Decision support in a distributed environment

by DANIEL T. LEE

University of Hartford

West Hartford, Connecticut

ABSTRACT

Traditional means of data processing, management information systems, and decision support systems cannot meet a new demand ushered in with the evolution of mini-micro computers. A modern computer end-user, especially a modern decision maker, needs a single pool of information that may be geographically dispersed. Therefore, a new combination of technologies is needed for coping with this new demand. The purpose of this paper is to develop a unified methodology for distributed-system design with distributed databases. The distributed systems designed under this unified methodology can satisfy geographical data independence in addition to logical and physical data independence in the traditional sense.
PREFACE

The increasing popularity of mini-micro computers has ushered in a new era of distributed systems. The end-users and knowledge personnel are increasingly using mini-micro computers in their daily data processing and decision making. They not only need transaction data but analytical information in an integrated fashion. This new demand requires a new combination of technologies for combining the distributed systems and distributed databases into a unified whole. This new combination of technologies will no doubt have a tremendous impact on the job performance of many people.

In the past four decades, computers have evolved through the eras of electronic data processing (EDP), management information systems (MIS), and the decision support system (DSS) eras. During the EDP era, computers made great contributions toward the automation of paper work and labor saving. Computer programs, however, are segmented, therefore redundancy and inconsistency are inevitable. MIS was conceived to integrate them for producing decision-making information. Unfortunately, it largely produces standard reports, which are either irrelevant or only indirectly relevant to the decision-maker's needs. In order to fill this gap, DSS was brought forward with the main emphasis on supporting decision-making.

Currently, DSS development basically follows the same traditional, yet inadequate means, which are only fit for static and structured tasks. The problems faced by a modern decision maker, however, are usually unstructured or semi-structured. In addition, traditional database technologies are usually conceived under centralized usage. Now the emerging mini-micro computers add another dimension of complexity: Data and processing might be geographically dispersed. This dimension introduces a new challenge to system design and database development.

The purpose of this paper is to integrate distributed system development with database design into a unified methodology for decision support, because data management is crucial for distributed systems in a dynamic environment, as Donovan indicates: "the database systems lie at the heart of decision support system tools." Sprague and Carlson indicate that distributed data systems comprise a class of information system that draws on transaction processing systems and interacts with other parts of the overall information system to support the decision-making activities of managers and other knowledge workers in the organization. The DSS aims at less-structured tasks and unspecified problems. This approach combines the use of analytical techniques and database technologies with main emphasis placed upon the ease of use, flexibility, and adaptability in order to accommodate changes in the environment. The characteristics of the DSS have been fully exemplified in References 1, 2, 9–11, and 31–33.

Geographical Data Independence

As indicated earlier, the DSS is facing a new challenge—the distributed environment. The analytical techniques and database technologies applied to DSS development in the traditional sense are inadequate. For example, traditional database technologies emphasize logical data independence and physical data independence. The former insulates the changes made in the external end-user's programs from the global conceptual schema, whereas the latter severs the changes made in the internal physical storage with the global conceptual schema. By satisfying these two data independences, the end-user and database designer can enjoy all the freedom in modifying the database without the constraints imposed by any one of the three schemas (external, conceptual, or internal). Traditionally, the databases satisfying these data-independence requirements are regarded as being very close to ideal, but under the new distributed environment, it would be necessary to satisfy one more requirement—geographical data independence. The end-users can obtain the data for their programs without having to know where it is really located. This capability requirement is very critical for an efficient and effective DSS, working in a distributed environment.

Generally, there are three basic capability requirements for a genuine DSS: data, model, and dialogue. Data means data management capability. It basically indicates database management systems (DBMS) necessary to satisfy the information needs of the end-users and decision-makers. Model means model management capability, because a modern decision maker needs not only transactional data but also analytical information. Dialogue means friendly query languages that end-users or decision-makers can use for interfacing with the computers. Now these three capabilities must be built upon a geographical transparency. The end-users and decision-makers can interact with models and data without having to know where they are located.
Traditionally, there are usually three components in DSS, as shown in Figure 1. In order to meet geographical data independence, Figure 1 should be extended to more than one DBMS or independent files that are interlinked through communication networks, similar to Figure 2.

