defined, decisions must be made about the means for supporting its dependencies. There are

Figure 7 Dependency diagram for logical distribution of on-line steel mill system



two choices: data communications or data duplication, each of which has advantages and disadvantages. An important advantage of data communications support is data currency and integrity; since the data are not duplicated there is just one valid set of records in the system. However, there are possible disadvantages to data communications support. These include questions of availability and excessive response time, due either to communications delay or to file overload—too many people using the same file.

The pros and cons of the use of data duplication as a means for supporting dependencies are the converse of the data communi-

Figure 8 Derived distribution

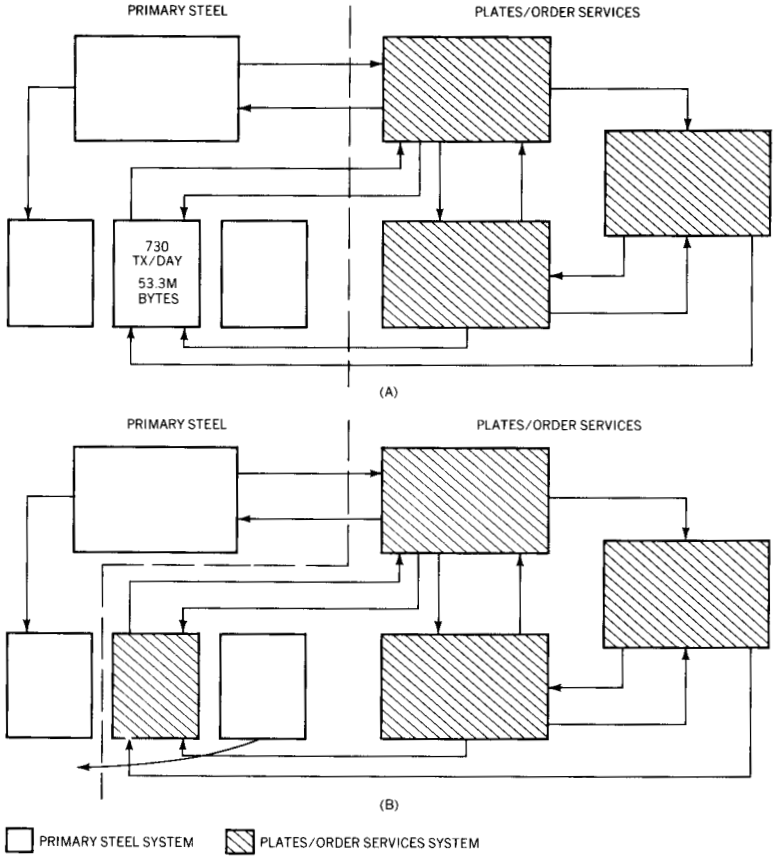
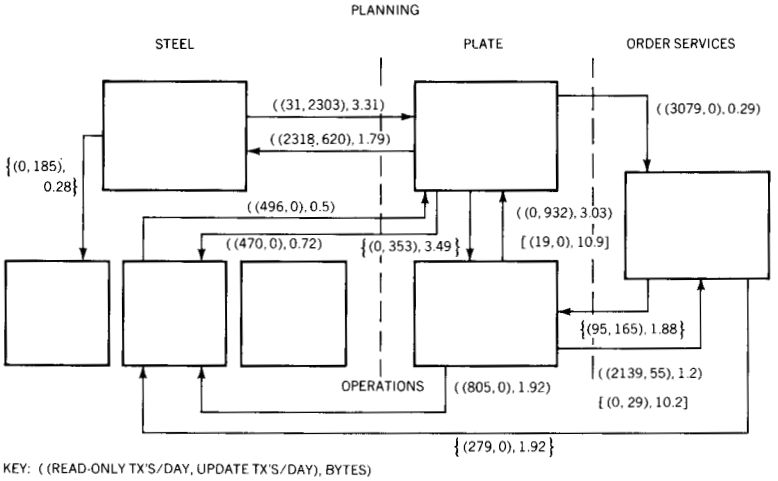


Figure 9 Dependencies—read-only or update



KEY: ((READ-ONLY TX'S/DAY, UPDATE TX'S/DAY), BYTES)

cations considerations cited above. Duplication of data raises questions about data currency and integrity, particularly when copies are updated. However, the use of copies enhances the autonomy of a logical grouping and therefore improves system availability, local management control, and response time.

