Issues and methods for practical distributed data processing applications—II

by MAURICE BLACKMAN

Arthur Andersen & Co.
Houston, Texas

and

HUGH RYAN

Arthur Andersen & Co.
Chicago, Illinois

ABSTRACT

This paper presents methods for the second of two key activities in the creation of practical distributed data processing (DDP) systems for business computing. The first activity, discussed in a companion paper, is to select a configuration of hardware and software to support the implementation of a multiapplication plan. The second, discussed in this paper, is to select data distribution and manipulation approaches for one application within the limits set by the results of the first activity. The paper assumes the definition of DDP established in the prior paper. It identifies some of the issues that constrain a commercial establishment with limited research funds and that justify the limitation of practical alternatives assumed as a basis for the methods.
INTRODUCTION

This paper continues the discussion of methods for designing practical DDP systems in view of the proven technology available to the commercial user. This discussion commenced in Paper I of this two-paper series. In the first paper a definition of DDP systems was selected to include all systems that would be recognized as DDP by a commercial user. A method for developing a DDP strategy was described. The definition is assumed for this paper, and the results of the strategy selection are assumed to be implemented. This paper addresses the second key task: data design.

Challenges and Gaps

The basis for the method proposed in this paper is a desire to select practical approaches to applications. We are seeking to avoid substantial effort in two areas:

- Overcoming the logical challenges of subdivided but dependent processing through additional application development
- Filling technical gaps with substantial system software development

The challenges and gaps include, for example:

- The correct handling of transactions found to be invalid at one node after having been validated, posted, and acted on at another node earlier in a cycle
- The maintenance of data integrity in answers to inquiries needing reference to several nodes while the late nodes in the sequence are continuing to post other transactions, unaware of the impending inquiry
- The preservation of the integrity of locally developed files related to local copies of centrally maintained files
- The preservation of the integrity of a distributed application that posts transactions on line at more than one node, so as to be able to back out a whole transaction upon the failure of any node, its database, or the communications network between them

The list goes on. These issues are substantial. In some cases they are only imperfectly understood. In time some will be routinely handled by software, especially when high-capacity data communications reduces the time for internode communication. For the present we believe that the average MIS director is better advised to avoid them.

The Complexity of Design for Distributed Data Processing

The system development process for a centralized environment includes the following segments:

- User requirements and functions are identified.
- A data design is defined on the basis of the requirements and functions.
- A technical architecture is designed by consideration of data design and business functions.
- A systems design is detailed from the technical architecture, using a combination of data and function-driven structured design.
- The entire design is implemented and converted.

This process is not trivial for a centralized design. The distributed environment adds an entire dimension to the design problem (Figure 1): the geographically distributed nodes of the network.

![Figure 1—Design process in a distributed development](From the collection of the Computer History Museum (www.computerhistory.org))

The design process must be applied to each class of node in the network. It is not sufficient to repeat the design process for each distinct type of node, because a design decision about one type may affect a decision about another. For example, a decision to allow changes of hourly wages at a central node may affect the timing, and even feasibility, of computing and printing paychecks at remote nodes.

Design in a DDP environment presents a set of complex design problems that require a well-structured approach to make the required decisions in a logical sequence. The methods presented address techniques for defining data allocation and operational data movement for a DDP system. However, they also affect functional analysis and the person/machine boundary.

These tasks take place in a context of other activities in the design process, as shown in Figure 2.
DATA ALLOCATION

The Process of Data Design

The designer begins the process of data design by identifying an initial data model. Then, for each business function to be supported by the machine, the designer must do the following:

1. Identify the aggregates and relationships required to support the function as a business function data model (BFDM).
2. Identify fields required to support the business function.
3. Assign fields to appropriate data aggregates in the BFDM.
4. Merge the BFDM into the prior project data model. This may require identifying new aggregates and/or relationships in the project data model.
5. Merge the fields into the appropriate aggregates of the project data model.

These steps of data design apply in both a centralized and a distributed environment.

Data design requires one additional step for a distributed environment:

6. On completion of the project data model (exhausting all business functions), minimize communication across nodes within each business function data model. (The basis for this additional step will be discussed in the section "Data Movement." )

The following is an example of the use of the method of data design for a distributed processing environment. Although it has been simplified by reducing the number of functions considered, it illustrates the key points of the approach. The example is based on parts distribution, in which computers at widely separated warehouses are used to keep track of local inventory. Stock status is reported centrally each week to support purchasing and allocation to warehouses. Predicted delivery data is then returned to the warehouses.

Example of Data Design for Parts Distribution Control

Consider two of the system's business functions: relieving inventory and purchasing new items (Figures 3 and 4).