Three Approaches

According to Lee there are currently two approaches in DSS design methodologies, the application-specific approach (ASA) and the integrated MIS approach (IMA). The ASA is under the notion that the knowledge personnel will try to improve their job performance by exploiting the new technologies in their specific applications. They feel that the institutionalization of DSS is very difficult, if not impossible, in terms of existing knowledge and cost–benefit justification, and that the MIS personnel are too busy producing standard reports and have no time or expertise to build up DSS for the knowledge personnel. Here the knowledge personnel are defined as the decision makers and their intermediaries, e.g., management science and OR personnel.

Unfortunately, the ASA is not general; neither is it efficient or economical. It never takes advantage of the existing information-producing mechanisms in the organization. The IMA was conceived under the notion of total system concept. The IMA tries to integrate everything in the system. It sounds great, but nobody knows “how.”

Lee developed a unified approach in accordance with the contingent model. It steps down a level of abstraction by grasping the things we can manage, without losing sight of the whole. It focuses on the information flows and groups them into subsystems. The things we can clearly define are first organized into the major operational databases, and spaces are provided for data, the contents of which are not known at the present, but whose relative positions (with respect to the whole construct) are clear. When the time comes that the data do become clear, we may add them to the system. Detailed discussion of the unified approach (TVA) applied to the DSS development is presented in References 35 and 38–40. The basics of TVA also can be applied to the design of a DSS in a distributed environment. This approach should be extended to accommodate the geographical data independence in four transaction transparencies—location, concurrency, replication, and failure. Discussion of these topics follows in the next three sections.

DISTRIBUTED SYSTEM DESIGN

The Characteristics of DS

A distributed system (DS) is defined as one in which application programs and/or data reside in separate interlinked sites and are designed in an integrated and tightly controlled fashion. This definition is somewhat biased toward an integrated approach. It should be modulated by the contingent model and the unified approach for designing the DSS that was discussed previously. After this modulation, it then can be applied to the development of a DS with DDB in a realistic and practical manner. Decision makers can then be allowed to access the data freely in an integrated fashion without being bogged down by complicated mechanisms.
The Characteristics of DDB

A distributed database is defined by Date\textsuperscript{17} as a database that is not stored in its entirety at a single physical location, but rather is spread across a network of locations that are geographically dispersed and connected by communication links. It may best be described as the union of a set of individually centralized databases, because a distributed system is considered to be a partnership among a set of independent but cooperating centralized systems rather than some kind of monolithic and indivisible object.

The Basic Functions of DS

The centralization of strategic management and the decentralization of functional operations should be one of the most important objectives. The maximization of the autonomy of the individual units and the minimization of dependence among them also are vital to an effective distributed system. Each processing unit should be self-contained, but can be connected through database management systems and communication network protocols. In this way, foreign entanglements can be reduced to a minimum, if not completely stamped out. The DS designed under these guidelines will have autonomy of individual units and integration of the whole system. This DS design is quite complex but can be done if the design methodologies are used properly. This is the topic of the next three sections.

Design Strategies of DS

There are usually two approaches—top-down and bottom-up. In designing a decent DS, both approaches are needed. The top-down is used for macro-system design. It tries to lay down the design strategies for databases, files, and distributed data; to establish standards for database design and responsibilities of development; and finally to decide data structure, subject databases and their locations.

At the end of this process, it will become clear which data structure should reside in a given location; their subject databases, application databases, independent files, and sub-schema files. Production systems vs. information systems also are clearly delineated.