It is evident that the selection of data communications or data duplication to support the dependencies of a derived distribution can be a complex problem that involves many trade-off decisions. The logical distribution and its various alternative derived distributions can aid system designers by clarifying their options.

concurrency control of copy updates

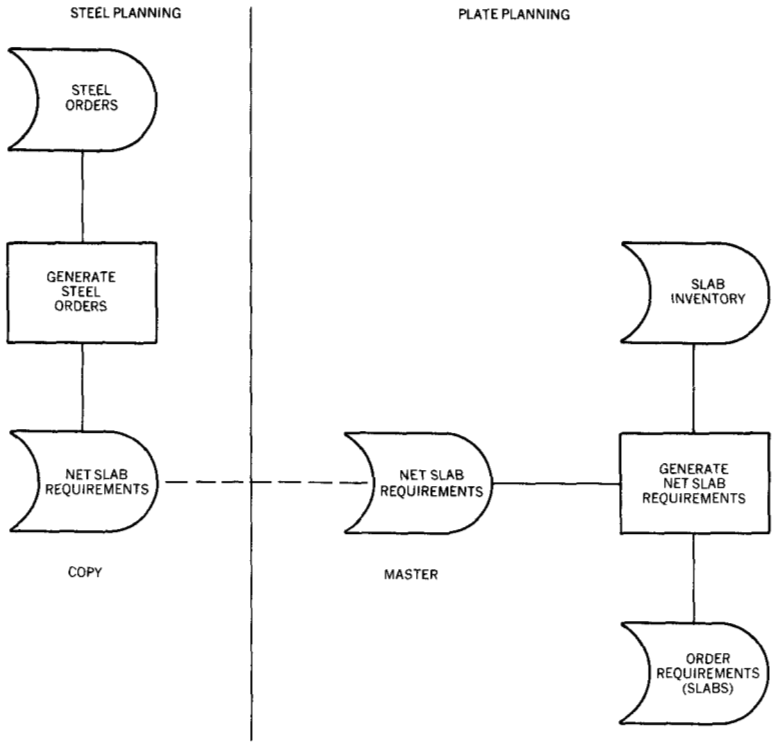
When dependencies are supported by the duplication of data, there are questions of data concurrency and integrity to be considered, particularly when updates are applied to the copies. To pursue this question with the steel mill, consider the logical distribution as in Figure 9, where the active components of the dependencies are separated into read-only usages and update usages. Examination of these data shows that in 11 of the 14 dependencies the update part of the active component of the dependency is either zero or below the one percent threshold. For these 11 dependencies we can post an interim update to the copy and send the update transaction to the master data base, where the true update is applied. Periodically the copy is refreshed from the master, after which the preceding interim updates are discarded. Obviously these copies also support the read-only transactions. With this arrangement, the update control of the data is maintained by the node that contains the master data base.

The significant update dependencies that remain are between primary steel planning and plate planning, and between plate planning and plate operations. For both interfaces we find the same result—it is never necessary for a given item of data to be updated simultaneously by two or more logical groupings. We illustrate this with the two planning systems.

Consider Figure 10. Here the plate-planning system is generating net slab requirements and posting them to the net slab requirements master file, which is an IMS physical data base that is organized by steel grade. The primary steel planning group uses a copy of the net slab requirements file and develops new steel orders—orders for the production of heats of steel. The system has a natural logical sequence that results in the update authority for a given data item being passed back and forth between the two planning systems, thus precluding the need for any simultaneous updating by the two systems. The logical flow is described with reference to Figure 11.

Update copy control is maintained by a system of flags in the root segments. The plate-planning system can read any data but may