Figure 3—Function chart for "Relieve Inventory Stock"

Figure 4—Function chart for "Order New Stock"
Next consider function 1.2, "Review Inventory Stock Status for Available On Hand." The BFDM (Figure 5) suggests that once the appropriate stock description is found, the warehouse stock status record is obtained. Fields are assigned to the warehouse stock status, including:

- Stock item identity
- Warehouse
- Short description
- Amount available

The existing project data model can satisfy this function, although the entities are now required by the local site.

Next consider the purchase function 2.0, "Order New Stock" (Figure 4). Examination of the related screen layouts and business function shows that a warehouse stock status entity and a purchase order entity are required. The BFDM suggested by this function is shown in Figure 6. The warehouse stock status gives the amount on hand and records the amount allocated to the warehouse. The purchase order entity records the purchase. In addition, it is related to the warehouse stock status. The relationship tells what purchase orders exist for a given item.

Merging this business function data model (Figure 6) with current project data model gives a new project data model (Figure 7).

Assigning Data

If one reviews the business function data models, it can be seen that the warehouse stock status is needed both at a local node, for relieving inventory, and at the central site, for purchasing. If only one version of the warehouse stock status entity is defined, one of the BFDMs will need to cross nodes.

Few approaches are available for minimizing the number of cross-node communications. For the business functions that require crossing nodes, the options are as follows:

- Replicate the data aggregate at each node.
- Copy a portion of the data aggregate at a node.
- Partition the data aggregate across nodes.

In replicating the data aggregate, one stores a copy of the data aggregate at all sites where it is required. In our example, we could replicate the warehouse stock status at the local and central sites.

Replicated data are appropriate in cases where any of these conditions applies:

- Most of the data of the aggregate are used.
- Planned data use is periodic.
- Noncurrent data have small impact.

Copying a portion of the data aggregate is a variation of the replication option. The identity of the data aggregate may be lost when a portion is redundantly stored. For example, we can store the purchase quantity allocated to a warehouse in the warehouse stock status record. Thus, data from the purchase order entity are redundantly stored in the inventory stock status entity. The identity of the purchase order entity, however, is lost at the remote site. Copied data is appropriate in cases where only a small portion of a data aggregate is used.

The third option is partitioning data, i.e., storing the data for a node only at the node. For example, the warehouse stock status could be partitioned by warehouse. Partitioned data are appropriate when the data can be clearly identified with a
given node type. The partitioning option often seems desirable, although it can increase the complexity of the design effort significantly.

The above suggests the project data model shown in Figure 8. Note that the geographic dimension has been introduced into the project data model.

![Figure 8—Project data model for distributed inventory processing](image)

**Structuring the Decision Process**

We are using a relatively simple data design problem for illustration purposes. Full-size problems need a more formal documentation tool. The form shown in Figure 9 has proved effective in the mapping process.

To use the form, the data aggregates of the project data model are listed on the form. Then all BFDMs that use the data aggregate are listed and cross-referenced to the appropriate source. Next, the sites where the business functions are performed are identified. When there is a mix of nodes in the "Where" column, a mapping approach must be defined. As noted, the options are as follows:

- Replicated data
- Redundant data
- Partitioned data

In addition, one can choose not to avoid crossing nodes and use messages instead. Messages are online transmissions of data between nodes. They are appropriate when use of the data is unpredictable and the data must be current.

**Concerns in Mapping the Project Data Model to Nodes**

A casual reading of the above would suggest that the DDP environment is accommodated by only filling out a few forms.

<table>
<thead>
<tr>
<th>DATA AGGREGATE</th>
<th>BUSINESS FUNCTION DATA MODEL</th>
<th>MAPPING</th>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warehouse Stock Status</td>
<td>Believe Inventory</td>
<td>Local</td>
<td>On-Line</td>
</tr>
<tr>
<td></td>
<td>Create Purchase Order</td>
<td>Central</td>
<td>Batch</td>
</tr>
<tr>
<td>Purchase Order</td>
<td>Create Purchase Order</td>
<td>Central</td>
<td>Batch</td>
</tr>
<tr>
<td></td>
<td>Check Available Stock</td>
<td>Local</td>
<td>On-Line</td>
</tr>
<tr>
<td>Stock Description</td>
<td>Believe Inventory</td>
<td>Local</td>
<td>On-Line</td>
</tr>
<tr>
<td></td>
<td>Stock Description Change</td>
<td>Central</td>
<td>On-Line</td>
</tr>
</tbody>
</table>

![Figure 9—Inventory control technical architecture](image)
This is not the case. As has been discussed, the DDP environment is quite primitive today compared to centralized online/DBMS environments. The system software, when available, is not very competent. Performance, when moving data between multiple machines, is often inadequate. Restart/recovery typically consists of having the user repeat work or try another approach, such as the telephone. The mapping suggested by Figure 9 must be performed in full awareness of node-machine capabilities, transmission software capabilities, business function volumes, and business function requirements (versus desires).