Design Strategies of DDB

As indicated, the above process establishes the basic strategies and guidelines for DS development. It is a macro-system design, from which we concurrently proceed to micro-system design—either during the construction of or after the completion of macro-system design. The following design guidelines follow the bottom-up approach and aim at a detailed DS design with major emphasis on DDB development. The methodologies for DDB design may use the traditional systems analysis and design methodologies. The major difference is that the complexity for DS with DDB development is much higher than that of traditional system design because the DS with DDB will have to satisfy geographical data independence. The four transaction transparencies—location, concurrency, replication, and failure—are the major concerns for an efficient and effective DS. These topics are discussed in the next two sections.

In summary, the top-down approach establishes an overall framework and a general architecture into which the end-user modules can fit into the overall architecture. The former is mainly concerned with the upper structure of the system, whereas the latter is used to establish the basic modules, especially the DDB, which is the major component of DS.\textsuperscript{46}

Design Procedures of DS

The practical steps in distributed-system development may be divided into two large phases: a subsystem delineation phase and a DDB development phase. In the subsystem delineation phase, there are five steps. First is establishing end-user sites in accordance with the processing and data requirements, such as a head office in Chicago, a warehouse in Atlanta, a warehouse in Miami, etc. Second is identifying the applications required in each end-user site for performing the functions of that site, such as credit-checking, shipping, accounts receivable, etc. Third is grouping these end-user sites with their applications into subsystems (e.g., in a manufacturing firm, there might be one head office, one or more factories, one or more branch offices, one or more laboratories). Fourth is tracing the internal and external transfers of data between internal subsystems or with the outside world, because some applications in each subsystem might share the same data or processing result. These applications should be grouped into the same subsystem, and their processing and data requirements should be closely coordinated and delineated. Fifth, separate computers for each subsystem should be identified. The interface between subsystems should also be clearly defined.

After the fifth step, a general picture of the distributed system is clearly shown, but it is still primitive and only a rough structure. A refinement is needed for its implementation. This will have to be done in the second phase.

In the DDB development phase, there are another six steps: First is establishing subject data bases and files by following the semantic data model (SDM),\textsuperscript{26,34} or Chen's entity-relationship model (E/R),\textsuperscript{14} because the SDM can be turned easily into an E/R model which, in turn, can be transformed straight into other database models, such as a relational data model (RDM), a network data model (NDM), or a hierarchical data model (HDM). The subject databases are constructed in accordance with the business subjects rather than applications, such as customers, parts, vendors, accounts, etc., rather than order entry, credit checking, inventory control, accounts payable, etc.

These business subjects are selected from narrative statements. The statements are recorded from the interview of the end-users, decision makers, and application programmers, or from checking the documents of the firm, and from the personal observation of the production process along with the
information flows of decision-making process. For detailed information on structured system analysis and design, please refer to References 24, 34, 54, and 55. Second is delineating the end-user sites and their applications—the same as in the first and second steps in the subsystem delineation phase. The third step is building a diagram of logical end-users, applications, and subject databases. This will clearly show that subject databases are shared by the applications, of which the applications are needed by the end-user sites. Fourth, determining the application programs needed for processing each application and the subject databases required for each application program. Fifth, classifying each application into four classes: SS, DS, SD, and DD. The classification is based on two factors: processing site and subject database. The SS class is determined in accordance with the application processed at the same location and the data required for processing is also located at the same location. The DS is decided by its being processed at different location, but the data required is located at the same location. The SD class is the most desirable. The DS is common with centralized systems. The DD class should be avoided. The transactions not in class SS must be handled by use of data replication, by data transmission, or both, to make the transaction class SS.

The sixth step is determining the traffic between end-user site and data location, showing whether the traffic is batch or on-line, and determining the volume as well as the frequency of traffic. The data distribution diagram (Figure 4) clearly shows these seven steps. Actually, the data distribution diagram could be drawn step-by-step.

