Data Communications

When the shoe's on the vendor's foot: A look at Tandem's corporate network

How multiprocessor nodes can become more sociable

Kent Madsen and David Foley, Tandem Computers Inc., Sunnyvale, Calif.

When the shoe's on the vendor's foot: A look at Tandem's corporate network

When a computer vendor sets up an internal network using its own products, outsiders may see what the machines can really do.

As part of an ambitious internal communications and information management strategy, Tandem Computers Inc. has used its own hardware and software products to build a vast corporate network. The data communications web contains 200 nodes and spans 18 countries. Users in such countries as Japan and Australia are tied to sites in the United States, Canada, and Mexico, as are offices in the major commercial centers of Europe.

Over a hundred different applications run over the in-house network. Perhaps the most important of these is electronic mail. Roughly 70,000 messages are originated, and 250,000 are delivered each week to and from users throughout the world.

The widely used electronic mail is joined by a number of more specialized applications. For instance, the company's various manufacturing groups maintain their records in a distributed database. A battery of financial packages is available to network users, including tools for order entry, invoicing, credit and collections, and budgeting. A network-based program is also available to process requests for product enhancements and to track the actions taken in response.

In addition to the applications, many databases and information resources are accessed via the network by domestic and international Tandem workers. A "public" database, accessible by anyone in the company, contains information on employee office locations, office telephone numbers, department affiliations, facsimile and mail drops, and so on. Customer lists, notes about software, and other marketing information are listed in a customer-reference database.

An innovative archive of technical information has been compiled primarily from electronic mail exchanges. Another database, set up as an electronic bulletin board, provides a central source of support information. Field salespeople, responding to requests for proposals, make use of a constantly expanding collection of proposal text files.

Resources like these have become indispensable to nearly all Tandem employees. Since data communications is so important to the way the company does business, developing and maintaining the corporate network has become a leading concern.

Topology

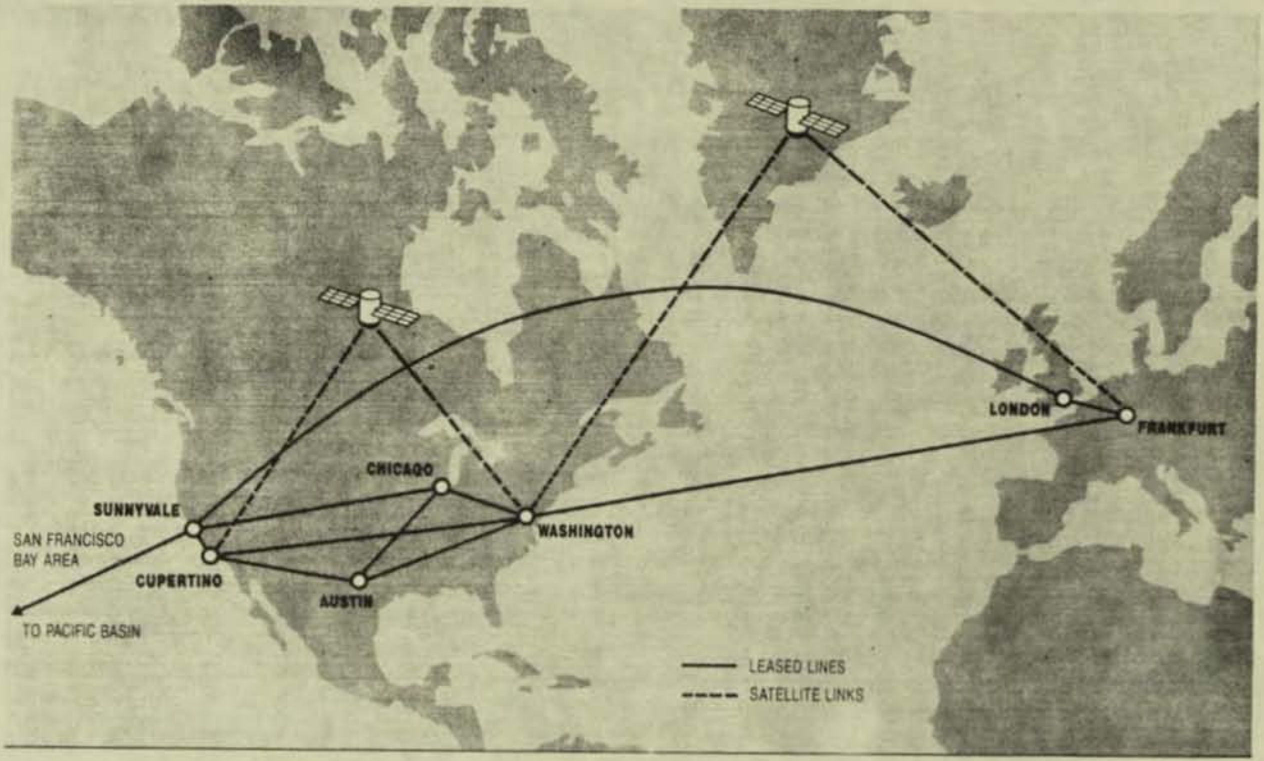
Management has insisted that the corporate network be built using standard Tandem products. Thus, each node consists of a multiple-processor computer in the NonStop line. Standard Tandem communications software and hardware are used, and databases are managed by standard Tandem products as well.

Of the 200 computers in the corporate network, 193 support applications and databases. These application nodes exist primarily to meet local word- and data-processing needs. However, they do handle communications for local users and applications, and they accept passenger traffic from other nodes.

The application nodes are built around seven "backbone" nodes that are dedicated to communications (Fig. 1). These nodes are linked by leased high-speed lines and, in several instances, by high-bandwidth satellite or microwave links. The backbone nodes have only one job: to be constantly available to move information between application processors. Roughly 1,500 Mbytes of data flow through them each day. There is, in addition, a substantial amount of regional traffic that never reaches the backbone nodes.

Connected directly to the backbone nodes are "Class I" nodes—machines that run accounting, manufacturing, and customer-support applications. These programs must be available if the company is to do

1. Over a billion and a half served. *Tandem branches from Osaka, Singapore, and Sydney to Neufahrn in West Germany can reach each other, as well as programs and data in the United States, via an extensive, internal computer network. The seven backbone nodes handle 1.5 billion bytes of traffic a day.*



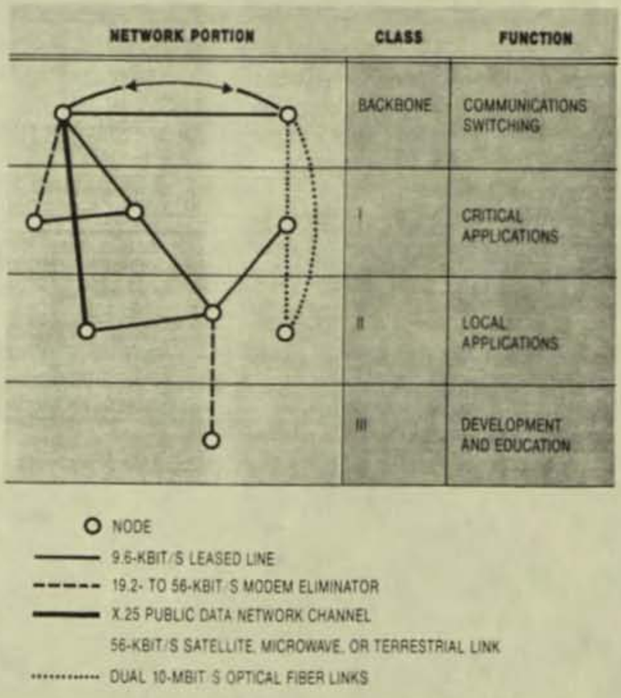
business, and thus the nodes in which they run must always be accessible from a backbone node.

The network's 26 Class I nodes are always linked directly to at least one backbone node and either directly or indirectly (through another Class I or Class II machine) to a second backbone node (Fig. 2). Each machine is thus part of a ring. This dual-path policy has been established to provide uninterrupted network service. It ensures that even if a backbone machine, a communications line, or a modem fails, the Class I node will not be cut off from the network.

Over a hundred network nodes are Class II. They typically serve field sales and service offices, running local applications and less time-critical network applications such as electronic mail. Thus, they need access to the network, but response-time and availability requirements are not as stringent as in the case of Class I nodes. Class II nodes are connected no more than two nodes away from a backbone machine (or a high-speed lightwave cluster, as in Figure 2) whenever possible. They also each have an alternate path to a backbone node—and thus to the rest of the network.

Class III nodes are used primarily for development work or customer education, not for running network applications. They are often intentionally overloaded, brought down, or crashed to debug and test the capabilities of software products and, therefore, are not always connected to the network. They are also used to give customers and internal support people experience in loading machines and handling recovery.

2. Architectural outline. *In this sample layout, all applications nodes except for the development machines have at least two paths to the backbone network.*



NETWORK PORTION
CLASS
FUNCTION
BACKBONE
COMMUNICATIONS SWITCHING
I
CRITICAL APPLICATIONS
II
LOCAL APPLICATIONS
III
DEVELOPMENT AND EDUCATION
NODE
9.6-KBIT/S LEASED LINE
19.2- TO 56-KBIT/S MODEM ELIMINATOR
X.25 PUBLIC DATA NETWORK CHANNEL
56-KBIT/S SATELLITE, MICROWAVE, OR TERRESTRIAL LINK
DUAL 10-MBIT/S OPTICAL FIBER LINKS

When any of the 63 Class III nodes is connected to the network, the connection is either through a ring or a spur composed only of Class III nodes. (A spur is a group of nodes strung along a communications path that is attached to the network at one end.) Thus, no higher-class node ever has to rely on a path through a Class III node for access to the network.

Application nodes of the three classes are usually connected to the backbone nodes (and to one another) via leased lines or satellite links. A microwave scheme from M/A-Com with Coastcom multiplexers joins the two California backbone nodes. Some application nodes, most notably in Mexico, Canada, and Europe, are linked via X.25 circuits. In addition to the node-to-node lines, there are numerous connections from terminals and terminal clusters to nearby nodes.

Tandem believes that, considering the size of the company and the network, it pays very little for communications. Expenses for domestic and international circuits, satellite links, modems, and other communications services are in the neighborhood of $180,000 per month.

Network management

Between 1979 and 1981, the Tandem corporate network grew from zero to about 40 nodes without any centralized management. Individual computers and applications were locally managed, and when local operations people wanted to interconnect their machines, they did so by whatever means seemed appropriate or convenient. Admittedly, this was haphazard, but it met the company's needs at the time.

During this three-year period, the average availability of Class I nodes over the network was low, not because of a problem with the computers themselves, but because no thought had been given to network architecture. At first, the 40 computers had been linked in star fashion to several central machines at corporate headquarters, to facilitate order-processing activities, communications between software developers, and so forth. However, disruptions in the star network could isolate users from resources in the network.

No provision had been made for alternate communications paths. Thus, line and modem failures inevitably isolated at least one node (and sometimes several) from the rest of the network. This also occurred when a node in the middle of a spur was brought down for maintenance or configuration changes.

In response to difficulties of this kind, a small network support group was formed in 1981 to evaluate the situation and address the problems involved in running a large multifunction network. Within four months, the backbone structure was put into place and rings were formed to provide less-interruptible service.

Network-oriented node-management practices were also instituted. For example, Class I nodes were not allowed to leave the network without being scheduled by the support group. Test software required approval before being let loose on the network.

As a result of these changes, the average Class I node availability rose dramatically and is now routinely at the 99 percent level. At first glance, this statistic may be misunderstood. Vendor hype usually includes claims of high availability. The respectability of these claims depends on how the term "availability" is defined. One must examine what underlies this kind of statistic.

To achieve complete availability with a standalone computer during five consecutive 8- or 12-hour business days requires only that the machine run during these days without a hardware failure. Maintenance and reconfiguration can be handled at night or on weekends without affecting the average. But achieving an average network availability of 99 percent running 26 Class I nodes for seven 24-hour days per week (as the network support group has done for almost three years) is far more complicated.

The Class I nodes must be available whenever the applications on them are likely to be accessed. In a domestic operation, this means 12 hours a day, since people work eight-hour business days in each of four time zones. Adding European users, and now users in the Pacific basin (Japan, Hong Kong, New Zealand, Australia, Singapore, and Hawaii), has put an unprecedented demand on the network.

Global demand for network access to Class I nodes imposes several stringent conditions. Maintenance and configuration changes requiring any of the Class I computers to be out of service count against the availability average. Whenever these computers are reconfigured, brought down for software changes, moved, or upgraded, the downtime is noted.

Not only must each Class I application node be available, but also, at least one communications path from each of these nodes to a backbone node is required continuously. This path may include several modems and lines and, on occasion, a Class II node, all of which must be available if the path is to be used. Finally, the backbone network itself must be available virtually all the time, to ensure that the primary and alternate communications paths are usable.

Given the above details, it is easy to appreciate what underlies the 99 percent availability statistic for Class I nodes. Global operations make incredible demands on network components and personnel. Even preventive maintenance is carefully scheduled and carried out.

The division of labor

Application nodes within the Tandem corporate network are locally managed. The applications that make use of the network are likewise developed, maintained, and managed by the groups that use them (manufacturing, capital management, marketing, etc.) or by specially designated organizations within the company.

The network support group is responsible for the backbone machines and related communications equipment. The backbone concept was implemented to separate the basic communications from the applications. This separation has made the nodes that handle the two functions more efficient and manageable. Backbone and application machines are configured differently to optimize the performance of each.

The primary role of network support is to manage the corporate network as a multifunction communications medium. Members of the support group collect data on network operations, manage the backbone machines, and troubleshoot line problems. They also train operations people at each node to consider the impact of their actions on the network at large.

Group members investigate and make recommendations on new hardware, software, and line services that might enhance the usefulness and responsiveness of the network. They must also plan for and maintain a sensible network architecture. This means treading a fine line between cost-effective implementation and satisfactory availability and response time.

The means to keep growing

Since 1981, the network support group has overseen the growth of the network from 40 to 200 nodes. Yet the group has never consisted of more than six people. The work of this group is simplified by the architecture and operating system of the computers used in the network. Each node consists of a computer designed for "failure tolerance" and expandability.

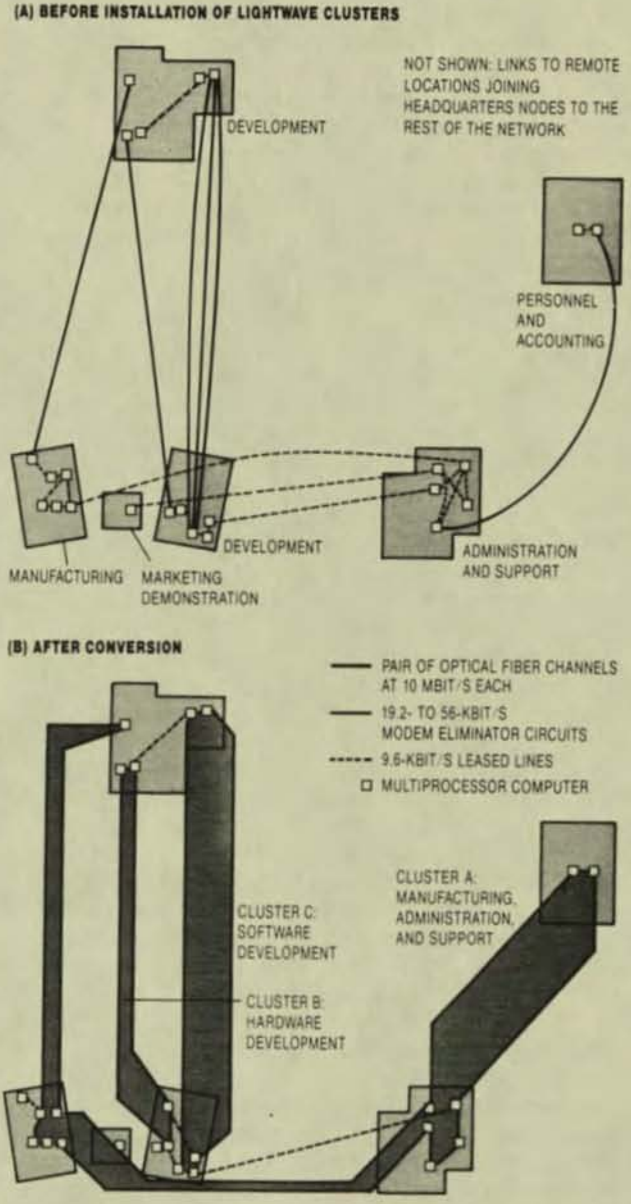
Failure tolerance refers to the ability of these computers to continue to function in the face of any single component failure, including a processor failure, and to the fact that it is possible to repair and reintegrate a failed component without shutting the computer down. This feature is important to the functioning of Class I nodes in Tandem's global network.

Expandability refers to the fact that a single machine can consist of anywhere from two to 16 cooperating processors. Guardian, the distributed operating system that manages resources for each multiprocessor node, allows the machine to grow through that range without requiring any reprogramming of applications. This means, for example, that operators of a NonStop TXP machine can increase the processing power of the computer from roughly four million instructions per second (MIPS) to 32 MIPS without having to change a single line of code.

Where even more local processing power is required, up to 14 of these computers (for up to 224 processors) can be linked locally in a ring via a Tandem software/hardware product known as the Fiber Optic Extension (FOX). This link is almost as fast as the internal bus that links processors within a single machine. The data transfer that takes place over the link is managed by the same operating system mechanism that handles traffic within a single multiprocessor node (independently of the input/output channels of the processors). As a result, the entire local subnetwork thus created can be used as if it were one large machine with a processing capability of 448 MIPS (14 nodes each with 16 two-MIPS processors).

The reliability and local expansion capability of the computers used in the Tandem corporate network make the network far easier to manage than it would otherwise be. As explained above, the operating system running in the local machine has the ability to make multiple processors appear to users and programmers as a single unified resource. In a network setting, this operating system also has the ability to blur node boundaries. The operating system and associated networking software permit operations people

3. Rings of light. *Computers in buildings at company headquarters are being linked into lightwave rings. The portions of the rings within buildings are now complete.*


(A) BEFORE INSTALLATION OF LIGHTWAVE CLUSTERS
NOT SHOWN: LINKS TO REMOTE LOCATIONS JOINING HEADQUARTERS NODES TO THE REST OF THE NETWORK
DEVELOPMENT
PERSONNEL AND ACCOUNTING
DEVELOPMENT
ADMINISTRATION AND SUPPORT
MANUFACTURING
MARKETING DEMONSTRATION
(B) AFTER CONVERSION
PAIR OF OPTICAL FIBER CHANNELS AT 10 MBIT/S EACH
19.2- TO 56-KBIT/S MODEM ELIMINATOR CIRCUITS
9.6-KBIT/S LEASED LINES
MULTIPROCESSOR COMPUTER
CLUSTER A: MANUFACTURING, ADMINISTRATION, AND SUPPORT
CLUSTER C: SOFTWARE DEVELOPMENT
CLUSTER B: HARDWARE DEVELOPMENT

and users to log on to their local machine and do work on remote nodes.

For example, they can type in successive two- or three-word commands that will start a program on a machine in New York, instruct that program to access a file in a disk volume in Atlanta, and print out the results for another employee on a device attached to a computer in Chicago. The command syntax by which these operations are carried out is identical to those that would be used locally for similar operations, except that, in each case, a node specifier must be added to the program, file, or device name.

Five of the seven backbone nodes in the corporate network are managed remotely from control points in Cupertino, Calif., and Frankfurt, West Germany. If, in the course of routine monitoring (or as a result of a telephone call from users), network support people detect a noisy line that is causing delays and timeouts, they can run tests to identify what kind of noise is present and then, if necessary, bring down the line.

The networking software will immediately detect this change, update the routing tables in each node, and automatically channel messages over an alternate path. Network support people can then simply call the telephone company personnel to report the problem and let them fix it. When the problem is fixed, network support brings the line back up and, at that point, network software updates the routing tables again to indicate that the old path is available.

Adding a node to the network involves little more than plugging it in. The local organization finds physical space for the new machine and sets it up. Meanwhile, the network support group orders the communications lines and assigns a node number and node name to the new machine. When everything is in place, the local operations people attach the machine to the line, activate the line handler with a single command, and let the networking software do the rest.

When the new node is attached, it announces its existence to its immediate neighbor. The neighbor sends the node a copy of its routing tables containing information about all the other network nodes. The new machine then sends greeting messages to those nodes. After receiving such a message, each node updates its routing tables. Only operations people at the nodes connected directly to the new one need to know that a change has occurred.

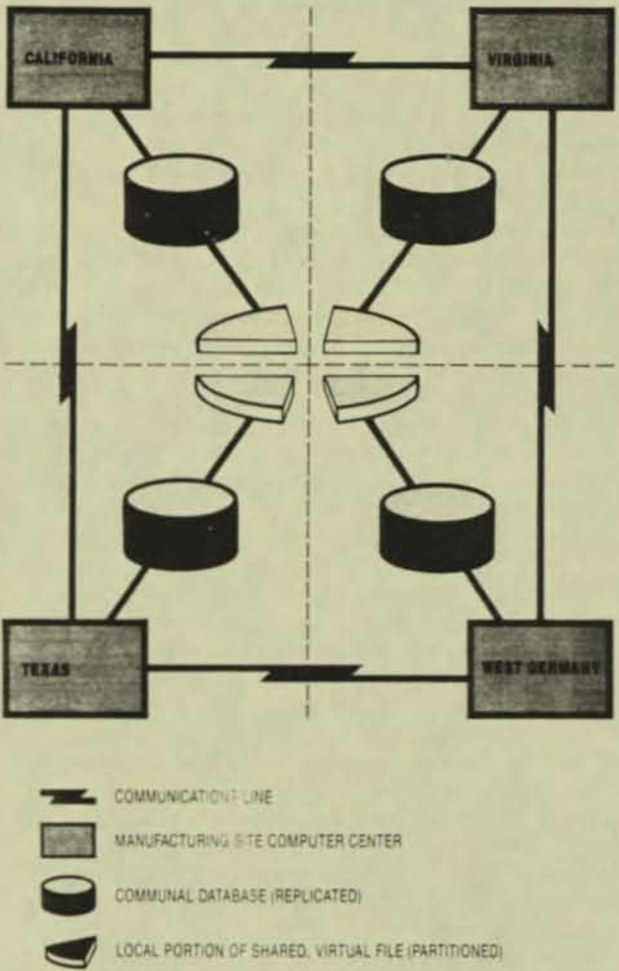
The network support group is currently using the lightwave product described above to link computers at company headquarters into rings (Fig. 3). The machines are joined by 9.6-kbit/s leased lines, with modems from Codex Corp. and Halcyon Communications Inc. Intrabuilding connections are 19.2- to 56-kbit/s RS-449 modem eliminators from Compre Comm Inc. or ARK Electronic Products Inc.

With lightwave links in place, up to 14 nodes will be able to communicate with each other almost as fast as the multiple processors within a given node. The link joining machines into a high-speed cluster consists of four fibers, two each for transmit and receive channels, configured in a ring at 10 Mbit/s per fiber.

Implementing the headquarters' architecture shown in Figure 3 will reduce the processing overhead associated with networking, since a controller, rather than the machines participating in the ring, will process pass-through traffic. In addition, functional groups of computers and users will be consolidated and certain replicated databases will no longer be needed, since it will be possible to access a database on another node in the ring almost as fast as if the database were locally attached.

Another reason for moving to lightwave technology is its improved reliability. The current architecture provides only two paths from most machines to the network at large, while the lightwave rings yield four

4. Breaking up is hard to do. *In this distributed database, communal data is replicated at each site, and local data is partitioned among the sites.*


CALIFORNIA
VIRGINIA
TEXAS
WEST GERMANY
COMMUNICATIONS LINE
MANUFACTURING SITE COMPUTER CENTER
COMMUNAL DATABASE (REPLICATED)
LOCAL PORTION OF SHARED, VIRTUAL FILE (PARTITIONED)

paths to each and every node in the cluster. Also, pass-through traffic can continue over a link even if the intermediate nodes are down.

A distributed database

Nearly all phases of Tandem's business depend in one way or another on services that the corporate network provides. As mentioned earlier, over a hundred different applications run over the network. Numerous databases and information resources are also available remotely. One sophisticated application developed by and for the manufacturing organization uses the network to maintain a distributed database.

Tandem has manufacturing plants in four locations: California, Texas, Virginia, and West Germany. Each one has a fair degree of local autonomy but similar information needs. Managers and employees at each plant need access to communal data, such as the company's comprehensive parts catalog and bills-of-materials (lists of parts that go into specific assemblies and finished products). For their own shops, they must keep a close eye on local information. This includes production schedules, materials requirements, purchasing of parts, receiving, inventory, interplant materials transfers, and work-in-process.

Originally, manufacturing information of this kind was maintained in separate databases at each site. This was good for autonomy because local information was under local control and communal information was always available, even when communications lines or distant computers were down. However, it also meant that communal data (such as the parts catalog and bills-of-materials) was often inconsistent from site to site. Monthly, there were typically 4,500 updates to the bills-of-materials files and 1,000 to the parts catalog. Thus, the copies of these files used at the various sites had to be updated and reconciled once a week.

Anticipated growth in the number of manufacturing sites was bound to increase the need for local autonomy. As each site's functions became increasingly specialized, so did its data requirements. At the same time, growth would aggravate the problem of consistency. Sites would need better ways to keep each other current and to share resources. Anticipating this, manufacturing information planners decided to use the network to provide an integrated, distributed resource.

The application they created distributes data across the network in two ways, as shown in Figure 4. Communal data, which is used heavily at each site, is replicated so that all manufacturing sites have ready access to it. Local data, which consists of records of interest only to users on particular nodes is stored at those nodes. The files containing those records are partitioned across the network.

Reads and updates of local data are easy because the information is on the local node and because there is no need to inform any other node of changes. Reads are also easy with replicated data, because the files are available on the local node. Updates of replicated data are more complex, however, because the local update cannot be considered complete until copies at all other remote manufacturing sites have been updated as well.

The designers of the application had a choice of how to handle these remote updates. One strategy would be to include the updates as part of the local transaction and not consider that transaction complete until the relevant records on all remote manufacturing nodes had been successfully updated. This would have a substantial negative impact on response time for the user requesting the local update, whose terminal or process would be suspended until update requests traveled to, and were completed by, all other nodes. It would also mean that if, for some reason, one of the remote nodes were inaccessible, the transaction could not be successfully completed, even on the local node.

Another approach would be to let the local software incorporate some kind of independent delivery mechanism. This mechanism would take responsibility for updating communal data on remote nodes as soon as possible after the local update transaction had been completed. The "asynchronous delivery" approach would mean that replicated files would be inconsistent for brief periods of time, until the independent delivery mechanism completed its work. It would also introduce the possibility of concurrent (and inconsistent) updates of the same replicated record by different nodes.

The developers decided to sacrifice absolute consistency of the replicated files at every moment in exchange for site autonomy and short terminal response times. To prevent conflicting updates to replicated data, they granted "ownership" of specific records to specific nodes and wrote the application in such a way that only the owner node could update a particular record. To prevent conflicting additions to replicated files, they pre-assigned various key ranges to certain sites and limited the additions to those ranges.

A customized delivery mechanism for delivery independent of the user was also developed. In it, each request to change a global record is put on a queue. This queue is emptied over a period of time by a software module that scrolls through the requests trying to update the remote databases. The module is programmed to perform the updates in the order in which they are received, preventing conflict.

The distributed manufacturing application was one of the first such programs to make extensive use of the network. It was implemented via standard Tandem products including a relational database manager and a terminal control program. If it were being developed today, there would be no need for the request queue or the customized delivery module, because a standard product now provides a reliable asynchronous delivery mechanism. This mechanism, known as Transfer, was developed to meet the future needs of distributed applications and interconnections between them.

The delivery mechanism consists of high-level (transport layer) software that gets information to people, devices, and processes in a specified time frame. Earlier approaches to network messaging (built into the operating system) were designed for interactive exchanges and could not be used unless the two communicating entities were available at the same time. If a particular node was not available, the user (or program) took responsibility for trying again at a later time.

The new software was designed to overcome this limitation. It attempts to deliver messages as soon as possible or within a specified time frame. If unsuccessful on the first try, it takes responsibility for periodic retry attempts thereafter. Delivery of the message or information package once and exactly once is guaranteed. If line failures, node failures, or disk controller failures make delivery impossible within the time period specified, the delivery mechanism notifies the requester of that fact.

Supporting support

Sales and service offices exchange information with hardware and software support centers by means of a product-reporting application. This network-based program provides a way for a field analyst (or, indirectly, a customer) to report a perceived engineering defect or bug, to request an enhancement to a product, or to ask a question concerning a product.

Field personnel enter product reports on software screens generated by the reporting application. Once a product report has been entered, the application forwards the report over the network to the appropriate support person. (If no support destination is specified on the report, an administrator decides where the report should go and forwards it.)

Although a report can be sent from any node to any other node (where both nodes have the application), it is normally sent from a field sales and service office to one of several regional technical support groups. In some cases, the regional group will be able to supply an answer and will simply return the report to the originating node. In other cases, the regional group will send the report to the corporate technical group, which will then either answer it or forward it again to the appropriate software or hardware development group.

Whenever a report gets forwarded, the application uses its electronic-mail interface to send a message to the report's originator. This keeps the person with the problem abreast of who is working on it. In such cases, the application also generates a mail message to the analyst to whom the report has been referred, as a reminder that someone is waiting for an answer.

Regardless of the exact path of a particular report, when a response is complete, the report is "returned to the field." All information pertaining to the problem is automatically collected and sent to the originating node by the application. To inform everyone concerned how the problem was resolved, and to make it easier to handle like problems in the future, an updated copy of the report (with the response) is automatically sent to all nodes that the report traveled to during its lifetime.

In addition, the application maintains a database on each individual node that contains all reports originated from that node as well as those that have been sent to it from other nodes. Thus, there is a fair amount of replication of the application's data throughout the network, even though each node has only a subset of the entire problem-reporting database. The database is frequently accessed by support personnel to identify outstanding problems that have already been reported, thus eliminating duplication of effort and ensuring faster resolution of problems for all customers.

Help for the business side

The network offers resources aimed at groups besides manufacturing and support. Business functions, from closing sales to processing orders to reporting financial data, have been computerized. Most of these are traditional, centralized applications, but some make extensive use of the network.

Products are built because someone wants to buy them. To help sales representatives sell them, the marketing department maintains a customer-reference database. Field salespeople who learn how customers or software houses use their products can submit that information to the database. Their colleagues can then view the data over the network and generate reports by industry, by application, or by product.

In this way, sales representatives can identify existing customers who might be able to help future ones. The customer-reference database is also a source of ideas on what to propose to prospective purchasers. And finally, a complementary-products listing provides a catalog of software packages available in the marketplace that can strengthen a representative's offerings.

Salespeople worldwide must often respond to "requests for proposals" because these requests usually present substantial opportunities. To eliminate the need to reinvent the wheel each time a proposal must be written, a headquarters proposal-assistance team maintains text files, accessible over the network. While they do not eliminate the need for writing and analysis by field sales, the text files substantially reduce the time it takes to prepare a customized proposal.

Once a sale has been made, it must be accounted for and the order administered. Contracts are sent to a sales administrator who verifies them and enters them into a marketing support application. The application sends an "electronic packing slip" to a manufacturing group. The message tells manufacturing to build and ship the order.

When the ordered equipment is shipped, a manufacturing person logs on to the marketing application and marks the order complete. (Order status is reflected in daily reports that are sent by the application to regional sales and service offices over the network.) The application then sends a message to an accounting and invoicing routine, telling it to bill the customer.

The accounting and invoicing application is tied to a database of ledgers, which it updates when bills are sent or payment received. It supplies sales reports to management people and answers their queries. It uses the network to broadcast reports to field offices and to tell accountants at the manufacturing site when a piece of equipment has been booked as a revenue item.

The budget model is another financial application that runs on the network. This tool is used by every organizational unit within Tandem in preparing capital asset and operating budgets for the coming year. Managers enter basic salary, hiring, and expense data on specially formatted screens, and the model calculates monthly, quarterly, and annual totals and generates reports that are used in evaluating spending plans

The budget model provides software that rolls, or merges, the budgets of various groups together automatically and generates an overall budget for larger organizational units. The results of local calculations can be forwarded over the network to headquarters where they are used in forecasting cash requirements and ensuring that a reasonable level of profitability is achieved by the company. ■

This is the first in a two-part series on Tandem's internal network operations. The second part will focus on electronic mail, the company's most widely used application, and take a closer look at network hardware and software.

Kent Madsen is the editor of the Tandem Application Monograph Series, produced by the company's field productivity program. David Foley is the technical manager of the Tandem network. Foley is responsible for architectural and strategic planning, analysis, and operations support.

Kent Madsen and David Foley, Tandem Computers Inc., Sunnyvale, Calif.

How multiprocessor nodes can become more sociable

This month's continuing look at Tandem's corporate network and its nodes shows a company increasingly dependent on distributed network applications like electronic mail.

At Tandem Computers Inc., electronic mail began as an ad hoc program allowing employees to send messages through existing machines used primarily for development, marketing, manufacturing, and order processing. However, today it is the most heavily used application in a worldwide internal network, and its importance is growing. During a recent 18-month period, network traffic doubled, but mail volume tripled.

Through electronic mail, Tandem salespeople from Singapore to Stockholm now collaborate and share information on a daily basis. Analysts from Montreal to Melbourne help each other respond to customer problems and to queries from prospects. Hardware repair personnel at distant sites communicate with manufacturing workers to resolve customer equipment problems. Managers at all levels use the network to keep in touch with employees and colleagues throughout the world. Moreover, electronic mail helps employees establish and maintain personal relationships across geographic or organizational boundaries. Thus, it contributes to a sense of community and teamwork.

One way of understanding the impact of electronic mail on the organization as a whole is to follow a fictitious employee through a typical day, noting how extensively this corporate resource is used. John is the manager of a technical support group of 40 or 50 people affiliated with the headquarters' marketing organization. The first thing he does when he arrives at the office in the morning is to log on to the local computer to scan his electronic mail, that is, to view a list of all messages in his electronic in-box.

Each item in the list of incoming mail gives the sender's name, the message type (original, forward, reply, and so on), and a subject line. Having scanned his mail, John can then select the most important messages to read first. Currently, there are four messages waiting. The first invites him to a strategy meeting that morning with the software development group. Another message contains minutes of a meeting he attended last week. John responds immediately to the invitation and files the minutes of last week's meeting (on disk) in electronic folders bearing names that he has specified.

John's third memo is a request from the vice president of marketing asking him to provide people to help with a design review for a large customer's application. John uses the electronic-mail editor to compose a short addendum and to add an enclosure to the vice president's request, providing more specific instructions. John then forwards the whole package to a subordinate, who will follow through for him.

He notices among his messages a request from a technical analyst in Australia for performance information on a new hardware product. John doesn't have the performance information needed, but he knows of someone at the performance research center in Germany who might. He forwards the message there and replies to the analyst in Australia, indicating what he has done, then heads out the door for his meeting.

All of this has been accomplished in less than five minutes. The messages John has sent are delivered almost immediately (or within the time frame he has specified). Thus, problems can be resolved and answers obtained very quickly, and John can be responsive without carrying details around in his head all day (or all week).

After returning from his strategy meeting, John again reviews his in-box. This keeps him in touch with people and problems from hour to hour, without being interrupted and without interrupting others. He typically checks his mail each time he finishes one of the day's major projects (a meeting, an interview, a lunch engagement, or a trip to the corporate library).

By using the in-house electronic mail, John avoids the frustration and inefficiency of "telephone tag" and cuts down on his long-distance telephone costs. He can leave lengthy messages on complex technical topics for someone who may not be available at the moment, and be assured that the message will not be garbled by an intermediary. In addition, John uses electronic mail to communicate easily with people on the other side of the world, whose business day may not overlap his own at all.

Letting digits do the walking . . .
John (and every other employee) is identified by a correspondent name. Electronic mail includes a way to look up the correspondent names of people on local and remote nodes. These names are listed in a directory application of the "public database," which contains such information as employee office locations, telephone numbers, correspondent names, and department affiliations.

It is easier to look up the correspondent name via computer than it is to find a number in a conventional telephone directory, because the computer does the searching. The correspondent database is updated weekly to incorporate all new hires, internal transfers, changes of address, and changes of status. With a query program, John searches the database for correspondent names by office location, node location, surname, or partial spelling of the surname.

John uses electronic mail to broadcast messages to large numbers of people via distribution lists. For example, when he wants to call a group meeting, he invokes a mailing list containing the correspondent names of everyone in his group. When he wants to announce some change in support policies, he invokes a much larger distribution list to send a message to everyone in the company. Thus, it is no more difficult to address many than it is to communicate with a few—or with one.

. . . and putting heads together
An interesting and well-used offshoot of electronic mail is an archive of technical information. Much of the mail that comes across the network is in the form of technical queries, broadcast to all employees. Workers send such queries when they need information on competitors, product performance, how to link particular devices, and so on. Answers are not hard to come by. Chances are that, out of the 5,200 Tandem employees who receive these messages, at least one will have the required experience or information.

At first, broadcast technical queries typically provoked dozens of secondary queries from people wanting to know what the original questioner had learned. To avoid having to answer these secondary queries individually, authors of technical query messages began to adopt the practice of identifying a file (accessible over the network) in which any replies would be stored. This allowed other interested parties to benefit from the exchange simply by accessing the reply file a day or two after the initial query was sent out.

The whole process has now been taken one step further. An administrative employee systematically reads and files (on disk) all technical queries related to a particular topic. After a week or so, the employee copies all the reply fields to a central node, stores them on disk (with the relevant query), and adds appropriate entries to a subject index.

This centralized repository, referred to as the "archive," is equipped with search software to facilitate information retrieval. Many employees may not be interested in particular informal exchanges of information at the time they occur. However, they can locate and access a stored record of these discussions whenever the need arises. The archive virtually eliminates the need for duplicate queries and provides an extremely valuable information resource, reachable from any network node.

There is a vast reservoir of information and insight within the organization itself, many times larger and more valuable than most formal, structured databases. The combined on-the-job experience of the 5,200 Tandem employees probably exceeds 30,000 years, and their combined college and university experience may amount to 20,000 years or more. Electronic mail and the archive allow people to share insights easily and store them for the use of others. These tools are instrumental in tapping a wealth of information and human experience.

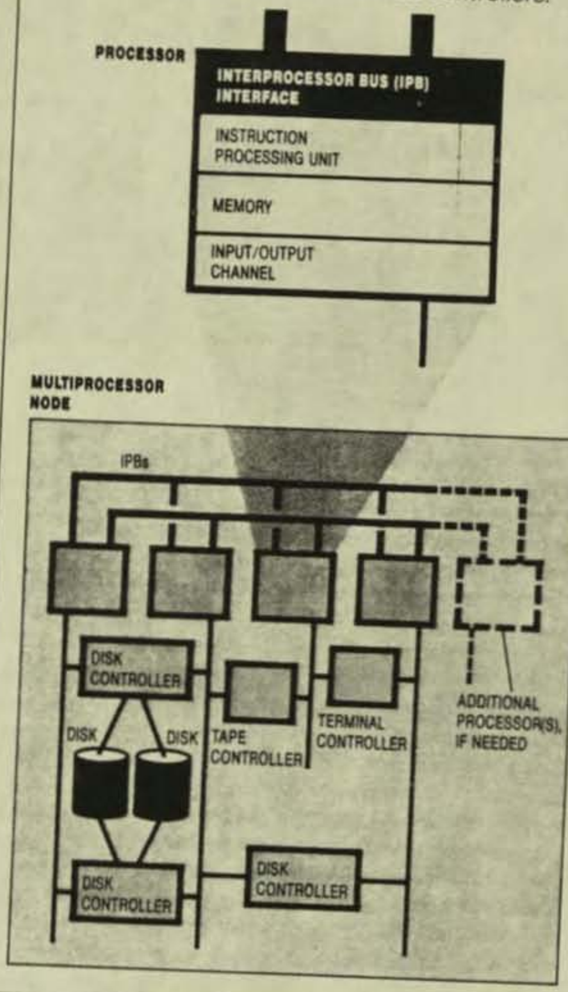
Electronic mail helps to preserve small-company interpersonal communications in the face of rapid growth. It also gives employees easy access to information resources within the company, regardless of where those resources may be. The archive and the electronic-mail network eliminate much duplication of effort and energy.

Overcoming hardware failures
While other applications are growing in use, electronic mail has become the lifeblood of the company's operations. It succeeds basically because users have come to trust the network to deliver their messages. This trust can be credited to one fundamental design principle, which underlies the network architecture and the hardware and software architecture at each node. The principle is that of "fault tolerance."

A fault-tolerant computer is one that can ensure continuous operation and data integrity in the face of any single component's failure. In addition, it must allow hardware service personnel to replace components and activate them without shutting the machine down. Such performance is crucial to computers in a network as large as Tandem's because the more computers there are, the more likely it is that a disabling failure will occur somewhere in the network.

A hypothetical computer running 24 hours a day and capable of offering 99 percent availability might have a mean time between failures of about two weeks. A two-node network composed of such machines would therefore average one failed node a week. With 200 nodes, a network would experience a node failure about every two hours. If we assume that communica-

1. Hardware. *Each node contains several processors (at least two) that communicate with a high-speed bus and dual-ported device controllers.*



tions lines go down as frequently as nodes and that the mean time to repair a line or a node is three or four hours, it is likely that a network that large would never be completely operational. Clearly, much higher standards of machine availability are required for large networks.

The overall availability of Tandem's corporate network rests on the continued functioning of its constituent hardware, as shown in Figure 1. Each node (including the backbone nodes) contains multiple processors linked by dual high-speed buses. Any one of these interprocessor buses (IPBs) can carry traffic at up to 13 Mbyte/s.

Each processor has its own IPB interface, instruction processing unit, memory, and input/output (I/O) channel (as well as its own power supply). However, the Guardian operating system distributed over these processors makes them appear to local or remote users as a single computer. It also allows them to cooperate with one another in processing individual transactions, to share the work load equitably, and to back one another up in the event of a failure.

The IPB uses a multiplexed, packet-interleaved protocol for high-speed interprocessor communications at the local node with a minimum of CPU interruption. However, it would be a mistake to view it as an ordinary communications link. From a logical standpoint, it is more like an internal bus in a conventional computer, since it ties cooperating elements of the local machine together and makes them appear as one.

As Figure 1 shows, designing for fault tolerance meant using multiple hardware components within a single computer. It also implied that techniques would have to be found for detecting failures, disabling problem components, and allowing for their repair and replacement without bringing down the rest of the computer.

Accordingly, hardware and software designers devised rigorous internal consistency checks for each processor. They believed that it was important to detect problems rapidly and halt a failed processor before it had a chance to contaminate data or disrupt the operations of the other processors. In keeping with this philosophy, if a processor finds that it cannot pass its own internal consistency checks, it will simply halt itself, allowing another processor to take over control of its peripheral devices.

If, by some fluke, a processor with a problem manages to pass its own internal checks anyway, there is yet another mechanism provided by which the processor can be restrained. Designers of the operating system decided that, once every second, each processor within a given node would send status messages over the IPB to all others indicating that it is alive and well. Also, every two seconds, each processor would check to make sure that it had received such a message from all the others.

When operational processors detect that one of the others is not following this established protocol, they can effectively quarantine the offender by declaring it "down." Control of its peripheral devices is then transferred automatically to the backup processor. In addition, backup applications program modules running in the other processors are activated to take over the work formerly assigned to the failed processor. When a processor is declared down in this manner, one of the other processors will also take corrective action to clean up any outstanding messages.

To allow the transfer of control of peripheral devices, hardware designers built dual-ported device controllers that can be connected to the I/O channels of two different processors. The controller is owned by only one processor at a time. However, if there is a problem either with the owning processor or with its I/O channel, operating system procedures switch ownership of the device and controller to the other processor (and its I/O channel). In this way, any device can be accessed even if the controlling processor or I/O channel fails. To ensure continuous operation even in the face of disk-controller failures, the disks themselves are dual-ported as well and can be connected to two different controllers, as shown in Figure 1.

The hardware designers also made provisions for the attachment of "mirrored disks" so that failure of a disk drive or its storage medium does not require that the computer be shut down. Mirrored disks are pairs of physically independent disk drives. Writes are performed on the disks in both drives; reads are taken from whichever disk has the shortest seek time. If one becomes inoperable, processing can continue with reads and updates directed to the healthy disk. When the failed disk is repaired, it can be restored to operation and its contents brought to a recent, consistent state. Then all updates performed on the other disk in the interim are transferred to it automatically.

Clearly, fault tolerance could not be achieved without the duplication of hardware components within a single computer. However, software runs on the hardware, and, therefore, if one processor is to take over for another, software components must be duplicated as well. This is accomplished through the use of "process pairs." (A process is a program module running in a particular processor.)

The operating system allows a "primary" process running in one processor to send periodic checkpoint messages to a "backup" process running in another processor. Checkpoint messages, usually sent before the primary process performs a critical task, such as I/O or updating a database, contain all the information that the backup process would need to take over for the primary one in the event of a processor crash.

If a processor goes down, backup processes running in other processors are activated so that they can continue the activities of the primary processes that were running in the now-failed processor. Because the backup process does not duplicate the activities of the primary one while the primary is still functional, it places only minimal demands on the processor in which it resides. Thus, processors can host backup processes as well as primary ones and do almost as much useful work as they would if the backup processes were not present.

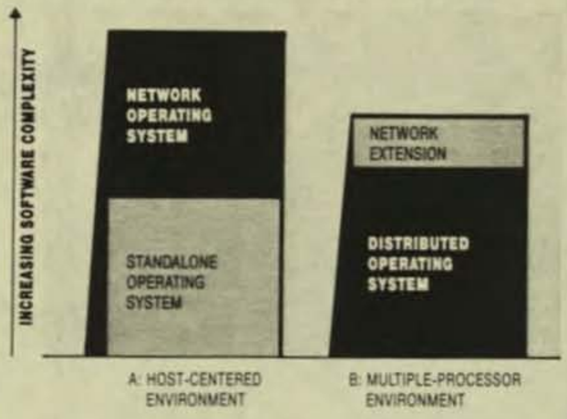
The operating system itself is protected by the implementation of process pairs. Early users of Tandem machines had to program their own primary and backup application processes (and devise effective checkpointing strategies) to get full protection. Now, however, a standard Tandem application development environment obviates the need for user programming of process pairs.

Sociable vs. self-centered computers

Hardware and software reliability are critical in a large network. However, there is an equally important need for machines that communicate and cooperate with their peers. Over the years, Tandem has come to believe that successful networking begins with the design of such "sociable" computers. A number of complex problems, which many people associate with networking itself, actually stem from the architectures of the machines being used as nodes. In most cases, such machines were originally designed with no thought for networking.

The economic need to preserve these architectures has forced many computer vendors to adopt an "add-on" approach to networking (Fig. 2). According to this approach, at each end of the communications line is a computer with an operating system designed for standalone processing. This operating system conditions the computer to think of itself as the main (if not the only) intelligent entity and to view anything attached to it as a peripheral.

2. Overhead of the loner. *To get a typical computer to interact takes a lot more code than if the machine already considers itself a network.*


INCREASING SOFTWARE COMPLEXITY
NETWORK OPERATING SYSTEM
STANDALONE OPERATING SYSTEM
NETWORK EXTENSION
DISTRIBUTED OPERATING SYSTEM
A: HOST-CENTERED ENVIRONMENT
B: MULTIPLE-PROCESSOR ENVIRONMENT

Layered on top of each node's operating system is a very substantial body of code, which must correct that myopic view. By clever and complex ruses, this "network operating system" overcomes the ingrained reclusiveness and egocentricity of the standalone computer, making it possible for that machine to converse with other computers in a network. Such communications is limited, though, and subject to rigid and somewhat arbitrary constraints, because each computer must be made to believe that it is still only talking to peripherals.

Inherent in this approach is a heavy communications processing burden that falls on the computers themselves (Fig. 2A), stemming from the fact that, in essence, the network operating system has to work against the local operating system. In addition, this approach entails a formidable burden of mounting software complexity. This burden is like an irrevocable tax on every network user, manager, and application, and it has a substantial negative impact on productivity for as long as the network exists.

This complexity increases exponentially as more (and different) computers are added to the network. And even if the linked computers are identical, networking is almost always a strange new world, with remote access procedures and syntax rules vastly different from those used at the local level.

Intranode communications

By contrast, the software that supports Tandem's corporate network does not present a new world to users and programs, but rather functions as an extension of the local environment. In a very real sense, networking is not layered on top of the nodal operating system, but built into it. No separate network operating system is needed.

Nodes in the corporate network routinely engage in "internal dialogues." Since each computer is, in and of itself, a local network composed of multiple independent processors, a machine's processing consists of conversations between its constituent parts. (Artificial intelligence theorists suggest that people may also think this way.) The same communications mechanisms that the operating system uses to blur processor boundaries on the local level are effective in blurring node boundaries within a network as well. When coupled with basic internode communications protocols, these local mechanisms (built into the operating system) contribute greatly to network operations.

The most important of these mechanisms is a "message manager," upon which the entire local operating system is based. Messages are important in the Tandem computing environment because the operating system itself is not a single monolithic program but a collection of "interrupt handlers" and processes. Each process has particular areas of responsibility and must communicate and cooperate with others (through messages) to get work done. These messages must flow, not only within a single processor, but also among the various processors (none of which shares memory).

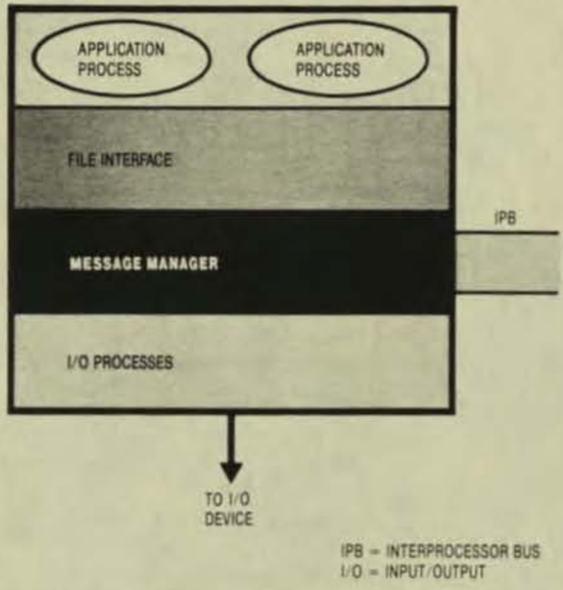
Copies of the most basic operating system processes (such as the "monitor" and the "memory manager") run in every processor. However, some functions, most notably input and output, must be handled by the particular processors to which the I/O devices are physically attached. This posed a problem for the early designers because all the hardware and software entities in a multiprocessor computer needed access to these I/O resources.

To resolve the problem, the designers developed the message manager, which allows a process to communicate with another process anywhere in the local machine simply by providing the destination entity's symbolic name. The message manager takes full responsibility for locating the named process and getting the message to it regardless of where that process may be running. Disks, tape drives, and terminals are all associated with processes. Thus, the message manager provides a way of addressing and accessing such devices from any location at all in the multiprocessor machine.

A message, as defined in the operating system, is bidirectional. It consists of a "request" for service and a "reply." Several such messages may be required to carry out a given operating system function. For example, a monitor process may be asked to create a new process. To do so, it must do some work and then make requests of several other operating system processes to gather the resources needed.

One of the requests would be to a "disk process," asking that space be allocated on disk as virtual memory for the new process. When the disk process has allocated the space, it replies, indicating that the work has been done. (This reply completes the message.) Other requests are made as well, and when the monitor process has seen to it that all of the necessary resources are in place, it replies to the entity that made the request, indicating that the process has been created.

3. Software. *Segmented code allows the operating system to span processors over an IPB and hide details of the hardware from user programs.*



It is easy to underestimate the value and uniqueness of the message manager. There is certainly nothing special about the concept of program modules passing information to one another. That happens all the time in conventional computing environments that lack a formal messaging scheme. Usually, one program module places information at a specified location in memory, and another picks it up. By contrast, the message manager is a general-purpose mechanism for getting a message between any two processes in a multiprocessor machine. It does not assume that communicating program modules will inevitably be running in the same processor or that memory is shared between the processors involved.

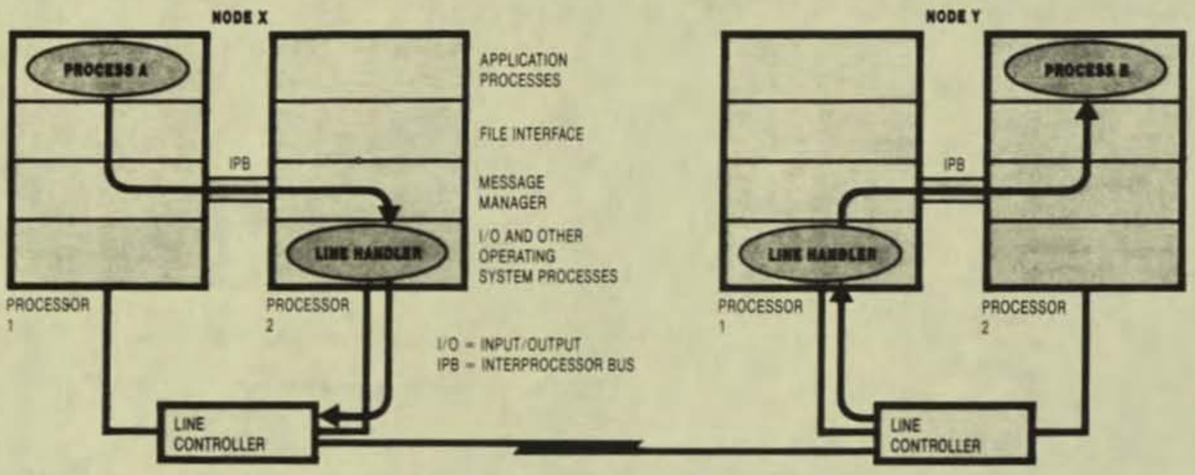
Accessing resources within nodes . . .