Data allocation is a key step in the definition of a DDP design. The data design approach needs a single extension for mapping the data design to the distributed network. The objective recommended is to minimize cross-node communications for BFDMs. The next step is to define the approach to data movement for a DDP environment.

DATA SYNCHRONIZATION

Selecting Data Movement Approaches

Data movement is defined for this paper as follows: a description of machine processes that relate data structures. The description specifies what the processes are, when the processes are run, and where the processes are run.

Inputs and outputs from the selection of data movement

The key inputs and outputs for selecting data movement shown in Figure 10, are as follows:

- The project data model
- Definitions of the application functions
- A description of performance and security requirements
- A description of the hardware/software environment from the strategy selection, that identifies hardware at each site, software (compilers, online monitors, database management systems) at each site, transmission hardware between sites, and transmission software to move data between use.

A thorough understanding of the capabilities and limitations of the hardware and software is necessary. Because of the rapidly changing hardware/software environment and the high cost of software development and maintenance, it is worthwhile to stay inside the constraints of available products.

The process of selecting data movement approaches

Relating data structures in a distributed environment means keeping the data synchronized (i.e., ensuring that at certain times the data at different sites have consistent and reasonable values). Thus, the process of selecting data movement for a DDP system is equivalent to defining the synchronization approach to be used for the system.

This task can be time-consuming, involving many decisions and reconsiderations. At this stage in a design, functions or features may need to be changed, reduced in capability, or completely foregone. Success is the result of creativity and compromise. Since the task is iterative, it is worthwhile to structure the design process.

The technical architecture form previously illustrated (Figure 9) is a means for structuring the process. As shown in Figure 11, the form can be extended with two additional columns. One column describes the synchronization requirement; the second describes the synchronization approach.

There are relatively few approaches available for synchronization:

- Transmit a message.
- Transmit a transaction file.
- Transmit a master file.
- Use the telephone.

Synchronization using messages

Synchronization using messages refers to a method of immediate transmission of data to other sites. An example of the software and hardware for this approach is shown in Figure 12. The example is based on a DDP network with IBM 8100s and a 370. In this environment, the 8100 online monitor DTMS provides an interface to CICS or IMS on a 370. A message can be sent from DTMS to CICS. The message is processed in CICS and a reply sent to the 8100.

The transmission of messages presents some significant problems in today's environment. One problem is record protection. When a record is acquired for update at a node, some currently implemented mechanisms provide no lockout protection for accesses from other nodes. Some implemented protection mechanisms fail to handle a program failure at one of the participating nodes. Finally, implemented schemes may require so many exchanges to acknowledge approval that performance is not acceptable.
### Performance, even without regard to protection, is a concern when using messages. As the example illustrates, a message involves the overhead of entry into two online monitors. In addition, there is the time for transferring data over lines between the sites. For large volumes of data, the line time can be significant. Thus the total time for a response from a message between processors may quickly become unacceptable to the user.

A third problem with messages is sensitivity to a node failure. In Figure 12, if the central site fails, the remote 8100 programs using the central site cannot run. One of the benefits of DDP, reliability through independence of sites, is lost when sites are connected through the use of messages. The problems posed by messages in DDP are significant. Messages should be used primarily for exception conditions. If, as the design evolves, it becomes obvious that many transactions require messages, use of a centralized system should be reconsidered.

#### Synchronization using a transaction file

Synchronization using a transaction file involves sending a queued set of transactions to other sites for asynchronous processing. Figure 13 gives an example of such a configuration:

- Personnel data are maintained at the remote site.
- Changes are posted to the local site database (1).