The seventh step is determining the data distribution in accordance with the following factors: transaction volume, data size, frequency of data transfer, frequency of update, complexity of update, complexity of data replication, cost of transmission, and cost of data storage. For example, if the transaction volume is high, the data size is low, the data are updated infrequently, the updates are simple, and the data replication structure is simple, then the data should be replicated. In Figure 4, it is clearly shown that the products subject data base are highly shared on-line by many end-user sites. Data transmission is too expensive. It should be replicated. Following the above distribution factors, we may proceed one-by-one to the subject database to determine whether it should be replicated, partitioned, or centralized. Detailed discussion on design of distributed systems is presented in References 22 and 46.

The above eight distribution factors do not exhaust the list; there are many more other factors that should also be considered, including concurrency control, failure recovery, processing requirement, software development. Because a detailed discussion of these topics is beyond this paper, they will be briefly treated in the next two sections.

**TRANSACTION PROCESSING**

A complete distributed system is supposed to be able to process any transaction at any site and to obtain data from any location. Unfortunately, so far there is no such complete system. Rather, a task that operates at several sites must be planned and programmed to be sensitive to data location and network communication. A comprehensive discussion of communication systems and their protocols is beyond the scope of this paper. It has been fully treated and documented in many other sources.\(^\text{8,17,22,25,41,44,45,49,50}\) This section is a discussion of transaction processing with four transparencies, and how to develop a distributed system with appropriate transparencies for decision support. Practical examples will be surveyed in the next section.

**Transaction Concepts**

A transaction is a unit of work. It consists of the execution of a sequence of operations. Traiger proposes a model of transactions in a distributed system with the highest levels of transparencies in location, replication, concurrency, and failure.\(^\text{13}\) The system may consist of a geographically dispersed collection of computers, called sites, which are connected by a communication network. The system supports a set of entities that are represented by one or more segments. These segments are identified by <name, site> pairs, where name is the entity name, and site is the place in which the segment is located. At any time, a value is associated with each segment. A segment may represent an entity or a part of an
entity. Several segments at a site might represent the same entity or they might represent the same entity at different sites.

If an entity is represented by multiple segments, then the entity is said to be replicated. An entity named E that is replicated at sites \( S_1, \ldots, S_n \) is represented by the segments \( < E, S_1 >, \ldots, < E, S_n > \). A system without replicas is called partitioned because each entity is at exactly one site. If all segments reside at the same site, the system is called centralized.

A particular application may need one or more transactions. Each transaction may associate a meaning with each entity; e.g., entity \( E_1 \) represents products, entity \( E_2 \) represents accounts, and so on. The collection of entities and their relationships is called subject databases. Their representation depends on the database models used.

**The Execution of Transactions**

A transaction issues requests to manipulate entities. These requests are translated by the system into one or more commands on the entity. Each site provides a group of commands that manipulate entities or segments at that site. The translator at the site keeps an entity-site directory, which gives the site addresses of segments. The format for reading a record (an entity or a segment) by transaction \( T \) of segment \( < E, S > \) which has value \( Val \) is represented by

\[ [T, \text{READ} \ (E, S), \ Val] \]

The READ is one of the commands. If the command is WRITE, it alters the value, \( Val \), to a new one. If the command is COMMIT, it means that the transaction is successfully executed.

Each command operates on one segment only and hence at one site only. There may be concurrency of execution in the network, but the commands at a site appear to happen in some order. All commands on segments are performed at the site of the segment. Whenever a site participates in a transaction execution, the site allocates an agent for that transaction. The agent keeps track of the local tracition state and performs the commands for that transaction at that site. Whenever a transaction requests nonlocal action, the requesting site issues requests to the requested site (or the owning site) for action. The requested site follows the same procedure of execution. Thus, the transaction is executed synchronously, completing one action (request) before issuing the next, and finally issuing a COMMIT action to each site visited. Actually, transaction execution control may migrate from site to site. The control protocol of the network must be more complicated.

**Data Distribution Architecture**

Lo proposes a three-level distributed-database design consisting of source level, user level, and control level. In the source level, the design consists of a complete set of subject (global) databases. Data in this level are fully replicated and synchronized. The user level consists of a subset of the subject databases that is derived from the source level and becomes the application databases used by the various control processing functions.