Applications processes are not allowed to communicate directly with the message manager or with basic operating system processes. However, the processes can make use of the operating system through a "file interface," which ensures that such interactions do not accidentally create problems for the user or for the machine (Fig. 3).

The file interface works with the message manager to allow application processes to communicate with entities such as other processes, files, and I/O devices, by a single set of calls. That is, such resources can all be referenced by means of pre-assigned symbolic file names. Application processes do not need to know physical locations since the file interface can access operating system tables that keep track of the entities.

To distribute the load over multiple processors (as

4. Network travel. *Application process A need only know the unique network name of process B, and the operating system handles the trip for any request from A. Thus, process A can access remote files, programs, and devices (through processes like B) almost as easily as it can reach local ones.*



well as for other reasons), application programs typically have a structure similar to the operating system in terms of requester and server processes. They also use the message manager's basic request/reply protocol.

For example, an application-type requester process running in one processor might be programmed to control a group of terminals and validate input from them, while an application server, running in another processor, might be programmed to formulate database queries. As a particular transaction is received, the requester validates the initial input and then sends a request to the server (via the file interface) that indicates what work needs to be done. The server process accesses the database (again via the file interface), retrieves the desired information (or updates the appropriate records), and then replies with the requested information or a confirmation that the work has been completed.

. . . and throughout the network

Since the computers that compose the network already consisted of multiple, communicating entities, it was far easier to interconnect them than it would otherwise have been. A group of four or five people spent only about a year developing the required networking software. This software, known as Expand, allows the local file interface and message manager to address and communicate with processes, files, and devices anywhere in a Tandem network.

The networking software consists of line handlers, a proprietary protocol for guaranteed data integrity, and a network control program (NCP). The NCP, which runs at every node, monitors and logs changes in network status.

Routing responsibility is distributed. The NCP maintains a copy of the network routing table (NRT) in each processor. The NRT lists the location of all of the other network nodes. NRTs are used to determine the best path to other nodes and to establish the communications link. (Thus there is no centralized routing that can fail and paralyze the entire network.)

The file interface bridges the network by allowing local processes to access files, processes, and devices anywhere in the network through a single set of calls. The addition of a node specifier to the symbolic name uniquely identifies these resources throughout the network.

Listening in on network dialogue

The networking software is basically an extension of the services provided by the local message manager. Just as the message manager software allows one process to send messages to others within the local machine, its networking extension allows a local process to send messages to other processes running at remote nodes in the network.

As mentioned last month, operations people and other users of the corporate network can, with proper security authorization, log on to a network node in California (or anywhere else) and do work on remote nodes. For example, with successive two- or three-word commands, they can start a program running on a machine in New York, instruct that program to access a file on a disk volume in Atlanta, and print out the results for another employee on a device attached to a computer in Chicago. Also noted was the fact that the command structures by which these operations are carried out are identical to those that would be used locally for similar operations except that, in each case, the program, file, or device name must be further qualified by a node name.

This sequence of events can be used to illustrate how the operating system entities work together. In fact, the different operations were achieved through one mechanism: messages. Process-to-process messages pass first through the local file interface and message manager, next through local and remote line handlers, and finally through the remote message manager and file interface software layers.

Consider process A running on node X within the corporate network (Fig. 4). If process A (an application process) needs to communicate with a process running at the local node, it gives the message, along with the name of the destination process, to the file interface. The file interface checks the message, makes sure that it is legitimate, consults operating system tables to find out where the destination process is running, and hands the message to the message manager.

The message manager takes responsibility for delivering the message and for returning the reply from the destination process. If the destination process is an operating system process, it replies directly through the message manager. If, on the other hand, it is a user application process, it must use the file interface to pass the reply to the message manager.

When process A wants to communicate with process B located at remote node Y, it proceeds the same way, giving a message and destination process name to the local file interface. However, this time the name consists of the process name appended to a node name (for instance, "nodeY.processB"). The file interface, discerning that this resource is not available locally, accesses the network routing table.

From the information contained in the table, the file interface determines that the message must go through a specific network line handler to reach its destination. It therefore preserves the name of the destination process but tells the message manager to deliver the message to that line handler. (In the event that there are two or more line handlers leading to the remote node, the routing table will indicate which path is best, based on the speed of the communications lines and the number of intervening nodes.)

When it receives the message, the local line handler compresses and packetizes the data and sends it over a communications line to another line handler at node Y. This line handler reassembles and decompresses the data and strips off the node portion of the destination process name. It then hands the message to node Y's message manager, which uses operating system tables to locate process B and present the message to it. Process B does whatever it was instructed to do and then replies. The message manager takes responsibility for seeing to it that the reply retraces the path of the request back to process A.

Through this basic mechanism, a user in California can log on to a local machine and enter a command that will start a program running on a machine in New York (Fig. 5A). First the user accesses a local "command interpreter" process, which reads input from the terminal through a terminal I/O process. In response to this input, the command interpreter (acting as a requester process, analogous to process A in Figure 4) sends a message to an operating system process, the "monitor" (a server, analogous to process B), running in the New York machine. The monitor process starts up the program and replies.

The new program started in New York then requests input from the user terminal in California (by sending a message to the terminal I/O process). In response, the user instructs the program to access a disk file attached to a machine in Atlanta. As shown in Figure 5B, the New York program (now analogous to process A) sends a message to the disk process in Atlanta (analogous to process B). The disk process does the necessary work and replies with the requested information. The program in New York then responds to the California terminal and requests further input.

In response, the user in California instructs the program in New York to print out the results of the disk access on a machine in Chicago. As shown in Figure 5C, the New York program sends a message to a line-printer process in Chicago. The line-printer process sees to it that the information gets printed, and replies.

In all these cases, the destination process does not have to be aware of the origin of requests. From its point of view, the message might just as well have come from a local process. This is because all the destination process has to do is to hand the reply to the message manager (or file interface), which will see that the reply finds its way back to the source process, following essentially the same path as the request.

Because Tandem chose to use standard software products at all nodes in the network, the syntax in the above operations is the same as would be used to perform similar operations locally. The only change needed is that the file, process, or device name would have to be qualified by a node name.

Managing growth and applications

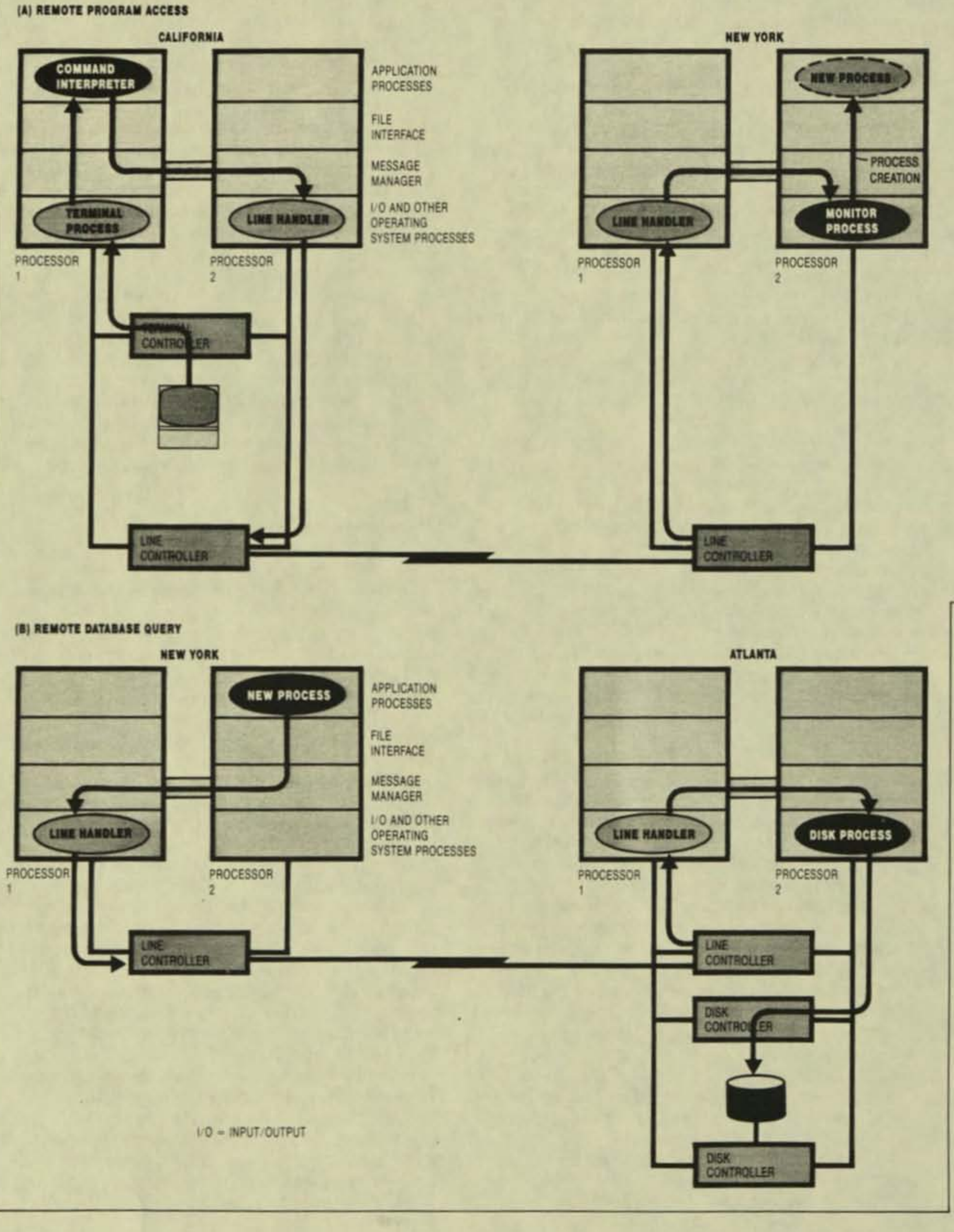
All applications, even those running on a single node, are designed in terms of the requester/server concept. As a result, these applications can be distributed easily when the need arises.

For example, a particular order-entry application started out with all orders called in to a central point and stored on disk. This application became distributed when people in the branch offices were able to key in the information themselves. However, there was still a need to keep a copy on a central disk. Since the application was written using requester and server processes, it was simple to move the requester portions to the regional nodes while the servers remained at the central site to update the database and to return acknowledgments to the requester.

Growth can affect user applications almost as drastically as distribution. Most environments can only add processing power by purchasing a larger machine, often with a different architecture and operating system. Such a move almost inevitably entails a significant software conversion effort. Even if manageable locally, a capacity upgrade can become difficult in a network setting. Dozens of applications and remote nodes that regularly access the newly configured local node must then also be changed.

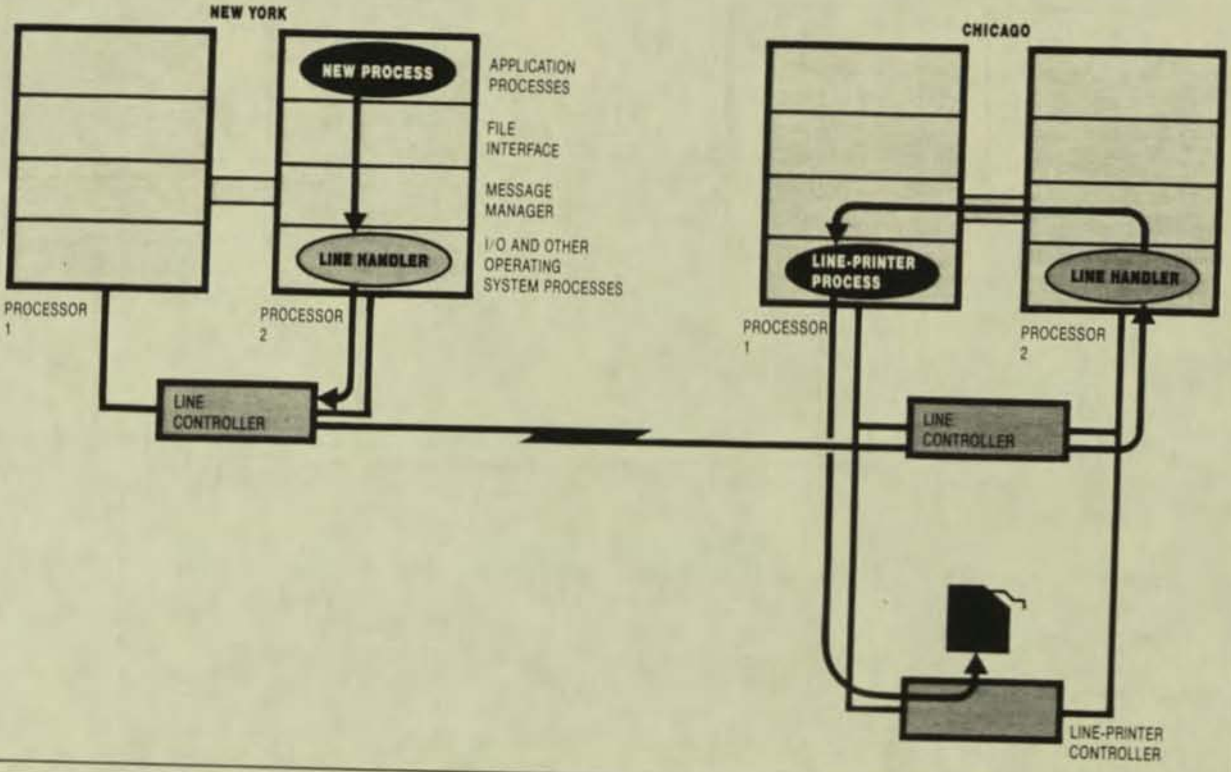
In Tandem's corporate network, however, increased data processing demands at local nodes have been met without producing waves locally and throughout the network. Nodes can be expanded from two to 16 processors, and duplicate copies of requester and server processes can be run in the new processors. Thus, the applications can handle twice the work load,

5. The network as single virtual machine. *Authorized users can log on to a network node in California (or anywhere else) and, with single commands, start a program on a node in New York (A), instruct that program to read a disk file in Atlanta (B), and have it print the file on a device in Chicago (C). Processes that be-*



gin as or are created by servers (such as the new process in New York) can turn around and make requests of other processes. They can also use the same techniques in communicating with remote nodes. User programs use the file interface, while operating system processes go directly via the message manager.

(C) REMOTE PRINTING



but programmers do not have to change a single line of code.

The network is remarkably homogeneous. All working nodes have the same architecture and run identical versions of the nodal operating system. Along with the local expandability of each node, this homogeneity greatly facilitates network management from a logical standpoint. It is much easier not having to deal with new operating systems as the network grows. The ability to reuse the same box also means that, physically, remote sites are more readily upgraded. Even at the central computer center, expanding a node is far less disruptive than replacing it.

Expandability also aids in the management of distributed applications. When the use of such applications grows, certain components such as disks, controllers, or CPUs may become bottlenecks. Modules can be added to replicate these components. In the example considered earlier, an order-entry application was replicated to the regional nodes. When a region grows, the requester program can be duplicated and run on a different CPU. The application then doubles its capacity without being rewritten, simply by having another copy of itself spawned.

New processors are often added to overworked nodes to relieve the burden on the existing processors. If several application processes are then moved from their original processors to the new ones to redistribute the load, the application programs do not have to be changed. They will run just as they did under the old configuration except that, overall, response times will be better.

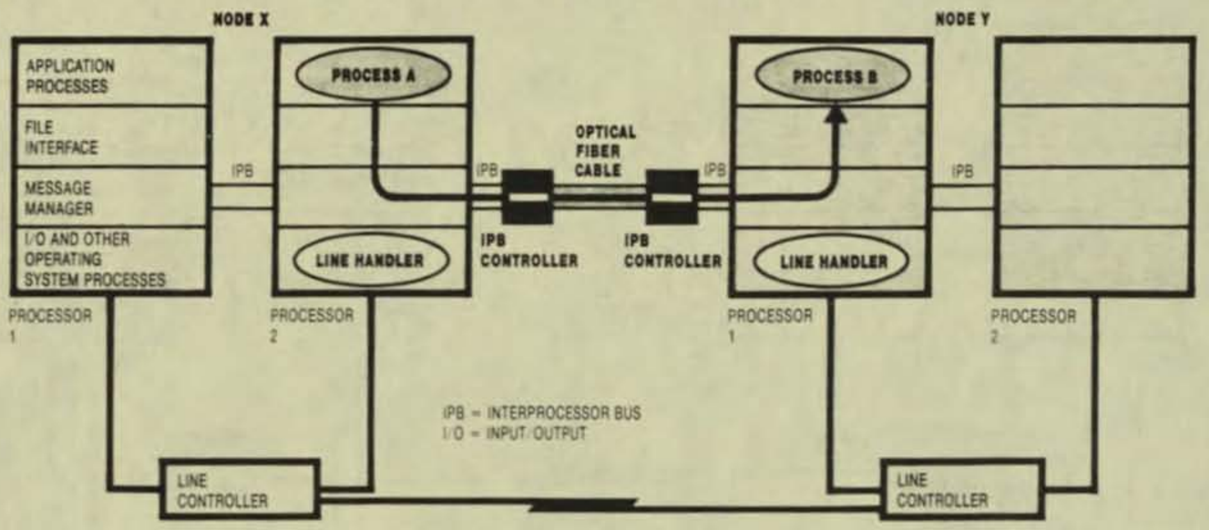
Looking to the future

In any structure as large and multifaceted as the Tandem corporate network, change is a constant. Use of the network's long-haul circuits has grown enormously in the recent past. For example, the bandwidth required on the backbone link between the two U. S. coasts began at 10 kbit/s in the middle of 1982. It doubled the following year. By 1984 it was up to 56 kbit/s, and by the end of this year it will have doubled again. Projections for the fourth quarter of next year show a need for another 56 kbit/s, a full T1 (1.544 Mbit/s) by the end of the decade.

These projections are for the network's data traffic only. The introduction of facsimile transmission capability to the network, which is currently taking place, will undoubtedly increase the rate of growth. Facsimile applications expected soon are a tie-in to electronic mail, store-and-forward facsimile switching, and dialing in or out to distribute and archive facsimile images on disk. The manufacturing division is starting with 40 to 80 pages a day, and the number is expected to increase.

To meet future needs for data transmission band-

6. Lightwave 'telepathy.' *Two nodes joined in a lightwave cluster exchange messages CPU-to-CPU, bypassing the traditional computer "talk-paths" (communications line, line handlers, and controllers). Transfers such as reads, writes, and protocol matters take place at the level of the message manager.*



width, the network support group is making plans to install satellite links between selected backbone nodes. These links will not replace existing terrestrial lines, which are needed for interactive traffic because of their low propagation delay. They will, however, provide the bandwidth needed to carry large volumes of mail and other transmissions for which rapid response times are not important.

Local traffic (particularly at the company's headquarters) is increasing roughly 50 percent faster than long-haul traffic. Clusters of machines linked by the company's lightwave product will therefore play an increasingly important role in the corporate network. Up to 14 computers can be linked in a ring via double-circuit optical fiber. The entire subnetwork thus created may contain up to 224 processors, each capable of processing 4 million instructions per second. The main headquarters subnetwork will initially contain over 100 processors.

A lightwave subnetwork is very much like a single, large, powerful node for two reasons. First of all, the transmission medium offers the speed and bandwidth needed to ensure that response times are essentially the same whether processing tasks involve communications within or between individual computers. Each of the four fibers (two full-duplex circuits) carries data at 10 Mbit/s, for an aggregate data transfer rate of 40 Mbit/s (the theoretical optimum; actual user throughput depends upon the application).

Secondly, the message manager allows users and executing programs to communicate with or access any other executing program, peripheral, or file in the corporate network simply by supplying its name and the relevant node name. As Figure 6 shows, the lightwave ring is designed to transport messages between processes. It sends them directly over the interprocessor bus, without using I/O channels or controllers. Clustering nodes on a lightwave ring takes advantage of higher-speed hardware. This method consumes up to 80 percent less CPU time than does the conventional way of handling data traffic, in which data leaves a node via a line handler. It also provides much faster response times. ■

(This is the second part of a two-part article.)

Kent Madsen is the editor of the Tandem Application Monograph Series, produced by the company's field productivity programs group. David Foley is the technical manager of the Tandem network. He is responsible for architectural and strategic planning, analysis, and operations support.

The authors gratefully acknowledge substantial contributions to this article by Denis Winn, operating system specialist with Tandem's software education department.

Further reading

Bartlett, J. F. "A NonStop Kernel." *ACM Operating Systems Review*, vol. 15, no. 5, December 1981, pp. 22-29.

Forsdick, H. C., Schantz, R. E., and Thomas, R. H. "Operating Systems for Computer Networks." *Computer*, vol. 11, no. 1, 1978, pp. 48-57.

Gray, J. and Metz, S. "Solving the Problems of Distributed Databases." *Data Communications*, October 1983, p. 183.

Holden, J. B. "Experiences of an Electronic Mail Vendor." *Proceedings, National Computer Conference*, AFIPS Press, 1980, pp. 493-497.

Holland, R. "Distributed Databases: Decisions and Implementations." *Data Communications*, May 1982, pp. 97-111.

TANDEMCOMPUTERS

Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, CA 95014-2599
(408) 725-6000

WORKPLACE DEMOCRACY IN TANDEM MANUFACTURING: A THEORETICAL FRAMEWORK

Dr. Nancy Dixon
Participative Management Advisor for Manufacturing
Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, Calif. 95014
408/725-6366

Workplace democracy or participative management, as it is sometimes called, is a systematic process designed to implement the five major corporate goals of Tandem in the manufacturing setting; people-oriented environment, satisfied customers, clear direction for employees, superior products and sustained profitability. As a total corporation Tandem has been able to make these goals not just words on paper, but a living reality for its employees. Tandem employees have always felt that they were trusted with high level information, listened to when something important needed to be communicated, and valued for the unique contribution each makes to the organization. Employee attitudes such as these exist at Tandem because the corporate leaders, who within a small organization, have been able to exert a strong impact on its culture. Positive employee attitudes have been purposefully nurtured through open door policies, open and shared information, tele-conferencing, beer busts and above all a corporate team that modeled its own beliefs.

Tandem has grown, and as it becomes still larger, there is need to supplement the informal processes of the past to retain the best of Tandem culture. Workplace democracy is a way to systematize the Tandem philosophy in the manufacturing setting. The basic beliefs of workplace democracy are rooted in the culture of Tandem. The systematic process used to implement those beliefs is modeled on the democratic process of the United States.

PHILOSOPHY OF THE LEADERS

The most important element of workplace democracy is the belief system that underpins it. That belief system is represented in the following statements:

Given adequate information, employees will make decisions that are in the best interest of both the organization and themselves.

Employees are better able to make decisions related to their job functions than are leaders who are further removed from the process.

Given a positive environment, employees want to do work of which they can be proud.

Employees want to be part of and to identify with something larger than themselves.

Employees want to learn and grow in their work.

These beliefs are so important to the success of workplace democracy that the first step in the implementation process is to have the leaders of the organization hold a discussion concerning their own beliefs about workers. If the management team arrives at a belief statement that is consistant with the spirit of the beliefs listed above, then the participative process can be implemented effectively. However if the management team, as a whole, leans toward opposite beliefs, i.e. employees are motivated only by their own self-interest, incapable of understanding management level information, and unwilling to work unless closely supervised, then a participative style of management should not be attempted at that site. The decision of whether or not to implement workplace democracy, is not meant as a judgement which implies that one set of beliefs is right and the other is wrong. The collective experiences of that management team may honestly lead them to the second conclusion. However sincere these beliefs are, workplace democracy should not be attempted under such circumstances because without holding positive beliefs about employees, leaders cannot honestly let go of some of their authority in order for employees to participate. Attempts to do so come off as phony, such as limiting participation to inconsequential decisions or attempting to manipulate employees around decisions that have already been made. Most employees have been manipulated often enough in their work careers to recognize a con job. Thus, workplace democracy should only be attempted where the beliefs of the leaders are consistent with the management philosophy they are attempting to implement.

ELEMENTS OF WORKPLACE DEMOCRACY

The systematic process of workplace democracy has three major elements. Each element is so interwoven with the others that it presents a somewhat false picture to separate them out. However, for the purpose of clarity, they will be discussed separately here. The three elements are alignment, representation and information.

The first element of workplace democracy is alignment. Of the three elements it is the most easily overlooked. The other two, information and representation, are visible as wall charts or committee meetings. Alignment means that the members of the organization agree upon a common goal that each feels is in the best interest of all. Alignment differs from a company goal, in that a goal is usually set by management, and is primarily addressed to profitability. Whereas alignment occurs when the goal or vision is one that will clearly benefit all. England, during the Second World War serves as a historical example of alignment. The people aligned around the belief that England as a country would survive, even if German bombs leveled every building. It was the fighting words of Sir Winston Churchill around which the people aligned. There are many examples of alignment in sports, matches when players have focused on winning the big game rather than on any personal glory the game might bring. Organizational alignment occurs when the goal is inviting enough that employees freely choose to work toward it. The goal in Tandem manufacturing might be to produce the world's finest computer, or to make our plant the best place to work in the United States.

An effective beginning to achieve alignment of a manufacturing site is to have the whole plant meet in a full day retreat to establish their common goal. The questions asked are, "What quality of product do you want to build? What kind of environment do you want to function within? and What kind of leadership will help you achieve these goals?" In this process the belief of the organization's leaders is put to test because the process must be honest and open if the group is to truely become aligned.

Representation is the second element of workplace democracy. Representation, as the term is used here, means a systematic process that allows employees at all levels an opportunity to share in making decisions. The representative process is philosophically based on the democratic principle that all, rather than an elite class, are capable of making decisions that will benefit all. In the manufacturing setting a number of groups serve as decision making bodies. The largest group is the weekly plant meeting attended by everyone in the plant. Here information on organizational performance is presented and discussed. Three committees also meet weekly, reporting their activities at the plant meeting. They are the Process, Productivity and Materials committees. These committees gather the information that is presented at the plant meeting, and have the responsibility of identifying and solving problems related to each of their three functions. Work group meetings are the third form of representation. These brief, daily meetings are the opportunity for employees to review how they did the previous day and to plan their daily work. All three types of meetings are facilitated by a member of that group.

Representation has little meaning unless employees have information with which to make decisions. Thus, an information system is the third element of workplace democracy. In the democratic process of the United States access to information is essential as exemplified by freedom of the press and open congressional records. In workplace democracy or participative management information is equally essential. Employees are taught to collect and understand management level information. Care is taken that the organizational performance of the whole plant is represented in these measurements. Thus productivity measures such as Cost per Standard Hour and Work in Process Cycle Time are used. Quality is measured by Mean Time Between Failure. Schedule is charted as Plan vs Actual hours. It is around this type of information that representative decision making occurs. These measures are reported by the employees at weekly plant meetings.

In addition to the information presented at plant meetings, a weekly newspaper is distributed that shows the collected information in graphs. The newspaper is created by the employees themselves. In addition to organizational performance information, the newspaper is used to communicate the agreed upon goals of the plant.

Another way information is kept and displayed is by having wall charts that employees maintain in each work area. These charts reflect the amount of work the group has committed to in their daily work group meetings, the work accomplished to date and various measures of quality. Thus feedback on each group's performance is daily and visible.

It is not uncommon in manufacturing for some type of performance measures to be computed. It is however unusual for those measures to represent anything other than direct labor effort. It is even more unusual for the information to be gathered and utilized by the employees themselves.

DEFINITION OF WORKPLACE DEMOCRACY AT TANDEM

Workplace democracy is defined as an organizational structure and systematic process which supports the utilization of the whole person (values, creativity, knowledge and skills) in the work environment through:

* open communication

* participation in decision making at all levels

* respect for each individual

* immediate feedback on organizational performance

One of the concepts incorporated into this definition is that the employee's ideas and planning skills are utilized as well as his/her manual labor. However, it is important to note the word "supports" rather than "demands" in the definition. Just as the democratic process of the United States does not require citizens to vote or run for congress, so an employee in workplace democracy is not required to participate. An employee may choose to simply do the assigned task and leave participation to others.

Workplace democracy as it is designed for Tandem manufacturing can be viewed as a framework upon which each plant or organization may construct its own unique system. That system may vary in terms of the extent of employee participation, the number and type of committees, the productivity and quality measures used and so on. It is assumed, however, that for workplace democracy to be successful in any organization it must incorporate the three elements of alignment, representation and feedback of organizational performance.

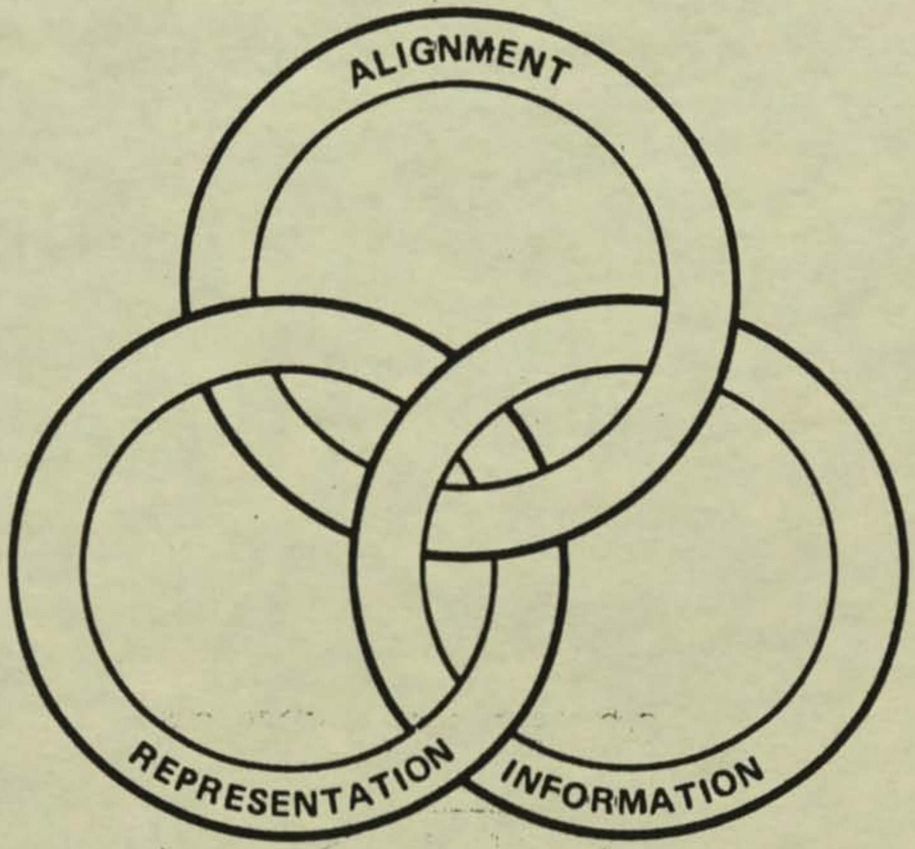
It is important not only to say what workplace democracy is but also what it is not. It is not a system that requires every employee to help make every decision. Some decisions are appropriately made by the leadership at each level of the organization. For example, the organizational charter for manufacturing is defined at the corporate level and individuals, including the plant manager, are employed to carry out that charter. In Tandem manufacturing the charter is to convert raw materials into usable finished goods with; a competitive cost, quality upon receipt of finished goods, reliability, on time delivery and predictability, while maintaining a positive work environment. Neither the employees nor the leadership of a manufacturing plant are free to choose not to convert raw goods into usable product. As illustrated by the organizational charter every manager has certain decisions he/she can make and others that are made at a higher level. Managers can only allow others to participate in decisions that are within the scope of their own responsibility. They cannot share power they do not have.

WORKPLACE DEMOCRACY AS A COMPLEX CONFIGURATION

As illustrated in figure one, the success of workplace democracy cannot be found in any one element. It is the combined effect of all three elements that makes it effective. Many organizations have attempted to put into place one of the three elements, believing that that element alone would increase productivity. For example, a great number of employee-involvement programs have been implemented in organizations with only brief success because the employees were given no information base to use in making decisions. Other organizations have attempted to align employees around an organizational goal, again achieving only limited success because employees had no systematic way to input their ideas into the organizational hierarchy.

It cannot be overly stressed that workplace democracy is not simply allowing employees to make some of the decisions. Rather workplace democracy, as it has been developed in manufacturing at Tandem, is founded on a positive belief in people and derives its effectiveness from the interrelatedness of its three elements, an information system that is used to give immediate feedback of organizational performance, dynamic leadership capable of aligning the organization around a common vision and systematic represenation of all levels in the decision making process.

WORKPLACE DEMOCRACY



A40035-14

Silicon compilers tame 10,000-gate-plus ASICs, gate arrays

Traditional design methods that manipulate gates or simple macrocells become unwieldy at densities of tens of thousands of gates per chip. However, silicon compilation offers a new paradigm for the management of these large, complex designs.

William J Stenzel, *Tandem Computers Inc*

Silicon compilation is an alternative to gate arrays and standard cells for implementing ICs. A silicon compiler creates layouts of part or all of a chip from a high-level description. Therefore, when using a silicon compiler, you can focus your efforts on high-level verification and system-timing issues rather than on low-level circuits and gates. Conceived for custom chips, silicon-compilation techniques also prove useful for gate arrays containing more than 30,000 gates.

Unlike gate-array or standard-cell designs, which require a gate-by-gate approach, silicon compilers allow you to manipulate high-level functions. You'll find silicon compilers' advantages most evident when you design data-path blocks and specify compiler functions in terms of latches, register files, multiplexers, arithmetic and logic units (ALUs), and shifters as well as in terms of their connections to data buses. In contrast, when using standard cells or gate arrays, you build up high-level functions from a library of low-level elements, such as gates and flip-flops, and then route signals between them.

Compiled chips, like standard-cell chips, require full wafer processing, in contrast to gate arrays, which require only the personalization of an existing base wafer's metalization. Consequently, nonrecurring expense (NRE) and time required to fabricate prototypes of a compiled chip are similar to those of standard-cell chips and higher than for small gate arrays.

Silicon compilation excels at highly structured, regular functions, such as data paths and memories. Such functions appear, for example, in digital signal processors, which are primarily structured blocks with most of their interconnections going to the external pins. Further, more complex chips may have islands of regularity in a sea of interconnection.

Look at your application to decide if your design is large enough to benefit from compilation. To consolidate only a few thousand gates of random logic, you might find a gate array more cost-effective. Designs in excess of 10,000 gates are the best candidates for compilation.

Compiler vendors offer two types of tools

When evaluating silicon-compiler vendors, you will find that some offer generic tools while others offer tools specific to their own foundries. Two that offer generic tools are Silicon Compiler Systems Corp. (San Jose, CA) and Seattle Silicon Corp. (Bellevue, WA). You may compile your design for fabrication at one of serveral foun-

Silicon compilers allow you to manipulate high-level functions; you needn't work at the gate level.

dries these two companies' compilers support. In contrast, VLSI Technology Inc. (San Jose, CA) offers compilation tools customized to VLSI Technology's fabrication process.

Other differences among compiler vendors lie in the features their compilers offer. Compilers from Silicon Compiler Systems and VLSI Technology are complete packages, including data entry, simulation, and timing analysis. Seattle Silicon's offering requires a third-party CAE workstation and software to provide many functions, such as schematic capture and logic simulation. Finally, Silicon Compiler Systems and Seattle Silicon offer tools that let sophisticated customers such as semiconductor houses create their own block compilers. This feature enables such customers to add proprietary functions or to tune functions to their own processes.

Changing your mindset

When first using a silicon compiler, you might need to make some changes to your design approach, particularly if your experience is with board-level TTL. Estimating whether or not a given design will fit onto a chip requires a set of guidelines that differ from those for pc-board layout. For example, when you size the pc-board area for a TTL design, half a dozen inverters and a 64k-byte RAM take about the same amount of space. Conversely, area within compiled chips depends more on the amount of logic in a function. Moreover, interconnections within a chip consume considerable space, occupying as much area as active logic in many cases.

Circuitry details, too, might dictate some changes in your design methods when you first use a silicon compiler. For example, although most systems use an edge-triggered register methodology that's familiar to TTL designers, Silicon Compiler Systems' Genesil differs in that it supports a latch-based scheme with 2-phase, nonoverlapped clocks and precharged buses, more typical of custom IC designs.

In contrast to edge-triggered schemes, which use registers as state elements that sample inputs at a clock edge (and which allow data an entire clock period to propagate through the subsequent logic before being sampled at the next clock edge), 2-phase latch-based schemes use transparent latches as state elements. The latches propagate data from their inputs to their outputs when the clock that enables them is asserted, and they hold their data when the clock is not asserted.

In the 2-phase scheme, a clock generator converts a master clock into two phases: phase A and phase B. Phase A is asserted when the master clock is high; phase B, when the master clock is low. These two phases are interlocked so that they are never simultaneously asserted (hence the term "2-phase nonoverlapped").

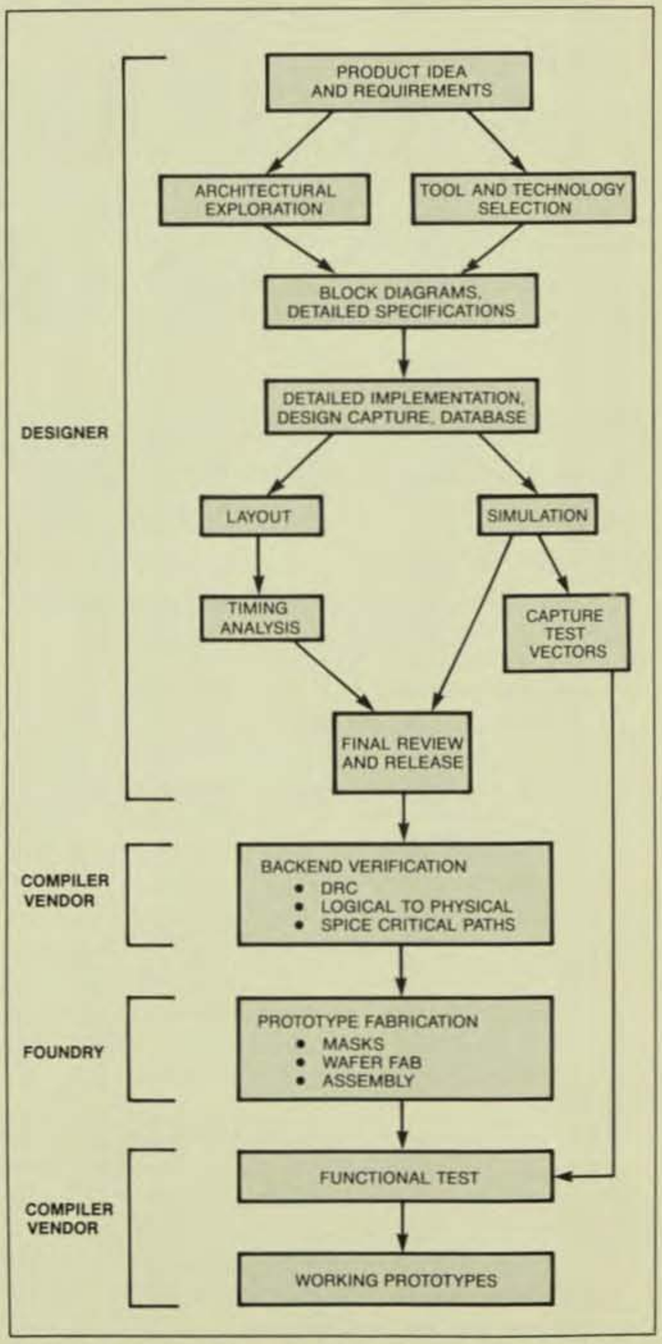


Fig 1—The designer, the compiler vendor, *and the silicon foundry must all cooperate to bring an IC from an idea to a working chip.*

You must alternate which phase controls the latches throughout your design: Signals originating in a latch controlled by phase A must terminate in a latch controlled by phase B. When a phase A latch is propagating, the subsequent phase B latch is holding, and vice versa, thus eliminating race conditions. Transparent latches require roughly half the space and internal logic as edge-triggered registers. Hence, they provide an advantage in chip size. After becoming familiar with the technique, you will find the latch-based scheme to be flexible and useful for pushing the performance of critical timing paths.

Design process

Fig 1 charts the steps—covering several design disciplines—that you must take to transform your product idea into a working chip. You must organize major projects with an awareness of these disciplines. During the specification and initial design phase, several engineers might each concentrate on a functional area of the design. As the project progresses, each engineer might focus on simulation, layout, or timing analysis.

In the architectural-definition phase, you must determine how to map your idea onto chips and evaluate design alternatives against system cost and performance requirements. Using the compiler, you can create trial designs of major blocks for evaluation. Because you describe blocks by their functional attributes, the compiler can capture the rough design and do the detail work necessary to determine size and speed estimates.

For example, within a data path, you could specify interconnections between ALUs, latches, register files, and multiplexers via major buses (as you would when drawing a conceptual block diagram). You can sketch the design alternatives and have the compiler expand the sketch into full detail to facilitate experimental designs for architectural evaluation. By the end of this phase, you will produce block diagrams and detailed specifications for the design.

Cost estimates

The major variables in a chip's cost are its size, package, and foundry. You can estimate size by experimenting with designs on the compiler and incorporating major logic blocks, signal buses, and an allowance for details not included in the rough design. High pin counts can create pad-limited chips in which I/O pads on the periphery determine the die size. As the die size increases, chip cost dramatically increases. Your foundry should help by estimating the cost based on die size.

Pin-count estimates should include power and ground pins. To reduce noise from simultaneously switched outputs, you need at least one power and ground pin for every eight to ten simultaneous outputs. The required ratio varies with the speed of the output drivers and the fabrication process. As pin counts increase, package costs also increase. Ceramic pin-grid-array (PGA) packages of 100 to 180 pins are available, but they can add $10 to $20 to parts cost.

Design capture

To implement your design, the compiler must capture several layers of information. Working from the inside of the chip to the outside, you should follow these steps in the design capture:

- Define the functional blocks and specify block interconnections.
- Specify chip I/O pads for function and location on the chip periphery.
- Connect the pads to the internal circuitry.
- Specify the package type and the bonding between chip I/O pads and package pins.

At the block-specification phase, you begin by selecting the desired type of functional block from a block-creation menu. The detailed specification varies with the type of block. For example, a random-logic block would comprise simple gates. Seattle Silicon and Silicon Compiler Systems support a schematic-capture mode using third-party CAE workstations. In this mode, you could specify the random-logic block by interactively drawing a schematic of the gates and identifying the external ports. Alternatively, Silicon Compiler Systems also supports a menu-driven, text-oriented means of specifying random-logic blocks.

A RAM or FIFO block, on the other hand, requires different information. The meaningful parameters for these blocks are size, organization, and timing. You may specify these parameters in a menu-driven fashion by filling in the appropriate blanks for the number of inputs, the number of words, the speed options, whether the block is to be single or multiple ported, and whether or not the output is latched.

Such blocks as read-only memory (ROM) or programmable-logic arrays (PLA) need this type of menu-driven structural information; in addition, they require programming information to specify their contents. For example, Genesil accepts truth tables, fuse maps, logic equations, or state-machine descriptions as formats for conveying PLA contents. Logic minimizers, such as Espresso from the University of California at Berkeley, *Silicon compilation excels at implementing highly structured, regular functions, such as data paths and memories.*

are available for PLA minterm reduction on several compilers.

You also have options in the method of specifying ROM contents. Genesil includes a macro assembler that can create symbolic macros and generate ROM contents by macro assembly. Alternatively, you can use a file containing the object code as ASCII ones and zeros. (At Tandem, designers develop microcode on a system other than the compiler and then port the finished object code to the compiler, formatting it appropriately along the way.)

Consider also complex blocks such as data paths. Genesil uses a menu-driven approach to specify data-path elements, bus connections, and orientation. Under Genesil, the entire data path is the same bit width (that is, number of bits in a word slice), although some elements allow masking of selected bits. (Although this scheme might sound restrictive, it's actually quite versatile.)

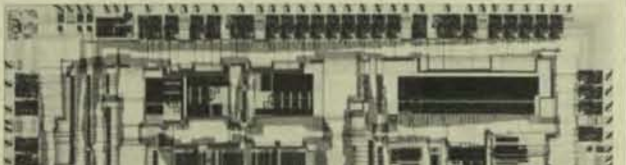
After you create the functional blocks, you must interconnect them. Genesil uses a textual net list to connect the external ports on each functional block to a named net. Seattle Silicon uses a third-party CAE-workstation schematic package to interconnect blocks.

Careful planning and good naming conventions are important to the interconnection phase. If several engineers work on different portions of a complex chip, prior agreement on net names and spellings will allow smooth and accurate net listing. Make the port names the same as the net names whenever possible.

Background

Silicon compilers began generating interest in 1983 and 1984 (**Refs 1, 2, and 3**), and they have since evolved into practical tools, yielding such products as graphics controller chips (**Ref 4**).

The information in this article is based on the experience of designers at Tandem Computers, who used silicon compilation to realize four CMOS chips employed in the company's NonStop CLX, a user-serviceable distributed on-line transaction-processing (OLTP) system. One of the chips—the CPU—measures approximately 500×500 mils and contains 60,000 transistors.



__Tandem Computers Inc produced this CPU chip__ with a silicon compiler for its NonStop CLX fault-tolerant computer. Part of a four-chip set, the CPU measures 500×500 mils and contains over 60,000 transistors.

Chip layout

The first step in chip layout is compiling the functional blocks, transforming them from a text description into a custom layout. After compilation, you can manipulate them as blocks having signal ports on their edges, without dealing with their internal layout. The rest of the layout phase consists of placing the blocks and routing their interconnections.

Compilers rely on an automatic signal router for interconnection. Because a chip's size is flexible at this stage, the router can push blocks away from each other to make room for the interconnections, guaranteeing that all the interconnections can be made. However, the result can be a very large chip. Designers exert the most influence on the layout through their placement of the blocks. Other factors affecting layout include the relative priority of signals and, on some compilers, assignment of routing channels.

You should have a layout plan in mind for the overall interconnection of the blocks (such as where major buses tie to which blocks). Although compilers have placement aids or even auto-placement algorithms available, your insight is important in producing a tight layout of a complex chip. Experimenting with alternate layouts will be fruitful, both for trying significantly different placements and for fine tuning the most promising placement. Achieving a good layout is very much an iterative process.

The compiler should assure adequate power and clock distribution. For example, Genesil calculates the power of each block and uses a power/ground bus width based on the total power of all blocks downstream from a given point. This procedure results in power buses that begin with a width of the power bonding pad and taper progressively. Clock lines may benefit from high-priority routing and several-times wider-than-minimal metal lines. In some cases, the clock lines can be isolated from other signals by surrounding them with ground lines.

Because interconnection delays stemming from wiring capacitance can be significant in CMOS, you must monitor the chip's critical paths while optimizing the chip size.

Simulation—verifying your design

Verifying your design through simulation is essential to maintaining the project schedule and budget. The expense, effort, and time required for chip iterations make a first-pass success highly desirable. Good simulation tools and a careful strategy make first-pass success attainable. The compiler generates a simulation model for your design automatically, ensuring that what you simulate corresponds to what you implemented.

Table 1 describes several levels at which to simulate. As you enter your design, you may interactively simulate small portions. For example, you may perform a check on a piece of control logic by manually applying a few patterns and observing the response. This level does not provide comprehensive verification, but it's a good indication that you're on the right track.

At the next level, you can apply hand-coded simulation vectors to the whole chip or to large parts of it. These vectors can become production-test vectors. This level is more rigorous and is reproducible as a regression test. However, it still relies on your ingenuity to specify the stimulus and response. Because simulation runs can require hundreds or thousands of vectors, the simulator must check for the expected response to a given stimulus and automatically flag any discrepancies. This rigorous simulation is far superior to the alternative of visually checking a printout of the simulator's output for erroneous responses.

Finally, you can embed the chip into a model of its intended application and perform system-level simulations. This level can expose subtle bugs and boundary conditions that you would not normally find in your lab.

TABLE 1—LEVELS OF SIMULATION

TYPE	CHARACTERISTICS/USE
INTERACTIVE	• GOOD FOR INITIAL DESIGN CHECKOUT OR TROUBLE SHOOTING. • POOR FOR COMPREHENSIVE VERIFICATION. • POOR REPRODUCIBILITY.
HAND-CODED VECTOR	• ALLOWS SIMULATOR TO CHECK EXPECTED RESULTS. CAN BE RERUN AFTER CHANGES AS A REGRESSION TEST. • CAN BE USED AS PART OF PRODUCTION TEST. • RELIES ON DESIGNER INGENUITY TO COVER ALL TEST CASES. CAN OVERLOOK SYSTEM-LEVEL PROBLEMS.
SYSTEM-LEVEL	• EXERCISES CHIP IN SYSTEM ENVIRONMENT. • GIVES HIGHEST DEGREE ON CONFIDENCE. • CONVERGES MICROCODE/SOFTWARE AND HARDWARE VERIFICATION. • CAN EXTRACT RESPONSES FOR PRODUCTION TEST.

Timing analysis

Accurate prediction of the performance of your chips is an important part of the design. You need to relate your system's goals and constraints and identify the areas of your chip that need fine tuning.

Two main components of CMOS circuit delays are the gate propagation delay and the additional delay arising from the capacitance and resistance of the interconnection. To obtain accurate timing numbers, you need to incorporate the interconnection loading derived from the actual layout of the chip. In a large chip, the interconnection delay of a signal crossing the chip can exceed a gate's nominal propagation delay. An excessive interconnection delay indicates a very long routing or a net with a large fan-out. You can shorten the delay by increasing the strength of the driver, altering the placement of the chip to shorten the interconnection length, or revising the logic to reduce the fan-out.

Two classes of timing-prediction tools exist. The more traditional tool is based on circuit simulation. Using a model of your circuit, you can stimulate it to trace specific circuit paths to obtain their delay. In general, the more accurate the calculations, the slower the simulator. Spice, a commercially available circuit simulator, is an example of a very accurate and computation-intensive tool. One of the main drawbacks of this type of tool is that it gives you information about only the paths you specify and already suspect—not unexpectedly slow paths.

The second class of timing tool, static analysis, can examine all paths and tell you which are the longest.

Two types of compiler vendors exist: those offering generic tools and those offering tools specific to their own foundries.

Static timing analyzers first characterize the delay from input transitions at each gate to the output transition at the gate's loads. They then trace the paths through the circuit, adding up the delays as they go. The most powerful form of analysis traces paths from their origin in a clocked element to their termination in a clocked element. In a synchronous system, this analysis identifies the critical paths limiting clock speed.

The static-analysis tool's obvious benefit is that it tells you about timing paths you may not have realized were slow. Other capabilities of a sophisticated static tool include incorporating external setup and hold requirements, checking margins on internal hold times, and the ability to specify timing paths to be ignored. Genesil also has the ability to automatically extract a Spice model of a specified path to allow more exact timing simulation where desired. The accuracy of a static timing analyzer depends on the sophistication of its models of transistor characteristics and resistive and capacitive loading. Static timing analyzers better suit synchronous circuits than asynchronous ones.

You can use both timing tools to observe the effects of temperature, variations in voltage, and variations in the fabrication process.

The next step is to assure that the physical mask that the compiler produced accurately represents your design. To accomplish this task, compiler vendors provide various types of post-design checks. The most basic is the traditional design-rule check (DRC). The DRC program processes the mask's data against the physical design rules for the foundry you have selected, checking that trace width and spacing meet a minimum specified size and that the transistors are properly formed, for instance. This checking assures that the chip can be fabricated.

Checking the mask

You can make an additional check by programmatically extracting the circuit's topology (that is, by deriving the circuit elements and the net list that they implement from the mask). This extraction can further verify the physical design, either by direct comparison to the logic design you previously simulated or by simulation against the same stimulus/response vectors you previously used.

You may wonder why compilers need this type of post-verification; after all, they should automatically generate correct silicon. However, one of the benefits of a silicon compiler is that it facilitates large, complex chip designs, such as microprocessors. Designers demand rich sets of design elements, and vendors enhance their products for higher density and performance. The result is that as designers push beyond what has been done before, a possibility of generating bugs exists. Even a single error in a chip can cause a malfunction; therefore, post-verification is essential to ensuring correct silicon.