### Figure 11—Technical architecture extended

<table>
<thead>
<tr>
<th>DATA AGGREGATE</th>
<th>BUSINESS FUNCTION</th>
<th>X-REF.</th>
<th>DESCRIPTION</th>
<th>WHERE</th>
<th>MODE</th>
<th>MAPPING</th>
<th>SYNCHRONIZATION REQUIREMENT</th>
<th>SYNCHRONIZATION APPROACH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warehouse Stock Status</td>
<td>Relieve Inventory</td>
<td>Local</td>
<td>On-Line</td>
<td>Partition</td>
<td>1</td>
<td>Data should be current as of when data is removed from inventory.</td>
<td>Synchronous on-line updating of transaction.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Create Purchase Order</td>
<td>Central</td>
<td>Batch</td>
<td>Replicate</td>
<td>Data should be current as of end of business on the day when purchasing runs.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purchase Order</td>
<td>Create Purchase Order</td>
<td>Central</td>
<td>Batch</td>
<td></td>
<td>Data should be current as of end of day.</td>
<td>Do updates in batch.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Check Available Stock</td>
<td>Local</td>
<td>On-Line</td>
<td>Redundant</td>
<td>Data should be synchronized with last purchase order.</td>
<td>When purchasing runs, create a trans file of P.O. to send to local site for updating.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stock Description</td>
<td>Relieve Inventory</td>
<td>Local</td>
<td>On-Line</td>
<td>Access local stocks.</td>
<td>Data needs to be current as of beginning of business at all local sites.</td>
<td>Transmit trans file of changes. Local site will post changes on overnight batch run.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stock Description Change</td>
<td>Central</td>
<td>On-Line</td>
<td>Redundant</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Figure 12—Synchronous message transmission between 8100 and 370 host
• A transaction file is maintained of personnel data changes made at the local site (2). An example of such change is the change of contact in event of an emergency.
• Periodically, the changes are sent to a central site for updating (3) and retransmission to all other sites (4).

Figure 13—Example of transaction file synchronization

Transaction file—asynchronous processing

In a DDP environment, this technique for synchronizing data structures is common. However, it presents some significant problems.

First, since data are posted immediately at the local site, they are not synchronized with the rest of the network. This method may be acceptable or even desirable, but it does mean that while the transaction is queued, different sites will possess discrepancies in data. The user needs to understand this and recognize it as a planned part of the design and not a system error.

Second, is a potential for conflicting transactions. In Figure 14, two sites are entering an emergency contact address for the same individual. It is accepted at each site. However, at the central site it is not clear which transaction is correct. This is a common problem with the use of transaction files, and many approaches exist to deal with it, such as designating the site with primary responsibility or timestamping. Every conflict should be reported, whatever the technique.

Another problem then arises: who is to receive the error report when two sites are involved. Once again, many approaches are possible:

• Report to the site causing the problem.
• Report to the site receiving the problem.

No general solution exists. In any approach, the impact of asynchronous processing of the transaction needs to be evaluated. In each case, responsibility for the resolution of problems should be identified and some means of follow-up defined.

Transaction file—transmission software

A second major problem with transaction files involves the development of transmission software to move the data files around the network. Such software must be able to do the following:

• Preschedule file transmissions.
• Send and receive sequential files between sites.
• Detect and report on error transmissions.
• Provide some form of a queuing mechanism to hold files until a site is prepared to accept them.

IBM's DSX software has some of these features and can be viewed as a representative example. This software has been undergoing design enhancements for over three years, which indicates that such transmission software may be a major effort by itself.

Synchronization using master files

Synchronization using master files resembles the use of transaction files, except that master file records are sent rather than the transactions causing the changes.

This technique does have some drawbacks. First, if the master file is large, performance may not be acceptable. The second problem involves the discontinuity of the file transmission. If files are not synchronized between two sites, the sudden revision of the file at a site may be disconcerting to a user. For one company, the central inventory files were used to maintain inventory balances. These balances were assumed to be correct and were used each month to update remote inventory balances. When the update occurred, an error report was produced of items with differing amounts, and an inventory check was made to resolve discrepancies. The discrepancies created a loss of confidence in the system, and there was user resistance to having "their" data overwritten by central-site data.

Another problem is the need to halt the application. Typically, while master files are being loaded, there are oper-
ational and application problems in running the application. Specifically, if changes are made while a file is being loaded, it may be difficult to predict which sites have what master file. To avoid this problem, the entire application must be stopped while loading occurs. Since users are reluctant to lose use of a system, even for short periods, and the software and operations procedures used to shut down the system may not be adequate to prevent concurrent online entrees, an effort should be planned to explain to users the need to halt the system on occasion. A subsequent effort during implementation is then needed to ensure that no one tries to enter “just one more.”

CONCLUSION ON DEFINING DATA MOVEMENT

A means for documenting the relations of data structures is the technical architecture form (Figure 11) identifying synchronization requirements and synchronization approaches. The limited number of approaches include the following:

- Transmit a message.
- Transmit a transaction file.
- Transmit a master file.
- Use the telephone.

All these techniques present significant problems and require evaluation based on application specifics. Figure 15 summarizes the tasks that have been discussed in the course of this paper to develop DDP design.

Figure 15—Summary of tasks to develop DDP design