The control level consists of four components: the transaction processor, data dictionary, subject database map, and communication software. The transaction processor coordinates the data flow in the system. The data dictionary documents all facts, including update information. The subject database map is more efficient than the data dictionary in locating subject databases. The communication software handles the actual data communication in the system.

Data are globally synchronized at the control level. The distributed-database concept discussed here is applicable to the wide applications of multinational corporations. Each division of the corporation resembles the regional operation centers. The data at the branch office can be updated locally and broadcasted to the other sites for updating. The database system design proposed here allows currently available database management systems to be used in the source and user levels, but it needs to develop the communication protocols required in the control level.

**User Transparencies**

Friendly interface with end-users with distributed systems is highly desirable in the execution of transactions. Location transparency, replication transparency, concurrency transparency, and failure transparency are vital for a successful distribution system. With these four transparencies, an end-user can concentrate on what he wants and does not have to worry about the location of data and the question concerning whether the data are replicated, partitioned, or centralized. During the execution, he will be assured that the data will be delivered accurately and that the results of data manipulation will be consistent. Detailed discussion is presented in References 4–7, 12, 17, 19, 23, 30, 43, 46, 50, and 53. The mechanisms of these transaction transparencies will be briefly examined during our discussion of the prototypes of distributed systems in the next section.

**THE PRESENT STATE OF THE ART**

Examining the prototypes of distributed systems is one of the most efficient methods of understanding the underlying theory of distributed systems. It will demonstrate the architecture, design methodology, distribution application, distributed database, transaction transparency, and especially it will unveil the state of the art in distributed-system development.

Six distributed systems are selected for analysis: IMS/MSC, CICS/ISC, Distributed INGRES, R* (R Star), Tandem's Encompass Systems, and SDD-1. Most of these are experimental systems. Some are installed successfully but their software protocols in network control and database management systems still need much improvement.
**IMS/MSC**

IMS's Multiple System Coupling (MSC) allows two or more IMS systems to be connected. End-users or programs can invoke one program on another. The input message from a user, or the IMS DC call from a program, will be placed on the input queue. IMS will then examine a local catalogue to see whether the program to be executed for this calling program resides at which remote site. When this is accomplished, the input message will be transmitted to that site for processing. The result will be transmitted back to the original site. The transaction invocation can propagate from one remote site to another with the result being transmitted back to the original invoking site. One must remember, however, that the MSC of IMS does not really support distributed transaction processing as in the System R or CICS. Each invoking site must complete the execution before the next one in the sequence can start. There is no parallelism among them or return of control from an invoked program to its invoking program. The system supports only transaction routing, not transaction processing. The term "transaction" is used in IMS to mean an input message rather than the execution of a program.

The end-user does not have to know where the data or programs reside, and can invoke a transaction from any site. Programs, however, can access only local data. They do need to know the precise location of remote data and this data distribution knowledge is built into the application logic. In a general sense, IMS does provide location transparency, but only in a limited form.

MSC is an IMS-only feature. IMS is also capable of participating in a different distribution scheme known as ISC (Inter Systems Communication), which is a set of protocols used for communicating with other systems, such as CICS.

Basically, the IMS/MSC provides location transparency for message handling among multiple sites. It also provides the transaction notion and failure transparency. The program isolation feature of IMS is similar to that of providing concurrency transparency for transactions within a single site. IMS has no notion of replicated or partitioned data and does not provide replication or location transparencies in a strict sense.

**CICS/ISC**

ISC (Inter-System Communication) is a set of protocols by which any systems conforming to those protocols can communicate with one another. Most of the ISCs are supported by Customer Information Control Systems (CICS). CICS/ISC allows two or more CICS systems to be connected in such a way that one application program can invoke another at a different site without ceasing execution itself, or can issue a DL/1 call against a database at another site.