Figure 10 Planning data and functions

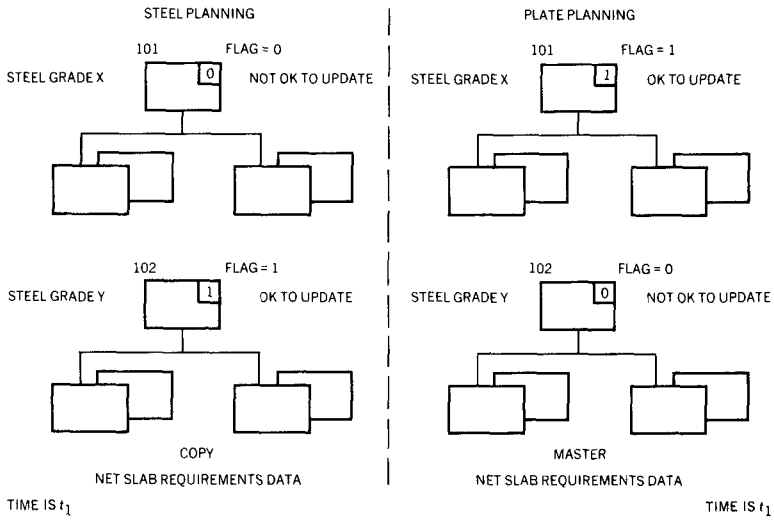


update only those data base records that have their flags set to one. Net slab requirements are developed within one steel grade at a time. When the slab requirements are completed for a grade of steel, the data base record, with its flag set to one, is sent to the primary steel system where it is put into the copy file. The same record is stored in the master file with its flag set to zero.

The same logical control applies to the primary steel-planning system. Only those data base records in the copy that have their flags set to one can be updated. After the new steel orders have been prepared, by grade, the updated net slab requirements data base record is returned to the plate-planning system, with its flag set to one. It is also put into the copy data base with its flag set to zero. Thus the most current version of the data base record, updated by the primary steel-planning system, has returned to the plate-planning system, ready for another processing cycle.

From the foregoing discussion we see that the update authority for any one data base record passes systematically back and forth between the two planning systems. There is no need for simultaneous updates. In this way the copy updating is controlled naturally by the application requirements. A similar logic applies to the copy update activities across the interface between the plate planning and plate operations groupings.

Figure 11 Update control flags



extended dependency measures

We have used a basic but relatively simple measure of the dependency between two groupings of a logical distribution. It seems likely that in the future more complex measures will be required. Resource utilization data, such as the number of data accesses per day, might be related to the active part of the dependency. Probabilities of data item usage could be attached to the passive part. Deferral of transactions is another means for characterizing dependencies, since some transactions must be processed immediately, while others can wait. The pattern of the traffic intensity of the transactions through the day suggests yet another way of characterizing dependencies. There are many other possible characterizations—copy dissemination delay, copy update traffic intensity, multipoint update simultaneity, etc. It is clear that the task of describing dependencies can be complex; it has been treated here in a relatively simple way.

Summary

A major goal of distributed system design is to bring data processing functions and data closer to the users than is possible with a centralized design. The expected advantages of this arrangement, such as local autonomy, responsiveness, etc., have been expressed at length in the literature on distributed data processing.

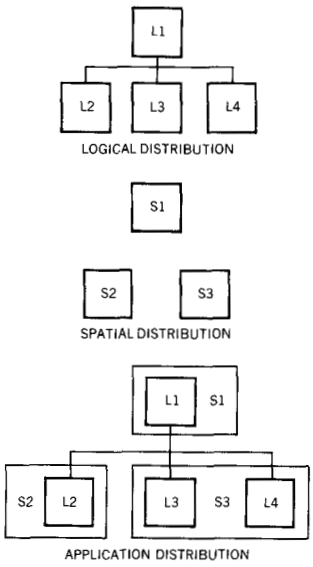
However, we have noted that a node of a distributed system does not function in a totally isolated, stand-alone mode. Other nodes depend upon it or it depends upon other nodes. If these interdependencies are too strong, the sought-after advantages of the distributed data processing system are lost. It is essential, therefore, that such a system be designed so that the nodal inter-

dependencies are at an acceptably low level. We have illustrated one approach to this task.

ACKNOWLEDGMENTS

The author is indebted to R. M. Gale, IBM Poughkeepsie, for his patience and cooperation in providing computing support, via the APL programs, for the numerous iterations that the distribution analysis required and for development of the clustering algorithm. The author is also indebted to R. Richards, IBM White Plains, for his excellent explanations of the intricacies of the steel business. An expression of thanks also goes to H. S. Arora, R. M. Gale, and B. Roth, IBM Poughkeepsie, for their many constructive comments about this paper. Finally we acknowledge the support of Dr. B. D. Rudin, Manager of the IBM Scientific Center in Los Angeles, whose foresight made this work possible.

Figure 12 Application distribution structure



Appendix: Definition of a distributed data processing system

At present there is no universally accepted definition of a distributed data processing system; moreover, there may never be one. It is conceivable, however, that a widely accepted definition will evolve for civilian data processing environments. The role of applications will be prominent in such a definition and is discussed thoroughly in References 3 and 4. Our purpose in defining the logical distribution is to provide the application component of a distributed data processing system definition.