TABLE 2—PROTOTYPE FABRICATION STEPS

TASK	TIME	COST
FINAL VERIFICATION	2 TO 4 WEEKS	SEE NOTE
DATABASE CONVERSION	1 WEEK	\$2k TO 10k
PLOT APPROVAL	2 DAYS	—
MASK GENERATION	1 TO 2 WEEKS	\$30k TO 40k
WAFER FABRICATION	6 TO 10 WEEKS	\$1k TO 20k
ASSEMBLY	1 TO 2 WEEKS	—
PACKAGE PART TEST	1 TO 2 WEEKS	SEE NOTE

NOTE: PART OF BROKERAGE PACKAGE FROM COMPILER VENDOR.

After verification, build prototypes

When your chip has completed final verification, you are ready to build prototypes. Several compiler vendors offer a brokerage service, for which they coordinate initial fabrication with the foundry and deliver the working parts to you. Seattle Silicon and Silicon Compiler Systems offer this service with guarantees of working parts.

To ensure a smooth path for your prototypes and to prepare for production, you need to begin a business relationship with your foundry well in advance of first silicon. This preparation can insure that the right resources and materials are in place when you need them. For example, if you specify a package type that the foundry does not stock, the lead time to procure it can be longer than the fabrication time.

Advance discussions can also clarify the time and charges for fabrication. The overall time and charges for getting tested parts are substantially more than for simply running a wafer lot. You need to make certain that you understand the steps in the process (**Table 2**). For example, starting from a database tape, the foundry converts database formats for the mask vendor and has the masks fabricated, inspected, and repaired before processing. After processing, the parts are assembled. You also need to allow time for testing, either at the foundry or at the compiler vendor. **Table 2** de-

You may need to make some changes to your design approach, particularly if your experience is with board-level TTL.

TABLE 3—CHARACTERIZATION AND PRODUCTION TESTS

TYPE OF TEST	AREAS TESTED
FUNCTIONAL	• OVERALL LOGICAL INTEGRITY OF CHIP.
DC PARAMETRIC	• VOLTAGE THRESHOLD OF INPUT PINS. • VOLTAGE AND CURRENT OF OUTPUT PINS. • LEAKAGE CURRENT OF INPUT AND 3-STATE PINS.
AC—SETUP/HOLD	• CHIP INPUT SETUP/HOLD FROM EXTERNAL CLOCK
—OUTPUT DELAY	• DELAY FROM CLOCK OR INPUT PINS TO OUTPUT PINS.
—CYCLE TYPE	• MAXIMUM OPERATING FREQUENCY FOR CLOCK.

scribes the steps in prototype fabrication, including the time and cost required.

If you use a brokerage arrangement, the foundry delivers untested parts to your compiler vendor. The compiler vendor then tests the packaged parts using test vectors from your simulation runs. This test assures functional parts and provides some speed testing. The compiler vendor's involvement usually ends here.

You must still do substantial test development to put your parts into production. Production tests have three broad components: functional tests, dc tests, and ac tests (**Table 3**).

Functional tests find fabrication faults quickly. The vectors used for simulation during design verification may not be appropriate for production. Verification typically assumes that the logic elements are perfect and checks to determine if your design functions as intended. These tests can have poor fault coverage or can be excessive in length. One current weakness of silicon compilers is poor support for fault-grading tools with which you improve production tests.

Compilers support transporting simulation vectors to selected IC testers. (With Genesil, Tandem designers needed to develop a format converter for Sentry Schlumberger Inc and Ando Corp for production. Once this tool was in place, they captured vectors from system-level simulations and incorporated them into production tests. This tool proved to be very effective: The chips could be tested with the same stimuli that they would ultimately encounter in the system.)

Dc and ac tests check the chip against its data-sheet specifications. The dc tests include input-voltage thresholds, output voltage levels at specified currents, and input and 3-state leakage currents. Ac tests include output delays, cycle times, and setup and hold times. Ideally, you should characterize these parameters over a range of supply voltages and temperatures corresponding to the specified operating range. Use a sampling of parts over different fabrication runs to include process variations. This testing lets you balance your system requirements against the actual performance of the chips and establish test limits to maximize your yield. It can also indicate problem areas requiring further attention. Your foundry can help in performing this characterization. **EDN**

References

1. Johnson, Stephen C, "VLSI Circuit Design Reaches the Level of Architectural Description," *Electronics*, May 3, 1984, pgs 121-128.
2. Johnson, Stephen C, "Silicon Compiler Lets System Makers Design Their Own VLSI Chips," *Electronic Design*, October 4, 1984, pgs 168-181.
3. Wallich, Paul, "On the Horizon: Fast Chips Quickly," *IEEE Spectrum*, March 1984, pgs 28-34.
4. Jones, Michael, and Bailey, Michael, "High Performance Graphics Via Silicon Compilation," *VLSI Systems Design*, March 1987, pgs 32-38.
5. "LSI Logic's Big Bag of ASIC Design Tools," *Electronics*, February 5, 1987, pgs 59-61.
6. "A Compiler for Semicustom Solutions," *Electronics*, February 5, 1987, pgs 62-64.

Author's biography

William Stenzel is a development engineering manager for Tandem Computers Inc (Cupertino, CA) and has been with the company for seven years. He is currently responsible for processor and memory development for distributed systems. Prior to Tandem, he spent five years with Hewlett-Packard, where his responsibilities included management of a custom CMOS processor project. He received his BS in Computer Engineering and MS in Computer Science from the University of Illinois.

TANDEM

TANDEM

NonStop™ Systems

Transaction Processing on the Tandem NonStop Computer: Requestor/Server Structures

Joe Collins

Software Education
and Design Support

SEDS-001 Jan. 1982

The SEDS Monograph Series

Editor: Kent Madsen

This is one of a series of technical papers published by the Software Education and Design Support (SEDS) department of Tandem Computers. The series was conceived as a means of sharing technical insights developed within the department with Tandem users and systems analysts. This information—in most cases dealing with the design of application software for Tandem systems—would otherwise be obtainable only in formal software education courses or through direct consultation with our design support group.

Transaction Processing on the Tandem NonStop™ Computer: Requestor/Server Structures

Joe Collins

TANDEM

Software Education and Design Support

SEDS-001 January 1982

CONTENTS

Transaction Processing on the Tandem NonStop Computer: Requestor/Server Structures

INTRODUCTION

Online Transaction Processing

Online transaction processing is, as the name implies, computer processing of data relevant to individual business transactions as they occur.[1] It is perhaps best understood in contrast to batch processing, the carefully sequenced posting of large numbers of transactions the night after, the week after, or the month after they occur. Online transaction-processing systems are attractive because (unlike batch systems) they are never out of date. They can provide accurate information on the state of a business at any instant. This allows managers to respond immediately (and intelligently) to unforeseen problems, changing conditions, and unexpected opportunities.

Online transaction processing has obvious applications in banking, inventory control, ticket and flight reservation processing, and many other areas. The major operational requirements are illustrated in Fig. 1. A large number of terminals must access and update a common data base in real time. Changes in the data base must be immediately available to all users. The system must be capable of handling large numbers of transactions of various types, arriving almost concurrently in an unpredictable sequence. The hardware used must be extremely reliable because business cannot go on when the system is down.

Transaction-processing software must be easily modifiable, capable of rapid response, and readily expandable:

- Ease of modification is important because, during the lifetime of the software, numerous changes will have to be made to keep pace with user needs and to fix bugs. The cost of making these changes must be considered as part of the cost of developing the software.
- Response time is critical because customers are frustrated if the system is slow to respond, and terminal operators are idle and unproductive while waiting for the system.
- Expandability is vital because software that handles 200 terminals today may within a few years have to deal with 1000, and systems used initially for inventory control may eventually have to perform billing and other functions as well.

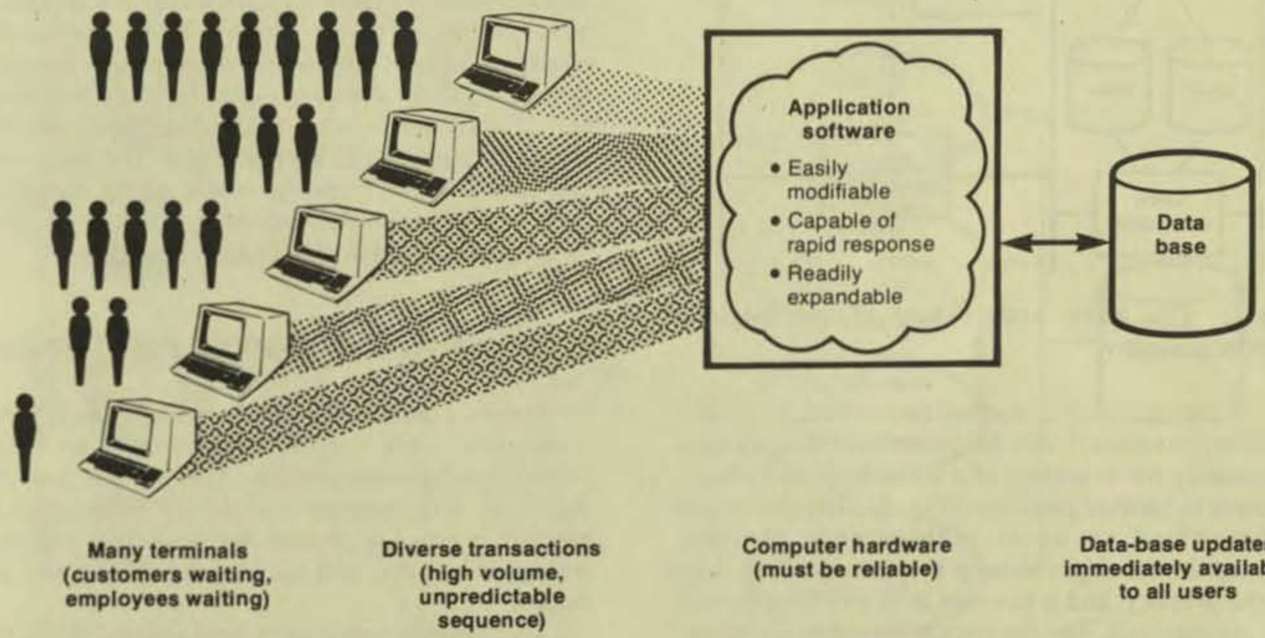

Application software
• Easily modifiable
• Capable of rapid response
• Readily expandable
Data base
Many terminals (customers waiting, employees waiting)
Diverse transactions (high volume, unpredictable sequence)
Computer hardware (must be reliable)
Data-base updates immediately available to all users

Fig. 1 The environment of online transaction processing.

The NonStop System

Tandem designed the NonStop system[2] with on-line transaction processing in mind. It is many times more reliable than any other commercially available system. No single component failure can shut it down or contaminate its data if it is properly configured.

As shown in Fig. 2, the system contains multiple CPUs, each with its own private memory and multiplexed input/output channel. The processor modules communicate with one another over a pair of high-speed interprocessor buses (DYNABUS). Peripheral device controllers are connected to the input/output channel of two processor modules so that the device is accessible even if a CPU fails. The system can be expanded simply by adding more processor modules (up to a maximum of 16 per system) as the workload increases. Then, with the aid of Tandem's EXPAND software, up to 255 systems can be joined into a network. Such expansion is possible without reprogramming.

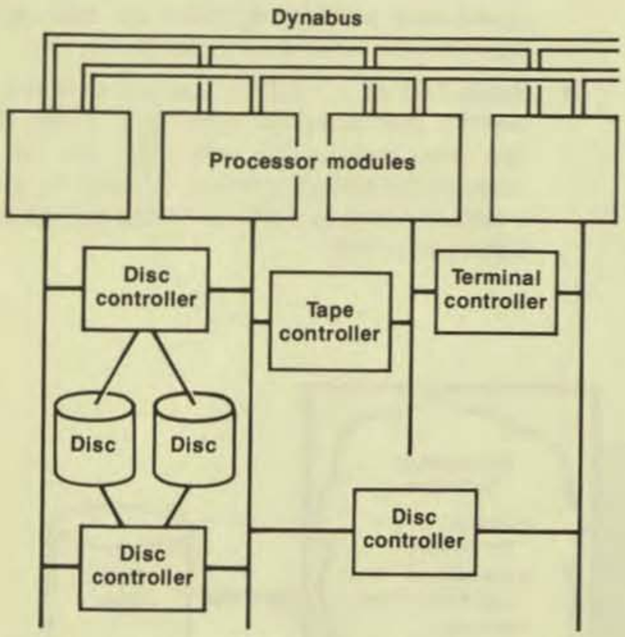


Fig. 2 The basic architecture of the Tandem NonStop system.

A process (i.e., a program running on the Tandem machine)[3] can be protected from system failures by the execution of a secondary, or backup, process in another processor (Fig. 3). This backup is programmed so as to require only periodic checkpoint messages to keep it apprised of the state of the primary, and it can step in at any time should the primary fail. The logistics involved in maintaining a collection of concurrently executing process pairs (primaries and backups), distributed over as many as 16 CPUs, would be horrendous were it not for a simple and extremely efficient message system built into GUARDIAN, the Tandem operating system. As shown in Fig. 3, the message system allows any process in the system to communicate with any other without detailed knowledge of its physical location. Furthermore, it positively confirms receipt of each message and keeps the sender's address on hand so that the receiver can reply as if the sender were still on the line. The message system makes every process running on a 16-processor system as easy to access as a file on a more conventional machine.

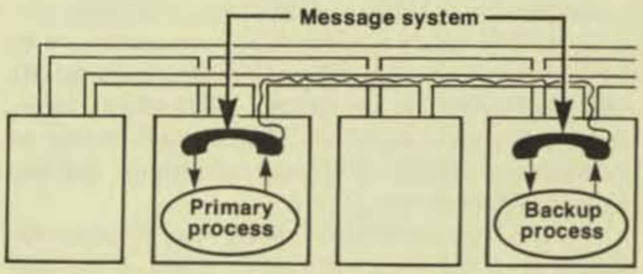


Fig. 3 The message system (part of the Tandem operating system) makes every process running on a 16-processor system as easy to access as a file on a more conventional machine.

The unique architecture of the Tandem NonStop computer system brings into play some equally unique approaches to software design. Because of the message system and the availability of multiple processors, there are performance and other advantages to be gained by breaking transaction-processing software down into "requestor processes" (independent software entities that control terminals and perform user-interface functions) and "server processes" (independent entities that access and manipulate the data base). The purpose of this paper is to discuss the nature of the requestor/server structure and the advantages that result from its implementation on the Tandem machine.

REQUESTOR AND SERVER PROCESSES

Figure 4 shows the division of labor between "requestor" and "server" processes in an online project management system. This system provides managers with accurate and timely information on projects in progress, events that occur in connection with each project, and participants in projects and events.

The requestor process is responsible for the following functions: terminal interface, field validation, data mapping, and transaction control.[4] The terminal interface deals with the various protocols that different terminals in the system employ. Field validation involves checking the input for numeric-only or alphabetic-only fields, required fields, etc. Data mapping is the conversion of data from its terminal display form to its internal form (e.g., June 27, 1981 to 810627) and back again. Transaction control includes all the logic necessary to display screens in the proper order, perform inter-field data consistency checks, and decide what actions to take in response to inputs from the terminal. It is desirable from the standpoint of efficient resource management that requestors be multi-threaded (i.e., capable of dealing with more than one request at a time), but this presents a formidable programming challenge.

Server processes (always single-threaded) are responsible for data base access, calculations, and other functions. In the sample application described in Fig. 4, there are four servers (Projects, Events, Participants, and Employee/Department), each one capable of processing several different transactions. For example, the Project server can add, update, delete, and list projects. (This does not mean that it only accesses project records, but rather that it performs all data base access required to process project-related transactions.)

Two examples will illustrate how requestors and servers work together in handling a transaction.

Example 1. A user at a terminal presses a function key to indicate that he wants to add an event. However, in entering information about the event, he forgets to supply the project name, which is considered by the software to be a required field. The requestor receives the input, recognizes that a required field is missing, and sends an appropriate error message to the terminal. In this case, because of the error, the server does not get involved.

Example 2. The user at the terminal receives the error message and repeats the request to add an event, this time supplying the required information. The requestor again checks to see that all the information is there and that it is in the correct format (numeric items in numeric fields, etc.). Then, it converts the data to the internal representation format and composes a message consisting of data relevant to the event and a code indicating that a new event is to be added to a particular project. It sends this message (the ''request'') to the Events server.

The Events server receives the message, identifies the ''add event'' code, and, in response to that code, takes action. First, it reads the relevant project record and increments by one the event-count number contained in that record. Next, it sets the event number for the new event equal to the event-count number, writes that event to the data base, and rewrites the project record. Finally, it composes a message containing an ''operation-successful'' code and sends that message back to the requestor.

The requestor recognizes the ''operation-successful'' code and sends an appropriate message to the terminal.

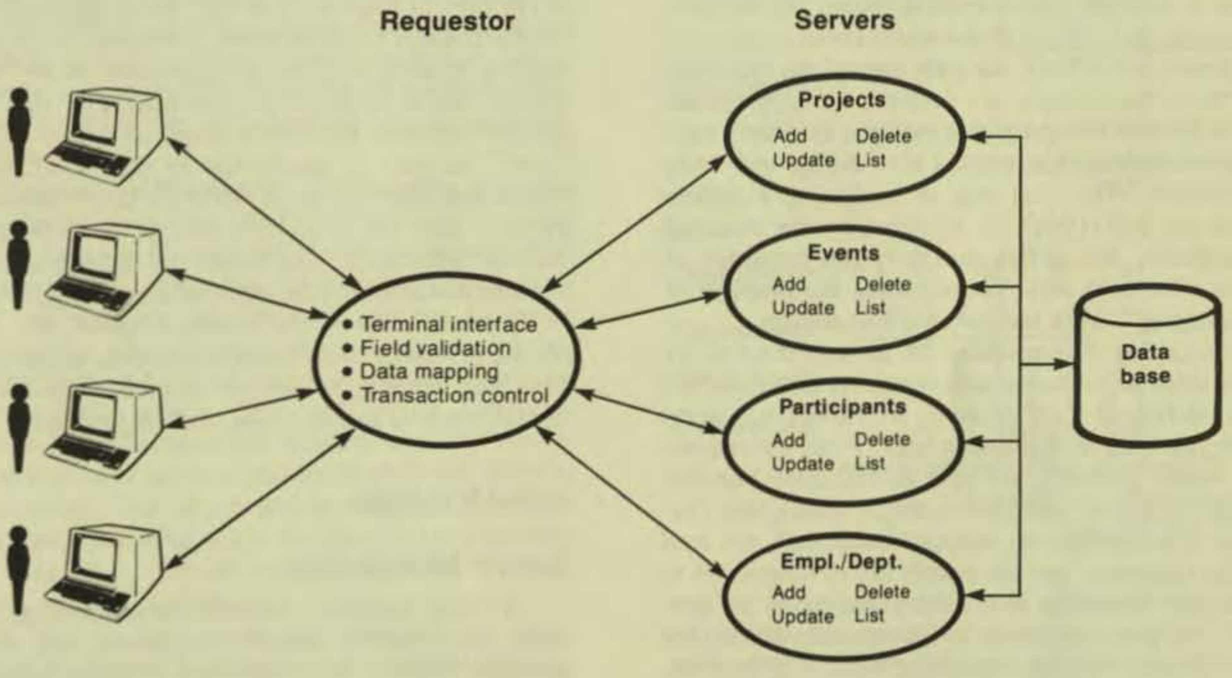


Fig. 4 Requestor and server processes in application software for an online project management system.

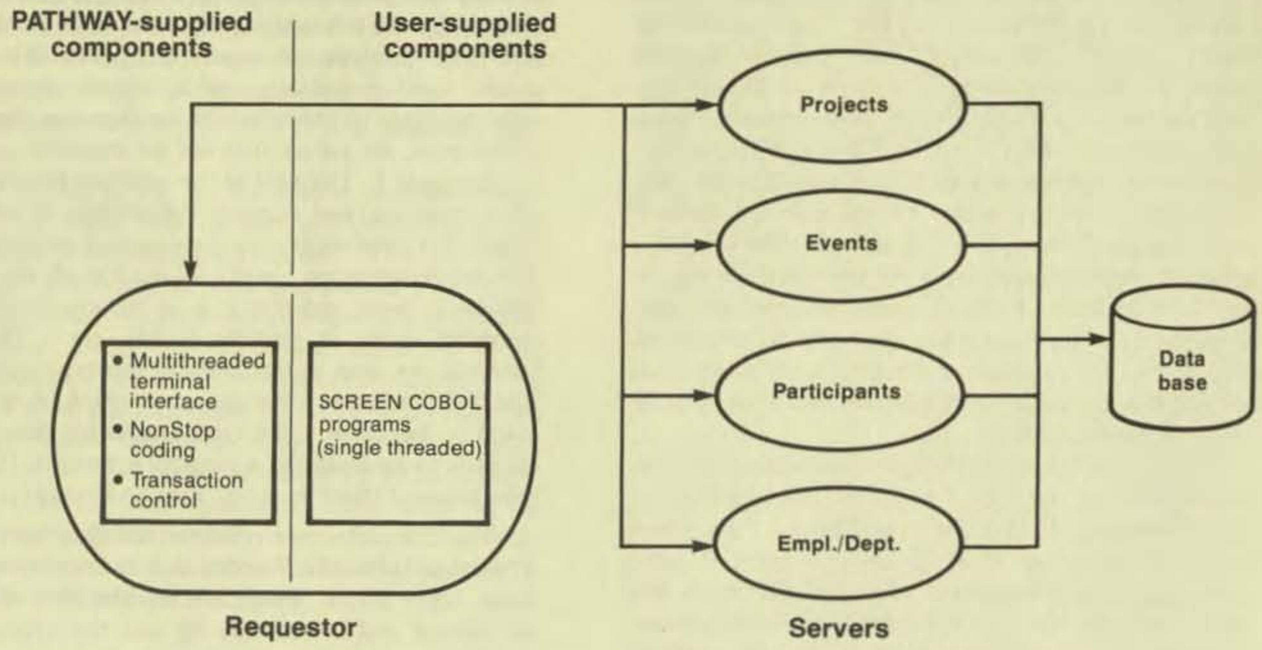


Fig. 5 PATHWAY, a Tandem software product, simplifies and standardizes the development of requestor/server-based application software.

Tandem offers a software product, PATHWAY,[5] which simplifies and standardizes the development and maintenance of requestor/server-based application software (Fig. 5). PATHWAY eliminates the need to consider multithreading of the requestors and the design of primary and backup requestor pairs. It also provides an environment for the configuration and control of the application.

Under PATHWAY, the only part of the requestor that users must supply is a collection of appropriate SCREEN COBOL programs (written as if they were single-threaded). Users must also design and write the servers. The first step in designing a system based on PATHWAY is to identify the required transactions and define precisely the sequence of actions for each one. Then, comes the problem of "packaging" these transactions into servers.

Packaging is something of an art, learned by experience with diverse applications, and there are many different ways of doing it. However, several guidelines can be mentioned here. First, the requestor should generally not have to call upon two different servers to complete a single transaction (because this doubles the message overhead, the load on the requestor, and the possibility of delays due to queuing). Secondly, if several transactions are handled by a given server (as in Fig. 4), they should not have widely varying execution times. If they do, users will find it difficult to tune and control the system because response times for the relatively simple transactions will be erratic, depending on whether a request for the complicated transaction is being executed at the same time.

Server processes should be context-free. They should not be required to remember past requests and should not concern themselves with the origins of the present request. The fact that a request came from a Tandem 6520 terminal connected to a remote Tandem system in Chicago connected to the local system via a packet network (and that the user pressed function key F5 in response to the "Enter Event" screen) is immaterial at this point in the transaction. Servers need not even know which requestor sent the request because, in effect, the Tandem message system holds the appropriate line of communication open until the server responds.

Because servers are single-threaded and concerned primarily with satisfying simple requests for data base records or sets of records, they are extremely easy to design, code, and understand.

ADVANTAGES

Ease of Modification

Writing complex transaction-processing software in terms of simpler requestor and server modules reduces the likelihood of errors in design and coding, and it also makes the software easier to modify. Each module is highly independent of every other. The interface between a requestor and a server is restricted to a well-defined and limited set of message formats and function codes. Thus, a change in any component is unlikely to have unexpected or subtle effects on other components.

The modular structure of properly designed requestor/server software greatly facilitates maintenance and the implementation of changes because such software can be readily understood. An analyst or designer who understands the requestor/server concept has only to ask a few basic questions to get an overview of the system: What functions does the application perform? What do the various requestors do? What do the various servers do? What is the format of the interprocess messages exchanged between the requestors and servers?

At the component level, he asks: What is the format of each screen (menu screen, add-event screen, etc.) presented to the terminal? What is the logical hierarchy of the screens?

With regard to a particular server, he asks: What requests does this server respond to? What data base files does it access? What transactions does it take part in?

Thus, the individuals assigned to maintain the system or to implement changes can begin their work with a comprehensive understanding of the system and its components. This greatly increases the probability that any changes made will be correct.

Rapid Response

Some of the most striking advantages of the requestor/server structure stem from its ability to take full advantage of the unique capabilities of the NonStop system. Among these capabilities are a capacity for parallel processing and system-tuning features based on the availability of multiple processors.

Transaction Pipelining. Traditionally, the term "pipelining" has been used to refer to the simultaneous execution by computer hardware of more than one instruction at a time. Each instruction is typically divided into two phases, and the hardware executes the first phase of one instruction simultaneously with the second phase of another. This improves performance. Because of its multiple-processor architecture, the Tandem NonStop system is capable of a tremendous amount of this parallel processing, and the requestor/server structure makes possible what might be referred to as transaction pipelining. The effects of transaction pipelining are illustrated in Fig. 6.

In Fig. 6a, we assume that three transactions, labeled A, B, and C, are received concurrently by a "monolithic" program labeled R/S, which performs both requestor and server functions. R/S takes 2 arbitrary time units (2t) to service each transaction. Under these circumstances, A will be served first while B and C wait, B will be served next while C waits, and C will be served last. As shown in Fig. 6a, transaction A will be finished in 2t, but transaction B will take 4t (because of the wait time), and transaction C a full 6t.

The average response time for this series of 3 transactions is (2t + 4t + 6t)/3, or in other words 4t. The throughput of the system under these conditions is 3 transactions/6t, or in other words 0.5 transactions per time unit.

In Fig. 6b, the same three transactions arrive at the system concurrently. However, in this case, the application software consists of a requestor (R) and a server (S) rather than a monolithic program, and the hardware is a Tandem multiple-processor system capable of parallel processing of requestor and server functions. We assume that it takes one time unit (t) for the requestor to do its work and another time unit (t) for the server to do its work (for a total processing time of 2t as in the previous example).

In the first time unit, A is serviced by R, while B and C wait. In the second time unit, A is serviced by S, and B is serviced by R, while C continues its wait. In the third time unit, B is serviced by S, and C is serviced by R. Finally, in the fourth time unit, C is serviced by S.

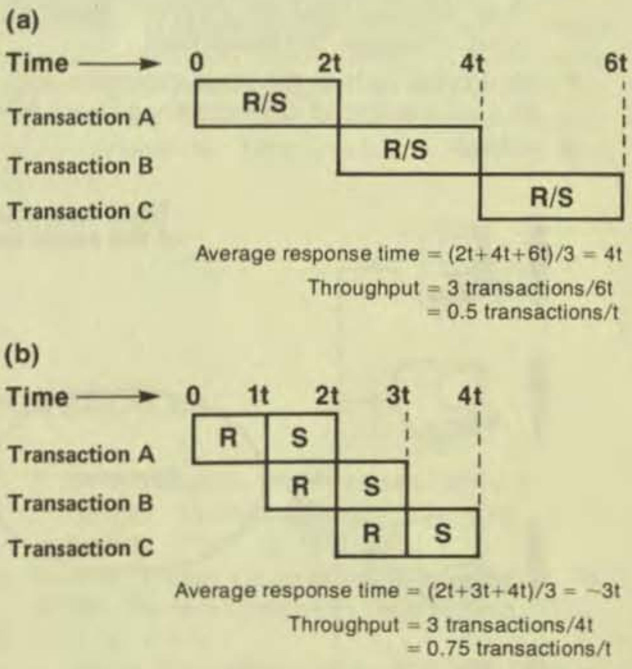


Fig. 6 Comparison of the performance of (a) monolithic transaction-processing software running on a conventional computer and (b) requestor/server software running on a Tandem machine.

As shown in Fig. 6b, under these circumstances, the response times are lower and the system throughput is higher because of processing overlaps. The response time for A is 2t, the response time for B is 3t, and the response time for C is 4t. Thus, the average response time is (2t + 3t + 4t)/3, or in other words 3t. The throughput of the system is 3 transactions/4t, or in other words 0.75 transactions per time unit.

This is, of course, an idealized case. It is unrealistic to assume that every transaction will take equal time in the requestor and the server, and, what's more, the time taken in sending messages has been ignored. (It is usually negligible.) However, the basic idea is sound — namely that performance gains can be achieved by structuring application software to take advantage of the Tandem system's inherent capability for parallel processing.

Tuning a Requestor/Server Application. Traditionally the tuning of application software to improve performance has been done by modifying the program itself. However, there are several dangers inherent in this practice:

- Modifying code may introduce bugs.
- The time required to modify code is considerable (often measured in days, weeks, or, in extreme cases, months).
- Tuning of this kind usually involves coding tricks that also make the programs harder to debug, maintain, and understand.
- Modifying code is the most expensive way to tune because of the manpower costs involved.

A more reasonable approach to tuning is available on the NonStop system if the software is based on the requestor/server model. The performance problem can be traced either to hardware or software overloading. If the performance problem is due to an excessive queue on a hardware component, more hardware can be added, as on any computer system. However, the NonStop system is unique in that it allows CPUs to be added as easily as any other type of hardware (and without reprogramming).

If the performance problem is due to software overloading, it can be traced to terminal software overloading or data base software overloading (via a Tandem performance-measuring tool known as XRAY,[6] which can measure the average queue length associated with a given process). Once the problem is identified, the application manager needs only to introduce additional requestors or servers into the system to correct it (Fig. 7). This involves no recoding because the requestor or server processes added are simply new executions of existing programs.

Load-balancing is another tuning method available on the NonStop system if the application software has been designed in accordance with the requestor/server model. Load-balancing is the redistribution of files on discs or of processes among the available CPUs to achieve the most efficient use of each.

Figure 8 illustrates one load-balancing operation. If CPU 2 in Figure 8a is chronically overloaded and XRAY measurements show that server S3 is the culprit, it would be possible in a matter of

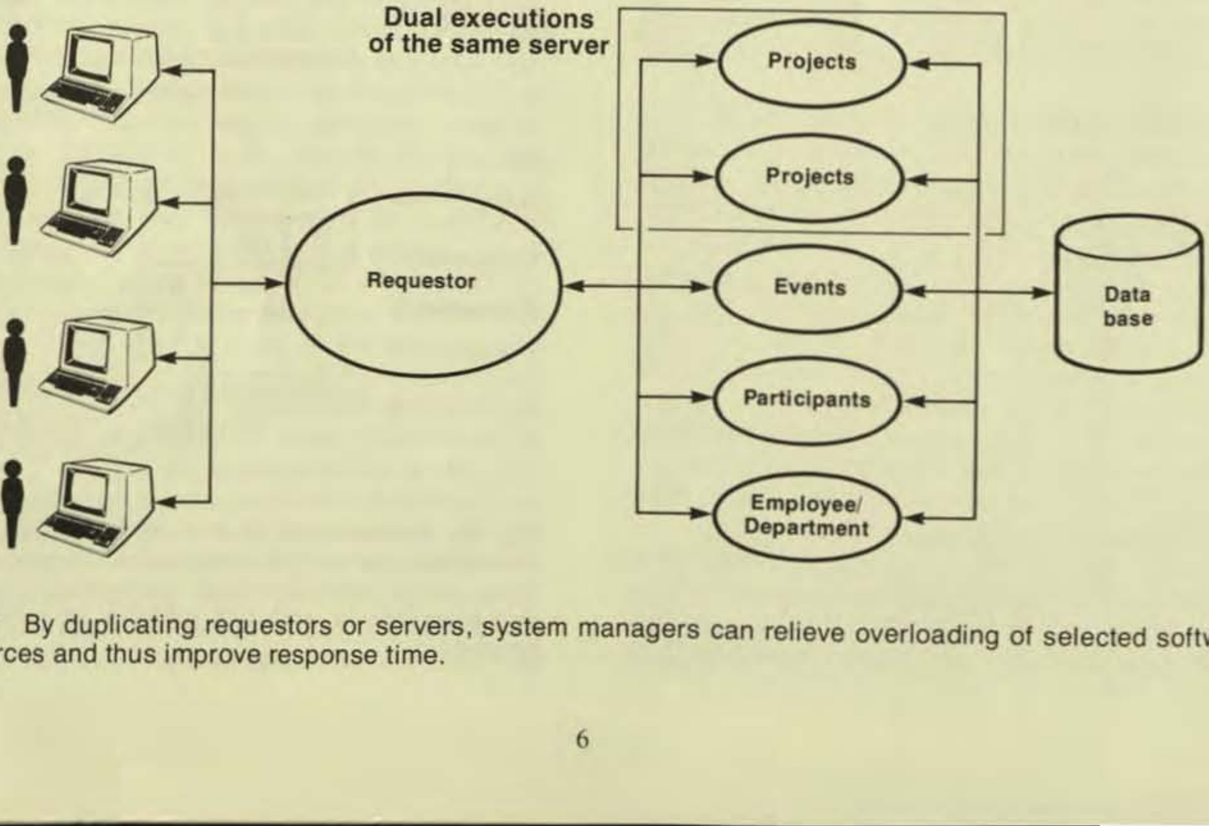


Fig. 7 By duplicating requestors or servers, system managers can relieve overloading of selected software resources and thus improve response time.

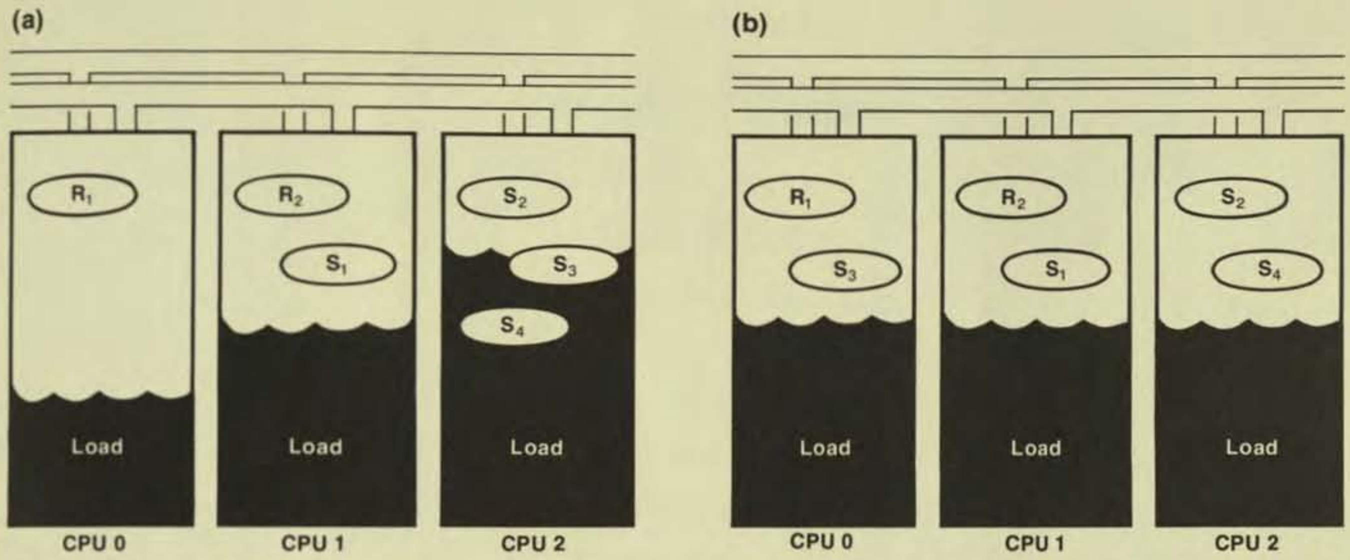


Fig. 8 (a) To relieve a bottleneck in CPU 2 due to overloading, the system manager could shift Server 3 to CPU 0, thus balancing the load (b).

minutes (Fig. 8b) to move server S3 from CPU 2 to CPU 0, which happens to be underutilized.

The NonStop system architecture allows any process to run in any processor module. The message system allows processes to communicate with one another regardless of which CPU they are running in and without the need for either process to know which CPU the other is running in. Therefore, there is no need to modify the code of a process when moving it from one CPU to another.

By adding software capacity in modular fashion and balancing the load on each CPU, the application manager can tune the application software without recoding, while preserving its logical structure, its modularity, and its simplicity.

Expandability

As explained above, the requestor/server structure makes application expansion easier. To add capacity to a NonStop system, one simply adds more processor modules, terminals, and disc drives as needed. Additional copies of the requestors and servers, identical to those in the original software, can then be run on the new hardware. The additional cost of software is minimal, consisting only of license fees for optional software products running in the new processors, and no additional programming is required.

Adding new functions is also simple. As the computing tasks increase in complexity, new servers are added to perform the new functions, and the SCREEN COBOL programs in the requestor are modified to make the appropriate requests.

CONCLUSION

The Tandem NonStop system architecture provides a favorable environment for online transaction processing, but poorly designed application software can negate the benefits offered by the machine. The requestor/server structure offers a proven way of achieving ease of modification, rapid response time, and expandability in transaction-processing software designed for the Tandem machine, and the PATHWAY software product significantly reduces the time needed to develop such software.

REFERENCES

1. F. G. Withington, *The Environment for Systems Programs* (Addison-Wesley, Reading, Massachusetts, 1978), pp. 181-202.
2. *Tandem System Description Manual*, Part No. 82000, Tandem Computers Incorporated, 1980.
3. Ibid., p. 4.2-1.
4. L. Smith, *Designing a Network-Based Transaction Processing System*, SEDS-002, Tandem Computers Incorporated, 1982.
5. *PATHWAY Operating Manual*, Part No. 82060, Tandem Computers Incorporated, 1980.
6. *XRAY User's Manual*, Part No. 82078, Tandem Computers Incorporated, 1981.

Jan' 82 SEDS-001

Joe Collins

AUTHOR

Transaction Processing on

TITLE

the Tandem NonStop: Requestor Structures

DATE DUE	BORROWER'S NAME
5/28	Mike Lucido 1
~~12/12/88~~	Carolyn Garbarino
JAN 12 '89	

GAYLORD 45

TANDEM

Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, California 95014-2599
(408) 725-6000

400070

TANDEM

A Fault-Tolerant Computing System

James A. Katzman
Tandem Computers
19333 Vallco Parkway
Cupertino, California 95014

Abstract

A fault-tolerant computer architecture is examined that is commercially available today and installed in many industries. The hardware is examined in this paper and the software is examined in a companion paper [*Ref. 4*].

Introduction

The increasing need for businesses to go on-line is stimulating a requirement for cost effective computer systems having continuous availability [*Ref. 1, 2*]. Certain applications such as automatic toll billing for telephone systems lose money each minute the system is down and the losses are irrecoverable. Systems commercially available today have met a necessary requirement of multiprocessing but not the sufficient conditions for fault-tolerant computing.

The greatest dollar volume spent on systems needing these fault-tolerant capabilities are in the commercial on-line, data base transaction, and terminal oriented applications. The design of the Tandem 16 NonStop* system was directed toward offering the commercial market an off-the-shelf, general purpose system with at least an order-of-magnitude better availability than existing off-the-shelf systems without charging a premium (*see Appendix A*). This was accomplished by using a top down system design approach, thus avoiding the shortcomings of the systems currently addressing the fault-tolerant market.

Except for some very expensive special systems developed by the military, universities, and some computer manufacturers in limited quantities, no commercially available systems have been designed for continuous availability. Some systems such as the ones designed by ROLM have been designed for high MTBF by "ruggedizing," but typically computers have been designed to be in a monolithic, single processor environment. As certain applications demanded continuous availability, manufacturers recognized that a multiprocessor system was necessary to meet the demands for availability. In order to preserve previous development effort and compatibility, manufacturers invented awkward devices such as I/O channel switches and interprocessor communication adapters to retrofit existing hardware. The basic flaw in this effort is that only multiprocessing was achieved. While that is necessary for continuously available systems, it is far from sufficient.

Single points of failure flourish in these past architectures (Figure 1). A power supply failure in the I/O bus switch or a single integrated circuit (IC) package failure in any I/O controller on the I/O channel emanating from the I/O bus switch will cause the entire system to fail. Other architectures have used a common memory for interprocessor communications, creating another single point of failure. Typically such systems have not even approached the problem of on-line maintenance, redundant cooling, or a power distribution system that allows for brownout conditions. In today's marketplace, many of the applications of fault-tolerant systems do not allow any down time for repair.

Expansion of a system such as the one in Figure 1 is prohibitively expensive. A three-processor system, strongly connected in a redundant fashion, would require twelve interprocessor links on the i/O channels; five processors would need forty links; for n processors, 2n(n-1) links are required. These links often consist of 100-200 IC packages and require entire circuit boards priced between $6,000 and $10,000 each. Using the I/O channel in this manner limits the I/O capabilities as a further undesirable side effect. The resulting hardware changes for expansion, if undertaken, are typically dwarfed in magnitude by the software changes needed when applications are to be geographically changed or expanded.

This paper describes the Tandem 16 architecture at the lowest level (hardware). Section I deals with the overall system organization and packaging. Section II explains the processor module organization and its attachment to the interprocessor communications system. Section III discusses the I/O system organization. Section IV discusses power, packaging, and on-line maintenance aspects that are not covered elsewhere in the paper.

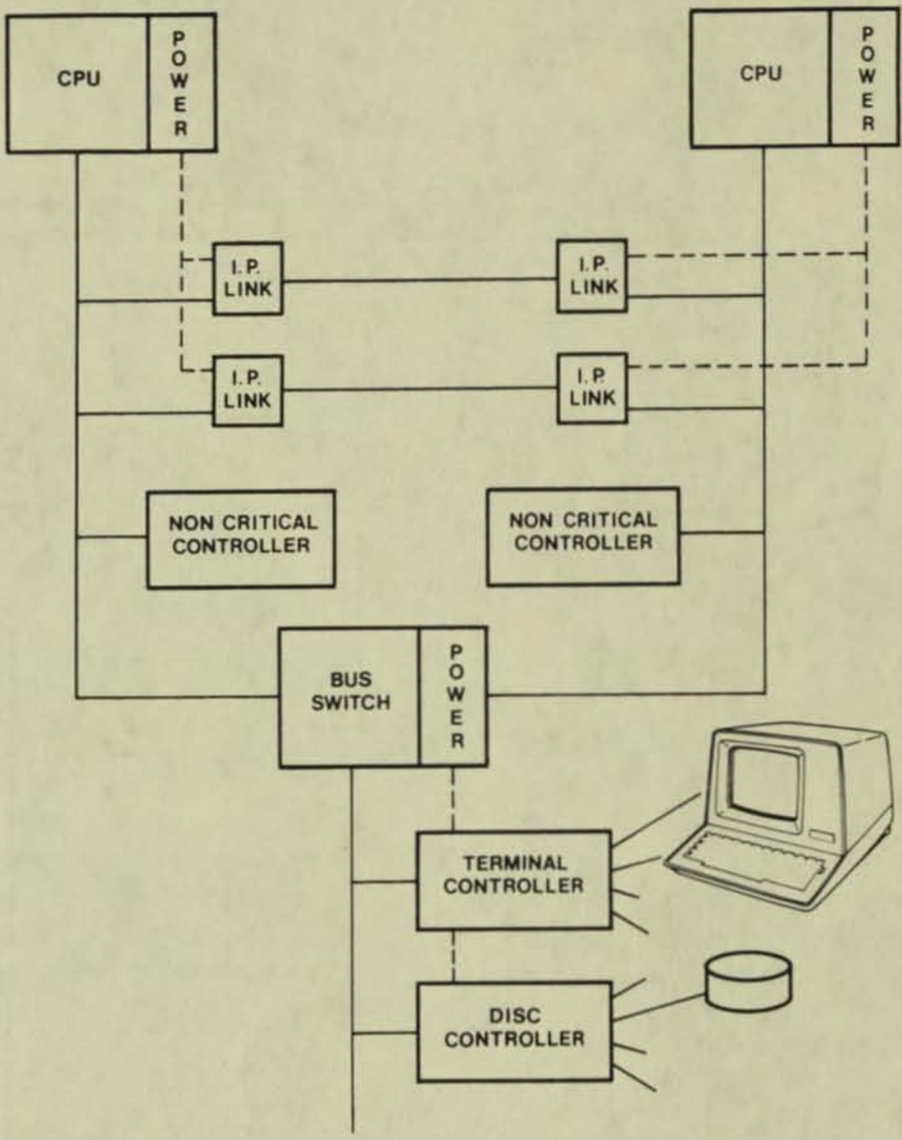


Figure 1. Example of Previous Fault-Tolerant Systems

I. SYSTEM ORGANIZATION

The Tandem 16 NonStop system is organized around three basic elements: the processor module, dual-ported I/O controllers, and the DC power distribution system (Figures 2 and 3). The processors are interconnected by a dual-interprocessor bus system: the Dynabus; the I/O controllers are each connected with two independent I/O channels, one to each port; and the power distribution system is integrated with the modular packaging of the system.

The system design goal is two-fold: (1) to continue operation of the system through any single failure, and (2) to be able to repair that failure without affecting the rest of the system. The on-line maintenance aspects were a key factor in the design of the physical packaging and the power-distribution of the system.

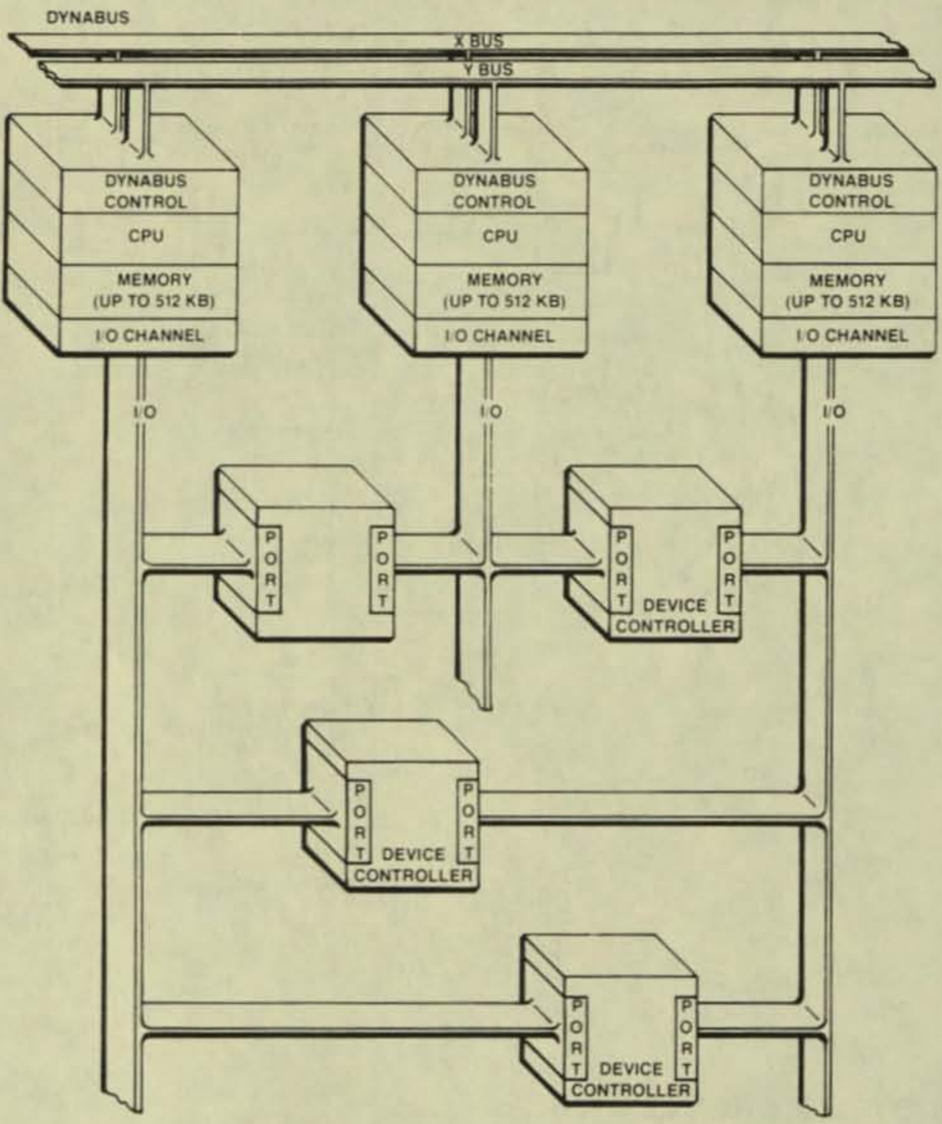


Figure 2. Tandem 16 System Architecture

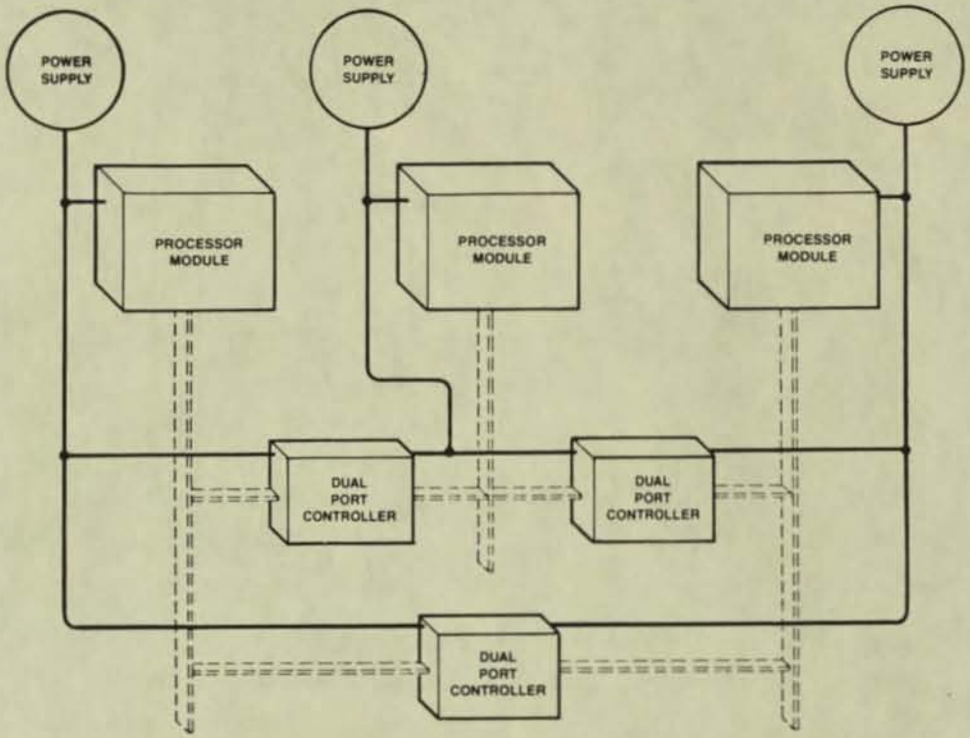


Figure 3. Tandem 16 Power Distribution

System Packaging

The cabinet (Figure 4) is divided into four sections: the upper card cage, the lower card cage, cooling, and power supplies. The upper card cage contains up to four processors, each with up to 2 M bytes of independent main memory. The lower card cage contains up to thirty-two I/O controller printed circuit (PC) cards, where each controller consists of one to three PC cards. The cooling section consists of four fans and a plenum chamber that forces laminar air flow through the card cages. The power supply section contains up to four power supply modules. Multiple cabinets may be bolted together and the system has the capability to accommodate a maximum of sixteen processors.

Each processor module, consisting of a CPU, memory, Dynabus control and I/O channel are powered by an associated power supply. If a failed module is to be replaced in this section its associated power supply is shut off, the module is replaced, and the power supply is turned on. Each card cage slot in the I/O card cage is powered by two different power supplies. Each of the I/O controllers is connected via its dual-port arrangement to two processors. Each of those processors has its own power supply; usually, but not necessarily, those two supplies are the ones that power the I/O controller (Figure 3). Each slot in the I/O card cage can be powered down by a corresponding switch disconnecting power from the slot from both supplies without affecting power to the remainder of the system. Therefore, if a power supply fails, or if one is shut down to repair a processor, no I/O controllers are affected.