In the first case, in the CICS sense of the term, this is called distributed transaction processing, which allows the total application to be divided into a distributed set of programs. The end-user will initiate the transaction by invoking the first of these programs. As it executes, that program—or agent—can invoke agents at another site. The set of all agents is considered as a unit for recovery. Therefore, it does support the transaction notion. CICS does not maintain a catalogue giving the location of each program; instead, an agent A that wishes to invoke another agent B must specify the site at which the program for B resides. The data distribution knowledge is built into the application logic. Location transparency is not provided under the first case.

In the second case, it is called data request shipping. It does support location transparency at the application level. The programs can issue DL/1 database calls against a remote database and CICS/ISC will intercept the call and ship it to the appropriate remote site, using a catalogue that gives the location of each database. An agent will be assigned by that remote site to perform the necessary processing or to issue calls on behalf of the original program and to return the result to the original program. Again, all agents are considered as a unit for recovery.

In a general sense, CICS does provide the transaction notion, location transparency, and failure transparency, but not concurrency transparency because there is no lock manager. Responsibility for concurrency control is delegated to the individual subsystems, such as DL/1, TOTAL, System 2,000, etc. Similarly, CICS has no notion of replicated data.

**Distributed INGRES**

The distributed version of INGRES does not provide a notion of transaction, but does provide location and replication transparency. In INGRES, a single QUEL statement is a transaction. This implies that it does not have either failure or concurrency transparencies for transactions that are groups of QUEL statements.

**R* (R Star)**

R* (R Star) is a distributed version of System R currently under development at the IBM San Jose Research Laboratory. The basic difference between SDD-1 and R* is that SDD-1 starts by choosing a workable strategy and then tries to improve on it, whereas the R* attempts to generate a whole set of workable strategies and then selects the cheapest one. The SDD-1 is characterized as somewhat "greedy," as Data puts it, in that it always looks for immediate improvements; it will find a solution that is locally optimal, but not necessarily the one that is globally optimal.

**Tandem's Encompass Systems**

The distributed EMPACT is an application of DS with DDB for business organizations. The design of distributed EMPACT illustrates the techniques used in DDB development, and the actual implementation of distributed systems.

The elements of the database are divided into two major categories, global data and local data. Private data do not enter the picture because they are used only by the individual site and are not visible to the other sites. Global data are shared by all sites, such as the list of parts that determine the TANDEM parts catalog (item master file). The local data...
consist of information that is uniquely important to the individual site using it but accessible by all sites, such as stock status and work-in-process data. Global data are necessarily replicated at all sites, whereas local data are single-site resident. There may be, however, a fourth kind of data called partial replication data, which permit requests by one site for material from another to be placed and processed. But these data are only known and resident at both sites, and not necessarily to the third party. Such data may be classified as semi-global.

The database consistency solution must satisfy two important objectives: continuous availability and site autonomy. Since global data are replicated at every site, query access to the database is guaranteed regardless of the status of other sites in the network. The problem is to find a way to maintain all copies in the network updated and consistent at all times.

One way is to broadcast the updates to all the sites in the network as a single transaction, but keep in mind that global files can be updated only when all sites are available. It usually requires a long wait and sacrifices site autonomy.

The solution chosen was to sacrifice the absolute consistency of the replicated files in exchange for site autonomy and short terminal response by using a suspense mechanism to maintain database consistency. Instead of immediately broadcasting the updates to all sites in the network, the server at the site where an update is initiated first updates that copy of the global data and then posts the update message to a suspense, or queue, file. A suspense monitor asynchronously polls the suspense file for transactions and, on an as-soon-as-possible basis, sends the transaction message to appropriate servers at remote sites, one at a time, as a separate transaction for updating.

The requirement of site autonomy is satisfied because updates to the global files can be initiated regardless of the status of other sites in the network. The propagation of the update to remote sites is performed asynchronously by the suspense monitor.

Because the suspense mechanism introduces a delay in the propagation of updates to remote sites, the possibility of conflicting adds and updates among the sites becomes a problem. To prevent conflicting updates from occurring when two or more sites update their copies of the same data simultaneously, ownership (by site) is assigned to global records and the initiation of updates is restricted to the owning site only.