The definition that follows is intended to stimulate interest, to identify the elements of a distributed data processing system definition, and to illustrate what such a definition might look like. It is not our intention to impose this definition upon anyone.

A *logical distribution* is a partitioning of a collection of related applications and their data into the maximum number of groups that have a specified low level of interdependence. A logical distribution is composed of at least two groups. Every proper subset of the logical distribution has an interdependency with at least one group not in the subset.

A *node* is a well-defined volume of space. Two nodes have no points in common.

A *spatial distribution* is a set of two or more nodes.

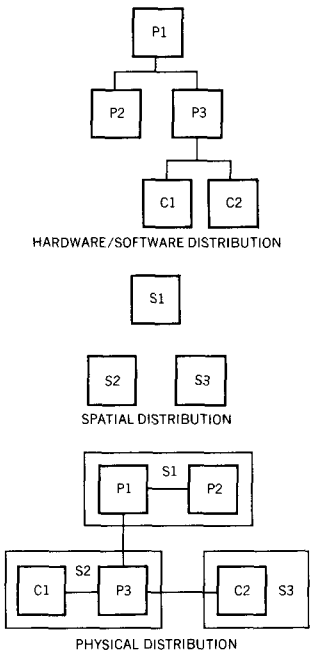
An *application distribution*, Figure 12, is composed of a logical distribution, a spatial distribution, and a relation between them. The relation is such that logical groups must be contained in at least two nodes and each logical group is put into one and only one node of the spatial distribution.

A *hardware/software distribution* is composed of a processor and one or more processors or controllers and their associated software. The processors/controllers must have the capability to provide at least one, possibly indirect, communication path between every pair of processors/controllers in the hardware/software distribution.

A *physical distribution*, Figure 13, is composed of a spatial distribution, a hardware/software distribution, and a relation between them. The relation is such that every node in the spatial distribution contains at least one processor or controller of the hardware/software distribution and each controller or processor of the hardware/software distribution is in one and only one node of the spatial distribution.

A *distributed data processing system* is composed of an application distribution and a physical distribution, both of which are defined with respect to the same spatial distribution. Mathematically, the structure of a distributed data processing system can be described by a pair of functions: one from the logical distribution into the spatial distribution and the second from the hardware/software distribution onto the same spatial distribution.

Figure 13 Physical distribution structure



CITED REFERENCES

1. A. L. Scherr, "Distributed data processing," *IBM Systems Journal* **17**, No. 4, 324-343 (1978).
2. P. J. Down and F. F. Taylor, *Why Distribute Computing?* NCC Publications, National Computing Centre Limited, Manchester, U.K., U.S. distributor—Hayden Book Company (1976).
3. P. C. Howard, Editor, "Performance implications of distributed systems—Part 1," *EDP Performance Review* (Applied Computer Research, Phoenix, AZ) **6**, No. 8, 1, 6 (August 1978).
4. P. C. Howard, Editor, "Performance implications of distributed systems—Part 2," *EDP Performance Review* (Applied Computer Research, Phoenix, AZ) **6**, No. 9, 1, 12 (September 1978).
5. C. J. Jenny, *Partitioning and Allocating Computational Objects in Distributed Data Processing*, Research Report RZ 984, IBM Corporation (October 1, 1979); available through the local IBM branch office. (ITIRC No. 79A 7097)
6. T. Lawson and M. P. Mariani, "Distributed data processing system design—A look at the partitioning problem," *COMPSAC '78 Proceedings*, IEEE, New York, NY (1978).
7. R. M. Gale, *A Methodology for Determining Logical Nodes in a Distributed Data Processing System*, Technical Report TR 00.3025-1, IBM Corporation (September 17, 1979); available through the local IBM branch office. (ITIRC No. 79A 6109)
8. H. S. Arora, C. T. Baker, and B. Roth, *Distribution of a Centralized On-Line Manufacturing Data Base System*, Technical Report TR 00.3023, IBM Corporation (August 23, 1979); available through the local IBM branch office. (ITIRC No. 79A 5996)

The author is located at the IBM Data Systems Division laboratory, P.O. Box 390, Poughkeepsie, NY 12602.

Reprint Order No. G321-5121.