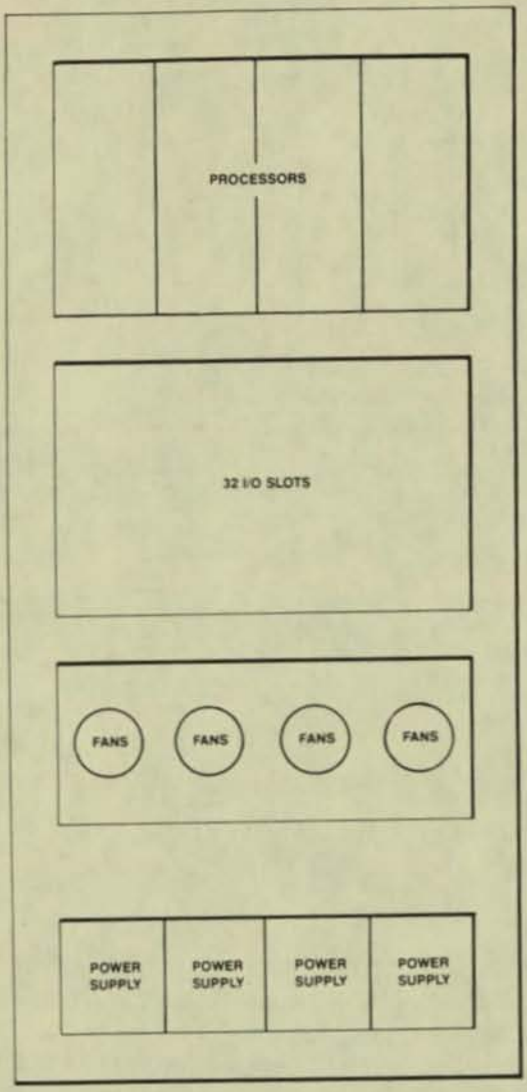


Figure 4. Tandem 16 Physical Cabinet

The dual-power sourcing to the I/O controllers was originally designed using relay switching. This plan was abandoned for several reasons: a) to contend with relay failure modes is difficult; b) the number of contact bounces on a switch-over is neither uniform nor predictable making it difficult for the operating system to handle power-on interrupts from the I/O controllers; and c) during the switch-over, controllers do lose power, and while most controllers are software-restartable, communications controllers hang up their communications lines. We therefore devised a diode current sharing scheme whereby I/O controllers are constantly drawing current from two supplies simultaneously. If a power supply fails, all the current for a given controller is supplied by the second power supply. There is also circuitry to provide for a controlled ramping of current draw on turn-on and turn-off so there are no instantaneous power demands for a given supply causing a potential momentary dip in supply voltage.

Both fans and power supplies are electrically connected using quick disconnect connectors to speed replacement upon failure. No tools are required to replace a power supply. A screwdriver is all that is needed to replace a fan. Both replacements take less than 5 minutes.

Interconnections

Physical interconnection is done both using front edge connectors and back-planes. Communications within a processor module (e.g. between the CPU and main memory) takes place over four 50-pin front edge connectors using flat ribbon cable. Interprocessor communication takes place over the Dynabus on the back-plane also utilizing ribbon cable. The I/O controllers use etch trace on the back-plane for communication among PC cards of a multicard controller. The I/O channels are back-plane ribbon cable connections between the processors and the I/O controllers.

Peripheral I/O devices are connected via shielded round cable either to a bulk-head patch panel or directly to the front edge connectors of the I/O controllers. If a patch panel is used, then there is a connection using round cables between the patch panel and the front edge connectors of the I/O controllers.

Power is distributed using a DC power distribution scheme. Physically, AC is brought in through a filtering and phase splitting distribution box. Pigtails connect the AC distribution box to one of the input connectors of a power supply. The DC power from the supply is routed through a cable harness to a laminated bus bar arrangement which distributes power on the back-plane to both processors and I/O controllers.

II. PROCESSOR MODULE ORGANIZATION

The processor (Figure 5) includes a 16-bit CPU, main memory, the Dynabus interface control, and an I/O channel. Physically the CPU, I/O channel and Dynabus control consists of two PC boards 16 inches by 18 inches, each containing approximately 300 IC packages. Schottky TTL circuitry is used. Up to 2 M bytes of main memory is available utilizing core or semicondutor technology. Core memory boards hold 32K or 128K 17-bit words and each occupy two card slots because of the height of the core stack. Semiconductor memory is implemented utilizing 16-pin, 4K or 16K dynamic RAMs. These memory boards contain 48K and 192K 22-bit words per board, respectively, and occupy only one card slot and are therefore 50% denser than core.

The processor module is viewed by the user as a 16-bit stack-oriented processor, with a demand paging, virtual memory system capable of supporting multiprogramming.

The CPU

The CPU is a microprogrammed processor consisting of a bank of 8 registers which can be used as general purpose registers, as an LIFO register stack, or for indexing; an ALU; a shifter; two memory stack management registers; program control registers (e.g. program counter, instruction register, environment or status register, and a next instruction register for instruction prefetching); scratch pad registers available only to the microprogrammer; and several other miscellaneous flags and counters for the microprogrammer.

The microprogram is stored in read-only memory and is organized in 512-word sectors of 32-bit words. The microinstruction has different formats for branching, sequential functions, and

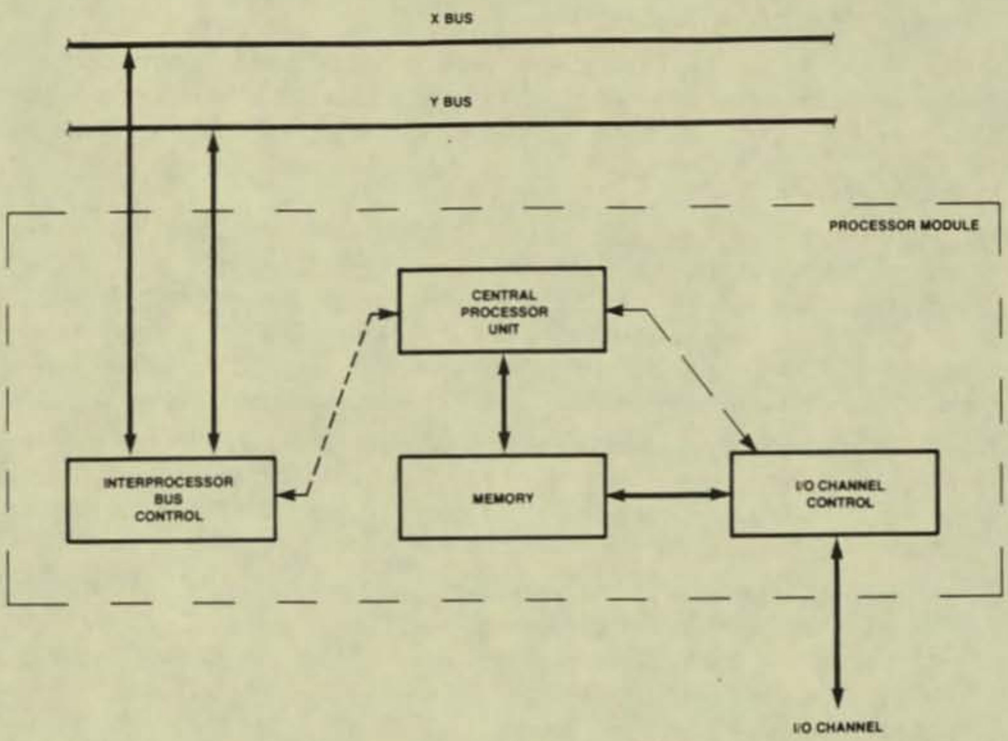


Figure 5. Tandem 16 Processor Organization

immediate operand operations. The Tandem 16 instruction set occupies 1024 words with the decimal arithmetic and the floating point options each occupying another 512 words. The address space for the micropogam is 4K words.

The microprocessor has a 100 ns cycle time and is a two-stage pipelined microprocessor, i.e., all microinstructions take two cycles to execute but one completes each cycle. In the first stage of the pipeline any two operands are selected by two source fields in the microinstruction for loading into the ALU input registers. In the second stage of the pipeline the ALU performs a primitive operation on the operands placed in the ALU input registers during the previous cycle and performs a shift operation on the results. In parallel, a miscellaneous operation such as a condition code setting or a counter increment can be done, the result can be stored in any CPU register or dispatched to the memory system or I/O channel, and a condition test made on the results. Each of these parallel operations is controlled by a separate control field in the microinstruction.

The basic set of 173 machine instructions includes arithmetic operations (add, subtract, etc.), logical operations (and, or, exclusive or), bit deposit, block (multiple element) moves/compares/scans, procedure calls and exits, interprocessor SENDs, I/O operations, and operating system primitives. All instructions are 16 bits in length. The decimal instruction set provides an additional 32 instructions dealing with four-word operands while the floating point instructions set provides an additional 43 instructions.

The interrupt system has 16 major interrupt levels which include interprocessor bus data received, I/O transfer completion, memory error, interval timer, page fault, privileged instruction violation, etc.

Provision is made for several events to cause microinterrupts. They are entirely handled by the CPU's microprocessor without causing an interrupt to the operating system. One event for example, is the receipt of a 16-word packet over the Dynabus. A packet is the primitive unit of data which is transferred over the Dynabus for interprocessor communication.The microprocessor puts the information in a predetermined area of memory and does not cause a system interrupt until the entire message is received.

The register stack is used for most arithmetic operations and for holding parameters for block instructions (moves/compares/scans) which need the parameters updated dynamically so that the instructions may be interruptable and restarted. The 8-register stack is a "wraparound" stack and is not logically connected to the memory stack.

Main Memory

Main memory is organized in physical pages of 1K words of 16 bits/word. Up to 1 M words of memory may be attached to a processor. In the core memory systems there is a parity bit for single error detection, and in semiconductor memory systems there are 6 check bits/word to provide single error correction and double error detection. Due to the relative reliability of these two technologies, we have found that semiconductor memory, without error correction, is much less reliable than core, and that with error correction, it is somewhat more reliable than core. Battery backup provides short term non-volatility to the semiconductor memory system for utility power outage considerations.

It might be noted that there are some memory systems using a 21-bit error correction scheme (5 check bits on a 16-bit data word instead of 6). While 5 bits are enough to correct all single bit errors, it does not detect approximately 1/3 of the possible double bit error combinations. In these conditions, this 5-check bit scheme will incorrectly deduce that some bit (neither of the bits actually in error) is incorrect and correctable. The scheme will then correct this bit (actually causing 3 bits to be in error), and deliver it to the system as "good" reporting a correctable memory error.

Memory is logically divided into 4 address spaces (Figure 6). These are the virtual address spaces of the machine; both the system and the user have a code space and a data space. The code space is unmodifiable and the data space can be viewed either as a stack or a random access memory, depending on the addressing mode used. Each of these virtual address spaces are 64K words long addressed by a 16-bit virtual address.

The physical memory address is 20 bits with conversion from the virtual address to physical address accomplished through a mapping scheme. Four maps are provided, one for each logical address space; each map consists of 64 entries, one for each page in the virtual address space. The maps are implemented in 50 ns access bipolar static RAM. The map access and main memory error correction is included in the 500 ns cycle time for semiconductor memory systems.

The unmodifiable code area provides reentrant, recursive, and sharable code. The data space (Figure 7) can be referenced relative to address 0 (global data or G+ addressing), or relative to the memory stack management registers in the CPU.

The lowest level language provided on the Tandem 16 system is T/TAL, a high-level, block-structured, ALGOL-like language which provides structures to get at the more efficient machine

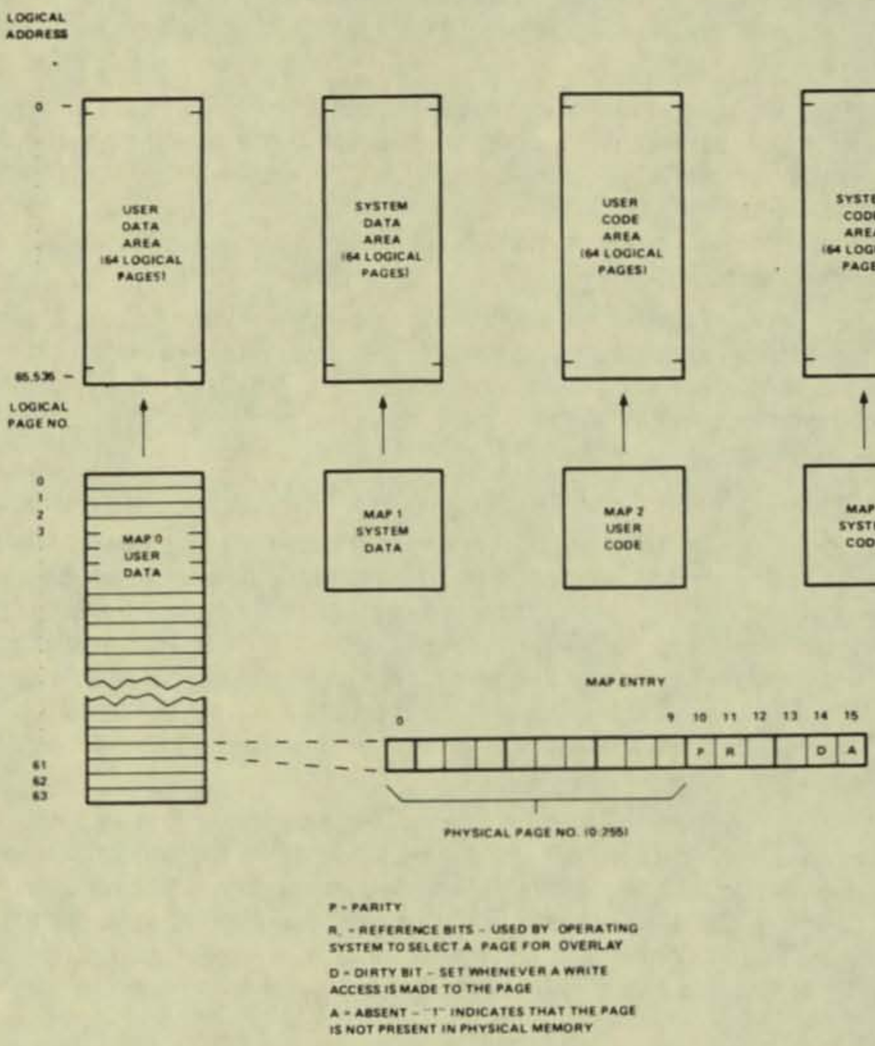

LOGICAL ADDRESS
0
65,535
USER DATA AREA (64 LOGICAL PAGES)
SYSTEM DATA AREA (64 LOGICAL PAGES)
USER CODE AREA (64 LOGICAL PAGES)
SYSTEM CODE AREA (64 LOGICAL PAGES)
LOGICAL PAGE NO.
0
1
2
3
61
62
63
MAP 0 USER DATA
MAP 1 SYSTEM DATA
MAP 2 USER CODE
MAP 3 SYSTEM CODE
MAP ENTRY
0
9
10
11
12
13
14
15
P
R
D
A
PHYSICAL PAGE NO. (0:255)
P - PARITY
R. - REFERENCE BITS - USED BY OPERATING SYSTEM TO SELECT A PAGE FOR OVERLAY
D - DIRTY BIT - SET WHENEVER A WRITE ACCESS IS MADE TO THE PAGE
A - ABSENT - "1" INDICATES THAT THE PAGE IS NOT PRESENT IN PHYSICAL MEMORY

Figure 6. Tandem 16 Logical Memory Address Spaces

instructions. The basic program unit in T/TAL is the PROCEDURE. Unlike ALGOL, there is no outer block, but rather a main PROCEDURE. T/TAL has the ability to declare certain variables as global. PROCEDURES cannot be nested in T/TAL, but a SUBPROCEDURE can be nested in a PROCEDURE and only in a PROCEDURE. A SUBPROCEDURE is limited in local variable access capabilities.

The memory stack, defined by two registers in the CPU, is used for efficient linkage to and from procedures, parameter passing, and dynamic storage allocation and deallocation for variables local to the procedure.

The L register (Local variables) points to the last stack marker placed on the stack. This marker contains return information about the caller such as the return address and the previous location of the L register. The contents of the L register are primarily changed by the procedure call and exit instructions.

Addressing relative to the L register provides access to parameters passed to a procedure (L−) and local variables of the procedure (L+). Parameters may be passed either by value (using direct addressing) or by reference (using indirect addressing).

The S register (stack top pointer) points to the last element placed on the stack. It is used for a SUBPROCEDURE's sublocal data area when S relative addressing (S−) is used.

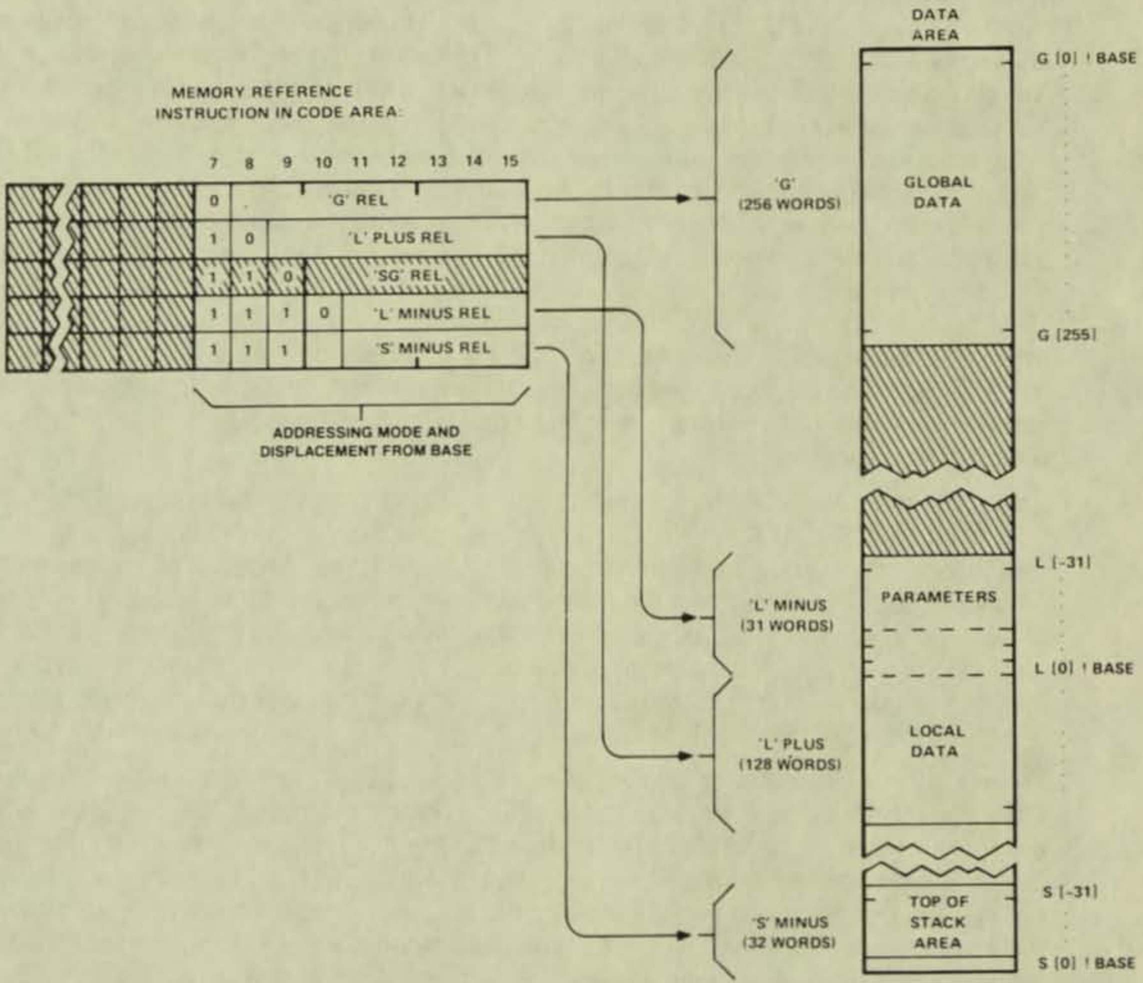


Figure 7. Tandem 16 Data Space

There is a special mode of addressing used by the operating system, called System Global (SG+) addressing. It is used by the operating system while it is working in a user's virtual data space (on his behalf) and needs to address the system data space. The system data space contains many resource tables and buffers and the need to access them quickly justifies the existence of this addressing mode.

There are three tables known to the operating system, the microprogram and the hardware: the system interrupt vector (SIV), the I/O Control (IOC) table, and the Bus Receive Table (BRT). These tables will be explained in later sections as appropriate.

The Dynabus

The Dynabus is a set of two independent interprocessor buses. Bus access is determined by two independent interprocessor bus controllers. Each of these controllers is dual-powered, in the same manner as an I/O controller. The Dynabus controllers are very small, approximately 30 IC packages, and are not associated with, nor physically a part of any processor. Each bus has a two-byte data path and control lines associated with it. There are two sets of radial connections from each interprocessor bus controller to each processor module. They distribute clocks for synchronous transmission over the bus and for transmission enable. Therefore, no failed processor can independently dominate Dynabus utilization upon failure since in order to electrically transmit onto the bus, the bus controller must agree that a given processor has the right to transmit. Each bus has a clock associated with it, running independently of the processor clocks and located on the associated bus controller. The clock rate is 150 ns on two- to eight-processor systems. The clock does need to be slowed down for the longer interprocessor buses of greater than eight processors. Therefore each bus on small systems transfers at the rate of 13.3 M bytes/second and on the larger systems at 10 M bytes/second. Performance measurements have shown that under worst case test conditions the Dynabus is only 15% utilized in a ten-processor system.

Each processor in the system attaches to both interprocessor buses. The Dynabus interface control section (Figure 8) consists of 3 high speed caches: an incoming queue associated with each interprocessor bus, and a single outgoing queue that can be switched to either of the buses. All caches are sixteen words in length and all bus transfers are cache to cache. All components that attach to either of the buses are kept physically distinct, so that no single component failure can contaminate both buses simultaneously. Also in this section are clock synchronization and interlock circuitry. All processors communicate in point-to-point manner using this redundant direct shared bus (DSB) configuration [*Ref. 3*].

For any given interprocessor data transfer, one processor is the sender and the other the receiver. Before a processor can receive data over an interprocessor bus, the operating system must configure an entry in a table (Figure 9) known as the Bus Receive Table (BRT). Each BRT entry contains the address where the incoming data is to be stored, the sequence number of the next packet, the processor number of the sender and receiver, and the number of words expected. To transfer data over a bus, a SEND instruction is executed in the sending processor, which specifies the bus to be used, the intended receiver, and the number of words to be sent. The sending processor's CPU stays in the SEND instruction until the data transfer is completed. Up to 65,535 words can be sent in a single SEND instruction. While the sending processor is executing the SEND instruction, the Dynabus interface control logic in the receiving processor is storing the data away according to the appropriate BRT entry. In the receiving processor this occurs simultaneously with the program execution.

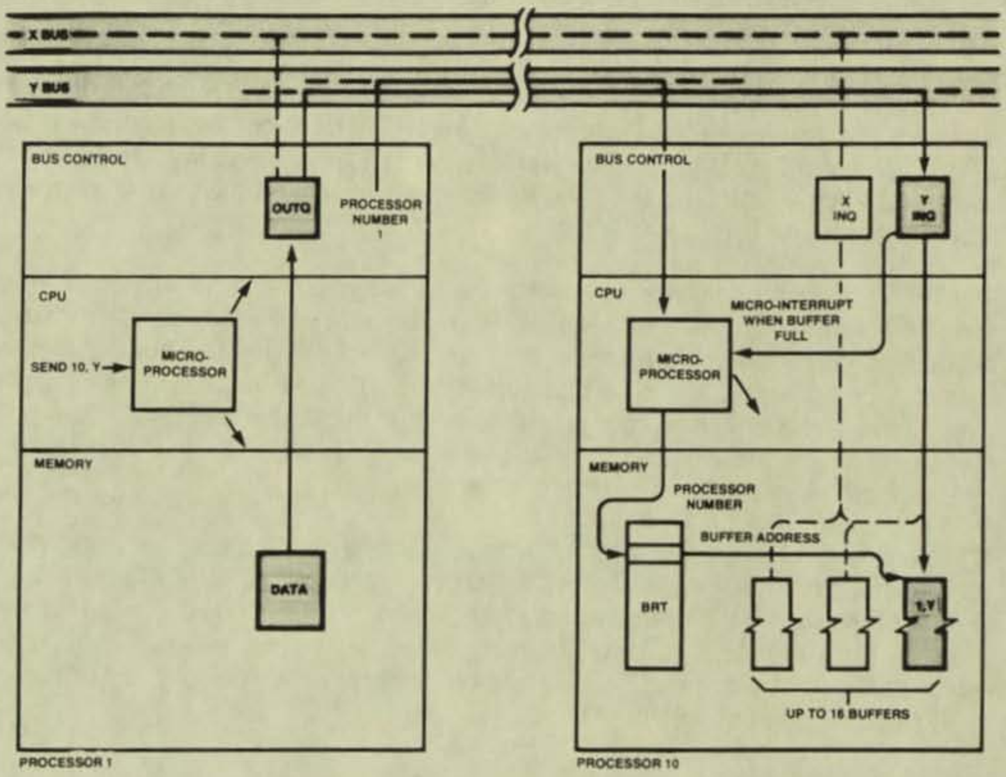
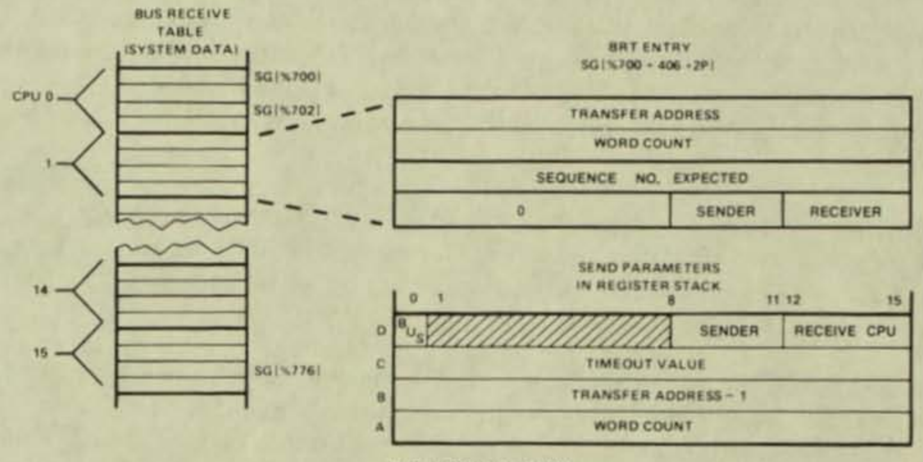


Figure 8. Tandem 16 Dynabus Interface & Control



BUS = X OR Y (0 = X BUS)
CPU = PROCESSOR MODULE 0-15
32768 - TIMEOUT VALUE IS THE NUMBER OF 0.8 μSEC
UNITS ALLOCATED TO COMPLETING THE SEND EXAMPLE
TIMEOUT VALUE = 0 THEN
32768-0 * 0.8 = 0.026 SECONDS

NOTE: "%" means base 8 notation.

Figure 9. Bus Receive Table

· The message is divided into packets of fourteen information words, a sequence number word, and an LRC check word. The sending processor first fills its outgoing queue with these packets, requests a bus transfer, and transmits upon grant of the bus by the interprocessor bus controller. The receiving processor fills the incoming queue associated with the bus over which the packet is received, and issues a microinterrupt to its own CPU. The microprocessor of the CPU checks the BRT entry, stores the packet away, verifies the LRC check word, and updates the BRT entry accordingly. If the count is exhausted the currently executing program is interrupted; otherwise program execution continues.

The BRT entries are four words that include a transfer count buffer address, sequence number expected, and the sender and receiver CPU numbers. The SEND instruction has as parameters the designation of the bus to be used, the intended receiver, the data buffer address in the system data space, the word count to be transferred, and a timeout value. Error recovery action is to be taken in case the transfer is not completed within the timeout interval. These parameters are placed on the register stack and are dynamically updated so that the SEND instruction is interruptable on packet boundaries.

There are several levels of protocol, beyond the scope of this paper, dealing with the interprocessor bus that exist in software [*Ref. 4*], to assure that valid data is transferred. The philosophy for the hardware/software partitioning was to leave the more esoteric decisions to the software, e.g., alternate path routing, and error recovery procedures, with fault detection and reporting implemented in the hardware. Fault detection was designed in those areas having the highest anticipated probability of error.

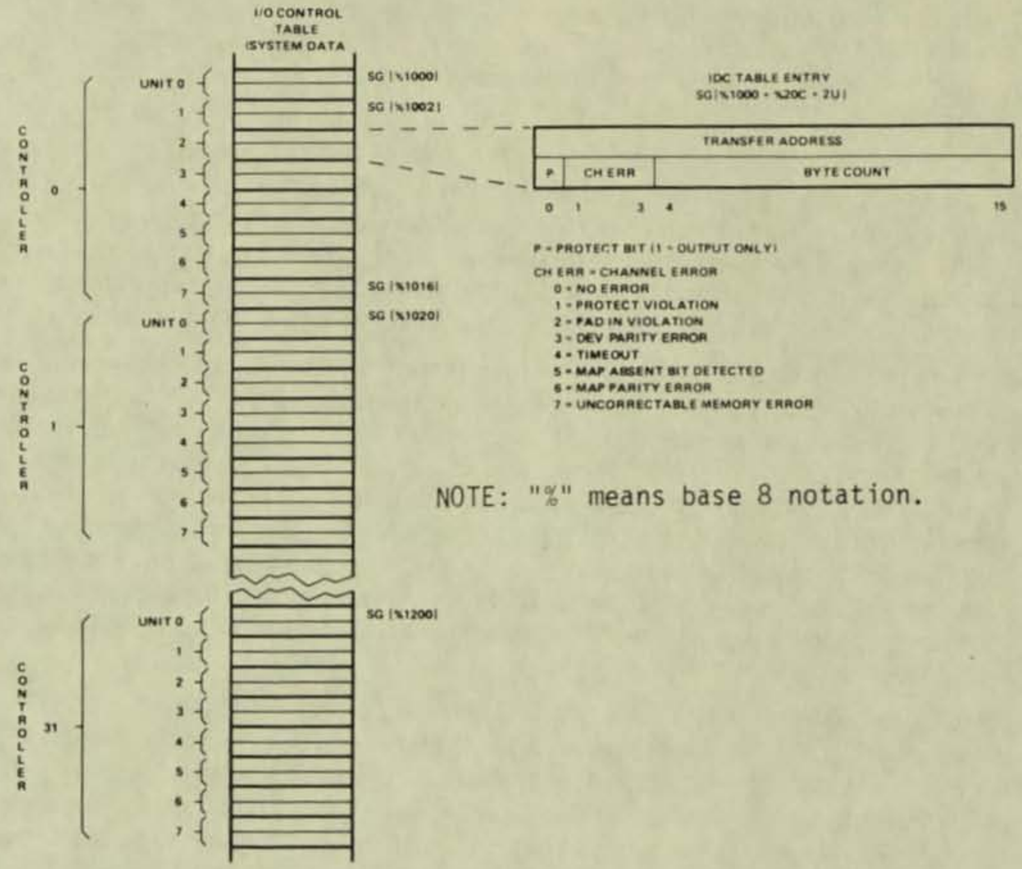
The Input/Output Channel

The heart of the Tandem 16 I/O System is the I/O channel. All I/O is done on a direct memory access (DMA) basis. The channel is a microprogrammed, block multiplexed channel with the block size determined by the individual controllers. All the controllers are buffered to some degree so that all transfers over the I/O channel are at memory speed (4 M Bytes/Second) and never wait for mechanical motion since the transfers always come from a buffer in the controller rather than from the actual I/O device.

There exists a table in the system data space of each processor called the IOC (I/O Control) table that contains a two-word entry (Figure 10) for each of the 256 possible I/O devices attached to the I/O channel. These entries contain a byte count and virtual address in the system data space for data transfers from the I/O system.

The I/O channel moves the IOC entry to active registers during connection of an I/O controller and restores the updated values to the IOC upon disconnection. The I/O channel alerts the I/O controller when the count has been exhausted and that causes the controller to interrupt the processor.

The channel does not execute channel programs as on many systems but it does do data transfer in parallel with program execution. The memory system priority always permits I/O accesses to be handled before CPU or Dynabus accesses (in an on-line, transaction oriented environment, it is rare that a system is not I/O bound). The maximum I/O transfer is 4K bytes.



NOTE: "%" means base 8 notation.

Figure 10. I/O Control Table

III. I/O SYSTEM ORGANIZATION

The I/O system had a design goal of being very efficient in a transaction, on-line oriented environment. This environment has constraints different from those of a batch environment. The figure of merit in an on-line system is the number of transactions/second/dollar that can be handled by the system. We also wanted an I/O system that had low overhead, fast transfer rates, no overruns, and no interrupts to the system until a logical entity of work was completed (e.g., no character by character interupts from the terminals). The resulting design satisfied these goals by implementing an I/O system that was extremely simple.

I/O controllers reconnect to the channel when their buffers are stressed past a configurable threshold, transfer data in a burst mode until their buffer stress is zero (buffer empty on input operations, full on output operations), and disconnect from the channel. When the transfer terminates, the I/O controller interrupts the processor. Controllers may interrupt for other reasons than an exhausted byte count, e.g., a terminal controller receiving an end-of-page character from a page mode terminal, or I/O channel error condition, or a disc pack being mounted.

Dual-Port Controllers

The dual-ported I/O device controllers provide the interface between the Tandem 16 standard I/O channel and a variety of peripheral devices using distinct interfaces. While the I/O controllers are vastly different, there is a commonality among them that folds them into the Tandem 16 NonStop architecture.

Each controller contains two independent I/O channel ports implemented by IC packages which are physically separate from each other so that no interface chip can simultaneously cause failure of both ports. Each port of each controller has a 5-bit configurable controller number and interrupt priority setting. These settings can be different on each port. The only requirement is that each port attached to an I/O channel must be assigned a controller number and priority distinct from controller numbers and priorities of other ports attached to the same I/O channel.

Each controller has a PON (power-on) circuit which clamps its output to ground whenever the controller's DC supply voltage is not within regulation. The PON circuit has hysteresis in it so that it will not oscillate if the power should hover near the limit of regulation. When the power is within regulation, the output of the PON circuit is at a TTL "1" level. A power-on condition causes a controller reset and also gives an interrupt to one of the two processors to which it is attached. The output of the PON circuit is also used to enable all the I/O channel bus transceivers so that a controller being powered down will not cause interference on the I/O channels during the power transient. This is possible because the PON circuit operates with the supply voltage as low as .2 volts and special transceivers are used which correctly stay in a high impedance state as long as the control enable is at a logical "0".

Logically only one of the two ports of an I/O controller is active and the other port is utilized only in the event of a path failure to the primary port. There is an "ownership" bit (Figure 11) indicating to each port if it is the primary port or the alternate. Ownership is changed only by the operating system issuing a TAKE OWNERSHIP I/O command. Executing this special command causes the I/O controller to swap its primary and alternate port designation and to do a controller reset. Any attempt to use a controller which is not owned by a given processor will result in an ownership violation. If a processor determines that a given controller is malfunctioning on its I/O channel, it can issue a DISABLE PORT command that logically disconnects the port from that I/O controller. This does not affect the ownership status. That way, if the problem is within the port, the alternate path can be used, but if the problem is in the common portion of the controller, ownership is not forced upon the other processor.

A controller signals an interrupt on the I/O channel if the channel has indicated an exhausted transfer count, if the controller terminates the transfer prematurely, or for attention purposes.

When simultaneous interrupts occur on an I/O channel, a priority scheme determines which interrupt is handled first. There are two levels of priorities designated "rank 0" and "rank 1". Each rank has up to sixteen controllers assigned to it. Jumper wires on each controller determine the rank and position within the rank (positions 0 to 15). The I/O channel issues a rank 0 interrupt poll cycle and each controller assigned to rank 0 can place an interrupt request, if it needs service, on a dedicated data bit of the I/O channel determined by the jumper wires. If there are no controllers on rank 0 requiring service, the I/O channel issues the interrupt poll cycle for rank 1. Note, only thirty-two controllers can be assigned to a given channel and each one has a unique rank and position designation. The highest priority controller is granted access to the interrupt system. Thus a radial polling technique allows the processor to resolve thirty-two different controller priorities in just two poll cycles. Each port of a controller has a separate set of configuration jumpers so that a controller can have different priorities on its primary and alternate path.

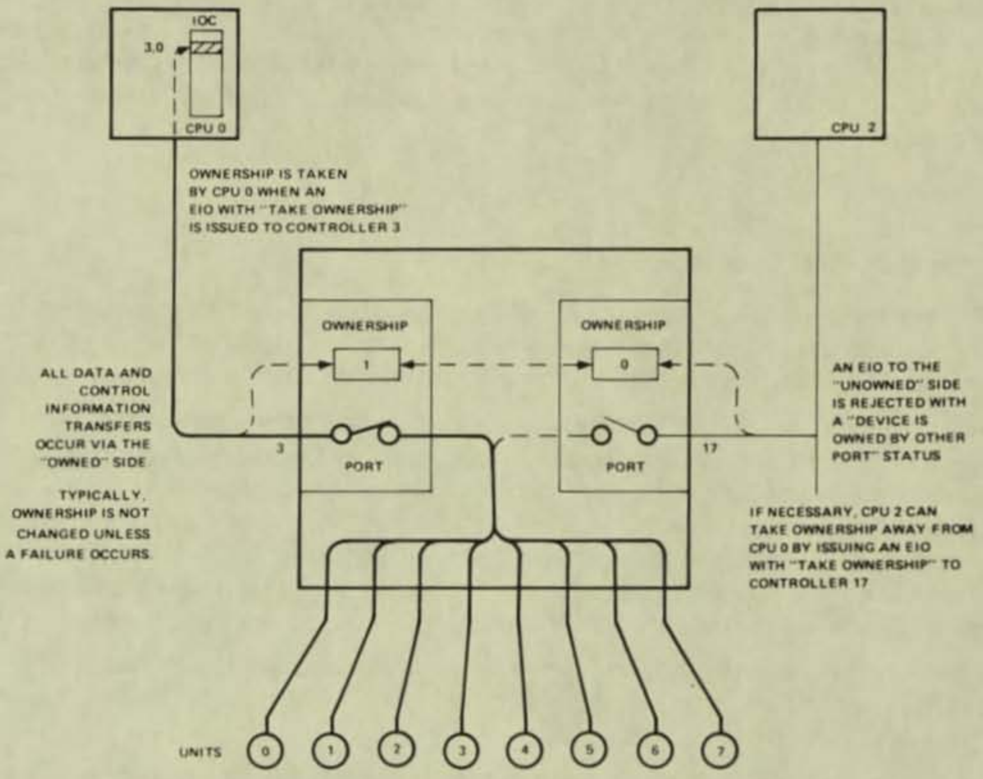

IOC
3,0
CPU 0
CPU 2
OWNERSHIP IS TAKEN BY CPU 0 WHEN AN EIO WITH "TAKE OWNERSHIP" IS ISSUED TO CONTROLLER 3
OWNERSHIP
1
OWNERSHIP
0
ALL DATA AND CONTROL INFORMATION TRANSFERS OCCUR VIA THE "OWNED" SIDE.
3
PORT
PORT
17
AN EIO TO THE "UNOWNED" SIDE IS REJECTED WITH A "DEVICE IS OWNED BY OTHER PORT" STATUS
TYPICALLY, OWNERSHIP IS NOT CHANGED UNLESS A FAILURE OCCURS.
IF NECESSARY, CPU 2 CAN TAKE OWNERSHIP AWAY FROM CPU 0 BY ISSUING AN EIO WITH "TAKE OWNERSHIP" TO CONTROLLER 17
UNITS
0
1
2
3
4
5
6
7

Figure 11. Ownership Circuitry

Controller Buffer Considerations

In the design of the Tandem 16 I/O system, a lot of attention was paid to the overrun problem. While overruns are possible on this system, they have been made a rare occurrence. Each I/O controller has three configurable settings: the I/O controller number, the interrupt priority, and buffer stress threshold reconnect setting.

Each I/O controller is buffered to some extent. The asynchronous terminal controller has 2 bytes of buffering, while the disc controller has 4K bytes of buffering. Considerations of device transfer rate, channel transfer rate, the individual controller's buffer depth, the controller's reconnect priority, and a given channel's I/O complement can be used to determine the buffer's depth (stress threshold) at which a reconnect request should be made to the channel to minimize the chance of overrun. Each controller with significant buffering (more than 32 bytes) has a configurable stress threshold. Buffer stress is defined as the number of cells full on an input operation, and the number of cells empty on output operations. In general, the I/O channel relieves stress while the I/O device generates more stress. Therefore the higher the stress, the more the buffer needs relief from the I/O channel, regardless of the direction of data transfer.

Tandem has developed a program which takes a system configuration and determines the appropriate stress threshold settings needed to guarantee no data overruns. Since reconnect overhead time is known, and all transfers on the I/O bus take place at memory speed, and the upper bound of the block length is known for each type of controller, it is a deterministic function as to whether or not an overrun is possible. If it is impossible to generate a no-overrun configuration, the program will output at minimum-overrrun threshold settings. Most times, however, it is possible to iterate on the configuration until threshold settings can be determined that prevent overruns.

Disc Controller Considerations

The greatest fear that an on-line system user has is that "the data base is down" [*Ref. 5*]. Many of these users are willing to pay the premium of having duplicated or "mirrored" data bases in case of disc drive fails. To meet this requirement, Tandem provides automatic mirroring of data bases.

A disc volume is a set of data contained on one spindle or one removable disc pack. A user may declare any of the disc volumes as mirrored pairs at system generation time (Figure 12). The system then maintains these pairs so they always contain identical data. Thus protection is achieved for a single drive failure. Each disc drive in the system may be dual-ported. Each port of a disc drive is connected to an independent disc controller. Each of the disc controllers are also dual-ported and connected between two processors. A string of up to eight drives (four mirrored pairs) can be supported by a pair of controllers in this manner.

Note that in this configuration there are many paths to any given data and that data can be retrieved regardless of any single disc drive failure, disc controller failure, power supply failure, processor failure, or I/O channel failure.

The disc controller is buffered for a maximum length record which provides several features important in an on-line system. For example, the disc controller is absolutely immune to overruns.

This disc controller uses a Fire code [*Ref. 6*] for burst error correction and detection. It can correct 11 bit bursts in the controller's buffer before transmission to the channel. Since overlapped seeks are allowed by the controller, when data is to be read from a mirrored pair it can be read from the drive which has its arm closest to the data cylinder. This is accomplished by using "split seeks," a SYSGEN parameter that requires one of the mirrored pair to only read from the first half of the disc cylinders with the other disc responsible for the second half of the disc cylinders. It is interesting to note that since the majority of transactions in an on-line system are reads, mirrrored volumes actually can increase performance.

NonStop I/O System Considerations

The I/O channel interface consists of a two-byte data bus and control signals. All data transferred over the bus is parity checked in both directions, and errors are reported via the interrupt system. A watchdog timer in the I/O channel detects if a non-existent I/O controller has been addressed, or if a controller stops responding during an I/O sequence.

The data transfer byte count word in the IOC entry contains four status bits including a protect bit. When this bit is set to "1" only output transfers are permitted to this device.

Because I/O controllers are connected between two independent I/O channels, it is very important that word count, buffer address, and direction of transfer are controlled by the processor instead of within the controller. If that information were to be kept in the controller, a single failure

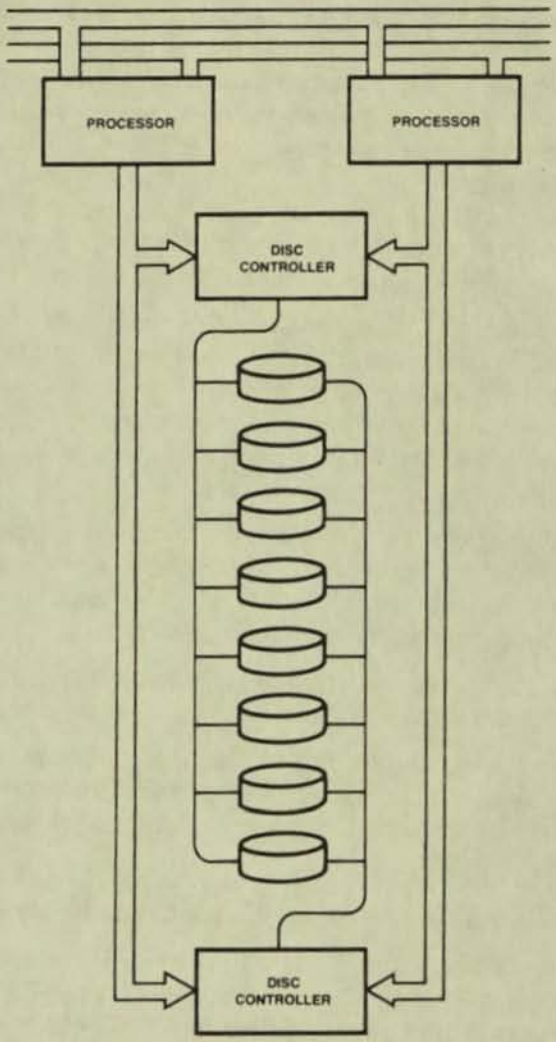


Figure 12. Tandem 16 Disc Subsystem Organization

could cause both processors to which it was attached to fail. Consider what would happen if a byte count register was located in the controller and was stuck in a situation such that the count could not decrement to zero on an input transfer. It would be possible to overwrite the buffer and cause system tables to become meaningless. The error would propagate to the other processor upon discovery that the first processor was no longer operating.

Other error conditions that the channel checks for are violations of I/O protocol, attempts to transfer to absent pages (it is the operating system's responsibility to "tack down" the virtual pages used for I/O buffering), uncorrectable memory errors, and map parity errors.

IV. POWER, PACKAGING, ON-LINE MAINTENANCE

The Tandem 16 power supply has three sections: a 5-volt interruptable section, a 5-volt uninterruptable section, and a 12-15 volt uninterruptable section. The interruptable section will stop supplying DC power when AC is lost while the uninterruptable sections will continue to supply DC power. The interruptable section powers I/O controllers and that portion of a processor which is not related to memory refresh operation. The uninterruptable sections provide power for the memory array and refresh circuitry. The 5-volt sections are switching regulated supplies while the 12-15 volt section is linearly regulated. The uninterruptable sections have a provision for a battery attachment so that in case of utility power failure, memory contents are kept for 1.5 to 4 hours, depending on the amount of memory attached to the supply.

The power supply accepts AC input of 110 or 220 volts ±20% to provide brownout insensitivity. At nominal line conditions, over 30 msec of ride-through is provided by storage capacitors. A power-fail warning signal is provided when there is at least 5 msec of regulated power remaining so that the processor can go through an orderly shut down. Some users must remain operational through utility power failure and have generator systems which provide continuous AC power for the entire system, including peripheral devices.

The power-fail warning scheme in the Tandem 16 power supply monitors charge in the storage capacitors rather than monitoring loss of AC peaks as is conventionally done. This has the advantage that the 5 msec required to do a power shutdown sequence in the processor is guaranteed even if it occurs after a brownout period.

The power supply provides all other prudent features required in a computer system, such as over voltage and over current protection, and over temperature protection.

The power-up sequencing on disc drives has been implemented with independent rather than daisy-chained circuits. In the daisy-chained approach, one bad sequencer circuit can cause the remaining drives in the chain not to sequence up after a power failure.

Further Packaging and On-line Maintenance Considerations

Modularity is a key concept in the Tandem 16 system. The maintenance philosophy is to make all repair by module replacement at the user site without making the system unavailable to the user. Therefore the back-planes, power supplies, fans, I/O channels, as well as the PC cards are modular and easily replaceable. Thumb screws are used when they can be so that a minimum number of tools is needed for repair. The package is designed so that there is easy access to all modules.

Processors and I/O controllers not only can be replaced on-line, but added on-line without system interruption if expansion is planned, all without applications software being changed.

Summary

The contribution of the Tandem 16 system lies in the synthesis of a system to directly address the need of the NonStop application marketplace. By avoiding the "onus of compatibility" to any previous system, an architecture could be designed from "scratch" that was "clean" and efficient.

The system goals have been met to a large degree. Systems have been installed containing two to twelve processors. Many application programs are on-line and running. They recover from failures, and stay up continuously.

BIOGRAPHY

James A. Katzman is a founder and Vice President of Marketing Support for Tandem Computers. From Tandem's start in November of 1974 through mid 1978, Mr. Katzman held the post of Vice President of Engineering and is one of the principal architects of the Tandem 16 NonStop system.

Previously Katzman was responsible for the design of the integrated I/O channel for the Amdahl 470V/6 system while at Amdahl Corp. from 1971 to 1974. While at Hewlett-Packard Company from 1968 to 1971 he was one of the principal architects of the HP 3000 computer system.

Mr. Katzman holds patents on all of the above machines. He is a member of the ACM and IEEE. Academically he holds a BSEE from Purdue University and a MSEE from Stanford University. He is a member of Tau Beta Pi, Eta Kappa Nu, and Omicron Delta Kappa honorary societies. He is listed in the 1978-1979 edition of Who's Who in the West.

References

[1] *Katzman, J.A., "System Architecture for NonStop Computing," Compcon, February 1977, pp 77-80.*

[2] *Tandem Computers Inc., Tandem/16 System Description, 1976.*

[3] *Anderson, G.A. and E.D. Jensen, "Computer Interconnection Structures: Taxonomy Characteristics and Examples," ACM Computing Survey, December 1975, pp 197-215.*

[4] *Bartlett, J.F., "A 'NonStop' Operating System," Hawaii International Conference of System Sciences, January 1978, this volume.*

[5] *Dolotta, T.A., M.I. Bernstein, R.S. Dickson, Jr., N.A. France, B.A. Rosenblatt, D.M. Smith, and T.B. Steel, Jr., Data Processing in 1980–1985, John Wiley & Sons, 1976.*

[6] *Peterson, W.W., Error Correcting Codes, The MIT Press, 1961, pp 183-199.*

[7] *U.S. Department of Defense, Military Standardization Handbook: Reliability Prediction of Electronic Equipment (MIL-HDBK-217B), 1965, Appendix A.*

[8] *Locks, M.O., Reliability, Maintainability, & Availability Assessment, Spartan Books/Hayden Book Company, 1973.*

APPENDIX A

The Tandem 16 system provides its high availability through architecture. In the literature [Ref. 7,8] we find that availability ranges between 0 and 1 and is defined as:

$$A = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}} \quad (1)$$

where

A	= Availability
MTBF	= Mean Time Between Failure
MTTR	= Mean Time to Repair

The availability of two redundant systems where only one is required is represented by:

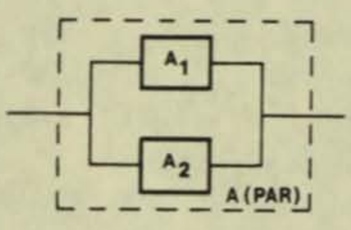


Figure 13.

and the parallel system, A(PAR), has an availability of

$$A(PAR) = A_1 + A_2 - A_1A_2 \quad (2)$$

If $A_1 = A_2 = A$ then,

$$A(PAR) = 2A - A^2 \quad (3)$$

When subsystems in series are required for operation, the system is represented by:

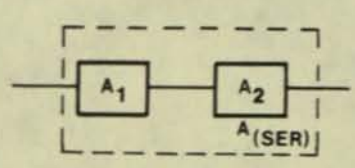
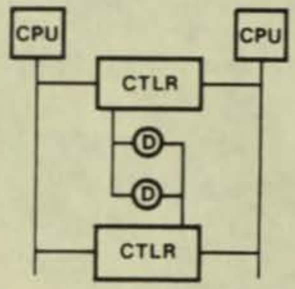


Figure 14.

and the series systems, A(SER), has an availability of

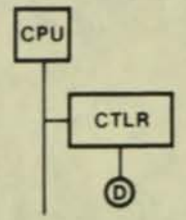
$$A(SER) = A_1A_2 \quad (4)$$

While it is not the intention of the author to give any more than these basics of theory of Availability, a comparison of three architectures of disc subsystems connected to host computers will serve as an example to demonstrate the order-of-magnitude more availability claimed for the Tandem 16 systems. The three architectures will be the following:



Tandem 16 System

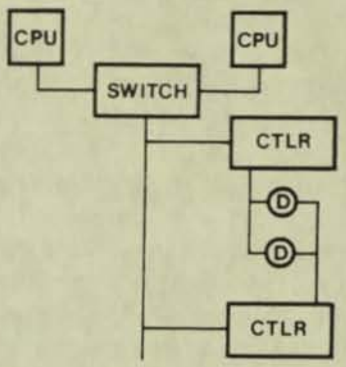
Figure 15.



Batch System

Figure 16.

and the typical "fault-tolerant" system



"f-t" System

Figure 17.

The availability models for the three systems are:

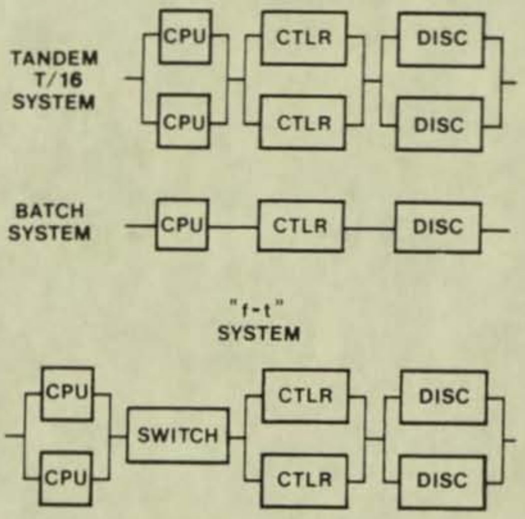


Figure 18.