Another problem introduced by the suspense mechanism is stale data. The data are out of date because an update to the file has been posted at a remote site but has not yet been propagated to the local site. However, because the propagation time for suspense updates is considered less than the time the user community takes to act on the update, temporary staleness is not a problem. Besides, a two-step protocol of check and update is used for updating transactions. Serialization of executing transactions is maintained by a counter at the site initiating global updates. The suspense file is key-sequenced, and the value obtained by incrementing the counter determines the relative position of the record in the suspense file.

This is an example of distributed system with DDB application. The organization of the database and software closely parallel the structure and organization of the business environment. Generally, Tandem’s Encompass Systems support location, concurrency, and failure transparencies, but not replication transparency. Detailed discussion of the Tandem’s Encompass Systems is presented in References 12 and 48.

SDD-1

SDD-1 (a system for distributed databases by Computer Corporation of America) is the first working DDB designed for naval command and control applications. It is also appropriate for general applications that require an integrated database and geographically distributed data. Multiple users need access to a single pool of information that is geographically distributed. It is highly desirable to have a system that can exercise decentralized processing and centralized control. The DDB poses a new technical challenge because its inherent requirements are for data communication and parallel processing. Overall architecture and basic techniques of SDD-1 will be briefly discussed. Detailed discussions are presented in References 5-7, 16, 25, 46, and 50.

SDD-1 supports a relational model. Users interact with SDD-1 through a higher level language called DATACOMPUTER. A single data-language command is called a transaction; this is the basic unit of interaction between SDD-1 and the users. This concept of transaction is similar to that of INGRES and System R.

An SDD-1 database consists of logical relations, which are partitioned into subrelations called logical fragments. These fragments are the units of data distribution. They are defined by horizontally and vertically subsetting relations. The assignment of fragments to sites is made when the database is designed. The end-users are unaware of data distribution or replication. They reference only relations, not fragments. The SDD-1 will translate from relations to fragments, and then select the stored fragments.

SDD-1 consists of three virtual machines: Transaction Modules (TMs), Data Modules (DMs), and a Reliable Network (RelNet). All data are stored in DMs under the supervision of TMs. DMs respond to four types of command: read, move, manipulate, and write to perform fragmentation, concurrency control, access planning, and distributed-query processing. The RelNet connects DMs and TMs and provides four services: guaranteed delivery, transaction control, site monitoring, and network clock.

Concurrency control

When multiple users access a shared database, two conflicts can occur. First, if T1 is reading a database while T2 is updating it, T1 might read inconsistent data. Second, if both T1 and T2 are updating the database, race conditions can produce erroneous results. Traditionally, this is solved by database locking, but this solution might cause long delay and affect site autonomy.

SDD-1 adopts serializability for concurrency correctness because serial execution maintains consistency. SDD-1 uses
two synchronization mechanisms that are different from locking. The first is called conflict analysis for detecting potential conflicts. Two transactions are in conflict if the read-set or write-set of one intersects the write-set of the other. The read-set of a transaction is defined as the portion of the database the transaction reads, and the write-set of a transaction is the portion of the database the transaction updates. The database administrator defines transaction classes, which are named groups of commonly executed transactions. Each class is defined by its name, a read-set, a write-set, and the TM at which it runs. A transaction is a member of a class if the transaction’s read-set and write-set are contained in the class’s read-set and write-set, respectively. Conflict analysis is performed on these transaction classes, but not on individual transactions, because transactions from different classes can conflict only if their classes conflict. The output of the analysis is a table that indicates for each class which other classes it conflicts with, and for each such conflict, what protocols are needed to ensure serializability.

Each TM might only be allowed to supervise transactions from one class. When a transaction issues a request, the system determines which TM should be sent in accordance with the transaction class to which it belongs. The TM synchronizes all transactions by global timestamping and pipeline rule.