Assuming the MTBF of all similar components to be equal:

CPU	9,000 hours
CTLR	12,000 hours
DISC	4,000 hours
SWITCH	15,000 hours

and the MTTR for any failure to be a conserative 24 hours, the availability of these systems are:

$$\text{One CPU} = .997340426 \quad (5)$$
$$\text{Parallel CPUs} = .999992927 \quad (6)$$
$$\text{One CTLR} = .998003992 \quad (7)$$
$$\text{Parallel CTLRs} = .999996016 \quad (8)$$
$$\text{One Disc} = .994035785 \quad (9)$$
$$\text{Parallel Discs} = .999964429 \quad (10)$$
$$\text{One Switch} = .998402556 \quad (11)$$

$$\text{Tandem 16} = (.999992927)(.999996016)(.999964229) \quad (12)$$
$$= .999953172 \quad (13)$$

$$\text{Batch System} = (.997340426)(.998003992)(.994035785) \quad (14)$$
$$= .989413082 \quad (15)$$

$$\text{"f-t" System} = (.999992927)(.998402556)(.99996016)(.999964229) \quad (16)$$
$$= .998355803 \quad (17)$$

Solving (1) for MTBF we get

$$\text{MTBF} = \frac{\text{MTTR (A)}}{1 - \text{A}} \quad (18)$$

Again assuming MTTR = 24 hours, the MTBF for the above systems are:

Tandem 16	Batch System	"f-t" system
512,490	2,243	14,573 hours
= 21,353	= 93	=607 days
=58.4	=0.25	= 1.66 years

The Tandem 16 architecture provides 35 times the MTBF of the typical "fault-tolerant" system architecture and 233 times that of the typical batch system. In this analysis it was assumed that dual controllers and dual ported discs were used, and that the two volumes were kept identical in each system except the batch system.

Tandem has completed extensive computer modeling of architectures. Empirical observations have substantiated our modeling data and product claim: the Tandem 16 architecture does, in fact, provide an order-of-magnitude more availablility than any past commercially available systems. The results seen here in this appendix, however, would not be observed normally on any of the systems mentioned. There are assumptions made which make these calculations unrealistic: all faults are not independent as assumed, faults do go undetected for long periods of time, and so forth. What this exercise does prove is that this architecture does provide a vehicle for order-of-magnitude improvement in availability which is empirically observable.

A NonStop™ Operating System*

Joel F. Bartlett
Tandem Computers Inc.
19333 Vallco Parkway
Cupertino, California

ABSTRACT

The Tandem 16 computer system is an attempt at providing a general-purpose, multiple-computer system which is at least one order of magnitude more reliable than conventional commercial offerings.Through software abstractions a multiple-computer structure, desirable for failure tolerance, is transformed into something approaching a symmetric multiprocessor, desirable for programming ease. Section 1 of this paper provides an overview of the hardware structure. In section 2 are found the design goals for the operating system, "Guardian." Section 3 provides a bottom-up view of Guardian. The user-level interface is then discussed in section 4. Section 5 provides an introduction to the mechanism used to provide failure tolerance at the application level and to application structuring. Finally, section 6 contains a few comments on system reliability and implementation.

*NonStop is a trademark of Tandem Computers Incorporated.

I. INTRODUCTION

Background

On-line computer processing has become a way of life for many businesses. As they make the transition from manual or batch methods to on-line systems, they become increasingly vulnerable to computer failures. Whereas in a batch system the direct costs of a failure might simply be increased overtime for the operations staff, a failure of an on-line system results in immediate business losses.

System Overview

The Tandem 16 [*Ref. 1, 2*] was designed to provide a system for on-line applications that would be significantly more reliable than currently available commercial computer systems. The hardware structure consists of multiple processor modules interconnected by redundant interprocessor buses. A PMS [Ref. 3] definition of the hardware is found in Figure 1.

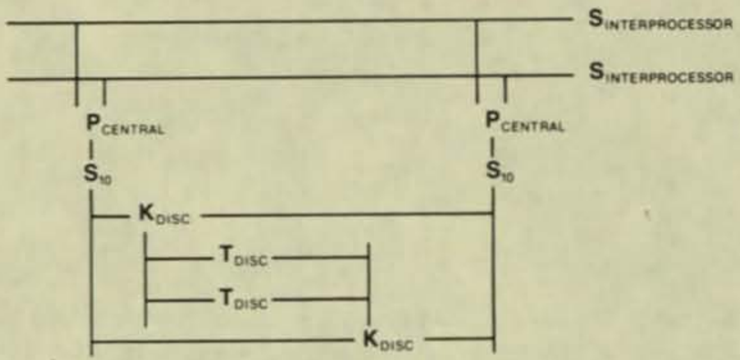


Figure 1. Hardware Structure

Each processor has its own power supply, memory, and I/O channel and is connected to all other processors by redundant interprocessor buses. Each I/O controller is redundantly powered and connected to two different I/O channels. As a result, any interprocessor bus failure does not affect the ability of a processor to communicate with any other processor. The failure of an I/O channel or of a processor does not cause the loss of an I/O device. Likewise, the failure of a module (processor or I/O controller) does not disable any other module or disable any inter-module communication. Finally, certain I/O devices such as disc drives may be connected to two different I/O controllers, and disc drives may in turn be duplicated such that the failure of an I/O controller or disc drive will not result in loss of data.

The system is not a true multiprocessor [Ref. 4], but rather a "multiple computer" system. The multiple computer approach is preferable for several reasons. First, since no module is shared by the entire system, it increases the system's reliability. Second, a multiple computer system does not require the complex hardware needed to handle multiple access paths to a common memory. In smaller systems, the cost of such a multiported memory is undesirable; and in larger systems, performance suffers because of memory access interference.

On-line repair is as necessry as reliability in assuring system availability. The modular structure of the Tandem 16 system allows processors, I/O controllers, or buses to be repaired or replaced while the rest of the system continues to operate. Once repaired, they may then be reintegrated into the system.

The system structure allows a wide range of system sizes to be supported. As many as sixteen processors, each with up to 512K bytes of memory, may be connected into one system. Each processor may also have up to 256 I/O devices connected to it. This provides for tremendous growth of application programs and processing loads without the requirement that the application be reimplemented on a larger system with a different architecture.

Finally, the system is meant to provide a general solution to the problem of providing a failure-tolerant, on-line environment suitable for commercial use. As such, the system supports conventional programming languages and peripherals and is oriented toward providing large numbers of terminals with access to large data bases.

II. SYSTEM DESIGN GOALS

Integrated Hardware/Software Design

The Tandem 16 system was designed to solve a specific problem. This problem was not stated in terms of hardware and software requirements, but rather in terms of system requirements. The hardware and software designs then proceeded in tandem to provide a unified solution. The hardware design concerned itself with the contents of each module, their interconnection to the common buses, and error detection and correction within modules and on the communication paths. The software design was given the problem of control; that is, selection of which modules to use and which buses to use to communicate with them. Furthermore, as errors are detected, it was the responsibility of the software to control recovery actions.

Operating System Design Goals

The first and foremost goal of the operating system, Guardian, was to provide a failure-tolerant system. This translated into the following design "axioms":

- the operating system should be able to remain operational after any single detected module or bus failure
- the operating system should allow any module or bus to be repaired on-line and then reintegrated into the system.
- the operating system should be implemented in a reliable manner. Increased reliability provided by the hardware architecture must not be negated by software problems.

A second set of requirements came from the great numbers and sizes of hardware configurations that are possible:

- the operating system should support all possible hardware configurations, ranging from a two-processor, discless system through a sixteen-processor system with billions of bytes of disc storage.
- the operating system should hide the physical configuration as much as possible such that applications could be written to run on a great variety of system configurations.

III OPERATING SYSTEM STRUCTURES

To satisfy these requirements, the operating system was designed to have the appearance of a true multiprocessor at the user level. The design of the system was strongly influenced by Dijkstras work on the "THE" system [Ref. 5], and Brinch Hansen's implementation of an operating system nucleus for a single-processor system [Ref. 6]. The primary abstractions are processes, which do work, and messages, which allow interprocess communication.

Processes

At the lowest level of the system is the basic hardware as earlier described. It provides the capability for redundant modules, i.e. I/O controllers, I/O devices, and processor modules consisting of a processor, memory, and a power supply. These redundant modules are in turn interconnected by redundant buses. Error detection is provided on all communication paths and error correction is provided within each processor's memory. The hardware does not concern itself with the selection of communication paths or the assignment of tasks to specific modules.

The first abstraction provided is that of the process. Each processor module may have one or more processes residing in it. A process is initially created in a specific processor and may not execute in another processor. Each process has an execution priority assigned to it. Processor time is allocated on a strict priority basis to the highest priority ready process.

Process synchronization primitives include "counting semaphores" and process local "event" flags. Semaphore operations are performed via the functions PSEM and VSEM, corresponding to Dijkstra's P and V operations. Semaphores may only be used for synchronization between processes within the same processor. They are typically used to control access to resources such as resident memory buffers, message control blocks, and I/O controllers.

When certain low-level actions such as device interrupts, processor power-on, message completion or message arrival occur, they result in "event" flags being set for the appropriate process. A process may wait for one or more events to occur via the function WAIT. The process is activated as soon as the first WAITed for event occurs. Events are signaled via the function AWAKE. Event signals are queued using a "wake up waiting" mechanism so that they are not lost if the event is signaled when the process is not waiting on it. Like semaphores, event signals may not be passed between processors. Event flags are predefined for eight different events and may not be redefined.

When a process blocks itself to wait for some event to occur or for a semaphore to be allocated to it, it may specify a maximum time to block. If the time limit expires and the event has not occurred or the resource has not been obtained, then the process will continue execution but an error condition will be returned to it. This timeout allows "watch dog" timers to be easily placed on device interrupts or on resource allocations where a failure may occur.

Each process in the system has a unique indentifier or "processid" in the form: <cpu #,process #>, which allows it to be referenced on a system-wide basis. This leads to the next abstraction, the message system, which provides a processor-independent, failure-tolerant method for interprocess communication.

Messages

The message system provides five primitive operations which can be illustrated in the context of a process making a request to some server process,Figure 2.The process's request for service will send a message to the appropriate server process via the procedure LINK. The message will consist of parameters denoting the type of request and any needed data. The message will be queued for the server process, setting an event flag, and then the requestor process may continue executing.

When the server process wishes to check for any messages, it calls LISTEN. LISTEN returns the first message queued or an indication that no messages are queued. The server process will then obtain a copy of the requestor's data by calling the procedure READLINK.

Next, the server process will process the request. The status of the operation and any result will then be returned by the WRITELINK procedure, which will signal the requestor process via another event flag. Finally, the requestor process will complete its end of the transaction by calling BREAKLINK.

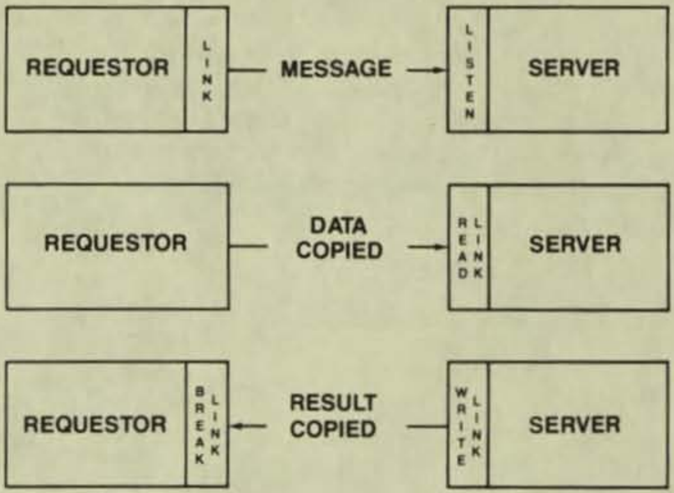


Figure 2. Message System Primitive Operations

A communications protocol was defined for the interprocessor buses that would tolerate any single bus error during the execution of any message system primitive. This design assures that a communications failure will occur if and only if the sender or receiver processes or their processors fail. Any bus errors which occur during a message system operation will be automatically corrected in a manner transparent to the communicating processes and logged on the system console. The interprocessor buses are not used for communication between processes in the same processor, which can be done faster in memory. However, the processes involved in the message transfer are unable to detect this difference.

The message system is designed such that resources needed for message transmission (control blocks) are obtained at the start of a message transfer request. Once LINK has been successfully completed, both processes are assured that sufficient resources are in hand to be able to complete the message transfer. Furthermore, a process may reserve control blocks to guarantee that it will always be able to send messages to process a request that it picks up from its message queue. Such resource controls assure that deadlocks can be prevented in complex producer/consumer interactions, if the programmer correctly analyzes and anticipates potential deadlocks within the application.

Process-Pairs

With the implementation of processes and messages, the system is no longer seen as separate modules. Instead, the system can be viewed as a set of processes which may interact via messages in any arbitrary manner, as shown in Figure 3.

By defining messages as the only legitimate method for process-to-process interaction, inter-process communication is not limited by the multiple-computer organization of the system. The system then starts to take on the appearance of a true multiprocessor. Processor boundaries have been blurred, but I/O devices are still not accessible to all processes.

System-wide access to I/O devices is provided by the mechanism of "process-pairs." An I/O process-pair consists of two cooperating processes located in two different processors that control a particular I/O device. One of the processes will be considered the "primary" and one will be considered the "backup." The primary process handles requests sent to it and controls the I/O device. When a request for an operation such as a file open or close occurs, the primary will send this information to the backup process via the message system. These "checkpoints" assure that the backup process will have all information needed to take over control of the device in the event of an I/O channel error or a failure of the primary process's processor. A process-pair for a redundantly-recorded disc volume is illustrated in Figure 4.

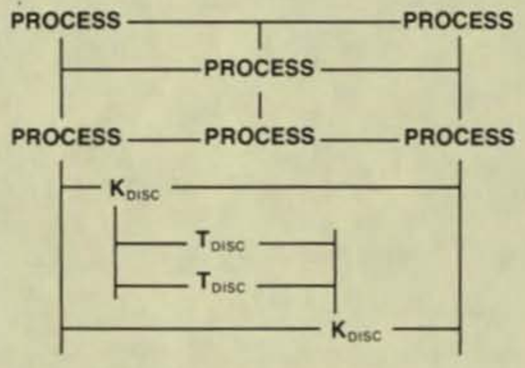


Figure 3. System Structure After the Addition of Processes and Messages

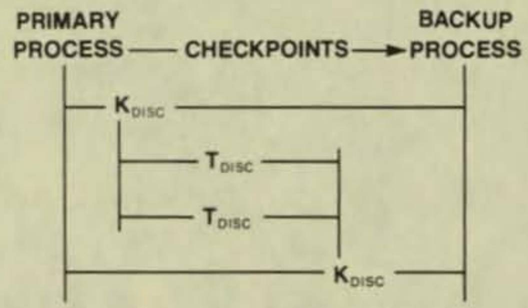


Figure 4. Process-pair for a Redundantly-Recorded Disc Volume

Because of the distributed nature of the system, it is not possible to provide a block of "driver" code that could be called directly to access the device. While potentially more efficient, such an approach would preclude access to every device in the system by every process in the system.

The I/O process-pair and associated I/O device(s) are known by a logical device name such as "$DISC1" or by a logical device number rather than by the processid or either process. I/O device names are mapped to the appropriate processes via the logical device table (LDT) in every processor, which supplies two processids for each device. A message request made on the basis of a device name or number results in the message being sent to the first process in the table. If the message cannot be sent or if the message is sent to the backup process, an error indication will be returned. The processid entries in the LDT will then be reversed and the message resent. Note two things: first, the error recovery can be done in an automatic manner; and second, the requestor is not concerned with what process actually handled the request. Error recovery cannot always be done automatically. For example, the primary process of a pair controlling a line printer fails while handling a request to print a line on a check. The application process would prefer to see the process failure as an error rather than have the request automatically retried, which might result in two checks being printed.

The two primitives, processes and messages, blur the boundaries between processors and provide a failure-tolerant method for interprocess communication. By defining a method of grouping processes (process-pairs), a mechanism for uniform access to an I/O device or other system-wide resource is provided. This access method is independent of the function performed within the processes, their locations, or their implementations. Within the process-pair, the message system is used to checkpoint state changes so that the backup process may take over in the event of a failure. This checkpoint mechanism is in turn independent of all other processes and messages in the system.

The system structure can be summarized as follows. Guardian is constructed of processes which communicate using messages. Fault tolerance is provided by duplication of components in both the hardware and the software. Access to I/O devices is provided by process-pairs consisting of a primary process and a backup process. The primary process must checkpoint state information to the backup process so that the backup may take over on a failure. Requests to these devices are routed using the logical device name or number so that the request is always routed to the current primary process. The result is a set of primitives and protocols which allow recovery and continued processing in spite of bus, processor, I/O controller, or I/O device failures. Furthermore, these primitives provide access to all system resources from every process in the system.

System Processes

The next step in structuring the system comes in assigning functions to processes. As previously shown, I/O devices are controlled by process-pairs. Another process-pair known as the "operator" is present in the system. This pair is responsible for formatting and printing error messages on the system console. Here is an example of where Guardian has not followed a strict level structure. The operator makes requests to a terminal process to print the messages, yet the terminal process wishes to send messages to the operator to report I/O channel errors. An infinite cycle is prevented by having the terminal process not send messages for errors on the operator terminal and having I/O processes never wait for message completions when sending errors to the operator. While it may be preferable to prevent cycles of any type in system design, they have been allowed in Guardian when it can be shown that they will terminate. The ability to reserve message control blocks assures that no cycle will be blocked because of resource problems.

Each processor has a "system monitor" process which handles such functions as process creation and deletion, setting time of day, and processor failure and reload cleanup operations.

A memory management process is also resident in each processor. This process is responsible for allocating a page of physical memory and then sending messages to the appropriate disc processes to do the actual disc I/O. Pages are brought in on a demand basis and pages to overlay are selected on a "least recently used" basis over the entire memory of the processor.

The choice of relatively unsophisticated algorithms for scheduling and memory management was a result of the fact that the system was not intended to be a general-purpose timeshare system. Rather, it was to be a system which supported multiple processes and terminals in an extremely flexible manner.

Application Process Interface

Above the process and communication structure there exists a library of procedures which are used to access system resources.These procedures run in the calling process's environment and may or may not send messages to other processes in the system. For example, the file system procedures do not do the actual I/O operations. Instead, they check the caller's parameters, and if all is in order a message is sent to the appropriate I/O process-pair. Likewise, process creation is seen as a procedure call to NEWPROCESS, which does nothing but check the caller's parameters and then send a message to the system monitor process in the processor where the process is to be created. On the other hand, a procedure such as TIME which returns the current time of day does not send any messages. In either case, the access to system resources appears simply as procedure calls, effectively hiding the process structure, message system, hardware organization, and associated failure recovery mechanisms.

Initialization and Processor Reload

System initialization starts with one processor being cold loaded from some disc on the system. The load file contains a memory image of the operating system resident code and data, with all system processes in existence and at their initial states. The system monitor process then creates a command interpreter process.

Guardian may be brought up even though a processor or peripheral device is down. This is possible because operating system disc images may be kept on multiple disc drives, I/O controllers may be accessed by two different processors, and the terminal that has the initial command interpreter on it is selected by using the processor's switch register.

After a cold load,the system logically consists of one processor and any peripherals attached to it. More processors and peripherals may be added to the system via the command interpreter command:

```
:RELOAD  1,$DISC
```

This command will read the disc image for processor 1 from the disc $DISC and send it over either interprocessor bus to processor 1. Once it is loaded, all processes residing in other processors in the system will be notified that processor 1 is up.

This command is also used to reload a processor after it has been repaired. Guardian does not differentiate between an initial load of a processor and a later reload. In each case, resources are being logically added to the system and processes must be notified so that they may make use of them.

The previous example of a reload message being sent to all processes is an example of how functions are split in Guardian. A mechanism is provided for informing a process of a system status change. It may then take some unspecified action (including doing nothing). Similarly, a system power-on simply sets the PON event flag for all processes. The operating system kernel must only insure that the process structure and message system are correctly saved and restored. It is then the responsibility of individual processes to do such things as reinitialize their I/O controllers.

Operating System Error Detection

Besides the hardware-provided single error detection and correction on memory, and single error detection on the interprocessor and I/O buses, additional software error checks are provided. The first of these is the detection of a down processor. Every second, each processor in the system sends a special "I'm alive" message over each bus to all processors in the system. Every two seconds, each processor checks to see that it has received one of these messages from each processor. If a message has not been received, then it assumes that that processor is down.

Additionally, the operating system makes checks on the correctness of data structures such as linked lists when operations are done on them. Any processor detecting such an error will halt.

All I/O interrupts are bracketed by a "watch dog" timer such that the system will not hang up if an I/O operation does not complete with the expected interrupt. If an I/O bus error occurs then the backup process will take over control of the device using the second I/O bus.

As previously noted, the interprocessor bus protocol is designed to correct single bus errors. In addition to this, extensive checks are made on the control information received over the buses to verify that it is consistent with the state of the receiving processor.

Power-fail/automatic restart is provided within each processor. A power-failure is detected independently by each processor module and as a result is not a system-wide, synchronous event. The system was designed to recover from either a complete system power-fail, or a transient which will cause some of the processors to power-fail and then immediately restart.

IV. USER-LEVEL SYSTEM INTERFACE

Tools are provided for interactive program development using COBOL or a block-structured implementation language, T/TAL. A file system with facilities comparable to or exceeding those offered by other "midi" computer systems allows access to disc files and other I/O devices. Process creation, intercommunication, and checkpointing primitives are also implemented.

The application process level facilities and the interactive program development tools have been heavily influenced by the HP3000 [*Ref. 7*] and by UNIX [*Ref. 8*].

Interactive System Access

General-purpose, interactive access to the system is provided by the command interpreter, COMINT, similar in many ways to the Shell of UNIX. Normally a command interpreter is run interactively from a terminal, but commands may be read from any type of file. The command interpreter is seen by the operating system as simply another type of application process.

Commands are read from the terminal, prompted by a colon (":"):

```
: command / process parameters / arguments
```

If the command is recognized, it will be directly executed. A command of this type is:

```
:LOGON SOFTWARE.JOEL
```

which is used to gain access to the system. If the command is not recognized, then a process will be created using the program file "$SYSTEM.SYSTEM.command" and the arguments for the command will be sent to this new process. The command interpreter will then suspend itself until a message is received indicating that the process has stopped. If this process cannot be created, then an error message is printed. For example, the text editor is accessed by typing EDIT followed by any command string:

```
:EDIT FILE
```

This will result in a process being created using the program file $SYSTEM.SYSTEM.EDIT and the command string, "FILE," being sent to it. Also a part of this command string message are the names of the files that are being used for input and output by the command interpreter. These are then used by the process for its input and output. If the previous command was typed at a terminal, the input and output files would be the device name of the terminal. Alternative names for the input and output files may be specified. For example:

```
: EDIT      /IN COMMANDS/
```

will create an editor process and pass it the file name "COMMANDS" for the input file and the terminal's file name, the default, for the output file. Finally, the processor to use and the priority at which to run the process may also be specified:

```
: EDIT      /PRI 100, CPU 3/
```

This will create an editor process in processor three with a priority of 100.

Additional features allow multiple processes to be started from one command interpreter and allow the previously typed command line to be edited.

Programming Languages

Compilers have been implemented for two languages, T/TAL and ANSI 74 COBOL. T/TAL is a block-structured implementation language. Its capabilities are similar to those offered by C on UNIX or SPL on the HP3000. All Tandem software is written in T/TAL as are most user applications.

Code generated by either compiler may be shared by multiple processes in the same processor. Both compilers generate an object file which may be immediately run without any intervening link edit operation. However, the object file also contains enough information so that an object editor, UPDATE, may combine the objects produced by several compilations or selectively replace procedures in an object file.

Tools

Program development tools include an interactive text editor, object file editor, text formatter, and interactive debugger. A screen generation program and access routines are provided to facilitate application interaction with page mode CRT terminals. File utilities exist which allow file backup and restore, file copying and dumping, and initial loading of key-sequenced files. A peripheral utility is provided to do such operations as disc formatting, disc track sparing, and mounting or demounting disc volumes.

Process Creation and Deletion

Processes are created by the command interpreter or by an application process call to the procedure NEWPROCESS. Parameters supplied include the name of the file holding the object code for the process, the processor number to use, and the priority at which to run the process. The parameters will be checked and then sent to the system monitor process in the appropriate processor. The system monitor will then create the process and return a "creationid" identifying the new process to the calling process. Part of this value is the processid previously defined, and the rest is the value of the processor clock at the time of process creation. The clock is kept as a 48 bit value which is the number of 10ms intervals since 12 a.m. on December 31, 1975, which assures that creationids will be unique over the life of the system.

Processes are not grouped in classical ancestry trees. No process is considered subservient to any other process on the basis of parentage. Two processes, one created by the other, will be treated as equals by the system. When a process, A, creates another process, B, no record of B is attached to A. The only record kept is in the process B where the creationid of A is saved. This creationid is known as B's "mom." When process B stops, process A is sent a stop message indicating that process B no longer exists. A process's mom is flexible and a process may adopt another process. For example (Figure 5), process A creates process B. Process B in turn creates a cooperating process, C. Since C would like to know if B stops, C will adopt B.

A process may stop itself or some other process by calling STOP. Process deletion is again a function of the system monitor process. Resources will be released and a stop message will be sent to the process's mom. If the mom process does not exist, then no message will be sent.

$$\begin{tikzcd}
\text{A CREATES B:} & A & \text{MOM} \arrow[l] \arrow[r, dash] & B & & \\
\text{B CREATES C:} & A & \text{MOM} \arrow[l] \arrow[r, dash] & B & \text{MOM} \arrow[l] \arrow[r, dash] & C \\
\text{C "ADOPTS" B:} & A & & B & \text{MOM} \arrow[l] \arrow[r] & C
\end{tikzcd}$$

Figure 5. Flexible Process Relationships

Application Process-Pairs

The process-pair concept introduced earlier is a powerful method for making some resource available to all processes in the system in a fault-tolerant manner. It is extended to the application processes as follows. When a process is created via NEWPROCESS, a process-pair name may be supplied. The creationid returned for this process consists of the processid and the process name rather than the processor clock value. For example (Figure 6), process A wishes to create a process with the name "$SPOOL." Once B has been created, any process in the system may send a message to that process via the name "$SPOOL."

Process B may now wish to create a process B′ in another processor to be its backup. B would then call NEWPROCESS, supplying the name "$SPOOL." Process B will keep B′ updated via checkpoints so that B′ may become the primary if B fails. B and B′ each wish to receive an indication if the other process is deleted. Therefore, B and B′ will automatically set to be each other's moms.

When the last process with the name "$SPOOL" is deleted, process A will be sent a message. Process A is known as the "ancestor" by the fact that this process was the one which created the first process by the name of "$SPOOL." Process A in turn may be a named process, in which case A's name will be sent the termination message. This allows a process-pair, "$A" consisting of processes A and A′, to create a named process, "$B" consisting of B and optionally B′, and guarantee that it will be sent a message when the process name $B no longer exists. This will occur even if the process which first created $B no longer exists.

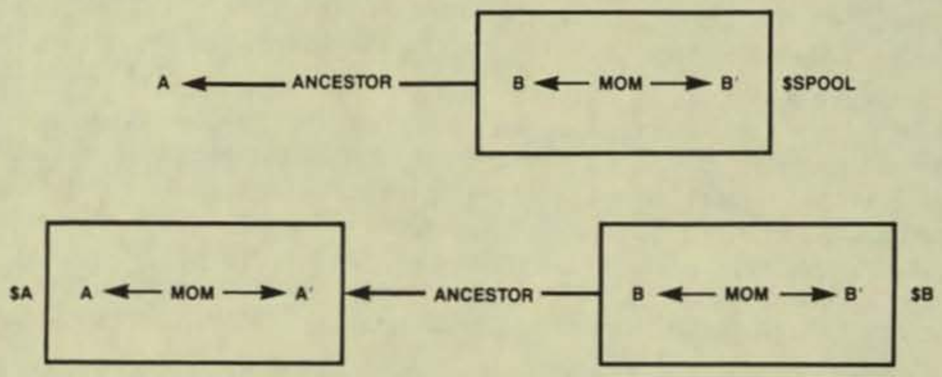


Figure 6. Application Process-pairs

File System

The Guardian file system provides a uniform method for access to disc files, unit record devices, and processes. All files are named: disc files have names such as "$DISC1.SUBVOL.FILE" and unit record devices and processes have names such as "$LP." Access by name allows any process running in any processor to access any file in the system. Direction to the appropriate process of the process-pair is handled by the file system in a manner transparent to the requesting program.

Files of all types are opened by calling:

```
CALL OPEN (filename,filenum,. . . )
```

The calling process supplies the file name. Security will be checked and then a file number will be returned to the calling process. This file number is then used for all further accesses to the file. A file may be opened for "wait" or "no-wait" access. If "wait" access is chosen, the process will be suspended until the requested operation on the file has been completed. On the other hand, if the "no-wait" access is requested then once the operation has been initiated, the requesting process may continue processing.

Disc Files

Each disc file is composed of between one and sixteen partitions. Each partition resides on a specific disc volume and is in turn composed of up to sixteen extents. Each extent is one or more contiguous disc pages of 2048 bytes each. Disc files come in several types. The first is "unstructured," similar to UNIX, where the file is treated as a contiguous set of bytes. A current file pointer is kept which is the byte address of the beginning of the next transfer. After each read or write operation:

```
CALL READ(filenum,buffer,cnt,transfercnt)
CALL WRITE(filenum,buffer,cnt,transfercnt)
```

the file pointer is incremented by the number of bytes transferred. The file pointer may be moved explicitly by:

```
CALL POSITION(filenum,fileposition)
```

The second type of file is a "relative-record" file. The file consists of fixed or variable-size records and may be randomly accessed. Rather than positioning to an arbitrary byte in the file, the process positions to the start of a specific record. If the process reads less than the record size, then the file pointer advances to the start of the next record.

The third type of file is "entry-sequenced." Records written to this file may be of varying lengths and are always appended to the end of the file. This type of file is normally used as a log file.

The final type of disc file is "key-sequenced." A file of this type may have a unique primary key and up to 255 alternate keys. Entry-sequenced and relative-record files may also have alternate keys. Each key may be up to 255 bytes long. Access to this file may be done on any key using the procedure:

```
CALL KEYPOSITION(filenum,key,keytag,keylen,positionmode)
```

The parameter "keytag" identifies which key is being used. The pointer "key" designates the value of the key which is "keylen" bytes long. The "positionmode" describes what type of access is to be made to the file. The first type of access is "approximate." Using this, successive reads to the file will return all records whose key values are greater than or equal to the "key" for "keylen." The second type of positioning is "generic." Here, successive reads will return all records whose key value is equal to "key" for "keylen." The final type of positioning is "exact." Successive reads will return all records whose keys are "keylen" long and equal to "key."

Files or individual records may be locked by:

```
CALL LOCKFILE(filenum)
CALL LOCKRECORD(filenum)
```

Record locking and unlocking may be combined with the actual I/O operation desired for increased efficiency:

```
CALL READUPDATELOCK(filenum,. . . )
CALL WRITEUPDATEUNLOCK(filenum,. . . )
```

The distributed nature of the system does not allow efficient detection of deadlocks caused by file locking. As a result, this type of checking is not done. A lock request on a file that has been opened with "no-wait" access will allow the application to do other processing if the requested file is not immediately available. A process may use this mechanism to assure that it will not wait indefinitely in the case of a deadlock.

Disc I/O

The disc processes in each processor share an area of main memory called the "disc cache." Each block read from the disc is placed in this area. Space is reused on a weighted "least recently used" basis. Thus, such items as index blocks for key sequenced files are kept available in memory so that successive accesses do not require that they be reread.

A logical disc volume, "$DISC1," may be recorded onto two different disc drives using two different I/O controllers. This second, or "mirror" volume provides a transparent duplication of data which protects a data base against loss via a failed disc drive or controller. All file writes are performed on both disc drives and file reads may be done from either drive. When a failed drive has been repaired, it may be "revived" while the application continues accesses and updates to files on that logical device.

Device I/O

I/O operations are done to unit record devices in a similar manner to disc file accesses. Here, Guardian does not support a record-structured, device-independent mode of access and as a result operations such as unblocking tape records must be done by the application program. While this lack of device-independent I/O can be considered a liability in some applications, it allows easy addition of new types of I/O devices to the system without requiring changes to the file system and allows device-dependent control by the application program.

Read and write operations are done in an identical manner for all files. Device-dependent operations such as skipping on vertical form channels on a line printer may be done by:

```
CALL CONTROL(filenum,control#,parameter)
```

Enabling or disabling terminal parity checking or other such access options are done by:

```
CALL SETMODE(filenum,modetype,. . .)
```

Guardian provides an extremely general purpose interface to asynchronous RS-232 or current loop devices. The file system and asynchronous terminal process provide a read after write operation:

```
CALL WRITEREAD(filenum,buffer,writecnt,readcnt,countread)
```

which allows a character sequence to be output to a device followed immediately by a read from the device. This allows the character sequence which causes a CRT terminal to transmit to be sent to it. The line will then be turned around and the terminal's buffer read into the processor. Since this write/read turn-around is done in the device controller, no data is lost because the read could not be started soon enough.

Normally, operating systems wish to enforce certain terminal characteristics such as inserting a carriage return and linefeed after each line written or interpreting certain characters on input for such operations as line and character delete. While Guardian provides such facilities, they may be disabled at the time the system is configured or after the file is opened. Other characteristics such as type of connection, character size, parity, speed, and character echoing are completely configurable. This allows arbitrary character sequences to be input and output without any interpretation or character editing being done by the operating system.

Communication software is also provided to handle multi-point asynchronous terminals. Point-to-point and multi-point Bisync software is also provided. Rather than attempting to emulate specific devices, the application program is allowed to specify the line control used.

Interprocess I/O

Each process in the system may have messages from other processes queued for it. Access to this message queue is provided via the file "$RECEIVE." A read on this file will return the first message. A process may check to see if any messages are queued and then continue processing if none are present. A process may ascertain the identity of the sending process via the procedure:

```
CALL LASTRECEIVE(sender)
```

This returns the "creationid" of the sending process. It is supplied by the operating system and thus may not be forged by the sending process. A process will receive indication of such events as a process being stopped,a processor failingor being reloaded,or the break key being pressed on a terminal that this process has open in the form of messages read from this file.

A process may open another process as a "file." Once opened, the process may use the file system procedures WRITE, WRITEREAD, SETMODE, and CONTROL to send messages to that process. The receiving process will read these requests from its "$RECEIVE" file. It will then process them and possibly return an error indication to the sending process. This allows the "server" process to simulate some arbitrary device. Using these tools, an output spooler or a process which could allow access to labeled magnetic tapes written on some other system can be constructed. The requesting process believes that it is communicating with a device, and the server process is able to simulate that device without requiring special privileged "hooks" in the file system.

V. APPLICATION PROGRAMS

Application Program Checkpointing

Application process-pairs are used to provide some service on a failure-tolerant basis. Requests are processed by the primary process and results are returned to the requestor process. On a failure of the primary process, the backup must be able to continue offering this service. This requires that any state changes in the primary process be sent (checkpointed) to the backup process. While such checkpoints could be sent on an instruction-by-instruction basis, this is clearly not feasible because of the overhead involved. Instead, a process need only checkpoint its state when it is about to make a non-retryable request to another process.

For example, at time T1, when the primary process and its backup are in the same state, the primary process starts some operation. Later, at time T2, when it is ready to write the result to a disc file, it will checkpoint changes made since time T1 to its backup. The processes will then again be in the same state. If the primary process failed at any point before T2, the backup process could restart at the last checkpoint, made at T1. The selection of states to checkpoint is analogous to defining restart points for jobs in a batch processing system. In a batch environment, these checkpoints are saved in a disc file; in process-pairs they are saved in a backup process.

Guardian provides system functions for checkpointing process state information between processes of a process-pair. The first type of item checkpointed is portions of the process's data space. This includes global data and/or the current stack, holding procedure return addresses, procedure local variables, and procedure parameters. Consider the following program segment, written in T/TAL, whose purpose is to output to a terminal the first 100 items of an array, "buffer:"

```
FOR  i := 1 TO 100  DO
  BEGIN
  CALL WRITE(terminal,buffer[i],itemlen);
  END;
```

This operation could be made failure-tolerant by two calls to the CHECKPOINT procedure. The first checkpoint copies the entire buffer to the backup process. This need only be done once as the data is not changed by later processing. The second checkpoint, before each write, saves the current process state, including the variable "i." This allows the backup process to take over the operation, duplicating at most one line of output.

```
CALL CHECKPOINT(,buffer[1],buffersize);
FOR  i := 1 TO 100  DO
  BEGIN
  CALL CHECKPOINT(stackbase);
  CALL WRITE(outfile,buffer[i],itemlen);
  END;
```

When the primary process fails, the backup will take over at the last checkpoint. The next logical extension to the original segment would be if the process were copying the one hundred values to be output from some disc file:

```
FOR  i := 1 TO 100  DO
  BEGIN
  CALL READ(infile,buffer,itemlen);
  CALL WRITE(outfile,buffer,itemlen);
  END;
```

In this case, not only would the process's data space contents need to be checkpointed as before, but so would the current file pointers for the input and output files. This ensures that they are correctly set when the backup process takes over. In order for file pointers to be checkpointed, both processes of the process-pair must have the files open. Special functions are provided which allow the primary process to cause a file to be opened or closed by the backup process:

```
CALL CHECKOPEN(filename,. . . )
CALL CHECKCLOSE(filename,. . . )
```

In the sample program, CHECKOPEN would be called following the call to OPEN when the primary process started. The program segment would now look like:

```
CALL CHECKPOINT(,buffer[1],buffersize);
FOR  i := 1 TO 100  DO
  BEGIN
  CALL CHECKPOINT(stackbase,,infile,,outfile);
  CALL READ(infile,buffer,itemlen);
  CALL WRITE(outfile,buffer,itemlen);
  END;
```

If a failure occurred after the read but before the write, the backup would take over and repeat the read using the same file pointer as was used by the primary. In both of these examples, a failure following the write but preceding the next checkpoint could result in a record being written twice. This would cause no problem if the record was being written to some absolute position in the file; however, an error would occur when writing to a key-sequenced disc file. In this case, the primary would successfully write the record to the file, but its backup process would get a "duplicate key" error when repeating the write. This problem is solved by having Guardian automatically generate an optional sequence number for disc file writes.

A part of the information copied to the backup process when a file is checkpointed is the sequence number for the next write to the file. When a write is done to a file that has been opened with this option, the sequence number passed by the file system is compared with the copy held by the disc process. If it is the same, then the operation is done and the status (error indication and transfer count) is returned to the application process and a copy is saved by the disc process. On the other hand, if sequence numbers do not agree, then no operation is done and a copy of the previous operation's status is returned. Using the previous example, the use of file sequence numbers is shown in Figure 7.

When a process-pair has a file open, any records locked in the file will be considered locked by the process-pair. When the primary fails, the backup may finish file modifications with locks still in effect, preserving the integrity of the data base.

While the primary process operates, the backup process receives the checkpoint information via a call to the procedure CHECKMONITOR. When the primary process sends a checkpoint message via a call to CHECKPOINT, this procedure moves checkpointed portions of the primary process's data space into the backup's data space and saves the latest file information.If a messge is directed to the backup process and the primary process still exists,it is rejected with an "ownership" error which informs the sender that the message is to be sent to the other member of the process-pair. Finally, when the primary process fails, CHECKMONITOR will transfer control to the correct restart point.

The Tandem implementation of COBOL provides a similar checkpointing facility. In each case, checkpointing is not an automatic operation. Careful attention during the application design phase will result in fewer checkpoints and will yield a checkpointing scheme that can be analyzed for correctness. Consideration must also be given to how the application will recover from failures occuring while a write operation is in progress to non-disc devices. Recovery when accessing a CRT terminal could be automatically done by rewriting the entire screen. Recovery while printing checks on a line printer would require some manual intervention and operator interaction with the application program.

ACTION	SEQUENCE VALUES AFTER ACTION:		
	PRIMARY	BACKUP	DISC PROCESS
CHECKPOINT (stackbase, ,infile, , outfile) sequence # of primary sent to backup	0	0	0
CALL WRITE (outfile, buffer, itemlen) sequence #'s match, operation is done, sequence #'s advanced	1	0	1
*** PRIMARY PROCESS FAILS, BACKUP TAKES OVER ***			
CALL WRITE (outfile, buffer, itemlen) sequence #'s don't match, operation is not done, backup's sequence # is advanced	—	1	1

Figure 7. File System Sequence Numbers

Application Structuring

The process, process-pair, and interprocess communication primitives of Guardian provide extremely general tools for application structuring. For example, consider an inquiry application such as hotel reservations. Requests come in from various types of terminals for reservations, cancellations, and hotel registration. Other requests come in for items such as management reports showing the number of rooms available at some hotel on some date. The application could be structured as follows:

Process-pairs will be defined for each type of terminal to handle actual terminal I/O (including any required line protocol and character conversion) and initial request verification. Each process-pair will be designed to handle some number of terminals. When a valid request has been received from a terminal, the terminal process-pair will route the message to the appropriate server process-pair.

Each server process-pair will be assigned a certain part of the application. In some cases, multiple copies of a server program will be run to allow multiple requests to be processed in parallel.

There are several advantages to this approach. First, the handling of terminals and processing of requests have been cleanly separated. New types of terminals can be added by simply adding a type of terminal control process-pair. New types of requests can be handled by adding another type of server process-pair. Likewise, software modifications and testing can be done on a modular basis. Finally, nowhere in this structure is there any requirement for a specific number of processors in the system or for the relative locations of processes. As the system load or the application changes, the number of processors, amount of memory, or physical location of processes may be changed without distrubing the application's internal structure.

VI. SOFTWARE RELIABILITY

When design of the operating system was started, we hoped to eliminate as much as possible the archetypal system crash. That is, once or twice a day, or week, the system crashes in a non-repeatable fashion. Our experience on an in-house system used primarily for software development and manual writing shows that we have achieved that goal. During a three-month period in the summer of 1977, a processor failed because of a software problem on two occasions. In each case, the problem was found at that time and the failure could be repeated by running a particular program.

I propose the following explanation of this reliability. First, the system was very carefully structured and much time was spent in initially specifying primitives. As experience was gained in trying to apply these primitives at higher levels, problems were found. This resulted in design changes at lower levels rather than "kludges" at a higher level. Implementation was also forced to stay within the designed structures because of the distributed nature of the hardware. If a problem could not be solved using processes interacting via messages, then it could not be "kludged" by turning off interrupts and changing some flag in memory. Given a single processor system, there is a very strong temptation to solve difficult problems in this manner.

Second, as the operating system and hardware were developed at the same time, another vendor's system was used to provide interactive text editing, a cross T/TAL compiler, a Tandem 16 processor simulator, and a downloader for the Tandem 16 prototypes. Implementation and checkout were not impeded by unreliable prototypes and as each level of the system was implemented, it could be extensively checked. These tools allowed initial implementation and checkout of all functions of the system through the level of the command interpreter. The wisdom of this approach can best be shown by the fact that when the first prototype processors were made available to the operating systems group, all operating system functions which ran on the simulator ran on the prototypes.

Third, debugging tools were built into the operating system from the start. A low-level interactive debugger was implemented which allowed breakpoints to be set at any level of the operating system, including interrupt handlers. Once this low-level debugger is entered in one processor, clocks in all other processors in the system are stopped so that they will not decide that the first processor is down. When the first processor continues, so will the rest of the system. A full maintenance panel only had to be used to track problems that managed to destroy the low-level debugger. Consistency checks were also coded into low-level routines. For example, before an element is inserted in a doubly-linked list, the list links of the element that the new element is being inserted behind are verified. These checks have proved to be extremely valuable in tracking problems or when implementing new features in the system. Even when extensive changes are being made to the system, it has the property that it will stop at one of these consistency checks very soon after something has gone wrong, allowing the problem to be rapidly found.

Fourth, formal testing was carried out at all levels of the system as they were implemented. A third person, whose only job was testing, was added to the initial project well before completion. By testing not just the external specifications of the system but also the underlying system primitives, it was assured that all system functions at all levels could be checked.

Finally, the primary design goal of the entire system was reliability. When the system design goals are clearly defined and understood by all involved, they can control implementation on a daily basis. Implied goals on the other hand are often forgotten when seemingly small decisions are made.

VII. CONCLUSIONS

The innovative aspects of Guardian lie not in any new concepts introduced, but rather in the synthesis of pre-existing ideas. Of particular note are the low-level abstractions, process and message. By using these, all processor boundaries can be hidden from both the application programs and most of the operating system. These initial abstractions are the key to the system's ability to tolerate failures. They also provide the configuration independence that is necessary in order for the system and applications to run over a wide range of system sizes.

Guardian provides the application programmer with extremely general approaches to process structuring, interprocess communication, and failure tolerance. Much has been said about structuring programs using multiple communicating processes, but few operating systems are able to support such structures.

Finally, the design goals of the system have been met to a large degree. Systems with between two and ten processors have been installed and are running on-line applications. They are recovering from failures and failures are being repaired on-line

Acknowledgements

An operating system is the work of many people. In particular I would like to acknowledge the contributions of Dennis McEvoy, Dave Hinders, Jerry Held, and Robert Shaw in its design, implementation, and testing.

REFERENCES

[1] *Katzman, J.A., "System Architecture for NonStop Computing," Compcon, February 1977, pp 77-80.*

[2] *Tandem Computers Inc., Tandem 16 System Description, 1976.*

[3] *Bell, C.G. and A. Newell, Computer Structures: Readings and Examples, McGraw-Hill, Inc. (1971), pp 15-36.*

[4] *Enslow, P.H., Jr., Multiprocessor Organization —a Survey, Computing Surveys 9 (March 1977), pp 103-129.*

[5] *Dijkstra, E.W., The Structure of the "THE" Multiprogramming System, Comm. ACM 11 (May 1968), pp 341-346.*

[6] *Brinch Hansen, P., The Nucleus of a Multi-programming System, Comm. ACM 13 (April 1970), pp 238-241, 250.*

[7] *Hewlett-Packard Journal, January 1973*

[8] *Thompson, K. and D.M. Ritchie, The UNIX Time-Sharing System, Comm. ACM 17 (July 1974), pp 365-375.*

BIOGRAPHY

Joel Bartlett received his M.S. degree in Computer Science: Computer Engineering and B.S. degree in Statistics at Stanford University in 1972. Before joining Tandem Computers in 1974, he was employed by Hewlett-Packard. The author is a member of the ACM.

Guardian/Expand: A NonStop™ Network*

James A. Katzman
H. Robert Taylor
Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, California 95014

ABSTRACT

This paper describes the features of the Expand NonStop Network system. Additionally, this paper explains how those features avoid pitfalls common to many other commercially available network architectures.

*Guardian, Expand, and NonStop are trademarks of Tandem Computers Incorporated.

INTRODUCTION

The Expand NonStop Network is unique because it is an extension of an existing network operating system. Every Tandem 16 computer system comprises from 2 to 16 separate central processors. Running in the NonStop mode requires constant communication among the processors. Consequently, the Guardian Operating System includes a sophisticated message handling system to control communications between processors and between processes (programs) running in one processor or running in separate processors. In effect, every Tandem 16 system is a local network controlled by the Guardian Operating System. The Expand NonStop Network expands the scope of the Guardian Operating System to allow communications among as many as 255 Tandem 16 systems.

The Guardian/Expand Network system, combined with the unique architecture of the Tandem 16 computer system, provides network users with a number of features unequalled by other computer vendors:

- NonStop Nodes

  The fault tolerant hardware and software of the Tandem 16 system eliminates the computer as a source of network failures [*Ref. 1*].

  NOTE: Individual Tandem 16 systems within the network are called "nodes" to distinguish the computer system from the network system.

- A Distributed System

  The Guardian/Expand Network makes it possible to configure a network of Tandem 16 fault tolerant computer systems so that a user of any node in the network can access the resources of any other node (processors, files, or physical devices) without regard for the physical location of the resource. To the user, the Guardian/Expand Network appears to be one large set of computer resources rather than a collection of separate systems.

- Dynamic Message Routing

  The Guardian/Expand Network constantly monitors the communications paths. When a transmission fails, the system retries until the transmission succeeds or until the system determines that the communications path has been broken. When the communications path has been broken, the Guardian/Expand system automatically reroutes the message via a different communications path.

- Best Path Message Routing

  The Guardian/Expand system monitors the communication lines and automatically selects the best path. The best path is the one that takes the least time. The system selects the fastest rather than the shortest path because this optimum end-to-end protocol can reduce communications costs. When a communications line fails, Guardian/Expand reroutes messages using the next fastest available path. When a new line is added, the system takes advantage of any new best paths created by the addition.

- Precisely Tailored Hardware

  Different nodes within a network typically have different computing requirements. The Tandem 16 system allows users to place exactly the right amount of computing power at each site. Even though the nodes within the network may range from a basic two-processor system to a sixteen-processor system with billions of bytes of on-line disc storage, the systems retain total compatibility of data, software, and application programs.

- Logical Growth

  The Tandem 16 architecture allows for incremental hardware expansion. A user can add memory, central processors, or peripheral devices as computing requirements grow. Similarly, the Guardian/Expand Network allows incremental expansion. Nodes can be added or removed and communications paths can be changed, all without reconfiguring existing systems.

  Notice that the Guardian/Expand Network can forestall the need for hardware expansion since the resources of every system in the network are accessible.

- Data Integrity

  The Guardian/Expand Network incorporates multiple safeguards to ensure that message packets are received correctly and that data cannot be lost in transmission.

- Human Engineering

  Because the Expand Network is an extension of the Guardian Operating System, the user interface to the network is through the Guardian command interpreter. For example, to run the text editor program on the local system, the user enters the command EDIT at his terminal. To run the program on a remote system, the user simply enters the symbolic name of the remote system before the command: \OHIO EDIT. In effect, this command connects the user terminal with the remote OHIO system and starts the text editor program on that system. The only difference the user may notice in running on that remote system rather than the local system is that response time may be slightly longer since the communication lines cannot match the performance of the local processor.

- Simple Programming Interface

  The Expand Network relieves programmers of the need to deal with a cumbersome telecommunication access method. Since programs communicate with each other via Guardian's message system, the programmer uses the same commands to communicate with a program in the same processor, another processor, or another system.

Full appreciation of the Expand Network system requires an understanding of how tightly the Expand Network system is integrated with the Guardian Operating System and the Tandem 16's architecture.

I. THE TANDEM 16 SYSTEM: A NETWORK IN A BOX

The Tandem 16 NonStop system is designed for the on-line, terminal and transaction, data base-oriented market where high availability is required [*Ref. 2*]. This requires a multiple processor system that can tolerate the failure of any single component, including even a central processor unit. Figure 1 illustrates the architecture of a typical Tandem 16 system.

Notice that at least two paths connect any two components in the system, and everything in the system, including even individual disc files, can be duplicated. Normally, all resources in the system function as independent, parallel resources. However, when a component fails, the remaining system components automatically take over the workload of the failed component. A system user at a terminal is typically unaware of the failure. Figure 2 illustrates a path switch resulting from the failure of an I/O channel.

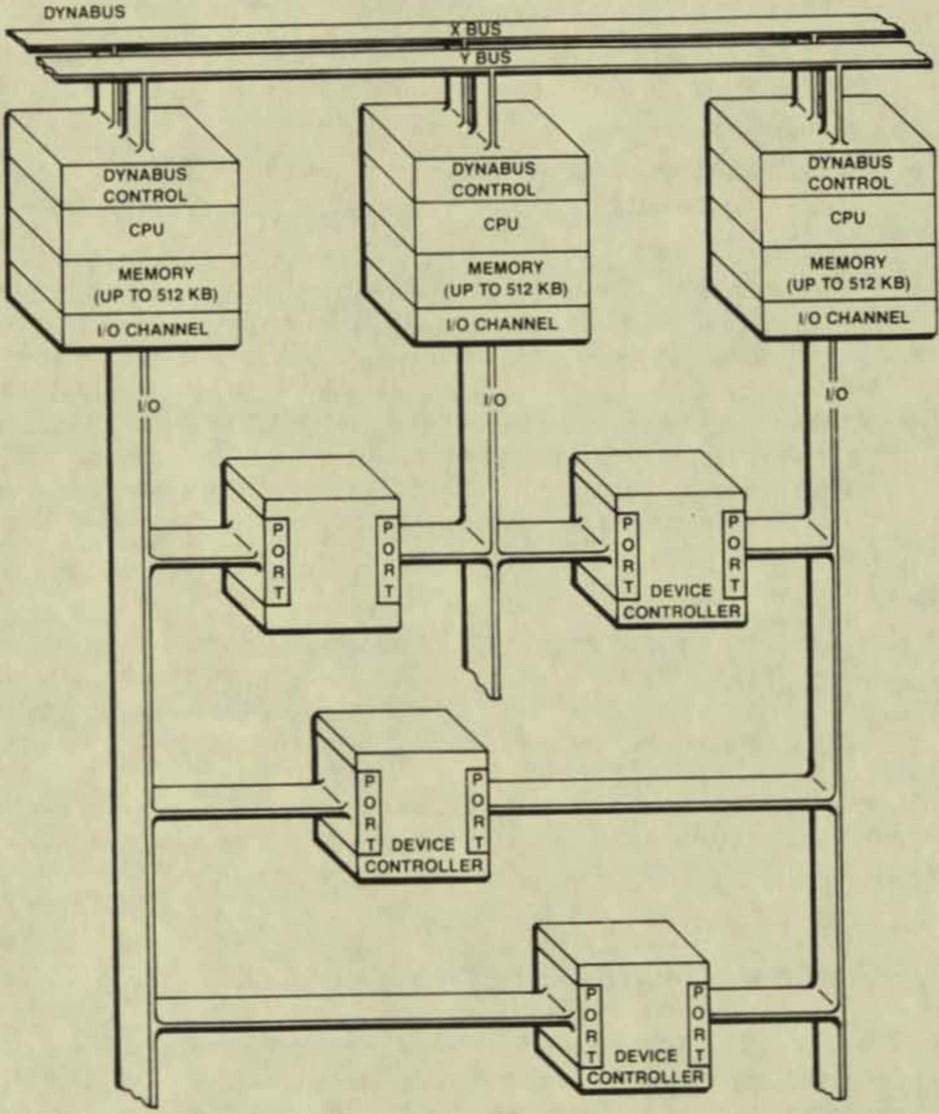
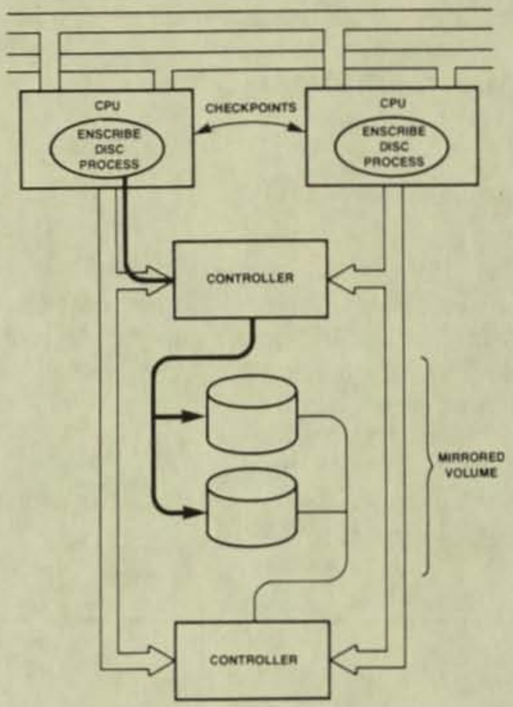


Figure 1. Typical Tandem/16 System

Typical Path



Path After Failure

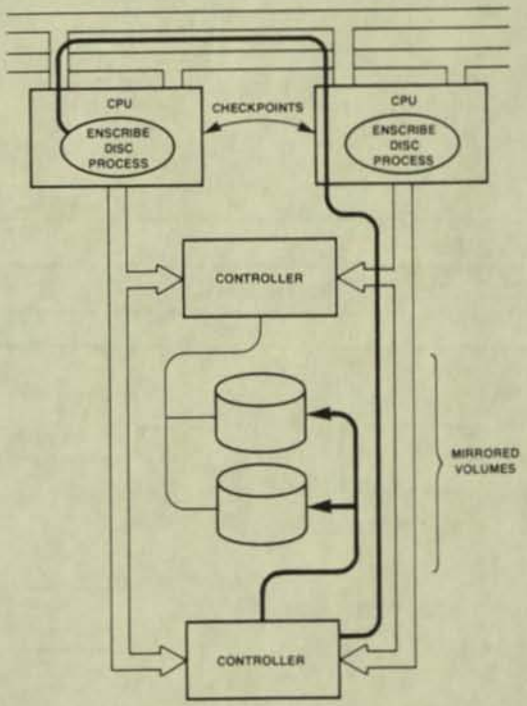


Figure 2. Path Switch on I/O Channel Failure

The Dynabus is used for interprocessor communication. Of course, the Dynabus is redundant, and each bus communicates at 13 megabytes per second. The regular I/O channels are block multiplexed and operate at 4 megabytes per second. Because of the speed of the Dynabus, the overhead for performing I/O operations via an indirect path as in Figure 2 is negligible.

Guardian Operating System

Guardian, the Tandem 16's general purpose multiprocessor operating system, manages the system's resources. The operating system resides in each processor in a system, but is aware of all the other processors. In fact, the operating systems in the different processors constantly monitor each other's performance. The instant one processor's operating system fails to respond correctly, the other processors assume that it is failing and take over its workload. Obviously, this requires a great deal of communication among the processors. Additionally, this requires a process to be able to address system resources by a logical name rather than by a physical address. The Guardian Operating System is designed in a top down manner with three levels and well defined interfaces as shown in Figure 3.

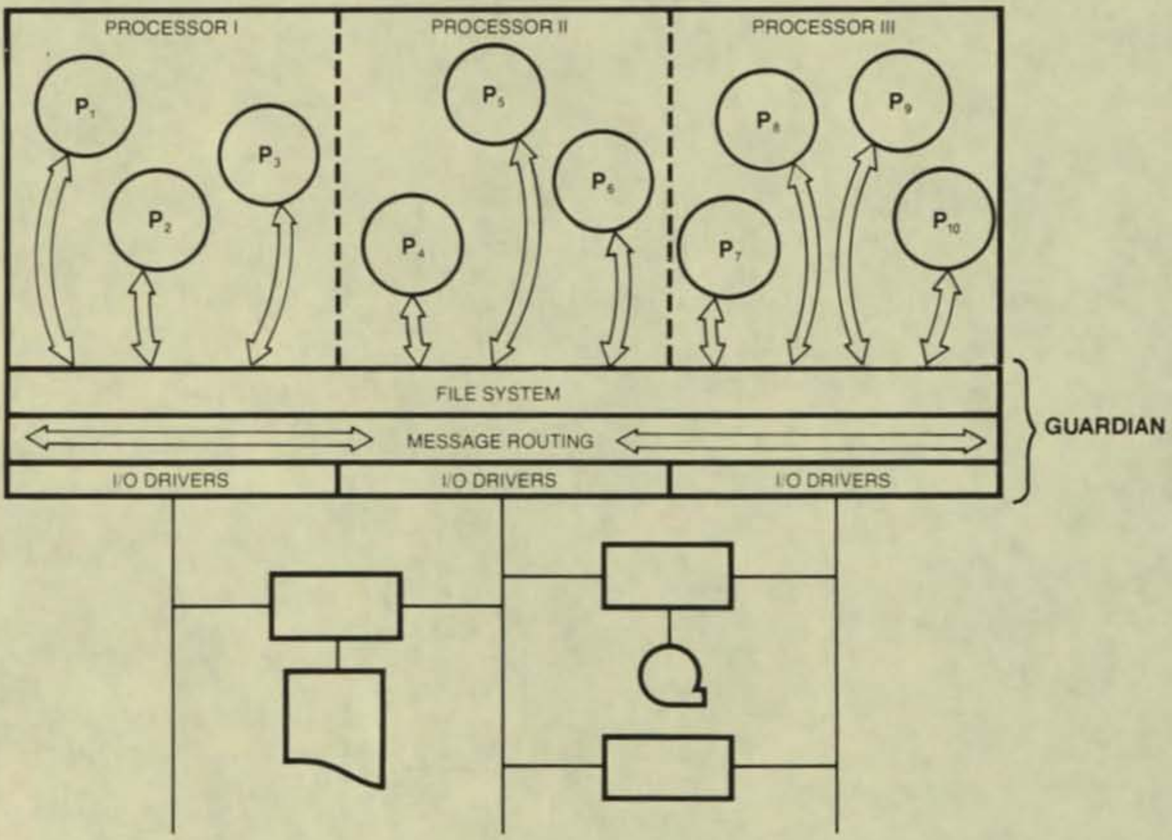


Figure 3. Guardian Operating System

The Guardian Operating System is based on the concept of processes sending messages to other processes. All resources in the system are considered to be files, and each resource has a logical file name. Communication between an application process, denoted by circles in Figure 3, and any other resource (disc, tape, another process, etc.) is via the file system. The file system knows only the logical name of the intended recipient of a message. The file system passes the message to the message system, which then determines the physical location of the recipient. The message system is a software analog of the Dynabus. The message system handles automatic path retries in case of path errors. Because application programs deal only with logical file names, the Guardian Operating System offers total geographic independence of resources. The programmer views this multiprocessor system as a single processor with resources available through file system calls.

Another advantage of geographic independence of physical resources is that it eliminates the need for reprogramming when hardware changes occur. For example, a user can add processors or I/O devices as the application load grows with time, but all existing programs can remain unchanged.

Data Integrity in the Tandem 16

The integrity of the data base in a Tandem 16 system is ensured by the Enscribe Data Base Record Manager. Enscribe uses a checkpointing scheme to perform data base updates in two separate processors. This scheme guarantees data base integrity even if a processor or I/O path failure occurs in the middle of an update. Optionally, two disc drives can be configured as identical copies of each other. Such discs are called a "mirrored pair." Further, each disc drive can be connected to a different controller. This technique maintains data integrity despite the loss of any single component, a processor, disc drive, disc controller, or power supply, even if the disc is in the middle of an update when the component fails. Thus, linkages are kept consistent and have high integrity within the system.

II. GUARDIAN/EXPAND: THE NETWORK EXTENSION TO THE MESSAGE SYSTEM

The Expand Network system extends the Guardian message system beyond the boundaries of a single system. The Guardian/Expand Network allows up to 255 separate Tandem 16 systems to be linked together. Conceptually, the Guardian/Expand Network system extends the Dynabus over miles or continents. To a user at a terminal, the entire network appears to be a single Tandem 16 system. (There may, of course, be a noticeable difference in performance since the Dynabus itself is more than 10,000 times faster than a typical "fast" 9600 baud private line.)

As an extension of the Guardian message system, the Expand Network maintains the geographic independence of resources. Any resource in the network can be addressed by its logical file name without regard for its physical location. However, a configuration option allows users to reserve processors for local processing requirements, thereby excluding those processors from the network.

III. THE COMPONENTS OF EXPAND

The major components of Expand are the End-to-End protocol, the Network Control Process, the Network Line Handler, the Network Routing Table, and the Network Utilities.

End-to-End Protocol

The End-to-End Protocol insures the integrity of messages from source to destination of a message regardless of how many intervening nodes and resources are utilized in the message transmission [*Ref. 3*]. The protocol is based upon the exchange of packets over a communications path. Expand supports communications paths of either a full duplex telephone line or a virtual circuit through an X.25 network.

Perhaps the greatest fear of network users is that the data base is down [*Ref. 4*]. The End-to-End Protocol makes sure that data transmissions are received correctly and don't get lost.

Network Control Process

The Network Control Process maintains the Network Routing Table, logs network changes, and services certain requests from remote nodes. Inquiries concerning remote resources are directed to the Network Control Process. Messages for all other processes are routed directly to the destination process. One Network Control Process resides in each node in the network.

Network Line Handler

The Network Line Handlers establish and maintain the communications path between two physically connected nodes. The line handlers also create a logical connections of nodes that are not neighbors by forwarding messages, thus implementing the End-to-End Protocol. When the X.25 network protocol is used, a line handler exists for each virtual circuit. These line handlers, in turn, communicate with a single X.25 access process that manages the physical line to the X.25 network.

Network Routing Table

Each Network Control Process builds and maintains its own Network Routing Table. The local Guardian message system uses the Network Routing Table to direct messages for other systems to the proper line handler. Line handlers also use the Network Routing Table to select the proper line handler when forwarding a packet. Network Routing Table maintenance is best explained with an example. Assume that a network is configured as in Figure 4. Notice that the normal route between nodes A and E is from A to B to C to E. The system uses this route because of the high speed link between nodes B and C. The failure of the link between B and C generates the following events:

1. The Network Control Process in node B informs its neighboring (directly connected) nodes (A and D) of the failure of the B to C communications line. Meanwhile, node C notifies its neighbor (node E) of the failure. Nodes D and E then notify each other of the change.

2. This change requires the establishment of a new communications path between nodes A and E (and others not described here for the sake of simplicity). Nodes A and E establish a new connnection by exchanging a number of control messages. Each node notifies its neighbor of the change in the shortest time distance to remote systems.

3. When the connection is established, each node updates its Network Routing Table so that its own packets are directed properly, and so that messages are forwarded over the fastest available path.

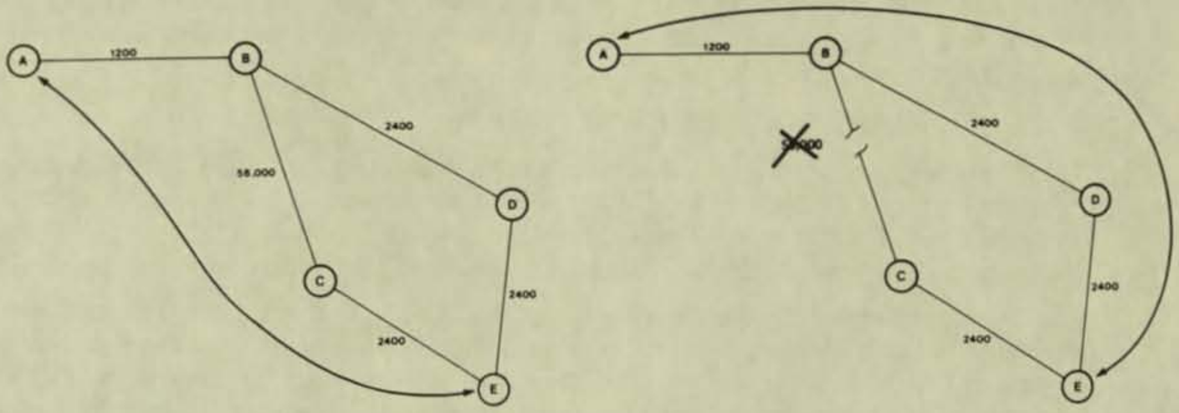


Figure 4. Network Routing

When a node updates its Network Routing Table, it checks to see if the change creates any new shorter (faster) communications paths. Therefore, the addition of a new line may generate a number of path changes. For example, restoring the path between nodes B and C in the sample network changes the path between nodes C and D. Messages are forwarded via node B rather than node E to take advantage of the faster line.

It is important to point out the Network Control Process and all Line Handlers run as NonStop process-pairs. This exclusive feature ensures a degree of reliability possible only in a fully redundant system such as the Tandem 16 computer.

Network Utilities

The Network Utility programs aid the user in monitoring the status of the network and in tracing problems. The tracing facility, NETRACE, creates a trace file that tracks the events that occur within a specific Line Handler. A companion utility, NETDUMP, formats this trace data for report generation.

The NETSTATS program displays accumulated statistics for a given line handler and optionally resets them. The statistics are reported by protocol level and include counts of the number of frames of a given type that have been sent or received, and the number of requests that have been initiated or processed by the line handler.

NETMAPS provides a facility to display the entire network status as viewed from a single system. It gives the current status of each remote system and the status and routing of each communications path to each system.

NETPROBE traces, system by system, the physical route used between two systems and gives the elapsed time required to traverse the route.

The user can run these utilities from any terminal in the network, and their output can be directed to any terminal or other output device in the network. Therefore, a user can check the performance of any system in the network.

Network Logging Center

As an option, the user can designate one node of the network as the Network Logging Center. When this is done, the status messages normally logged at each individual node are forwarded to the logging center.

III. AVOIDING THE PITFALLS OF NETWORKS

Understanding the evolution of networks is helpful in avoiding the pitfalls of many of the network systems currently available commercially from computer venders. The concept of creating an interlinked network of computers results from two different approaches to solving the problems of organizations with geographically scattered requirements for computing resources.

The first approach simply places a computer at each site that can justify the cost, a common technique in batch oriented environments. This approach tends to isolate the end user from the computing resources. Also, the approach makes it difficult for the organization to exchage information easily among the computer installations. Sharing data among installations typically involves shuttling about magnetic tapes or even decks of punched cards. As a result, the organization's data base is seldom completely accurate. Clearly, it would be an advantage if the computers could communicate directly with one another.

The second approach places one large host computer at a central location with terminals at remote locations. This has the advantage of placing the organization's data on-line and allows non EDP personnel greater access to the computing resource. However, this approach ignores the practical economics of the computer industry. In the past two decades, computer hardware has become increasingly less expensive while communications costs have remained relatively high. A more serious disadvantage of this approach is that a failure of the host computer can cripple the organization's ability to conduct its business [*Ref. 5*]. Clearly, it would be an advantage if some computing power could be located at the remote terminal sites.

Both approaches lead to network systems. In the first case, existing computer installations were made to communicate with each other. In the second case, some computing responsibilities were moved from the host computer to the remote locations. Eventually, the so called "80/20" rule emerged. This rule states that when 80% of the processing against a partition of a data base is performed on behalf of a remote site, it is most cost effective to move that partition to a comptuer system at the remote site in a computer network configuration [*Ref. 6*]. This way only about 20% of the system transactions require communication facilities.

Establishing a network is an attempt to meet one or more of the following objectives:

- Distribute data bases (put the work close to the end user)
- Share resources
- Reduce communications costs
- Increase performance and decrease response time by partitioning tasks
- Achieve higher system availability

Past Approaches

Several computer manufacturers have implemented and market network systems [*Ref. 7*]. Typically, these systems reveal their heritage in the evolutionary approaches to networks. For example, most network operating systems have been imposed "on top" of existing operating systems. This gives rise to network "architectures" that have no relationship to the systems they serve and to cumbersome access methods.

In systems where computer systems have replaced terminals, the concept of a host computer with a centralized data base generally remains. Unfortunately, the problems caused by a failure of the host computer also remain. If the host suffers intermittent failures, data integrity can be lost throughout the network, not just in the host's data base.

In general, commercial networks have linked the nodes of a network system in a point-to-point manner as shown in Figure 5.

In the network shown in Figure 5, all nodes can communicate with each other since each is connected to all other nodes by communications lines. While this is convenient, it is also very costly. By contrast, Figure 6 shows a typical Guardian/Expand network.

The Guardian/Expand Network reduces communications costs because its automatic message forwarding eliminates the need for point-to-point interconnection of the network. Most other commercially available networks require the user to write special applications programs to relay messages from node to node.

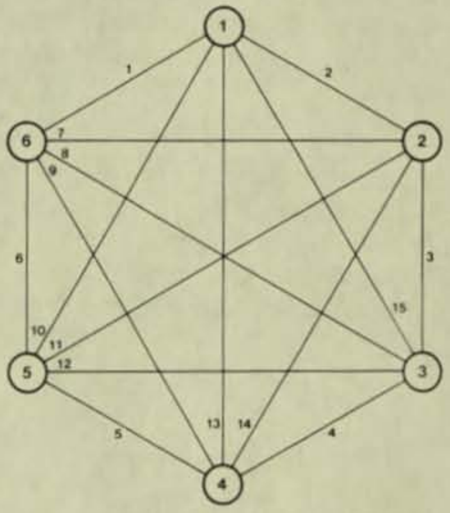


Figure 5. A Typical Network

Recognizing the Risks of Distributed Processing

Several risks face the user contemplating a network of computing systems. However, these risks can be avoided if they are recognized in time.

- System Availability

  Networks that rely on a host computer or that have a master/slave hierarchy can never be more reliable than the central computer.

- System Compatibility

  Selecting the computer systems for each node of a network requires consideration of factors other than just the hardware costs. Considering only hardware costs, it may make sense to install different kinds of computers at different sites. But in a network, costs increase as the number of vendors increase. Different programming groups are required for the different processors. Incompatibilities in data types, operating systems, and programming languages must be resolved. Making a relatively simple change at the network level may involve extensive reprogramming requiring extensive coordination.

- Expansion Capabilities

  System expansion must be carefully planned, as mistakes here can be very expensive. The user should plan for both the expansion of the network and the hardware expansion of individual systems. In regard to the latter, it should be noted that even with a single vendor different sizes of computers often used different, incompatible operating systems and access methods. Thus, outgrowing a particular processor may involve extensive reprogramming. Reprogramming without gaining additional system capabilities is the most expensive kind of programming.

- Data Integrity

  The safeguards for data integrity offered by a network vendor should be examined closely. Data bas integrity at each site is just as important as accurate data transmission throughout the network. A single corrupted data base can quickly contaminate the entire network, not because the data is duplicated, but because the data at one site is typically based on data from other sites. In a 20-node system where each node interacts with four other nodes each minute, the entire network can be corrupted in less than three minutes.

- Network Support Software

  Vendor supplied support programs are an important part of a network. Tracing an error back to its source can be difficult since the error may pass through a number of nodes before it is detected. The network should include a method for isolating errors anywhere within the network from any given node. More typically, error analysis functions are centralized in a host or master computer, which leaves the entire network vulnerable to a failure in the host computer.

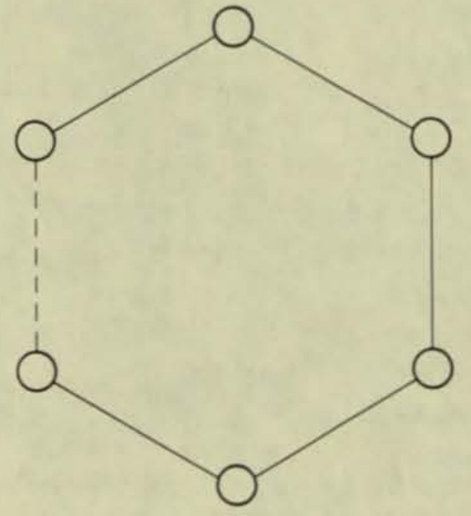


Figure 6. A Typical Guardian/Expand Network

REFERENCES

[1] *Katzman, J.A. "A Fault Tolerant Computer System," Proceedings of the 1978 Hawaii International Conference on System Sciences, 1978.*

[2] *Tandem Computers Incorporated, Systems Introduction, 1975.*

[3] *Tajibnapis, William D. "A Correctness Proof of a Topology Information Maintenance Protocol for a Distributed Computer Network," Communications of the ACM, July 1977, Association for Computing Machinery.*

[4] *Dolotta, T.A., M.I. Bernstein, R.S. Dickson, Jr., N.A. France, B.A. Rosenblatt, D.M. Smith, and T.B. Steel, Jr., Data Processing in 1980–1985, John Wiley & Sons, 1976.*

[5] *Auerbach Staff, "What is Network Architecture?," Computer Decisions, June 1976, Hayden Publishing Co.*

[6] *Hunter, John J., "Distributing a Database" Computer Decisions, June 1976, Hayden Publishing Co.*

[7] *International Data Corporation, "White Paper: Distributed Processing/Data Communications," Fortune Magazine, March 1977, Clifford Crum Publisher.*

Summary

The Guardian Operating System is perhaps the only operating system designed as a network system. The multi-processor design of the Tandem 16 system necessarily requires an operating system structured as a local network-in-a-box. In effect, the Expand NonStop Network simply eliminates the box.

DESIGNER'S OVERVIEW OF TRANSACTION PROCESSING

Lloyd Smith
Tandem Computers
19333 Vallco Parkway
Cupertino, California 95014

Abstract

This paper presents information for the individual responsible for designing a transaction oriented system. It covers the major design considerations that should be taken into account. The paper is divided into the following topics:

I. Introduction

II. A Transaction Processing System Model

III. System Control

IV. Implementation Considerations

V. TANDEM's Transaction Processing System

I. Introduction

One of the important points in any design, and the most important in a transaction environment, is that the system as a whole should be viewed as a "SERVICE". In any service organization the first goal is to find a service that people need. The second goal is to offer a service that people can depend on, and the third goal is to respond to the changing demands of those who use the service. If all three goals are not established from the beginning the success of

NOTE: TANDEM, GUARDIAN, EXPAND, NonStop, and PATHWAY are trademarks of Tandem Computers Incorporated.

the service may be negligible. The one goal most neglected in the past has been responding to change. The combination of user needs, building a reliable system, and the ability to react to change quickly are explored in this paper. The intent of this paper is to offer a better understanding of transaction processing and suggest general guidelines for successful implementations in the future, by examining these three points.

The first design consideration must be to fulfill a user defined need. This need or service is referred to as a "USER FUNCTION". Once a function is offered, a user gains confidence in the service based on reliability and the system's responsiveness to change as the needs of the business change. This ability to change and evolve is referred to as "REACTION TIME". The following design considerations have a direct impact on reaction time.

INSTALLABILITY:

A system can succeed only if it can be installed in a reasonable amount of time. Ease of installation also applies to any major enhancements or user functions.

FLEXIBILITY:

Determines how well the system reacts to change. If a system is designed properly with change in mind, changes can be applied quickly. Reaction time is reduced significantly.

EXPANDABILITY:

Allows the system to accommodate new users and new functions. After a system's initial success, two events typically occur:

1. More users want to use the system. A well-designed system must incorporate the ability to handle an increase in the number of users.
2. New functions are requested. The ability to handle new user functions without affecting the current user functions must also be considered.

MAINTAINABILITY:

The ability to correct problems and "tune" a running application. Too often, this consideration is overlooked in the original design. The problems that occur not only affect the existing functions but future functions as well. If significant resources are required to maintain existing functions, the ability to provide changes and enhancements is reduced. The total reaction time increases and the probability of overall success decreases.

RELIABILITY:

To ensure "service" to the user, the system must be reliable. The best implementations fail if the user cannot access his system. In addition to being constantly available, the system must ensure the integrity of its data base.

Note: Performance considerations are important; however, the above mentioned considerations should not be sacrificed for efficiency. The key to performance is through a good design.

II. A Transaction Processing System Model

Figure 1 shows the components of a typical transaction oriented system. A video display unit provides the human interface to the system. The remainder of the diagram describes the software components of a computer system including a Data Base stored on a direct access device external to the computer. Notice that the flow of control in this diagram is bi-directional. Notice also that operator request may not need the services of all five components. For example, if an operator enters non-numeric data into an entry field defined as numeric data only, the terminal I/O and the field validation routines are the only two components exercised. The request for a new function might involve the display of a new format. In this case four components are exercised: terminal I/O, field validation, data mapping, and transaction control.

This diagram is used as the foundation for the design of a transaction oriented system. The components describe the functional activities that are common to all transaction oriented applications. A brief definition of each component follows the diagram.

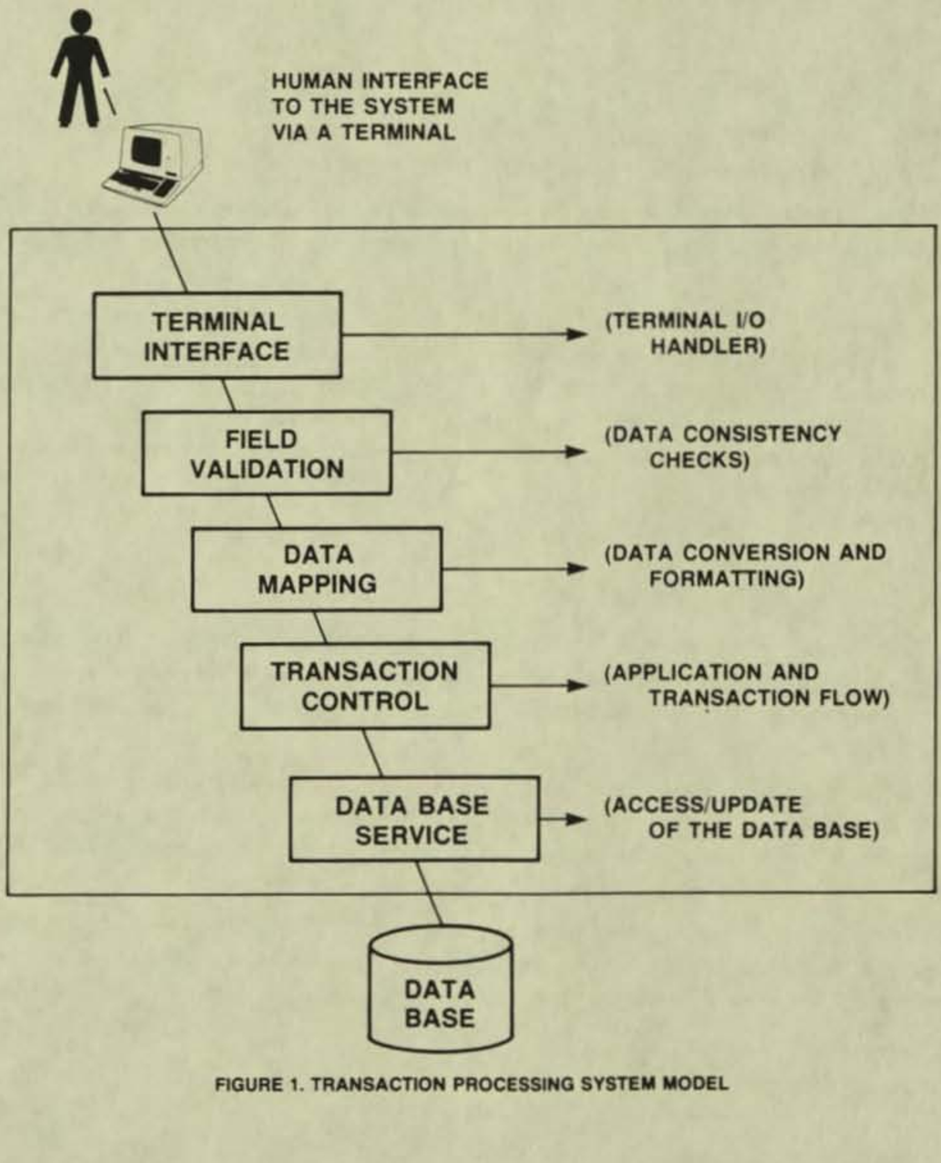

HUMAN INTERFACE TO THE SYSTEM VIA A TERMINAL
TERMINAL INTERFACE
(TERMINAL I/O HANDLER)
FIELD VALIDATION
(DATA CONSISTENCY CHECKS)
DATA MAPPING
(DATA CONVERSION AND FORMATTING)
TRANSACTION CONTROL
(APPLICATION AND TRANSACTION FLOW)
DATA BASE SERVICE
(ACCESS/UPDATE OF THE DATA BASE)
DATA BASE

FIGURE 1. TRANSACTION PROCESSING SYSTEM MODEL

Major Program Components

1. Terminal Interface

The terminal interface is responsible for the following general functions:

- All physical terminal I/O
- Control of device-dependent characteristics

Because the physical I/O to various terminal types often involves different protocols, it is advantageous to isolate the code that actually communicates with terminals into one place. This approach enhances the capability to test and install new terminal types, isolate and fix problems, and easily take advantage of new features that may become available on existing terminal types.

The data transmitted by the physical I/O also requires different handling by different terminal types, e.g., the codes to delimit a field may differ.

2. Field Validation

The field validation facility is responsible for data consistency on a field-by-field level. Normally these edits are defined when the screen format is built.

Field validations should be applied as close to the actual operator as possible. Notification of a problem with a data field entered should be timely, and its impact on the running system should be kept to a minimum.

The mechanism by which field edits are defined should be independent of physical terminal type. There should be one consistent way to define a field as numeric data only, a field that must be entered, or a range of acceptable values for a particular field. By separating the type of edit from the physical mechanism of applying the edits, which are defined at the terminal interface, terminal types can be added to the system without altering the logical view of entry fields and their associated edits.

3. Data Mapping

This facility is responsible for the conversion and formatting of data from an external to an internal representation and back again. By developing application tasks which refer to the data in its internal form the external characteristics of a system may change without affecting existing applications.

Data mapping is the key to writing terminal independent applications which process requests with no concern as to how that request was actually constructed. Therefore, whether requests are stored in a batch file on disc or entered through a video display should have no bearing on the application design below this point.

The data mapping facility should be at the symbolic field name level. This allows the ordering of fields to change on a particular screen display; however, the order in which the fields appear in the logical request will remain the same.

4. Transaction Control

This facility is responsible for the overall application flow. It is analogous to the top level implementation of a structured program which:

- Initiates all logical terminal I/O;
- Interprets and validates the request received from the data mapping facility;
- If Data Base access is required to service the request, the appropriate data along with the proper control information is passed to the appropriate Data Base routine; and
- Interprets and validates the Data Base routine replies.

The main function of transaction control is application flow, along with the routing of requests to one or more Data Base routines. It is a relatively small portion of actual application code; however, it is the heart of any application. The major benefits of this approach are:

- Since the actual amount of code necessary to control any one function is relatively small, it is a simple task to add and change user functions;
- Since the approach is structured in a modular fashion, new functions can be integrated and tested easily as part of the whole application; and
- Application flow and control can be tested as a separate piece of the total application and therefore have little or no impact on the Data Base services.

5. Data Base Service

This facility is responsible for all activity on a Data Base. This is normally a very important application function because it alters the state of the Data Base.

A Data Base service routine should be written using the simplest approach possible. The most straightforward and simple approach contains the following components:

- Get a request from a transaction control facility:

  This is the point of entry for a Data Base service. Getting a request from the transaction control facility is similar to reading a record from a disc file,
- Access the Data Base:

  The request may be for a read, write, update, delete or any combination of the four. The specified requests are applied against the Data Base.
- Build a reply based on the results of the Data Base access:

  The reply could contain actual data from the Data Base, control information describing any error condition that occurred, or any combination of the two.
- Reply to the transaction control facility:

  This is the exit point of the Data Base service. Replying to the transaction control facility is similar to writing a record to a disc file.

Moreover, each Data Base service should process requests uniformly from one or more user functions. By doing so the Data Base service will be independent of any particular user function, and the service can be viewed as a general utility, accessible by any user function. This eliminates redundant code and simplifies the implementation of new user functions requiring Data Base services already established.

The Data Base service must be written in a context free environment. The Data Base service should not be responsible for the retention of data between requests. Once a request is received, the Data Base service should be able to process the complete request and then forget about it. This approach simplifies the code significantly and creates an environment that is easy to understand and maintain.

One critical part of any transaction oriented system is input validation. There are three major types of input validation:

TYPE OF VALIDATION	EXAMPLE
FIELD	NUMERIC MUST FILL RANGE 100 THRU 199
REQUEST	INTER-FIELD RELATIONSHIPS (IF PAYMENT-METHOD = "CASH" THE AMOUNT ENCLOSED MUST BE > 0) (IF CITY WAS ENTERED STATE AND ZIP CODE MUST ALSO BE ENTERED)
DATA BASE	DOES ACCOUNT # 12345 EXIST IN THE DATA BASE? DOES THIS SALES TRANSACTION IN THE AMOUNT \$245.00 FOR ACCOUNT # 12345 EXCEED THE ESTABLISHED CREDIT LIMIT?

Figure 2 shows each level of edit within the transaction processing system model.

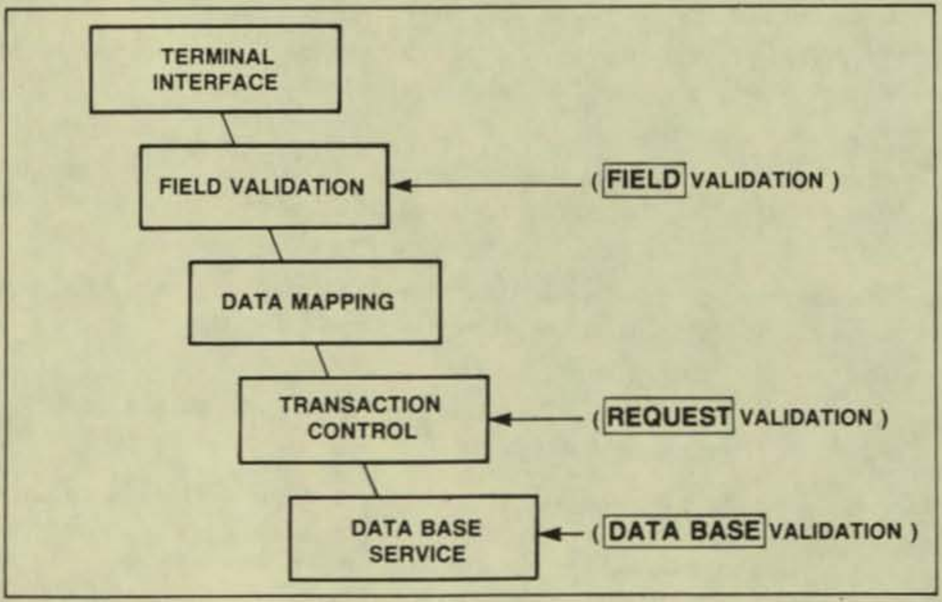


FIGURE 2. EDIT LEVELS

At this point the model is defined. The following illustrations show the use of the model in an application environment.

The internal components of the transaction processing system model can be grouped into the following categories:

Request oriented:
Components 1 through 4 have the combined responsibility for gathering, interpreting and responding to requests. From this point on the combination of components 1 through 4 will be referred to as the "REQUESTOR" portion of our model.

Service oriented:
Component 5, the Data Base services, are written as general utility functions accessible by any user function within a REQUESTOR. The Data Base services will be referred to as "SERVERS".

Figure 3 shows the REQUESTOR/SERVER relationships:

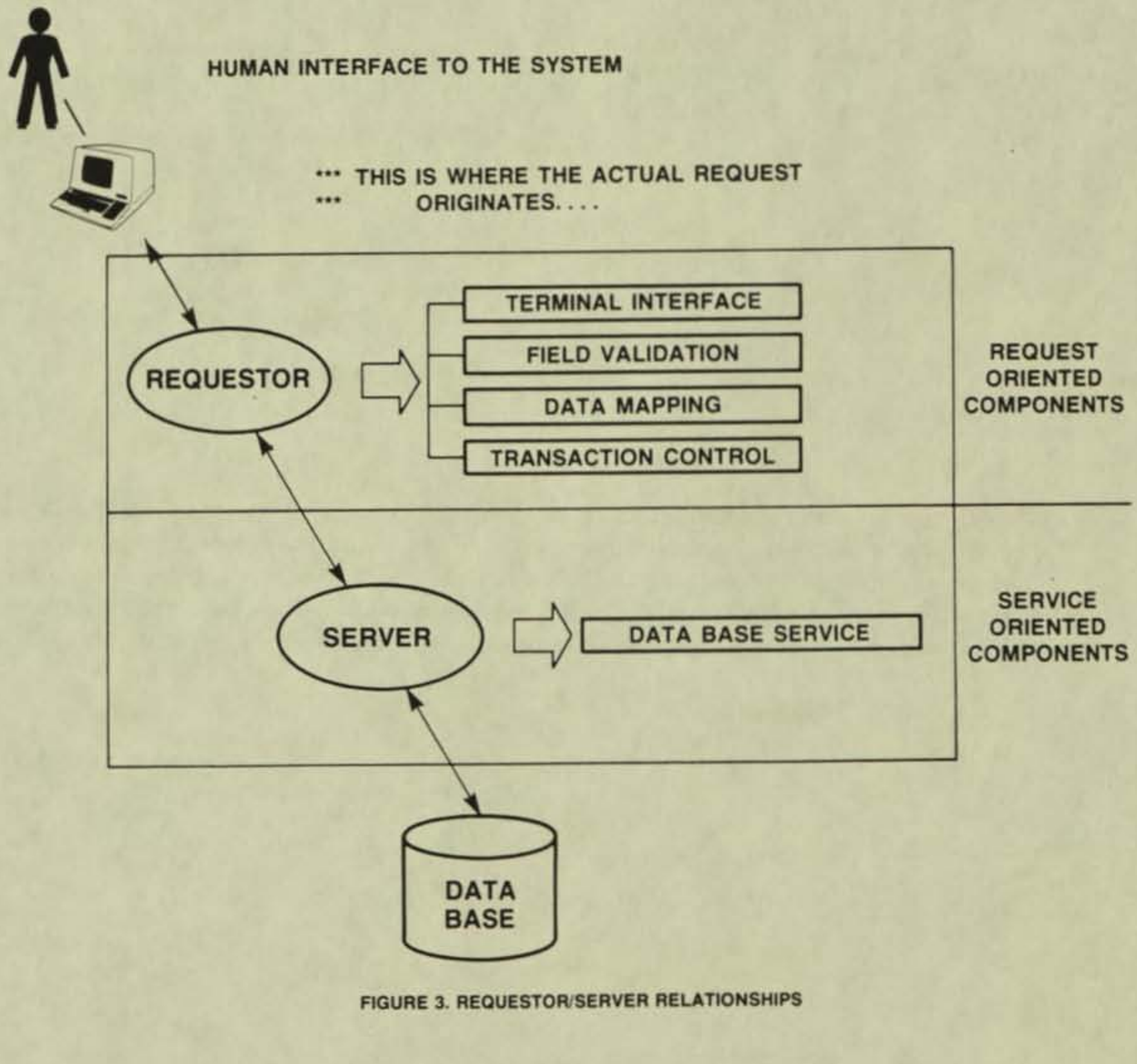


FIGURE 3. REQUESTOR/SERVER RELATIONSHIPS

Figure 4 illustrates an order entry application with three basic functions: credit checking a customer, adding a new order, and updating an existing order.

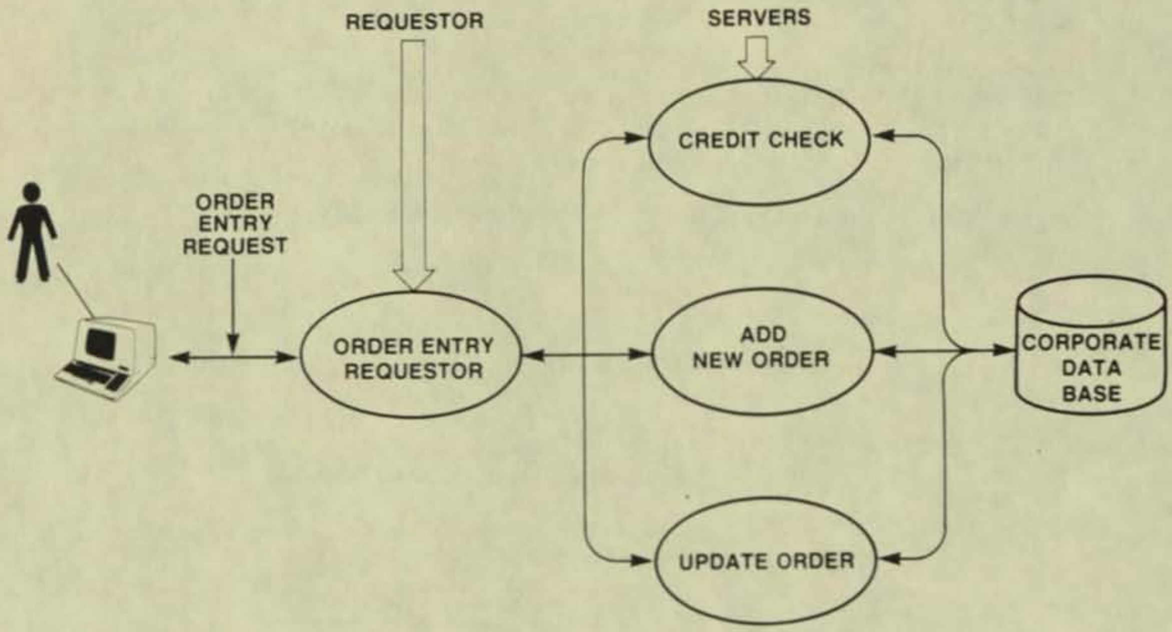


FIGURE 4. ORDER ENTRY APPLICATION

This example illustrates a terminal operator building an order entry request and sending it to the system. It also shows the relationship established between the requestor, who is responsible for the overall application control, and the Data Base servers, which are responsible for all Data Base activity. The system is partitioned into small modular components that are easy to define, write, debug and enhance. Because of its modularity, the system is extremely easy to maintain.

Figure 5 shows the relationships established between multiple applications running under the control of one transaction processing system.

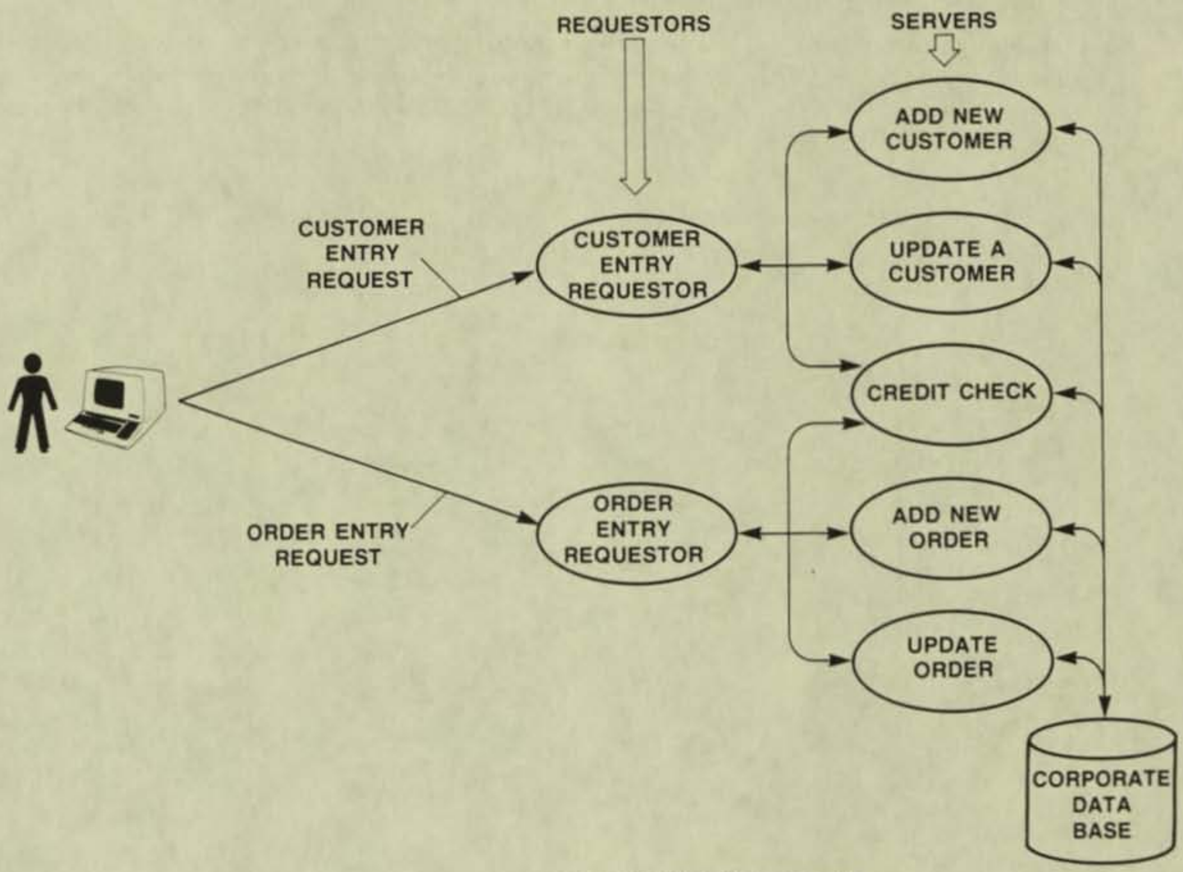


FIGURE 5. MULTIPLE APPLICATIONS IN A SINGLE SYSTEM

The above example illustrates the following points:

- Multiple application requestors can exist within a transaction processing system.
- Each terminal operator should have access to the number of application requestors needed.
- Data Base Servers may be shared among requestor applications.

III. System Control

Taking advantage of modular design techniques improves the overall quality of the system. However, as the components of a system are divided into smaller, more manageable segments, the problem of overall control becomes increasingly difficult.

The solution to this problem is to create a command center that has a global view of the entire application and, therefore, can create, delete and monitor the total application environment. This facility will be referred to as the APPLICATION MONITOR. Figure 6 illustrates the application monitor's view of the system:

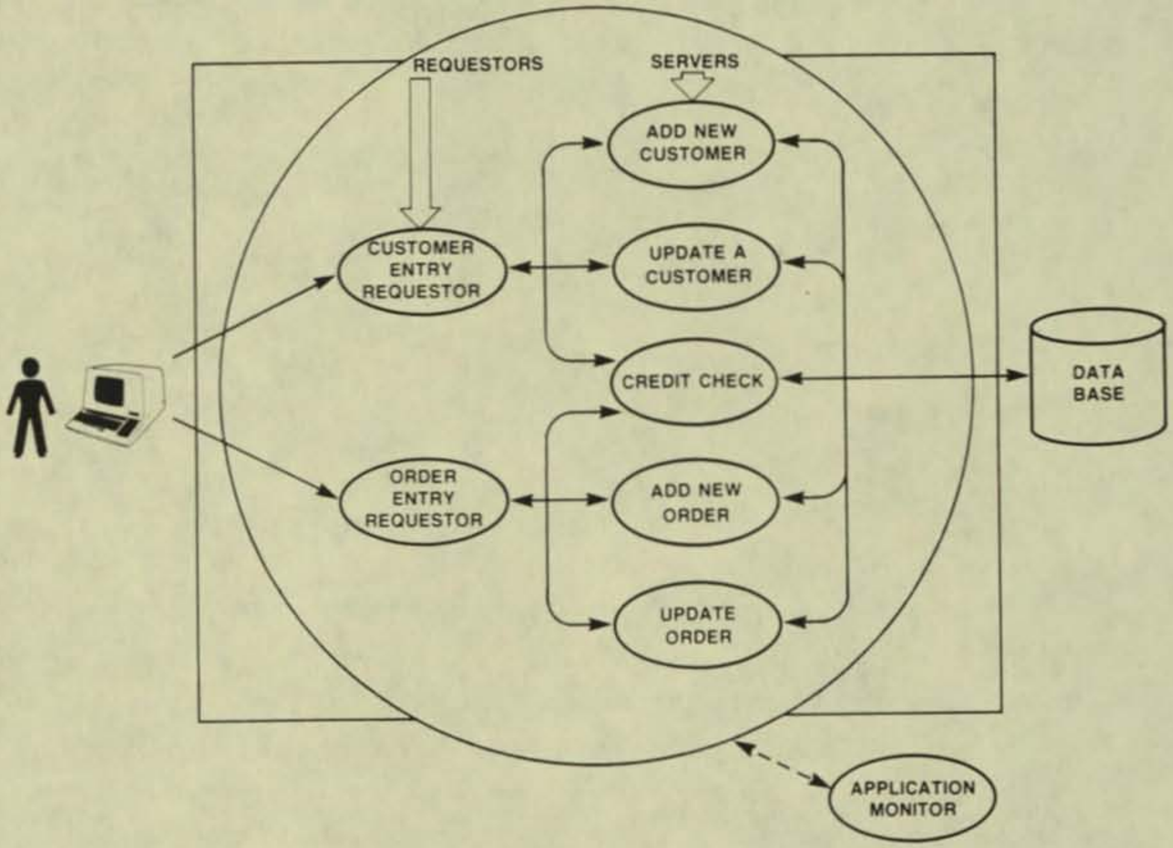


FIGURE 6. SYSTEM CONTROLLED BY APPLICATION MONITOR

The transaction processing system model established previously can be viewed in the following way:

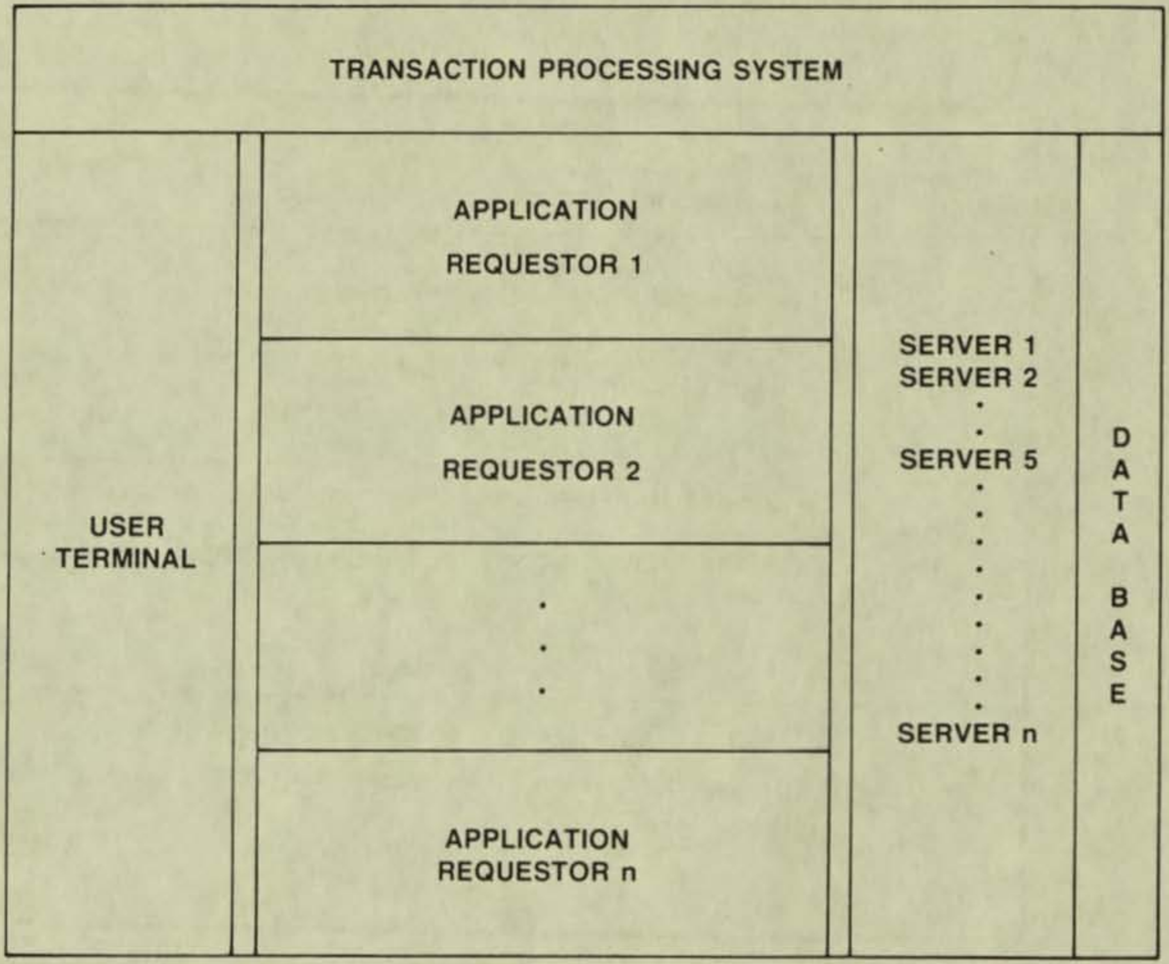


FIGURE 7. TRANSACTION PROCESSING SYSTEM OVERVIEW

The application monitor's view of a transaction processing system is analogous to the view a system operator might have at his console. The system operator controls the hardware environment and the application monitor controls the application environment.