The second synchronization mechanism is the global timestamp and the pipeline rule. In traditional locking, the execution order is determined by the order in which the transactions request locks. In SDD-1, the order is determined by a total ordering of transactions induced by timestamps. Each transaction submitted to SDD-1 is assigned a globally unique timestamp by its TM and is sent to the DMs. When a DM receives a READ command, it defers the command until it has processed all earlier WRITE commands. The pipeline rule requires that each TM send its WRITE commands to DMs in timestamp order.

The access planning minimizes intersite communication. Two-phase commit guarantees delivery. SDD-1 treats directories of data as ordinary user data, but the data directories also can be fragmented, distributed, and updated. A copy of the directory locator is stored at every DM. SDD-1 maintains directories that contain relation, fragment definitions, fragment locations, and usage statistics. TMs will use them for every transaction manipulation.

SDD-1 is the first working DDB and employs ARPANET’s communication network and able to use the world’s X.25 packet-switching networks. The work was supported by the Defense Advanced Research Project Agency. SDD-1 was designed for Naval command and control applications. The techniques can be used for DS with DDB in general. The development team analyzed the problems of directory, conflict, and efficiency of the system, which was implemented successfully.

In summary, the SDD-1 does provide both location and replication transparencies, allowing the user to think in terms of entities (files) rather than segments. It does not support the notion of transaction; so, strictly speaking, it does not provide failure or concurrency transparencies. In SDD-1, a single data-language statement is a transaction. An application usually requires several data-language statements to perform an operation.

### Summary Notes on Prototypes

We have surveyed some, but by no means all, of the major DDB prototypes. Most of them surveyed are experimental systems. Some are implemented successfully, such as SDD-1 and Tandem’s Encompass Systems. The IMS/MSC, CICS/ISC, and distributed version of INGRES also have been used as the basis for much of the discussion. In Figure 5, five prototypes of distributed systems with DDB are listed for comparison in terms of four transparencies and the notion of transaction. This is used only to show the general character of each prototype.

SDD-1 is atypical prototype distributed system designed and implemented in an integrated fashion to provide the user with a single, consistent view of a complete database. The system also is designed to support databases that can be physically distributed with arbitrary redundancy over a network of potential worldwide distribution. The control is completely distributed. The system will continue to function even if any one of the sites fails. New sites can be added freely. This will increase the survivability of the system. Furthermore, these distribution features can be applied to business applications. It will naturally lead the EDP, MIS, and especially the DSS into a new era of distributed systems.

### CONCLUSIONS

The fundamentals of DSS have been discussed; basic characteristics of DSS, current design methodologies, and capability requirements of DSS have been covered briefly. Distributed system development and distributed database design have been discussed intensively. A step-by-step method has been used for illustrating the design process of DS and DDB. A three-level distribution architecture of DS has been shown. The notion of transaction and the four transaction transparencies are important concepts in DS and have been briefly treated.

Typical prototypes of DS with DDB have been closely scrutinized to peep into the mysteries of DS. Six types of current DS have been used for investigation. Some are installed successfully and are commercially available, such as SDD-1, but

<table>
<thead>
<tr>
<th>Prototypes of DS</th>
<th>Location</th>
<th>Replication</th>
<th>Concurrency</th>
<th>Failure</th>
<th>Notice of Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDD-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tandem’s Encompass</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INGRES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMS/MSC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CICS/ISC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5—Survey of prototypes
most of them are experimental in nature. This prototype investigation may provide valuable information for DS designers in the selection process; when the situation arises, they may use this information as a guideline to DS development or for choosing the appropriate DS.

Currently, there are no perfect distribution systems on the market; the major bottlenecks are in the software development of communication systems and network database management protocols. Technology is progressing at a rapid pace, but it will gradually ease off in the near future. We can predict that the next decade will be the era of distributed systems, especially since the use of mini-microcomputers has become widespread. Distributed systems with distributed databases will become the major carrier for data management in the decade to come.

REFERENCES