The application monitor should maintain control over the following functions within a transaction processing system:

- Overall Transaction Processing System
- Each Application REQUESTOR
- Each Data Base SERVER
- Each Operator TERMINAL

IV. Implementation Considerations

Once the system is designed, its timely implementation, plus its ability to change as the needs of the business change, will be the deciding factors that determine its ultimate success or failure. Each system may be composed of many applications and each application will require the following components:

APPLICATION	
REQUESTOR	• **TERMINAL INTERFACE**
	• **FIELD VALIDATION** **(ONE COMPONENT OF THE SCREEN DEFINITION)** **COMPONENTS OF A SCREEN DEFINITION:** 1. INFORMATIONAL DATA FOR PROMPTS (PROTECTED) 2. DATA ENTRY FIELDS (UNPROTECTED) 3. INITIAL ENTRY FIELD VALUES 4. ENTRY FIELD VALIDATION SPECIFICATIONS
	• **DATA MAPPING (TO AND FROM THE SCREENS AND MEMORY)**
	• **TRANSACTION CONTROL WHICH IS RESPONSIBLE FOR:** 1. INITIATING ALL LOGICAL TERMINAL I/O 2. INTERPRETING AND VALIDATING REQUESTS 3. ROUTING REQUESTS TO THE PROPER SERVER 4. INTERPRETING AND VALIDATING REPLIES FROM THE SERVER
SERVER(S)	• **DATA BASE SERVICE WHICH IS REPONSIBLE FOR:** 1. ACCEPTING AND INTERPRETING REQUESTOR MESSAGES 2. ANY DATA BASE ACTIVITY: (READ, WRITE, REWRITE OR DELETE) 3. BUILDING A REPLY BASED ON THE SUCCESS OR FAILURE OF THE DATA BASE ACTIVITY 4. REPLYING TO A REQUESTOR

FIGURE 8. FUNCTIONS OF REQUESTORS VERSUS SERVERS WITHIN AN APPLICATION

Requestor Procedures

Because the requestor provides the logic that communicates with the end user, it must be the most flexible component of the system. A means of designing, changing and deleting screen formats is essential. Internal record formats produced through data mapping should be kept and maintained in a data definition library similar to those libraries associated with record definitions on a Data Base Management System. The actual transaction control might be written in a procedural language that is easy to use but flexible enough to handle total application flow.

All the above facilities should be maintained in a library accessible at run time. This allows smooth integration of each function within a requestor. It also allows modular expansion of functions within an application with little or no impact on current running functions.

This approach to implementing an application requestor ensures the reaction time necessary to effectively handle user demand. Moreover, it allows the system to expand in small, well-controlled increments, thus increasing the integrity of the overall system.

Server Procedures

The back-end component of any application is the Data Base server. Back-end functions must be handled with care, for they maintain the most critical aspect of any application, the Data Base. One of the principal advantages of this design concept is that it greatly simplifies the implementation. Server procedures can be designed and tested in the familiar — read a record, update the Data Base and Write a reply — fasion. Input transactions can be read from a disc file or magnetic tape. Test Data Bases can be created for the purposes of testing, and server procedure testing can take place independent of the overall application implementation.

The following diagram illustrates the two step integration of any Data Base server:

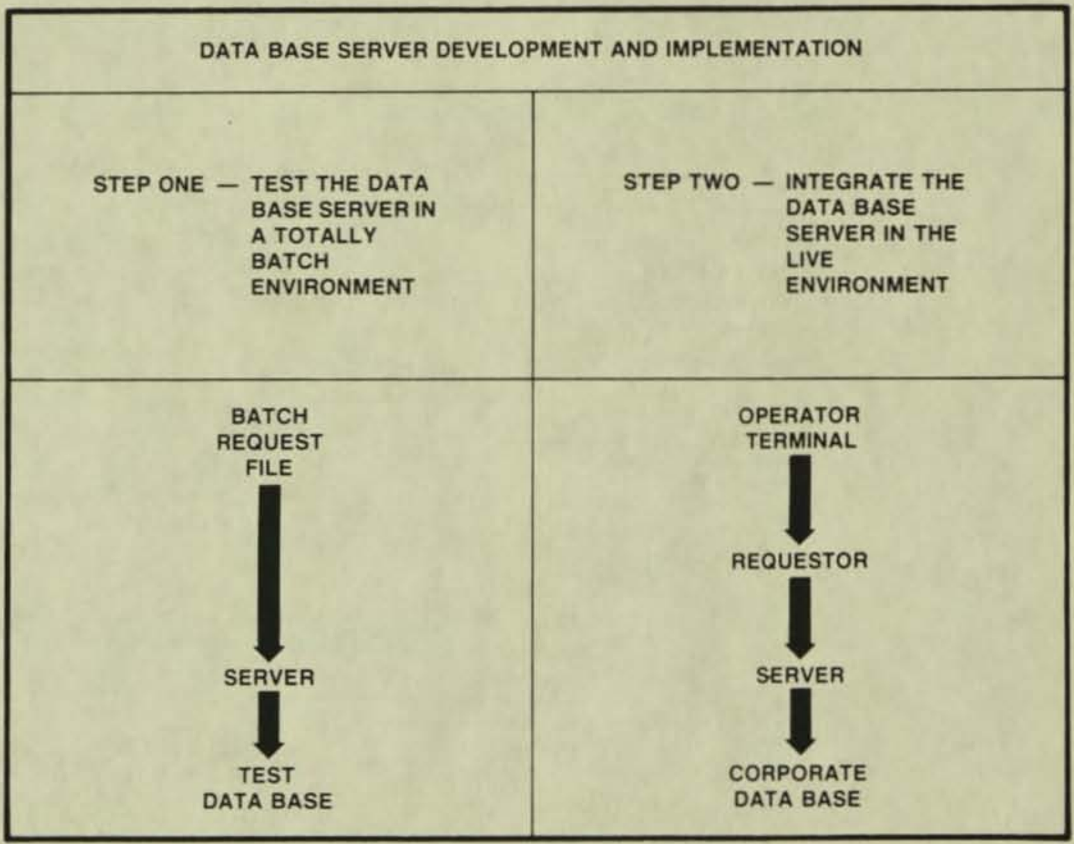

DATA BASE SERVER DEVELOPMENT AND IMPLEMENTATION
STEP ONE — TEST THE DATA BASE SERVER IN A TOTALLY BATCH ENVIRONMENT
STEP TWO — INTEGRATE THE DATA BASE SERVER IN THE LIVE ENVIRONMENT
BATCH REQUEST FILE
SERVER
TEST DATA BASE
OPERATOR TERMINAL
REQUESTOR
SERVER
CORPORATE DATA BASE

FIGURE 9. INSTALLING A DATA BASE SERVER

The integration of both REQUESTORS and SERVERS into the live system should be handled via the APPLICATION MONITOR. The application monitor should be able to logically start and stop any component within the system.

V. The TANDEM Transaction Processing System

TANDEM offers a total environment for transaction processing. The GUARDIAN OPERATING SYSTEM was specifically designed with NonStop transaction processing in mind. The FILE SYSTEM within Guardian allows separately running processes (in the same CPU, different CPUs within a single system or different systems with an EXPAND network) to communicate with each other at a simple READ/WRITE level. Guardian allows one logical computer system to incorporate up to 16 processors. The EXPAND network allows the interconnection of as many as 255 logical systems within a network and still maintains the simple READ/WRITE level communications between application processes. With this as a base, TANDEM has introduced a new product called PATHWAY. PATHWAY allows a user to take advantage of the unique TANDEM architecture, and it significantly reduces the time necessary to develop a transaction processing system. The functions enclosed within the inner box are addressed by the PATHWAY product.

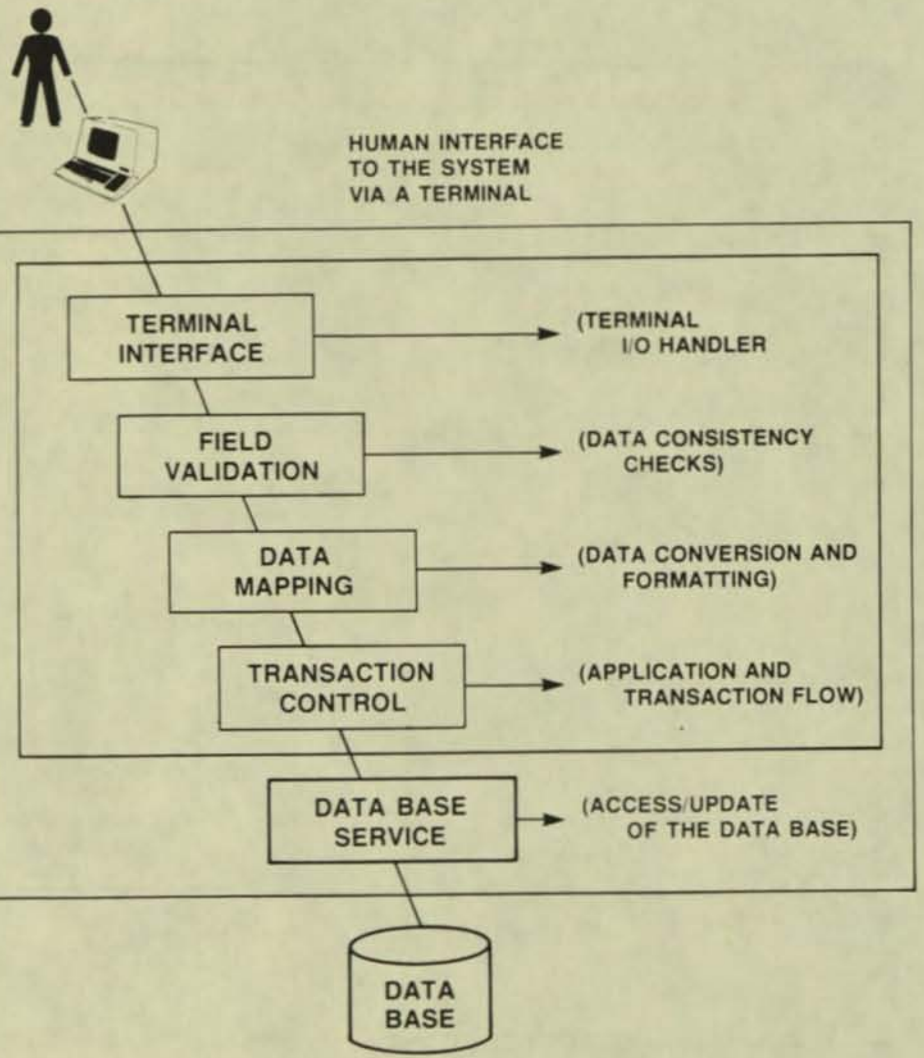


FIGURE 10. THE PATHWAY SYSTEM WITHIN THE TRANSACTION PROCESSING SYSTEM MODEL

PATHWAY Product Overview

The goal of the PATHWAY product is to simplify the design and development of transaction oriented applications. The PATHWAY product addresses four of the major components necessary to implement a transaction oriented application.

1. Terminal Interface	(Multi-terminal I/O handler)
2. Field Validation	(Data consistency checks)
3. Data Mapping	(Data conversion & formatting)
4. Transaction Control	(Application & transaction flow)

The fifth component (Data Base Server) can be implemented using any of the TANDEM standard languages — COBOL, FORTRAN, TAL, or MUMPS.

The PATHWAY product has the following components:

- *Interactive Screen Builder*

  Allows the user to build screens interactively at a terminal.

- *Screen COBOL compiler*

  A COBOL-like terminal oriented language. The compiler creates and maintains a pseudo code library accessed by the Terminal Control Process at run time.

- *Terminal Control Process*

  Interprets the pseudo code-library created by the Screen COBOL compiler and performs the four major application functions mentioned above in a NONSTOP environment.

- *Application Monitor*

  Responsible for creating, monitoring and altering the application run time environment.

- *AMCOM – Application Monitor Command Language*

  The mechanism by which an operator may communicate with an active Application Monitor.

If we assume a successful installation of a transaction oriented system, we now must deal with EXPANDABILITY. By using the unique TANDEM architecture, an application can be written and then expand smoothly as the demands placed on the system increase. Most successful systems first expand because of an increase in the number of users who need to use it. Figure 11 shows the addition of new users and interjects a new question: "Do I run more than one copy of the total application or do the users share the application?" Figure 11 shows users sharing the application. It should be noted that TANDEM's terminal control process, a part of PATHWAY, handles all the multi-tasking between more than one terminal of the same type. In Figure 11, notice that the application remains unchanged even though the number of users increases.

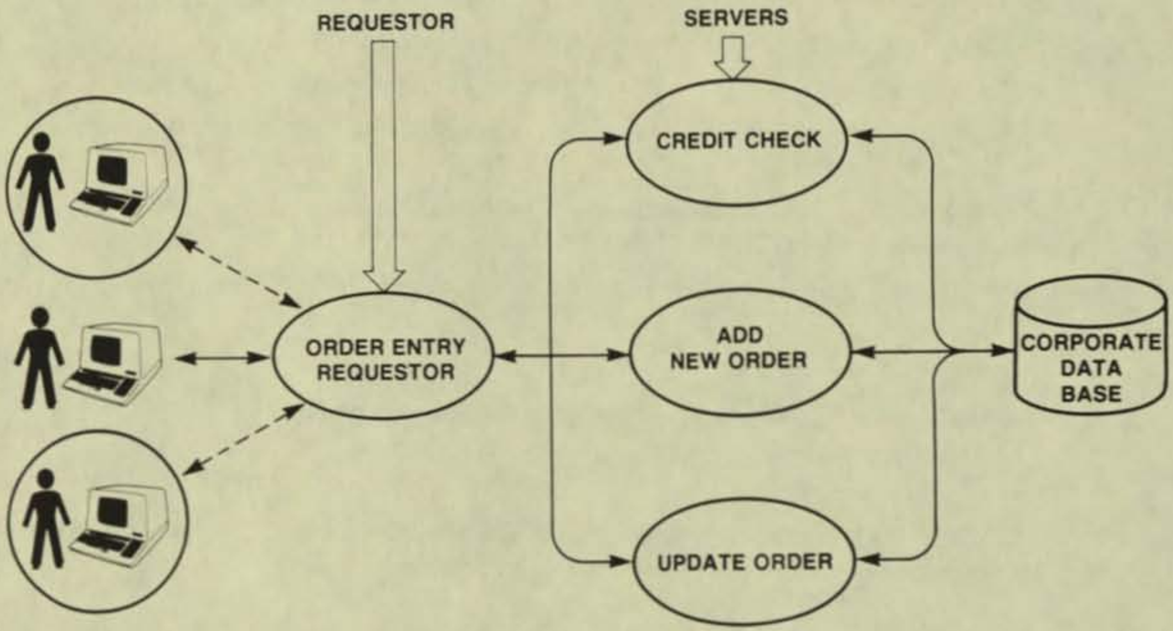


FIGURE 11. ADDING USERS TO PATHWAY SYSTEM

In a successful application, the addition of users can degrade system performance. At that point, most systems go through either a major change or a rewrite. The unique TANDEM approach offers an alternative.

To alleviate the problem of increased user load, the system must be able to distribute the application into more than one physical process. This is the key to expandability.

The description of the system to this point views the system as one self-contained application. However, all expansion limitations can be eliminated if one logical application can comprise multiple physical processes or running programs. This leaves the original design of the system unchanged, but increases system throughput by expanding the modularity of the application beyond the physical boundaries of a single program unit.

The following diagrams illustrate the various ways of functionally distributing the application to allow for expansion:

- Terminal operators can be connected to multiple requestors running the same application.
- Multiple copies of the same server can be created to increase throughput.
- Sharing a server between more than one application requestor.

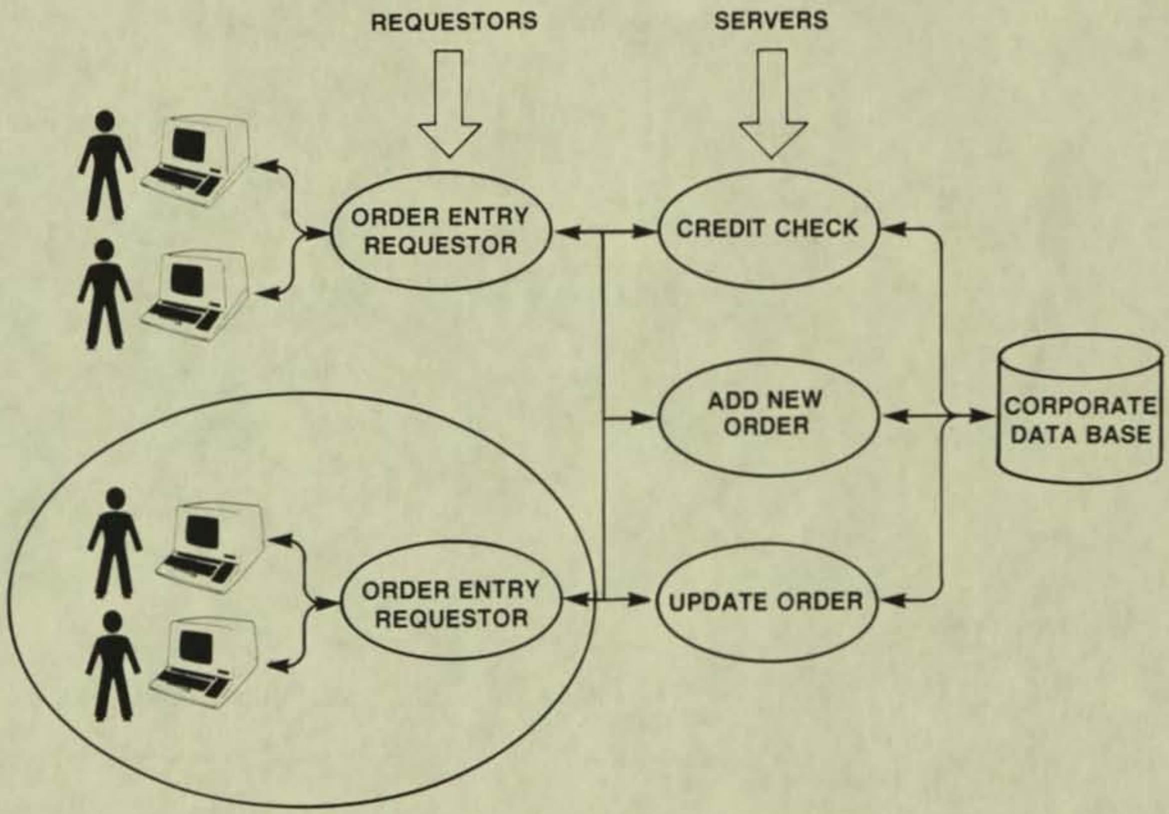


FIGURE 12. SYSTEM EXPANSION BY ADDING REQUESTOR

Notice that the users are distributed between two requestors and that the requestors share the server functions. Therefore, the total number of servers remains constant even though the system includes multiple requestors.

Another expansion problem is caused by increased demand on any one server. This problem can be overcome by duplicating a particular server to create what is known as a "SERVER CLASS". For example, if the majority of user demand on the system is to run credit checks, using a single server for credit checking could create a bottleneck and impact the total applicaton. This bottleneck can be eliminated by creating another copy of the credit check server and distributing the requests between the two copies. Figure 13 illustrates a server class:

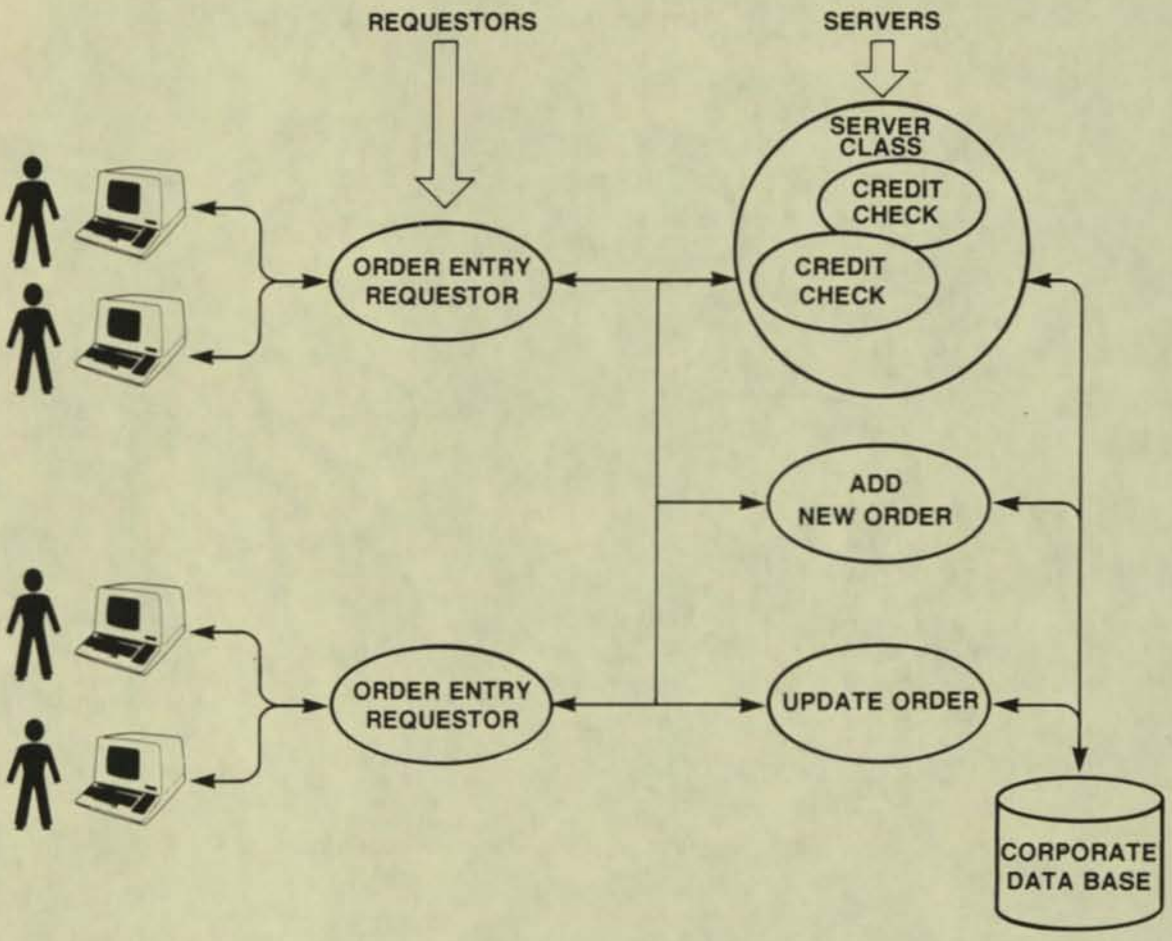


FIGURE 13. SERVER CLASSES

Figure 14 illustrates the addition of a new application. Notice that even though there are two unique applications, the requestors can still share servers or server classes.

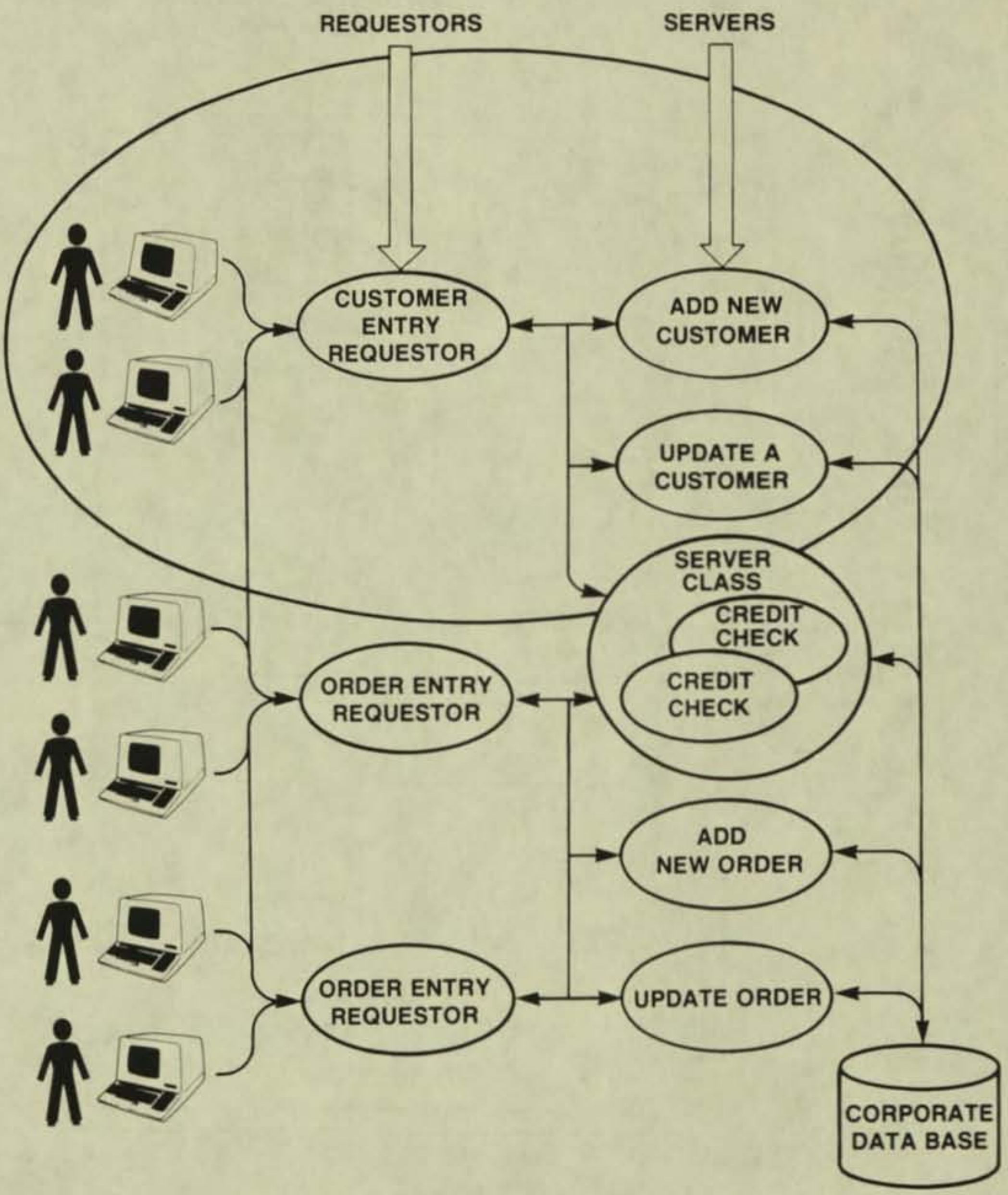


FIGURE 14. ADDING A NEW APPLICATION REQUESTOR

Figure 14 shows that both the order entry and customer entry applications need to be able to check credit. The credit check server class is therefore shared by both applications, yet each application also has its own private servers to fulfill the individual requirements of a particular application.

One of the major reasons for distributing an application into multiple processes was for expandability. Figure 15 illustrates the distribution of application functions within a local TANDEM system. Terminals, Requestors and Servers may be distributed among multiple CPU's, maximizing parallel operations and increasing throughput. It should be clear that when demand forces the system to grow that no design changes will be necessary.

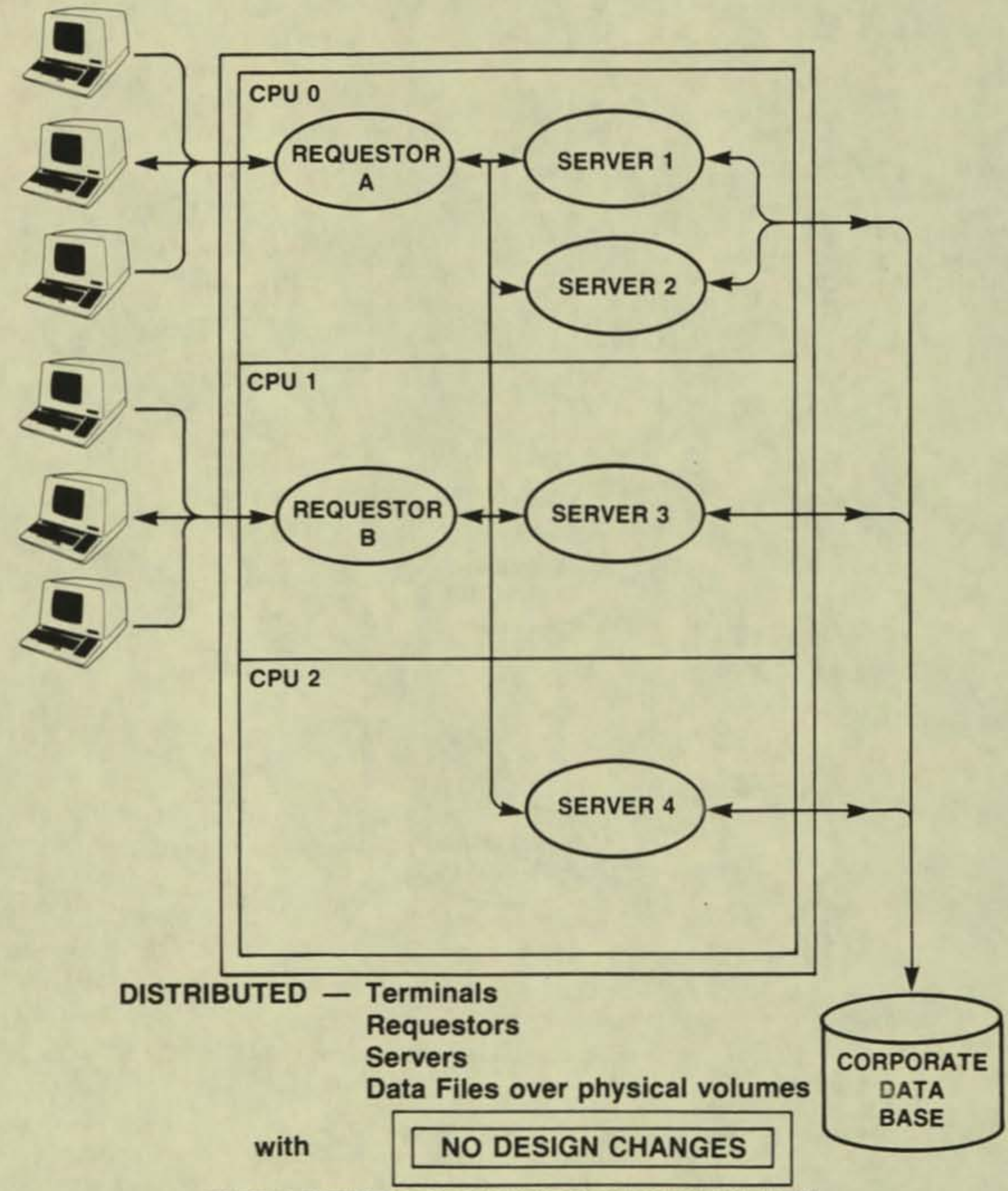


FIGURE 15. DISTRIBUTED LOAD BALANCING ON A LOCAL TANDEM SYSTEM

The next step in distributing application functions is over a network. The mechanism for communicating between processes within the TANDEM environment is consistent (Read / Write). Therefore, distributing application functions over a TANDEM network using EXPAND does not impact the original design. Figure 16 illustrates the distribution of application functions over a simple EXPAND network.

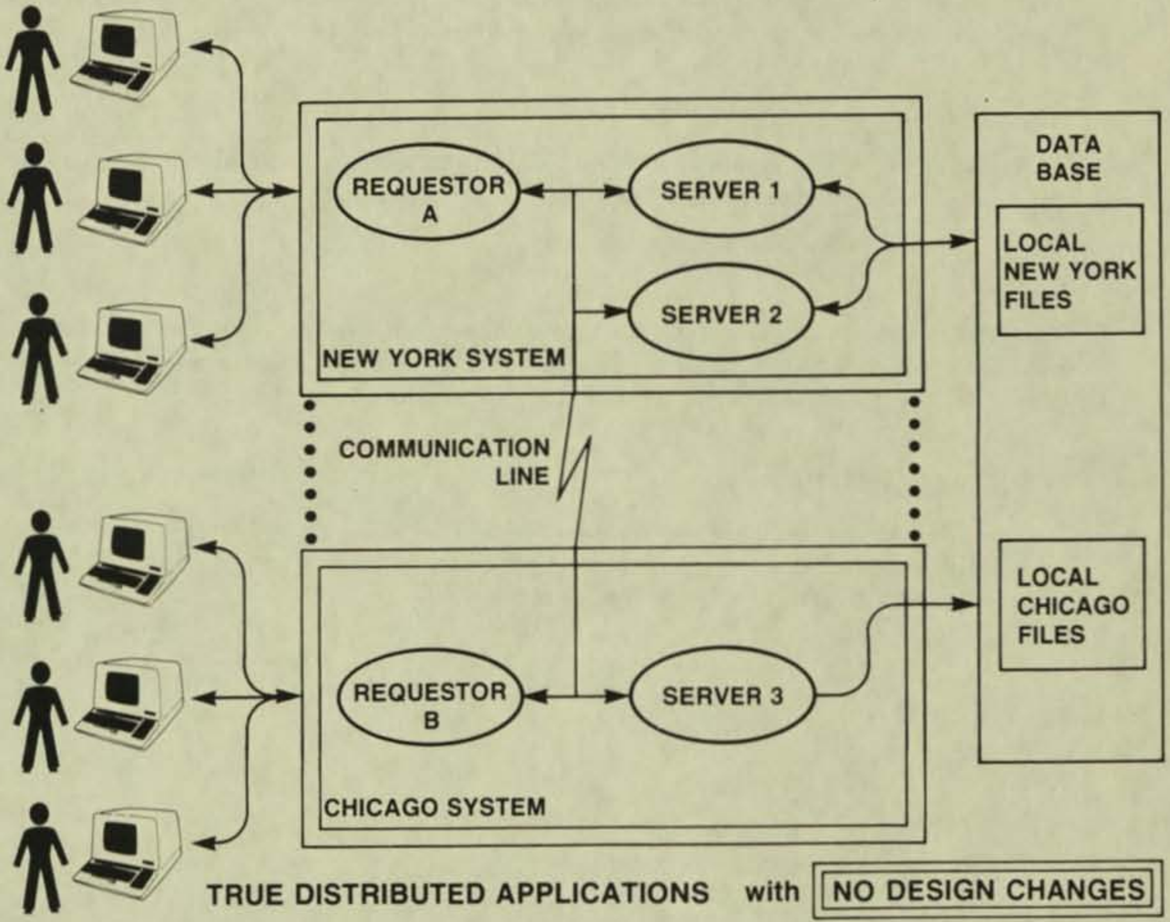


FIGURE 16. DISTRIBUTED LOAD BALANCING ON A SIMPLE EXPAND NETWORK

In summary TANDEM offers a total solution to transaction processing:

RELIABILITY:

NonStop transaction processing assures continuous system availability and data integrity

INSTALLABILITY:

PATHWAY offers a quick and easy way of developing transaction oriented applications, which significantly reduces the cost of applications development.

FLEXIBILITY:

PATHWAY offers the ability to make on-line additions, modifications, or deletions of transaction types, screen characteristics, applications, and terminals.

EXPANDABILITY:

The combination of the GUARDIAN Operating system and the EXPAND network offers true distributed processing, which allows applications to run in any processor in any system without regard for the physical location of terminals or the data base.

MAINTAINABILITY:

Because of the structured approach taken by the PATHWAY product, modules may be written and implemented in small, single threaded, and easy to understand components. Therefore, the maintenance task will be kept to a minimum.

THE MANAGEMENT JOURNAL FOR IBM SHOPS

SYSTEMS USER

One database does it all

by Kevin Weigler

Today, DP managers of global organizations believe they must accept the cost of performance trade-offs in relational databases. However, innovations in database technology will soon provide a solution to this problem.

Each vendor's relational database management system (RDBMS) is optimized for a specific environment. For example, an RDBMS that is designed for batch processing may provide poor transaction and query processing performance. An on-line transaction processing (OLTP) RDBMS that provides rapid inserts, updates and deletes may provide poor sequential reads. For query processing, an RDBMS may provide good decision support but poor OLTP performance.

"Parallel processing, distribution and resultant high performance are all natural extensions of a message-based operating system running over a loosely coupled architecture."

To gain optimum performance for all accesses -- transaction, sequential (batch), and query processing -- organizations must maintain multiple databases on multiple vendors' equipment. Typically, one system is used for batch and/or query, another for OLTP. The first database is updated at night, the second on-line database provides constant access to current information. But, with multiple databases, organizations do now know the state of their business at any one time and they incur the cost of maintaining extra hardware and software.

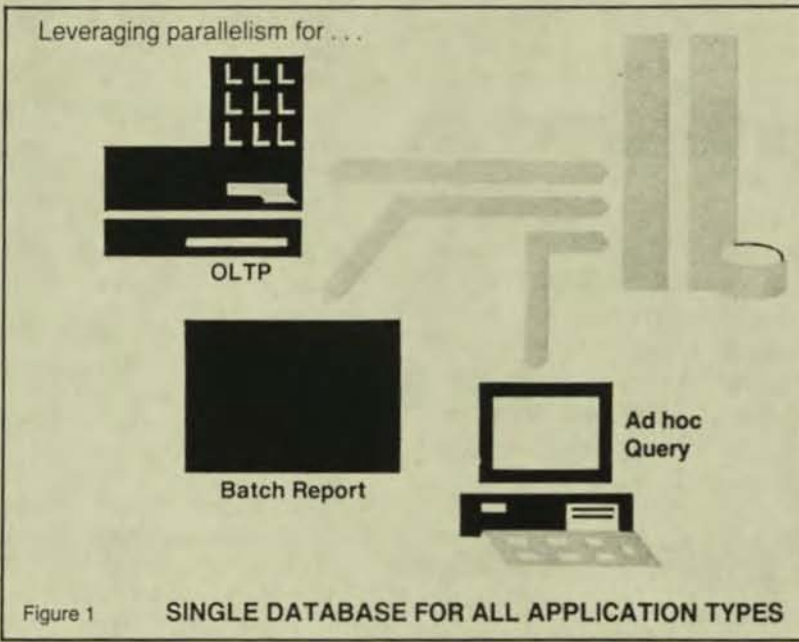


Figure 1 SINGLE DATABASE FOR ALL APPLICATION TYPES

Many organizations are switching from a hierarchical to a relational database management system because the latter provides a more flexible means of accessing and combining information. An RDBMS allows new types of data to be added and accommodates different ways of viewing data without the need to restructure the database.

The emerging data manipulation language of choice is the American National Standards Institute (ANSI) Structured Query Language (SQL). SQL allows portability of programming skills across the multiple vendor environments that are often found in global organizations. These organizations are demanding one database of record that performs well for all types of accesses using a language that is portable across heterogeneous systems.

According to the Gartner Group, Inc., a market research firm based in Stamford, Connecticut, "In the near future, customers won't have to trade off performance in one type of access for another. There will be a general-purpose, high-performance database capable of running OLTP, batch and query, available in 2 - 3 years." (see Figure 1). The added performance gained from this type of database could greatly increase an organization's competitive advantage.

Distribution for Performance

As organizations spread to provide better service to their geographically distributed customers, disseminating information via data distribution provides better performance for users than a centralized strategy. When users are widely dispersed, response time is dominated by time required for data communication. By moving the database close to the user, communication time is minimized. As communication time decreases, the cost of transferring data also decreases. Thus, data distribution can be more economical for global organizations as well.

Many organizations will have strategic reasons for both centralizing and distributing data. By using one flexible RDBMS, suitable for both centralized and distributed applications, an organization gains the advantage of disseminating data as business needs dictate.

With a distributed RDBMS of record, the organization maintains one logical database that is spread across any number of nodes. Users at local and remote nodes perceive the entire database as if it were stored locally.

A distributed RDBMS can ensure data integrity in the event of application, node or network failure. It can manage

"When users are widely dispersed, response time is dominated by time required for data communication. By moving the database close to the user, communication time is minimized. As communication time decreases, the cost of transferring data also decreases."

a transaction that is updating data on multiple nodes so that either the transaction will completely commit or back out.

A distributed network provides continuous availability, which also improves performance. Each node operates independently and even if other nodes become unavailable the network continues to operate.

Many smaller processors connected to process a query (for example) in parallel cost less to design, and hence to purchase, than one large processor. Using a multiprocessor system with non-shared memory that is capable of parallel processing, an organization can use multiple operating systems, disks, channels and applications to gain better overall performance.

Parallel Performance

Parallel processing can be accomplished in several ways, including the use of tightly and loosely coupled multiple processors.

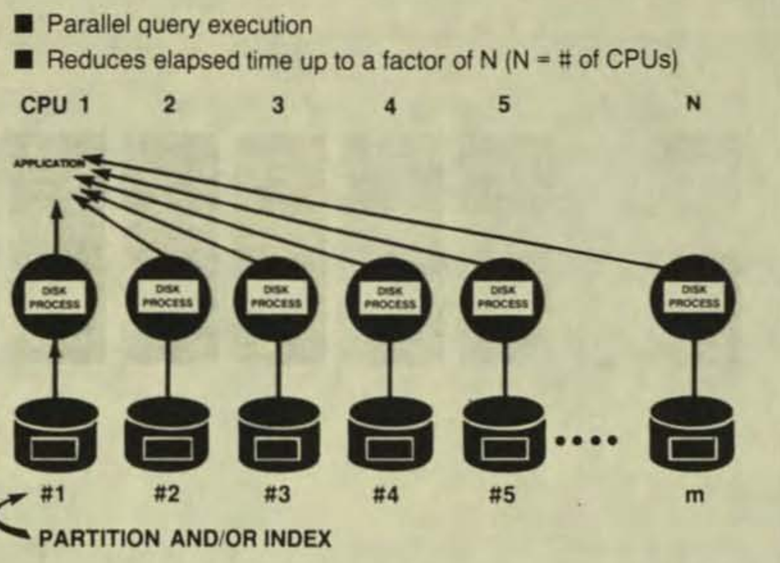

■ Parallel query execution
■ Reduces elapsed time up to a factor of N (N = # of CPUs)
CPU 1
2
3
4
5
N
APPLICATION
DISK PROCESS
#1
#2
#3
#4
#5
m
PARTITION AND/OR INDEX

Figure 2 **LEVERAGING A PARALLEL ARCHITECTURE**

Tightly coupled processors share one or more key resources, including memory. As contention increases for key resources an ever greater percentage of system resources will be devoted to managing the system and its resources, not to productive work. Adding an additional processor provides much less than 100 percent of its capacity due to the interlocking of shared memory. There is also a limit to how many processors can be added before the memory runs out of bandwidth.

Tightly coupled systems run a significantly greater risk of failure than loosely coupled systems because an error in one processor can corrupt the operations environment of another. If the one copy of the operating system fails, all the processors fail.

Running a message-based operating system, *loosely coupled processors* communicate by sending messages from one program to another. Even system level requests become messages flowing over a high-speed bus to each processor.

With loosely coupled processors, when one processor fails, the others will continue to be productive and no data is lost. Load sharing can spread out the work and give better, more consistent response time. Adding an additional processor provides close to 100 percent performance improvement from each processor. Hence, linear performance growth can continue almost indefinitely with the addition of processors. With loosely coupled processors, a distributed environment is a natural extension of a central site.

Loosely coupled multiprocessor systems provide even better performance when used for parallel processing. In parallel query processing with a distributed RDBMS, the master query executor(s) decomposes a query into N fragments. N server executors search disks containing the desired tables, anywhere in the network, and the result is brought up to the application. This can improve response time by as much as a factor of N, where N is the number of processors that are available to the query. For example, with five fragments a request that would normally execute in 10 seconds could take as little as 2 seconds (see Figure 2).

Parallel processing, distribution and resultant high performance are all natural extensions of a message-based operating system running over a loosely coupled architecture. Thus, hosting a single database of record on a parallel processing system provides a substantial performance advantage.

Platform for Performance

In addition to the need for parallelism, the one RDBMS of record places additional requirements on its system platform:

• Data Integrity - A transaction is completed as a whole or not at all, even if the system crashes in the process of completing the transaction.

• Modular Expandability - More processors, disks, workstations, devices, etc. can be added to the system without taking it down or changing any application or system code.

• High Availability - A loosely coupled multiprocessor architecture provides very high availability because, when one processor fails, the other will continue to do the failed processor's work -- without interrupting the user.

• Node Autonomy - In a distributed network of systems, some nodes will occasionally be unavailable. Each node must be able to provide data without reliance on any other component or node in the network.

• Security - An RDBMS should be integrated into the operating system to prevent subversion by a user who opens and writes to data files without going through the RDBMS.

• Tools - An RDBMS should allow easy access, with comprehensive tools to increase productivity and keep down the increasing cost of system operations and development. Decision support tools are important in enabling an organization to use data in new strategic ways.

• Workstation Integration - Workstation and personal computer use will continue to expand into the 1990s. These devices will generate increasingly more complex transactions, growing from hundreds of bits to hundreds of thousands of bits in length. For maximum user productivity and best utilization of inexpensive MIPS, workstation users need to share data with host systems on a peer-to-peer basis.

• Connectivity - The ability to connect diverse systems and software protects an organization's investment and improves productivity of users and equipment. On-line applications can be off-loaded from a back-end host designed for batch processing to a system designed for OLTP to free host capacity. This system will require excellent connectivity across multivendor systems, i.e. X.25 packet-switching networks, SNA systems and networks and workstation local area networks (LANs). This connectivity should be as transparent as possible to the application.

One Database for All

Increasingly, organizations are recognizing the strategic importance of enterprise-wide transaction processing networks, considering the database to be "mission critical" to their operations. If the database is unavailable, business stops.

This network is becoming a transaction-rich environment, requiring data sharing across wide area and local area networks, workstations and mainframes and heterogeneous systems. When transaction processing is done on-line, a user at a terminal, personal computer or workstation can capture transactions, make changes to the database, and get response in seconds.

"To gain optimum performance for all accesses -- transaction, sequential (batch) and query processing -- organizations must maintain multiple databases on multiple vendors' equipment."

The network provides immediate access to the exact state of the business, extremely important to a global operation where multiple demands are made for strategic information 24 hours a day. Compare this with batch processing, which updates transaction information in a few hours or by the next day.

For the enterprise transaction processing network of the future, the one RDBMS of record will need to service heterogeneous systems with an industry standard communications interface, such as IBM's SNA LU6.2. It should adhere to the full ANSI SQL standard to meet multivendor network requirements. To perform well for all types of accesses, the RDBMS of record should accomplish the following:

• On Line Transaction Processing - I/O architecture should be optimized for random access. Fixed overhead for initializing and terminating transactions must be minimal.

• Batch Processing - Batch processing requires effective sequential I/O via techniques such as block buffering and successful disk cache management.

• Query Processing - For complex transactions, throughput can be improved if each query within a transaction can be evaluated in parallel, taking advantage of loosely coupled multiple processor architecture.

• Operations - As organizations move critical information to a single RDBMS, it becomes necessary that the database be available 24 hours a day, seven days a week.

Typically, to rebalance files for rapid physical access, the database must be shut down. A "snapshot" copy of the database must be made, compressed and then swapped into operation.

With on-line reorganization, files can be rebalanced during an access while on-line, without incurring the expense of shutting the operation down. On-line file reorganization translates into increased availability and higher performance.

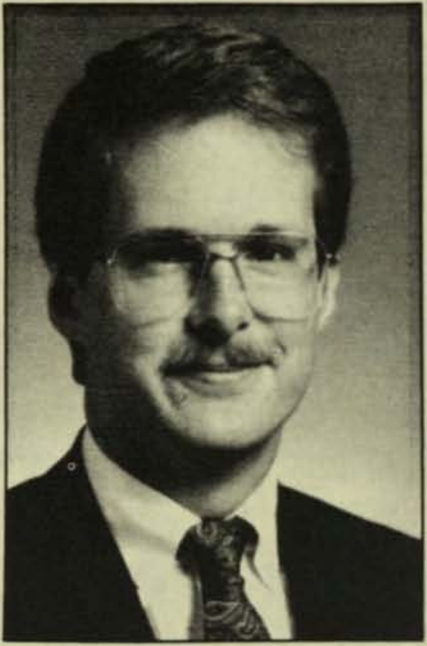
Key to 1990s Performance

Current RDBMS offerings are designed for particular processing environments: transaction, batch and query. Parallel processing technology offers the greatest potential to provide a platform for an RDBMS to perform well for all three environments. In addition to requiring parallelism, a single database of record demands several additional services from its operational environment, i.e., data integrity, modular expandability, high availability, security, tools, workstation integration and connectivity. DP managers must consider the total operational requirements when evaluating vendor offerings. **SU**

About the author

Kevin M. Weigler is product marketing manager for NonStop SQL, a distributed RDBMS from Tandem Computers. He has specialized in DBMSs and application development on a variety of hardware platforms for thirteen years.

After studying physics at Evergreen State College in Olympia, Washington, Mr. Weigler earned a B.S. in mathematics from the University of LaVerne in California.



a special reprint from the

NATIONAL UNDERWRITER

INSURANCE & THE INFO INDUSTRY

DDP Offers Flexibility At Low Cost

By STUART LEVIN and JORDAN SCHELL-LAMBERT

The insurance industry can become more competitive and cost-effective by sharing data processing tasks among multiple computers or sites, using an approach called "distributed data processing".

Firms using this approach experience, among other benefits, improved customer service levels, greater flexibility in the use of their systems, and reduced data communications expenses.

Distributed data processing is a technique that relies on multiple computer systems. In a distributed processing environment, computers share processing workloads while regularly sharing data among themselves.

Computing tasks are distributed across processors, each of which is designed to optimally handle its processing demands. Central mainframes could be devoted to large batch runs while "transaction processing engines" would handle on-line inquiries and database updates. They work in parallel, in contrast to a single central computer processing one task at a time.

An example of this can be illustrated at the insurance point of sale.

A carrier has placed a kiosk in a shopping mall, with a personal computer and electronic funds transfer capabilities integrated within it.

POLICYOWNER SERVICE and claims personnel are more productive using on-line transaction processing distributed systems.

Ready For Inquiry

People in the mall may come up to the kiosk and inquire about costs for certain products, such as term life, or personal auto or homeowners insurance.

All rate quote processing could take place at the mall location, or at a regional processor. The system may want to check the claims and payment history (if any exists) of the applicant within the home office system.

The kiosk prints out the quote. The system could then ask the prospect if he or she would like to make a purchase. If yes, then the prospect could use a credit card, have the purchase authorized, and receive a binder. The home office systems would be notified of the sale across an electronic network.

DDP can be approached in a variety of ways. For simplicity, this article covers three possible approaches: specialized "transaction processing engines," geographic expansion, and networks optimized for DDP.

1. Transaction Processing Engines. A fairly simple form of DDP can be prescribed for a common headache: integration of existing systems that can't communicate with each other, especially if they run on different "host" processors. The solution uses computers as "go-betweens" to handle the translations.

An example of this approach would be using screens existing in different host computers or simply in different applications. On-line transaction processors could easily bring all screens together, quickly creating "new" applications, maximizing the productivity of all involved.

This approach also allows policyowner service and claims departments better access to information, raising service levels and providing competitive advantages.

A large direct-response insurance company has implemented a DDP system in this category. The system integrates back-end mainframes with 1,500 personal computers in a nationwide network.

The new system provides sales and service employees a way to access both Honeywell and IBM host systems. Also, agents and service personnel need not be concerned about logging onto multiple hosts. Once they have accessed the network, it manages security and allows users easy access to whichever applications they need.

2. Geographic expansion. As a strategy, growing distributed "nodes" is a way to make users more productive. Processors at each location can be tailored to the people or customer they service. This flexibility is not easy to achieve with a single central machine.

The computers used in DDP have been developing more rapidly than large general-purpose processors. It is not unusual for a product's performance to dramatically increase while the price decreases from year to year. Distributed processing takes advantage of this trend. As new systems are designed, more processing can be put on machines closer to the user. The central host may still be used for special calculations or high-volume runs like check writing.

Major Cost Savings

Major cost savings can be achieved by using the local machines yet another way. They can help contain communications costs by handling long-distance voice and data switching for an office or a region.

Agency automation or field underwriting applications could be accomplished with this kind of system. Since home-office underwriting procedures can be enforced through the system, a significant amount of authority (generally for simpler, common-place products) can be distributed without hiring high-level expertise in every office.

3. Corporate processors and networks optimized for DDP. Though each company will define "optimum" differently, the ultimate in distributed processing might resemble a map of the carrier's office locations and agencies. Some processing chores are likely to remain centralized. Yet, with this approach, even a traditional large central computer site may take advantage of DDP.

After outlying nodes have been developed, the systems planner should take a look at what remains on the central computer. It might make sense to break up the central tasks, creating high-speed "transaction processing engines" within the network.

Higher Performance

Before a transaction is completed, it may pass through multiple devices and processors. Even if these machines are located in the same room at an operations center, the company can still use DDP techniques to achieve higher performance and flexibility.

Before installing a distributed system, insurers would want to evaluate the economic variables of the plan. Potential cost savings must be balanced against development costs.

As cost and competitive pressures in the insurance industry increase, the interest in DDP is growing. Look for the use of distributed processing systems to increase within the insurance industry in the near future. □

Mr. Levin is insurance industry marketing manager for Tandem Computers Inc., Cupertino, Calif. Mr. Schell-Lambert is manager, management information consulting division, Arthur Andersen & Co.

BUILDING FOR THE FUTURE

Establishing an In-house Custom Design Capability

by Al McBride
Director of Engineering, VLSI Technology
Tandem Computers Incorporated
April 14, 1986

Introduction

Tandem Computers Incorporated, founded eleven years ago, today ranks as a Fortune 500 company with $624 million in revenues (fiscal year ended September 30, 1985). Tandem's success is derived from the wide market acceptance of its unique computer system, designed to satisfy the full range of requirements for on-line transaction processing (OLTP). The market for OLTP systems is growing twice as fast as the worldwide market for commercial data processing systems.

Tandem systems are widely used to run automated teller networks, messaging systems, factories, stock exchanges and point-of-sale systems.

Unlike computer systems from any other vendor, Tandem systems provide the combination of data integrity, distributed data base, system security, fault tolerance, modular growth without reprogramming, the ability to process thousands of transactions per second, networking, ease of programming and integration with existing information processing devices.

In-House VLSI Design Capability

In late 1979 and during the very early 1980s, Tandem began investing in an in-house integrated circuit (IC) research and development facility to develop VLSI technologies for use in future products. Tandem's goal was to minimize development time for new products using increasingly complex circuit technologies, and to deliver new systems to market faster.

Today, Tandem has a leadership position in VLSI design capability. The company has in-house facilities for computer-aided design (CAD), simulation and VLSI prototype fabrication.

The most significant outcome to date of Tandem's investment in VLSI technology development is the new NonStop VLX system. The VLX is a transaction processing mainframe that delivers higher transaction throughput than the industry's largest computer system. It features new ECL/TTL gate array technology in its processors, and CMOS technology and off-the-shelf microprocessors in many other parts of the system.

By designing new circuit technology internally, Tandem was able to reduce by one year the development time for the VLX. New circuit technologies that Tandem is continuing to develop, combined with the advantages of Tandem's unique parallel system architecture, will provide the foundation for Tandem's future.

Tandem's VLSI Philosophy

In 1979, Tandem made several fundamental decisions that would govern its development and use of VLSI technology. These decisions were ambitious for the Tandem of 1979 -- a $50 million company only five years old.

Tandem's first decision was to use gate array technology because of the benefits it offered. Gate arrays could be personalized and lent themselves to quick design turnaround. They also offered a choice of technologies: bipolar for high speed and CMOS for high density and low power. Tandem would apply both of these technologies as appropriate throughout its systems. In addition, entire gate array designs could be be done with one base wafer type, and they met the system requirement for design contribution and uniqueness.

Another key decision was to establish more than one source for the gate arrays. Because supply would be fundamental to Tandem's success in this venture, the company could not gamble on any one vendor for the components.

Tandem also decided that it wanted to have other technologies available, in addition to gate arrays, for implementing different products such as processors and controllers. So the company decided to use microprocessors as well as bipolar and CMOS gate arrays.

Finally, Tandem decided to build a prototype chip fabrication facility so that it could turn around new designs quickly, and -- in the face of ever more complex system designs -- provide new systems in a timely manner.

Computer-Aided Design (CAD) and Simulation

In 1979, CAD and simulation capabilities were the "crown jewels" of only a few large companies. Advanced chip and multiple chip simulations were still academic subjects. Consequently, Tandem's first job was to write CAD and simulation software that could run on its own computer systems.

One of Tandem's primary objectives for its CAD capability was that it had to give Tandem the ability to design circuits in such a way that they could be multiple sourced. With these CAD tools, Tandem is able to map its chip designs to multiple vendors and have interchangeability of chips on the boards. This helps Tandem ensure low price, high quality and availability.

In just 15 months, Tandem's CAD and simulation software were up and running. The first processor the company designed using these tools was the NonStop TXP, introduced in 1983. Although there were no gate arrays in the TXP, Tandem proved that its design and simulation software worked.

The company estimates that the in-house tools cut as much as six months from the design cycle of the TXP. Tandem was able to make modifications early in the design cycle and test them in a few hours.

Tandem Systems Help Design New Systems

One of the most important advantages of Tandem's design tools is that they run on Tandem computers and can exploit the power of Tandem's distributed data base and networking capability. In addition, CAD and simulation tasks run much faster on a Tandem system than on large conventional uniprocessor systems because the tasks run in parallel on separate processors.

Another advantage of Tandem's design tools is that they allow designers to frequently interact with the program to see how the design is progressing. They can modify the design throughout the process and experiment and innovate -- one of the most important advantages of a responsive design system.

Tandem's in-house CAD and simulation capabilities have a direct impact on the success of its products. Effective design can result in less hardware in a system, which means lower cost, better reliability, lower cost of ownership and higher performance. The NonStop II, for example, contains 30,000 gates on three boards -- about 10,000 per board. The TXP has 50,000 gates on four boards. Tandem's new system, the NonStop VLX, has 78,000 gates on only two boards, three times the integration density of TXP.

Tandem's CAD layout tools enable Tandem to achieve 10 to 15 percent better circuit utilization on bipolar and CMOS chips than the industry average. The new VLX system has an average chip circuit utilization of 95 percent over 33 parts. If Tandem had followed the recommendation made by one of its semiconductor vendors -- to have 80 percent utilization -- the VLX would have resulted in 50 percent more boards, 20 percent more chips and a 25 percent higher cost. Tandem has been told that it is the only company to be successful at laying out 100 percent of the usable gates on the chip.

From Design to Prototypes

In 1979 when Tandem selected bipolar gate arrays as the technology for future high-performance systems, Tandem's research showed that there were no gate array vendors who had what Tandem required for the VLX. Available gate arrays were all emitter-coupled logic (ECL). But Tandem's systems experience showed that transistor-to-transistor logic (TTL) for the input/output interface to the printed circuit board was the most cost-effective for its systems.

The result of this research was a joint chip development between Tandem and Motorola, one of the world's leading semiconductor suppliers. Working together, Tandem and Motorola jointly developed the Motorola MCA 2800 ALS, now a standard product in Motorola's semi-custom bipolar line. The chip is optimized to Tandem's needs in terms of density, power dissipation and performance, and takes advantage of Motorola's state-of-the-art Mosaic II bipolar production process.

Reduced Design Time

Also in 1979, Tandem had decided that fast turnaround for gate array designs would be critical to its ability to bring new products to market quickly. But at that time, all the vendors Tandem surveyed were only beginning to develop their processes and improve yields. Turnaround time for a new chip -- or even a design change -- was months, not weeks. Tandem knew it could not build a wafer fabrication facility, but realized it would have to do some part of the process on its own.

Consequently, Tandem built a very large-scale integrated (VLSI) prototype fabrication line in which Tandem could metalize, test and assemble its own unique designs, starting from a vendor's standard wafer. With gate arrays, the uniqueness of each design is in the connection of the gates on the wafer. The semiconductor vendor does two-thirds of the manufacturing and maintains the wafer inventory, then Tandem does the final one-third in one to five weeks (instead of the 10 to 16 weeks it would take from the vendor). Following its multivendor strategy, Tandem qualified two vendors for the wafers. Tandem's first prototype chip was a 2,000-gate bipolar device with three layers of metalization. Tandem's fab line could add three layers of metal to the base silicon and process CMOS and bipolar wafers. Only a few chip fab lines in the world can match this.

The lab was built during 1983. The initial designs were fabricated in the fall of 1984. In 1985 the prototype line turned out 52 bipolar and 18 CMOS designs. About 1,000 parts were produced, including some for manufacturing beta systems. Some chips were turned out in a single weekend, although the average turnaround was five weeks. Tandem's in-house VLSI lab even produced chips for manufacturing VLX beta systems to keep the project on schedule.

Part of Tandem's in-house capability is a rigorous testing plan. For instance, heat transfer simulations began on the new chip even before any prototypes were available. As a result of these simulations, Tandem designed a heat sink that substantially improves the reliability of the end products.

Tandem's joint development work with Motorola resulted in a standard product that today is the most popular gate array in Motorola's semi-custom bipolar product line. Tandem's CAD and prototype fabrication capability have been successful in helping Tandem develop new products in a timely manner. In just five years, Tandem has moved from absolutely no in-house capability to a position of leadership in bipolar and CMOS gate arrays.

Advantages for the Future

Tandem's new product development has benefited from its in-house CAD, simulation and prototype fabrication capabilities. The knowledge base and the facilities Tandem has gained from this investment will provide it with advantages in the future.

Today Tandem systems incorporate 2,000 circuit bipolar chips and 4,000 circuit CMOS parts. Tandem engineers are working on CMOS designs with as many as 20,000 circuits per chip. Developments such as these could result in a 30,000-gate, three-board NonStop II on two chips.

Tandem intends to continue to develop and exploit state-of-the-art technology jointly with its semiconductor vendors.

Tandem already designs full custom chips using silicon compilation techniques. Tandem is also examining a gate array connection technique that recovers space on the chip by using more than three layers of metal, plus other innovative process and circuit design inventions.

The company expects that these advances will allow it to continue to increase densities -- 8,000 gates in bipolar and up to 50,000 gates in CMOS by 1990. In addition, Tandem will continue to explore new packaging schemes to accommodate the power dissipation of the new designs. Tandem also is watching developments in gallium arsenide as a possible alternative to silicon.

Staying at the forefront of these technologies is vital to Tandem's future. By applying the latest VLSI technologies within Tandem's patented parallel architecture, Tandem believes it can continually offer its customers the best on-line transaction processing systems in the industry.

-- END --

Al McBride is Tandem's director of engineering, VLSI technology. He joined Tandem in 1980 after 15 years at IBM, where he held senior engineering positions in advanced technologies. During his tenure at IBM, Mr. McBride worked on the advanced computer system project which was managed by Gene Amdahl. Mr. McBride also worked on IBM's 801 RISC processor development and the initial designs of the IBM PC. He also managed IBM's development of a 1500 circuit bipolar gate array microprocessor and he managed the development team for IBM's first single chip computer. Mr. McBride holds MSEE and BSEE degrees from the University of California, Berkeley.

Jeri Edwards, Tandem Computers Inc., Cupertino, Calif.

Time-staged delivery networks save time, enhance productivity

When resources, facilities, and recipients are unavailable, a new type of network allows data to reach its appointed rounds.

Time-staged delivery allows a sender to input information to a network and then go on to other tasks while the information is being sent. Unlike interactive communications, time-staged delivery does not require that the information receiver be present at the time data is transmitted. A delivery mechanism transfers the information to a location in the network designated as the receiver's reception "depot" and stores it there. Message receivers can then pick up messages at their convenience. Time-staged delivery is often termed "asynchronous," or without regard to time, because the sender and receiver do not have to be in lockstep for information to pass between them. Several requests may be sent at once. The responses are returned later, without regard to time or sequence.

Before time-staged delivery, all information exchange was interactive. That is, senders and receivers exchanged information in real time (Fig. 1). This meant that the sender, receiver, and network all had to be active at the same time. Such setups are generally called synchronous communications because the sender and receiver match, or synchronize, responses with requests. A response is the information returned directly after a request, and it must be received before another request can be made.

Interactive and time-staged delivery parallel the development of the modern telephone network and electronic-mail implementations, respectively. That is, people use the telephone when they must talk directly to another person. The sender, receiver, and communications resource must all be active for the message to get through. But often it is more productive to use an electronic-mail facility, especially if time is not critical. This type of implementation accepts messages (from the sender) and guarantees that they will reach the intended receiver, thereby eliminating the need for direct contact between sender and receiver.

This does not mean that all electronic-mail implementations are time-staged delivery networks. Conventional electronic-mail networks incorporate time-staged delivery but only for a particular terminal type and software application. General time-staged delivery networks, on the other hand, are multipurpose implementations that can be used by any application.

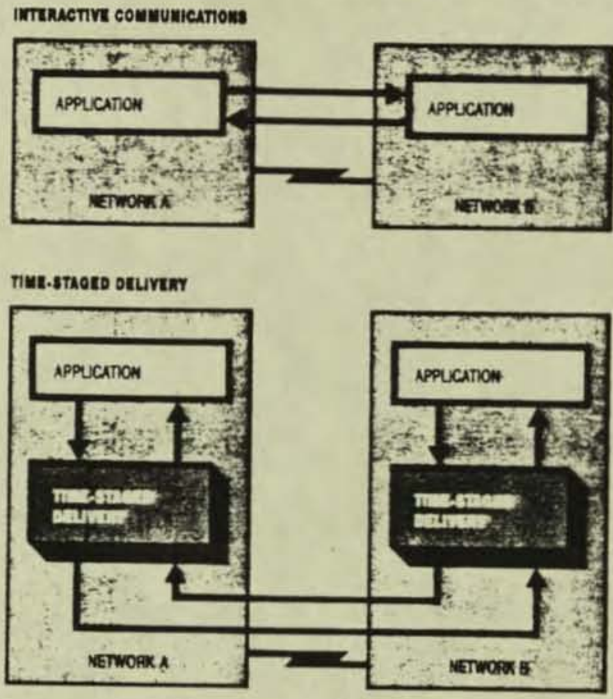
Occasionally, time-staged delivery networks are confused with store-and-forward implementations, such as Arpanet. Packet-switching networks have been labeled store-and-forward networks because they take packets off incoming communications lines, put them on an outgoing queue, and then transmit them (see "X.400 compared"). Time-staged delivery networks often operate on top of such packet-switching networks, providing an added layer of service. The delivery network, however, manages the entire transmission (which may consist of multiple packets) through the packet-switching network on behalf of the sender.

Transfer vs. SNADS

Tandem Computers' time-staged delivery network is called Transfer. Tandem customers write time-staged applications that use Transfer for the distribution and storage of information. Tandem's electronic-mail and facsimile transport products use Transfer for office communications.

IBM also has a time-staged delivery capability known as Systems Network Architecture Delivery Systems (SNADS). Like IBM's Systems Network Architecture (SNA), SNADS is also a network architecture. To date, it has been implemented in IBM's Distributed Office Support System (DISOSS) version 3.2, System/36, and the 5520 word processing system. These products

__1. Interactive vs. time-staged.__ With interactive communications, sender and receiver exchange data in real time. In time-staged, the network is independent.



implement SNADS for document distribution. The SNADS implemention in such instances is closed: Applications cannot use the SNADS facility. The recent SNADS implementation in IBM System/38, however, supports a method for applications to use its SNADS capabilities.

Any applications that interact with each other could probably be enhanced by using a time-staged delivery network, and there are four categories of applications that can be immediately identified:

X.400 compared

The International Telegraph and Telephone Consultative Committee (CCITT) approved the X.400 message handling standard in October 1984 (DATA COMMUNICATIONS, May 1984, p. 159). Since then, several vendors have announced support for the standard. Electronic-mail vendors are examining X.400 for use in linking private electronic-mail systems to public services.

X.400, Systems Network Architecture Delivery System (SNADS) coupled with Distributed Office Support System, and Transfer have certain elements in common. Conceptually, all three involve a delivery system with a postman, envelopes, and document contents. It appears there may have to be gateway products to link X.400 with SNADS and other products. Transfer is similar to X.400, but it is not currently compatible at all X.400 levels.

- Job networking.
- Distributed database maintenance.
- Document distribution.
- Intelligent networking.

Job networking. This first category is also known by the terms "extended transactions" and "functional distribution." There are instances where processes need to be tied together for full automation into a sequence of steps. Information then flows between the steps and triggers the execution of dependent processes. An example of job networking might be a chain store where orders taken at branch offices are then relayed to regional offices. At the regional office daily sales records are taken and supply orders are filled out and sent to a regional warehouse. When the stock arrives at the store's receiving dock, the receiving clerk records its arrival (Fig. 2).

The Japanese are developing a new twist to job networking, a time-staged method of manufacturing they call "Kanban"—also known as just-in-time manufacturing in the United States. With Kanban, factory parts are worked on in various locations and ultimately arrive at predetermined locations just in time to be incorporated with other arriving parts. Accompanying the parts is documentation regarding identification, history, and the next station destination for the parts. Once the parts reach the next station, the information regarding them is updated.

Robots and cards

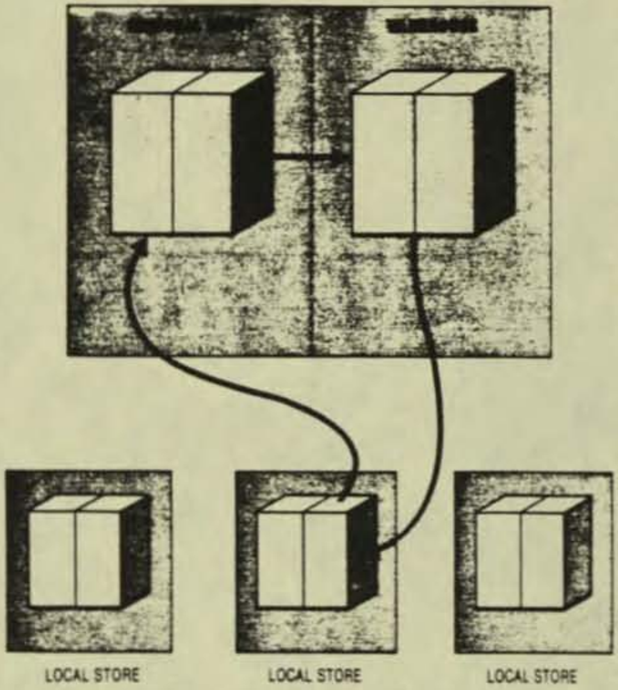
Although the Kanban method promised to save manufacturing companies millions of dollars, it has not done so yet. As it turns out, the task of automating parts flow is easier than automating information flow. The Japanese used cards to record parts information (*Kanban* means card), which worked well when people manned stations, but not when robots manned stations.

Time-staged delivery solves this problem by automating the flow of information between processes. After information is delivered to a process-receiving depot, an application can get the information and display it for a human operator, or a robot can access the same information. After the part is processed, the information can be updated and handed back to the time-staged delivery operation for transport to the next process.

Distributed database maintenance. Managing a network containing a distributed database can be a difficult job. The entire database must be available and accessible when updates occur, otherwise it can become inconsistent. For example, a worker jackhammers through all eight lines in a communications path, or computers are taken off an application after business hours in London to work on another job while updates are being sent from California during business hours. Thus, vital updates during such downtime can be lost.

A time-staged delivery network can alleviate this maintenance headache by delivering information when resources are available to handle it. The network can hold the update until the links damaged by the jackhammer are repaired, deliver the update to the destina-

2. Chain store. *Time-staged delivery networks are useful for chain stores where orders must be coordinated with merchandise shipments and delivery dates.*



tion node, and hold it there until the application has returned.

Document distribution. This application covers information transportation between individuals. These applications are the most obvious for timed-staged delivery because people are usually too busy to wait for information. A time-staged delivery system would take information from the sender and hold it until the receiver is ready to pick it up.

Electronic mail is the most common document distribution application. File transfer and forms routing are also document distribution applications. File transfer applications take documents created outside the system (for example, a microcomputer or word processing equipment) and transport them to receivers (one person or a group of people). Forms-routing applications typically automate the flow of business forms through a corporation.

Tools that automate business activity

Some document distribution applications can be similar to job networking in office environments. For example, a person fills out an expense report that then gets routed by several individuals in management for approval before it reaches the payroll department. Unlike job networking or distributed database maintenance (which is transaction based), document distribution applications are productivity tools that automate business activity.

Intelligent networking. These applications connect diverse computers and devices through a central network of processors. Applications on diverse computers usually are created to work in standalone environments. Connecting them through a central network facilitates communications, while placing the burden of connection on the network. This arrangement also tends to be least disruptive to the diverse computer applications.

Requiring synchronized communications for passing information on an intelligent network would often require programming modifications. In many cases, a time-staged delivery network eliminates most of the need for programming modification. The network handles the communications for the applications. Because the network holds the information at the destination for the recipient, it allows the receiving station to respond to the information when it is convenient to do so, rather than interrupting the recipient's local work flow.

Transfer and SNADS represent solutions to the requirements for time-staged delivery. The goals for the two architectures are similar. Both move data between the sender and receiver asynchronously; both add functionality to a network built for interactive communications; both are meant to be used in mixed interactive/time-staged applications, as well as "pure" time-staged environments; and both have goals of application independence, ease of use, manageability, efficiency, and extendability.

As with any network, there is a trade-off between functionality and resource use. A full-function time-staged delivery network offers a variety of features. A basic network (distributed spooler) offers a skeleton of the functionality of the full system, but it consumes fewer resources. This is not a bargain, however, if you end up adding the missing features yourself.

Automatic process invocation

Automatic process invocation is an important feature for time-staged delivery systems. Three of the four types of applications for which time-staged delivery is appropriate (job networking, distributed database maintenance, and intelligent networking) require processes to be triggered because of the arrival of certain types of messages.

Even document distribution applications can be enchanced by having such a facility. A document distribution application could use this feature to invoke a filing process that would automatically file each incoming document in an appropriate folder.

Figure 3 compares the architectures of SNADS and Transfer at a high level. With the Transfer architecture, a user can be a person, device, or application. An application program, called a "client," interfaces to Transfer for the user. When the user is an application, it can be its own "client" and interface to Transfer directly. Clients interface to Transfer through a well-defined, high-level language called "Units of Work." There are more than 40 Units of Work available for clients to use (although most clients will not use all of them).

After Transfer has accepted the message from the sender's client, it sends that message through the network. At the destination node, another Transfer

SCITEC

DS1 SAT15 T-CARRIER MODEM

- Converts RS422/V.35 to full rate **1.544 MBPS DS1**
- CSU or line drive application
- Fully equipped with all connectors standard-MS34, DC37, DA15, COAX
- Front access controls and indicators, loop up/down diagnostics
- Loop, Independent or Station clocking
- **SAT20 CCITT 2.048 MBPS** option
- **NEW! SAT15 LD, 448, 224, 112K, 56 kbps**
- Even lower costs no frills SAT10 line driver

Scitec Corporation
850 Aquidneck Ave., Middletown, RI 02840
P.O. Box 30, Newport, RI 02840
(401) 849-4353 TLX 294104

CIRCLE 261 ON READER SERVICE CARD

Voice Digitizer
Model DVU-960

LNR's Model DVU-960 Voice Digitizer operates from 16 to 64 Kbps to provide voice communication over synchronous digital transmission facilities. This compact, cost-effective unit has the ability to operate full or half-duplex over terrestrial or satellite systems.

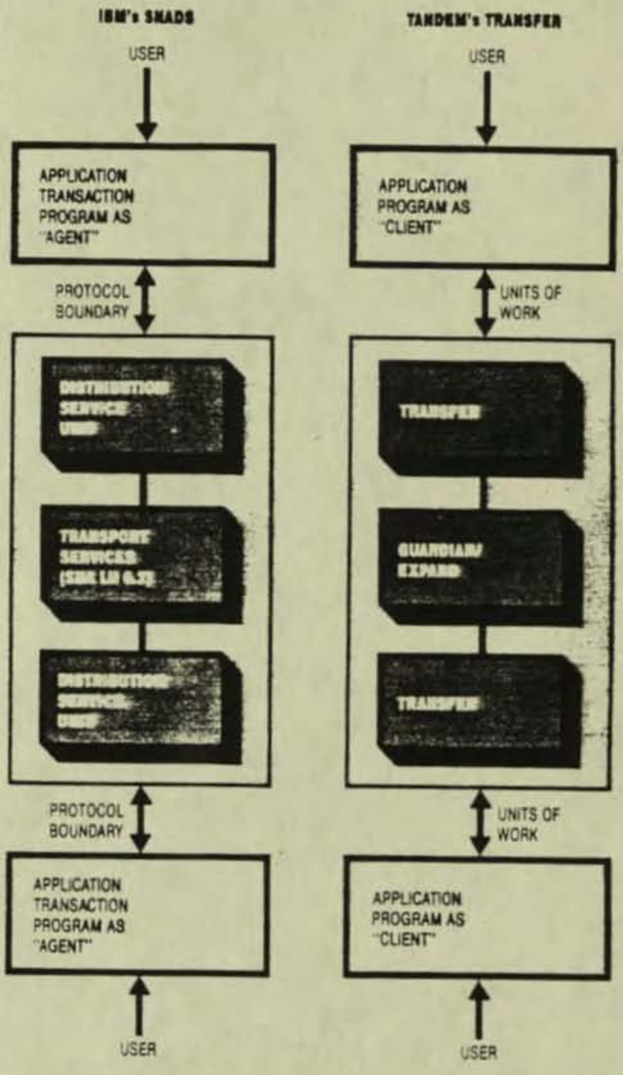
- **Natural Voice Quality with Speaker Recognition**
- **High Noise Immunity**
- **RS-232 C Digital Interface**
- **Real-Time Syllabic Compander**
- **Continuous-Variable Slope Delta Modulation**

For more information or a demonstration, call or write...

LNR COMMUNICATIONS INC 180 MARCUS BLVD. HAUPPAUGE, N.Y. 11788
TEL: (516)273-7111•TWX: 510-227-9871

CIRCLE 150 ON READER SERVICE CARD

3. Comparison. *The operational architecture of IBM's SNADS and Tandem Computers' Transfer are similar. Both are time-staged delivery networks.*


IBM's SNADS
TANDEM's TRANSFER
USER
APPLICATION TRANSACTION PROGRAM AS "AGENT"
APPLICATION PROGRAM AS "CLIENT"
PROTOCOL BOUNDARY
UNITS OF WORK
DISTRIBUTION SERVICE UNIT
TRANSPORT SERVICES (SNA LU 6.2)
TRANSFER
GUARDIAN/EXPAND

subsystem holds the messages until they are picked up by the recipient user's client.

SNADS performs document delivery between DISOSS applications; that is, when DISOSS needs to distribute a document outside its node, it uses SNADS. SNADS requires several other IBM architectures to distribute a document for DISOSS. It uses SNA (Logical Unit 6.2 sessions) to move the documents. The documents themselves use DIA/DCA (Document Interchange Architecture/Document Content Architecture) formats to define tabs, pages, paragraphs, centering commands, and so forth and to indicate what the recipient should do with the documents. ■

Jeri Edwards holds a master's degree in cybernetic systems from San Jose State University. Before working for Tandem Computers Inc., Edwards was PBX specialist for Rolm Inc. of Santa Clara, Calif.

Five dollars

Data Communications

McGraw-Hill's Magazine of Distributed Data Processing and Information Networks

April 1983

The different flavors of SNA compatibility page 89

1983 Harry R. Karp Award Winner page 108

Sandy Metz, Tandem Computers Inc., Cupertino, Calif.

The different flavors of SNA compatibility

A straightforward discussion of breaking into an SNA network. The author distinguishes between host access, device support, device emulation, and passthrough.

IBM's commitment to making systems network architecture (SNA) the basis for all its future data communications products has made SNA unavoidable—for both users and vendors alike. According to current estimates, roughly 70 percent of installed computers in large organizations are either IBM or IBM-compatible. Consequently, independent vendors are very likely to sell into an SNA environment—a likelihood that is increasing. With this rise in the importance and visibility of SNA has naturally come an increase in non-IBM products designed to work with SNA networks.

SNA is an architecture, rather than a single entity, that allows non-IBM vendors to design products that work in the SNA environment. It is unlikely that IBM will radically change SNA. Inevitably, SNA will continue to evolve, but its basic structure should remain stable so that the existing investment of the large IBM customer base will be preserved. To understand how SNA compatibility is achieved, and how implementations vary, requires a brief overview of the networking scheme.

Non-IBM vendors are aided in designing compatible products by the separation of function provided by SNA's layered architecture. This essentially permits products that work with SNA to manage a subset of SNA functional capabilities. A note of caution, however: Those layers not managed by the SNA-compatible product must be managed by the applications programmer. There are currently six SNA layers, which provide the following functions:

- Data link control—uses link-level protocols to transmit data between network nodes
- Path control—responsible for message routing and integrity between network-addressable units (NAUs)
- Transmission control—controls data sequencing and the rate at which data flows to a node
- Data flow control—deals with the logical organization of message flow, such as response mode, send/receive modes, and message grouping
- Presentation services—provide device-specific data formats
- NAU services manager—interfaces to the user or network manager.

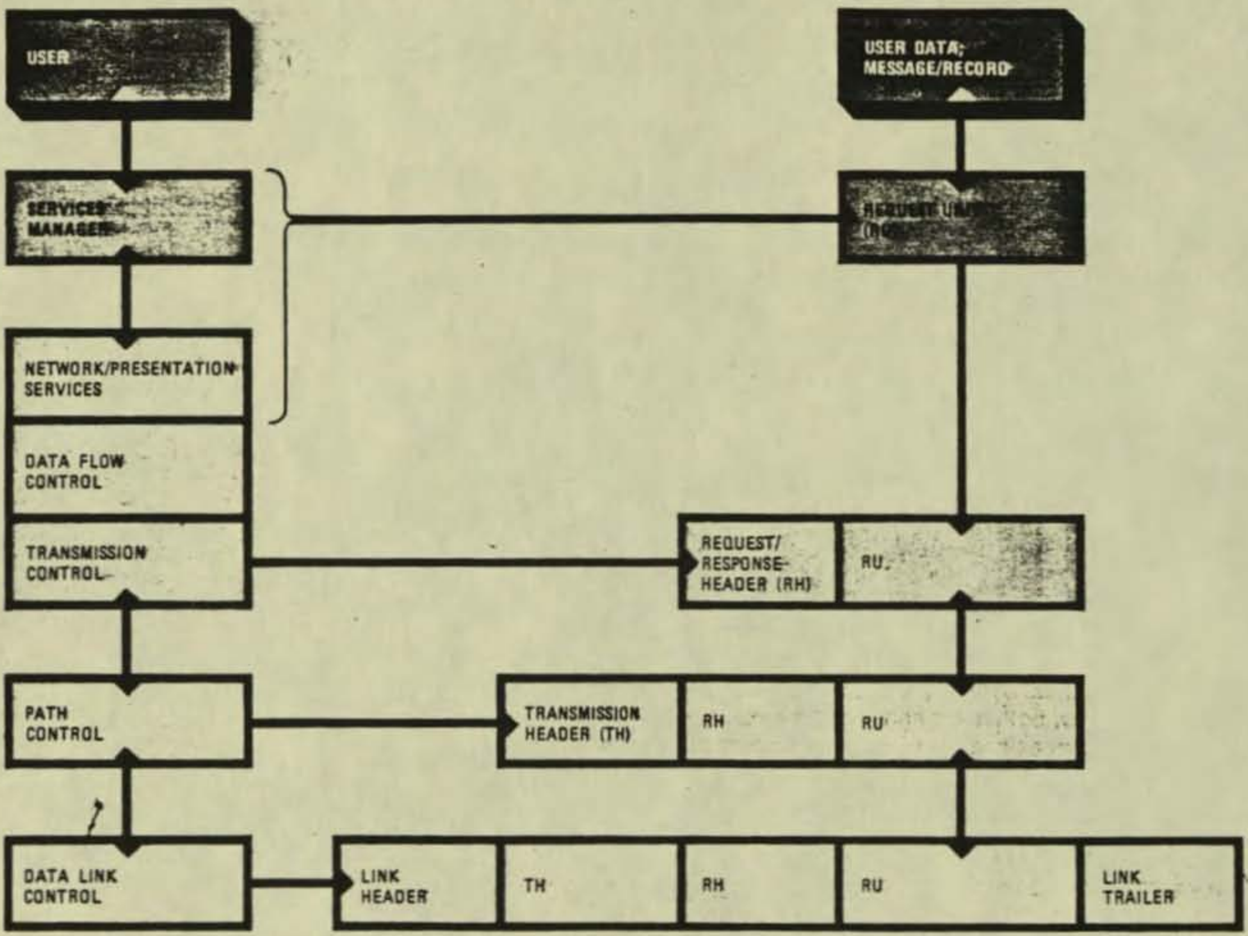
Within an SNA network, a message percolates through each of these layers at both the source and destination nodes. The most basic piece of SNA data, called a request unit (RU), usually contains the user application data (Fig. 1). The NAU services manager and presentation services layers interact with the RU by providing, respectively, user interfacing and device-specific data formatting.

The transmission control layer then adds a request/response header (RH) to the RU. This header contains transmission and data flow control information, including a request or response indicator, the RU type category, chaining information, bracket information, response type information, a pacing (flow control) indicator, and a sense data indicator (indicates the presence of user data).

The RU-RH combination is then passed to the path control layer,where a transmission header (TH) is added to the message. This header includes addressing information (source and destination), segmenting information, expedited or normal flow indicator, and sequence number information, and it may contain complex route flow-control information as well. The data link control layer then adds a link header (LH) and link trailer (LT) to the RU-RH-TH message. The LH and LT add the link level protocols necessary for transmission over common-carrier facilities.

The data link control layer manages the transmission

1. Requirements. *Any software-based SNA-compatible product must contain elements of the corresponding IBM SNA layers, though non-IBM vendors have more discretion in the complexity of the subsets they add to their products. This is necessary for the non-IBM applications to interact with IBM SNA programs.*



of the message from the source node to the destination node, where the headers are stripped from the message in the same fashion as they were added, until the basic request unit is all that remains. The more layers that can be managed by the SNA interface product—whether from IBM or an independent vendor—the less the applications programmer, or user, has to do.

Kingdoms

Figure 2 shows the SNA network topology, consisting of a "sub-area" network and a "peripheral" network. The differences are significant. For example, moving data from the peripheral to the sub-area network requires a 2- or 6-byte transmission header (depending on the source and destination node types), while moving data through the sub-area network requires a 10- or 26-byte transmission header (depending on the IBM software release being used).

Most SNA-compatible products currently on the market interface only as a peripheral-network node. An example of such a product is a terminal that emulates an SNA terminal, appearing to the SNA network as an actual IBM SNA terminal. To perform more-complex tasks, such as communications between IBM SNA applications and non-IBM SNA applications, it is necessary to interface with the sub-area network.

One of the few examples of a sub-area network interface is Tandem Computers' SNA gateway product, called SNAX (for SNA Communications Services), which interfaces with the sub-area network to perform IBM host-level device control. This interface manages the data link control, path control, and transmission control layers. The applications programmer need only provide the device-specific data format and the user interface to control terminal-node and cluster-controller-level devices from the non-IBM processor.

Messages are sent and received across an SNA network between logical network components (the NAUs). IBM's SNA defines three different types of NAUs: the systems services control point (SSCP), physical units (PUs), and logical units (LUs). All SNA-compatible devices need to be able to assume one or more of these roles.

The SSCP performs network management functions such as controlling the communications access to resources associated with the PUs and LUs. Somewhat less complex, a PU performs control functions for the device in which it is located and in some cases for other devices attached to the device containing the PU. Every node in an SNA network is associated with one physical unit type (see "Unit layers"). An LU is either a device (terminal) or a program (processor-resident) through which a user gains access to the data communications network. Figure 3 shows an architectural view of an SNA network and the relationship between these network entities.

Network-addressable units communicate with each other by establishing temporary logical connections called sessions. Session parameters, which define the manner in which data will be exchanged between NAUs, are established through a handshaking protocol.

To effect an LU-to-LU session, the SSCP must first establish an SSCP-PU session with each physical unit that is active in the network configuration. After this initial contact with the physical units, the SSCP then establishes SSCP-to-LU sessions with the active logical units of each physical unit. Only after this is completed can the LUs establish sessions with other LUs for normal data-traffic flow.

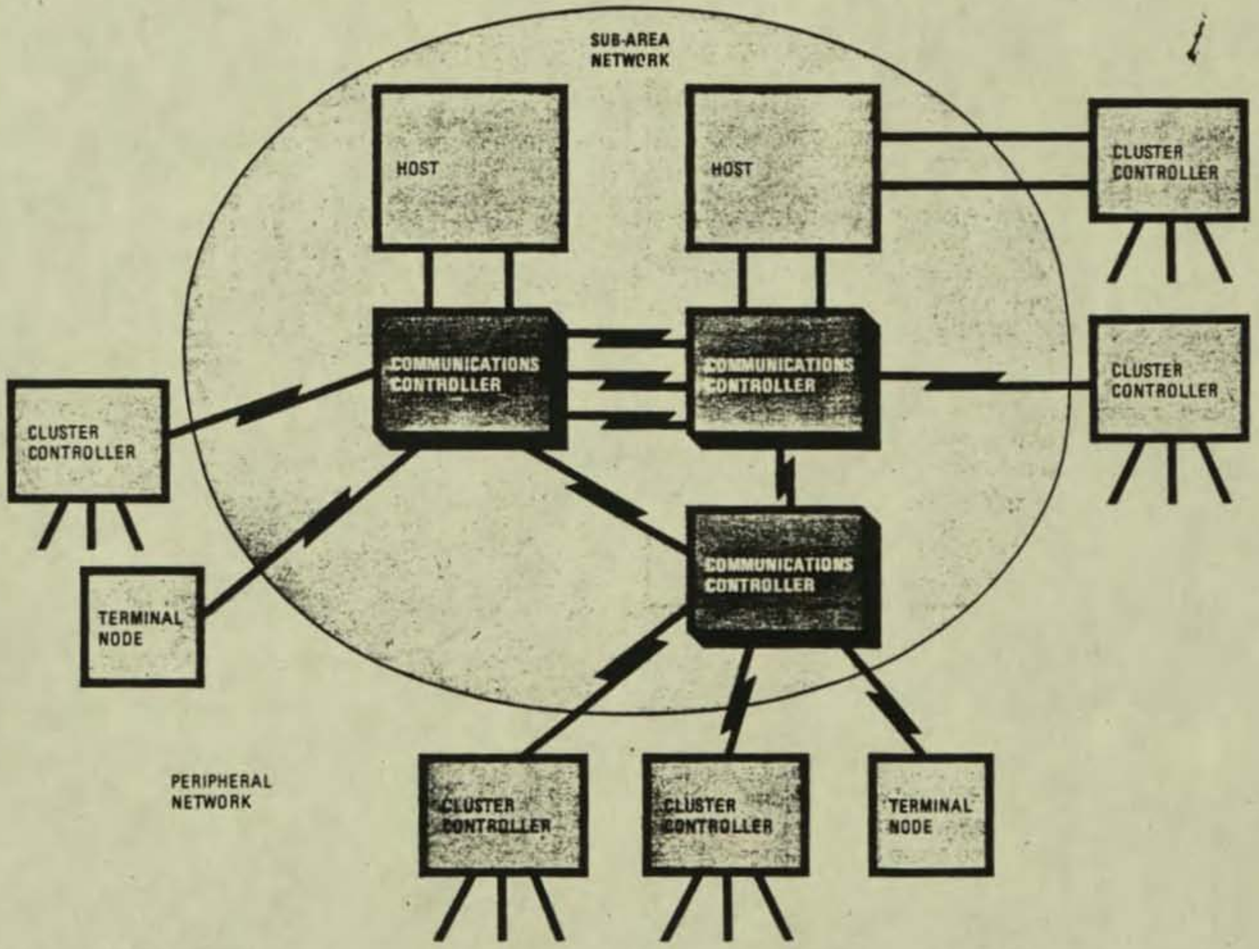
Non-IBM products are interfaced with SNA by establishing the proper sessions with IBM LUs in the SNA network. Each type of session is characterized by sets of parameters, or profiles, that define the functional subset of SNA capabilities required for that session. Non-IBM vendors can determine and implement one or more of these functional subsets to achieve differing levels of compatibility.

Penetrating SNA

There are several ways to gain access to SNA networks. One method is by writing a software-based interface for device control. In this case, an application program running on a non-IBM processor controls an SNA device.

Another method of accessing an SNA network is through a software-based interface for accessing a host. This method permits an application running on a non-IBM computer to communicate with an application

2. Demarcation. *SNA networks are divided into two major areas: the sub-area network and the peripheral network. Most SNA-compatible products interface only at the peripheral network level. For non-IBM applications to work with IBM SNA applications, interface in the sub-area network is necessary.*



Unit layers

Physical and logical units are identifiable in existing IBM products. Below is a listing and brief description of the different PU and LU types.

- PU type 1—a terminal node, such as an IBM 3271.
- PU type 2—a cluster controller node, which supervises the behavior of terminals (PU type 1) and other peripherals.
- PU type 3—not currently defined by IBM.
- PU type 4—a communications-controller node, such as an IBM 3705. These are typically front-end processors, which off-load from the host much of the work, such as interrupt handling, associated with data communications.
- PU type 5—a host node, typically containing a system services control point (SSCP). The physical unit type 5 is usually mainframe-based, although with new microprocessors some controllers are assuming more and more host-like functions.

LUs represent networking intelligence, usually software-based, in the SNA network. The LU types are distinguished by the degree of capability they afford users in network communications:

- LU type 0—provides a generalized user-definable process-to-process communications capability for user applications.
- LU type 1—defines the formats and protocols for SNA support of a remote-job-entry workstation when operating in batch mode. An LU type 1 supports the SNA character-set data stream.
- LU types 2 and 3—define the session rules for communications with interactive displays and hard-copy printers, typically 3270 terminals and other cluster-controller-attachable devices. LU types 2 and 3 support a different character-set data stream.
- LU type 4—defines session rules for communicating with the 3770 data processor and other SNA word processing devices.
- LU type 5—not currently defined by IBM.
- LU type 6—defines SNA process-to-process communications rules for distributed database environments.

running on an IBM (or compatible) SNA host. This "compatibility" provides access to SNA host databases, allowing the non-IBM-computer user to benefit from existing SNA applications programs.

Data General's DG/SNA software product (akin to Digital Equipment Corporation's) provides one such interface capability for its line of 32-bit processors. It allows applications running on an Eclipse computer to communicate with SNA host applications.

Designing a programmatic, or software-based, interface for SNA host access involves selection and implementation of the desired session services. This requires that a subset of every SNA layer be implemented. Once each layer is defined, the interface can then support varying levels of SNA complexity.

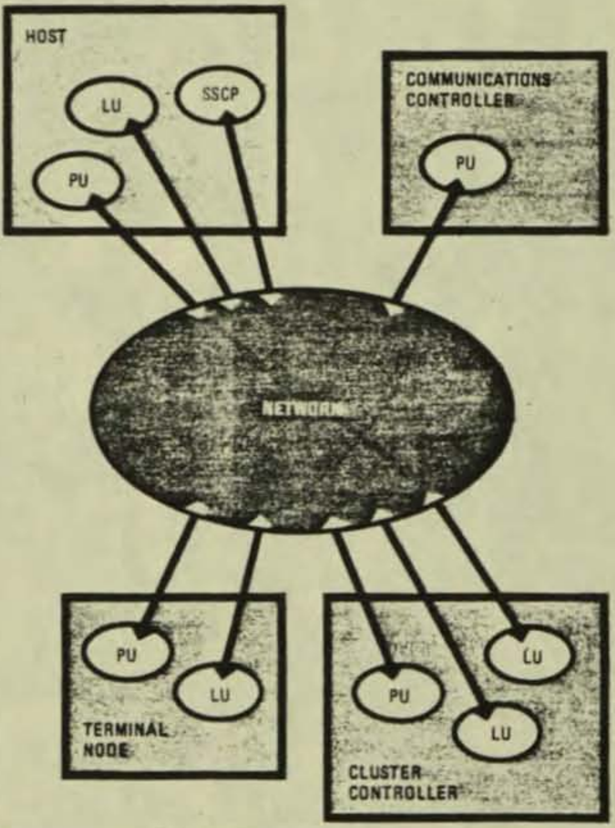
One special type of software interface for host access is used strictly for hardware emulation. This approach is used mainly where non-IBM terminals are made to appear to SNA hosts as IBM SNA terminals.

Compatibility degrees

Making terminal equipment SNA-compatible is the easiest type of host-access interface to implement, because it involves only the SNA peripheral network. Providing compatible terminal equipment allows a vendor to sell into the large existing SNA customer base without requiring a large investment in research and development. There are many companies offering this type of equipment (see table), although degrees of compatibility are widely disparate.

To emulate an SNA device such as a 3270 terminal, the vendor must address all SNA layers and all the functions within those layers that are appropriate to the types of sessions being used. Hardware functions such as keyboard input must be mapped to the non-IBM device, and protocol and format conversions must also be efficiently performed in order to offer complete compatibility. Protocol converters perform similarly,

3. Network addressable units (NAUs). *SNA-compatible products must assume the role of one or more "authorized" SNA entities: a systems services control point (SSCP); physical units (PUs); or logical units (LUs).*



ux

id

ine!

channel
ynchronous
MM

channel
ynchronous
MM

RK
PRODUCTS

pril 1983

SNA interface capability matrix

	PROCESSING CAPABILITY						TERMINAL CAPABILITY	
	SOFTWARE-BASED INTERFACE FOR DEVICE CONTROL		SOFTWARE-BASED INTERFACE FOR HOST ACCESS		AUTOMATIC PASSTHROUGH		COMPATIBLE TERMINAL EQUIPMENT	
	SNA 3270	OTHER	SNA 3270	OTHER	SNA 3270	OTHER	SNA 3270	OTHER
AT&T							✓	
BEEHIVE							✓	
CONVERGENT TECHNOLOGIES			✓				✓	
COURIER TERMINAL							✓	
DATA GENERAL			✓	GENERAL CLUSTER CONTROLLER				
DEC			✓	RJE	FROM VT100			
HARRIS			✓		✓			
HEWLETT-PACKARD			✓		FROM H-P TERMINALS			
IBM	✓	✓	✓	✓	✓	✓	✓	✓
INNOVATIVE ELECTRONICS							VT100 TO 3278 CONTROLLER	
LANIER			LIMITED		FROM LANIER TERMINALS			
LEE DATA							✓	
MEMOREX							✓	✓
MOHAWK DATA SYSTEMS			LIMITED		LIMITED			
NCR COMTEN					✓	✓		
RAYTHEON							✓	
TANDEM	✓	✓	✓	GENERAL CLUSTER CONTROLLER	✓	✓		
TELETYPE							✓	
TELEX							✓	
WANG			✓	RJE	FROM WANG TERMINALS			

NOTE: THIS MATRIX IS INTENDED AS AN OVERVIEW OF VENDORS PROVIDING SNA INTERFACE CAPABILITIES, WITH A FOCUS ON PROCESSING VERSUS EMULATION FEATURES. THE TABLE DOES NOT INCLUDE ALL VENDORS.

but are not included in this survey because their operation does not also support user applications.

Transparency

Passthrough, another SNA interface method, involves using a non-IBM computer or network to move information between an SNA host and an SNA terminal device (Fig. 4). This is particularly useful to users who have an existing non-SNA network and wish to retain that investment while at the same time being able to interface SNA devices and processors to it.

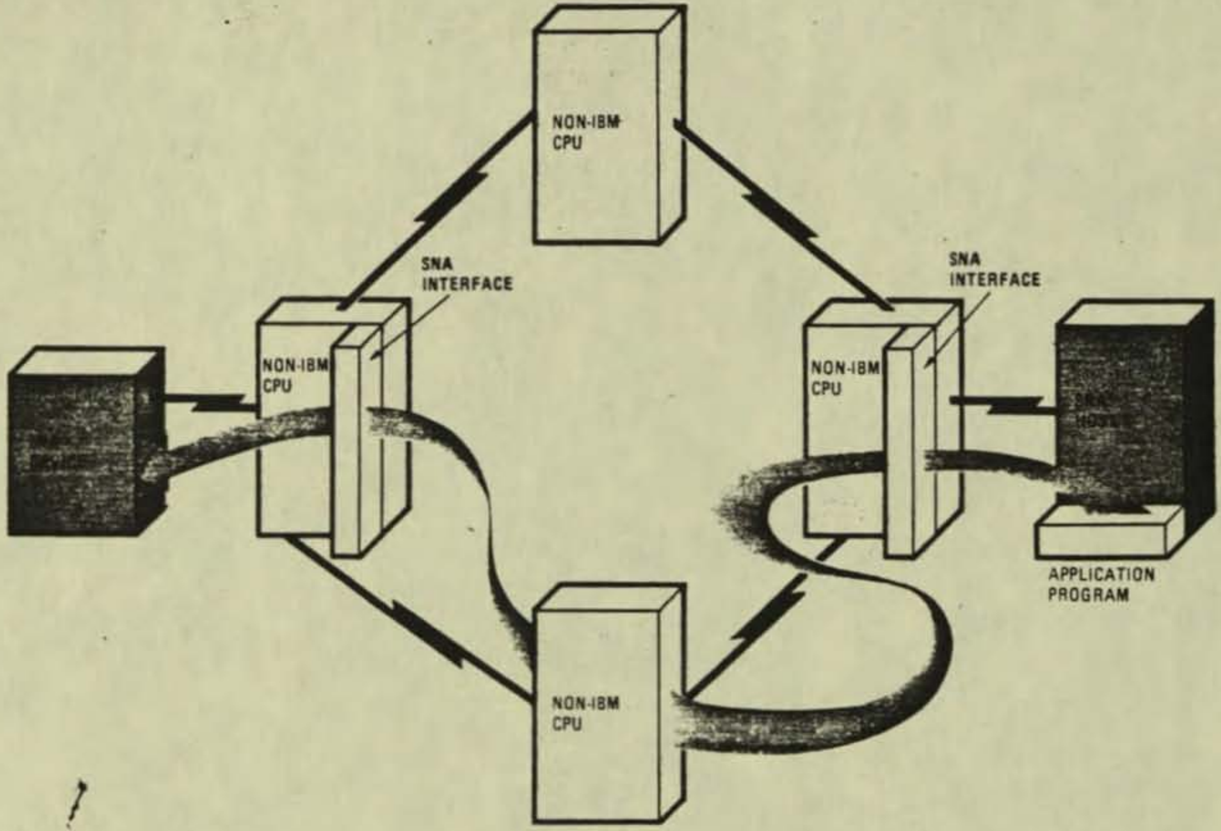
Passthrough software coexists with programmatic SNA interfaces. It connects users without requiring applications programs in the middle. Hence the user applications retain the use of the network.

Hewlett-Packard's DSN/Interactive Mainframe Facility uses the passthrough method to enable H-P terminals and printers connected to an HP3000 to behave as though they were IBM terminals or printers connected directly to an SNA host. A few passthrough products allow more than one vendor's devices to be used (see table).

Some SNA-compatible products combine these SNA interface capabilities. The software combines passthrough with programmatic interfaces both for host access and device control. This allows for the management of more-complex applications. For example, a single entry into an SNA electronic cash register, directly attached to a non-IBM processor, would be able to access credit-check applications automatically in an SNA host at a remote location.

Using non-IBM equipment with IBM hosts is not a

4. Passthrough. *Among the most complex forms of SNA compatibility, passthrough permits an IBM SNA device to access IBM SNA applications via a non-IBM network by combining SNA-compatible device support with SNA-compatible host access. Few passthrough products support more than one vendor's devices.*



new idea, and many major IBM customers use a variety of non-IBM gear to fulfill certain needs. Price is perhaps the main reason for this. Non-IBM vendors can often make a comparable product for less money, with performance equal to or better than that offered by IBM.

Smaller vendors also have more flexibility to customize products for particular applications. They have the resources and the time to create a product with the features and functions best suited to a particular application (and perhaps a particular customer). This flexibility also often allows the small IBM-compatible vendor to offer better delivery time.

Another boon to non-IBM vendors is that many users want to be able to buy software from a variety of sources. Many specialized vendors provide products and services that IBM does not offer.

Vendor selection

Naturally, choosing a vendor for SNA compatibility will be based mainly on the specific requirements of the customer's application. SNA software-based interfaces for general host access, device control, and passthrough capabilities are all available from more than one vendor. In many cases, however, a vendor can supply one capability but not the other. Another word of caution: The SNA-compatible product may be so closely tied to current IBM software releases that updating it—to keep up with mainframe software—could be a problem.

Once application needs have been determined and analyzed, the vendors must be studied. While technical SNA-compatibility details are certainly key, the capability and availability of the vendor's support organization and the level of customer training offered cannot be overlooked.

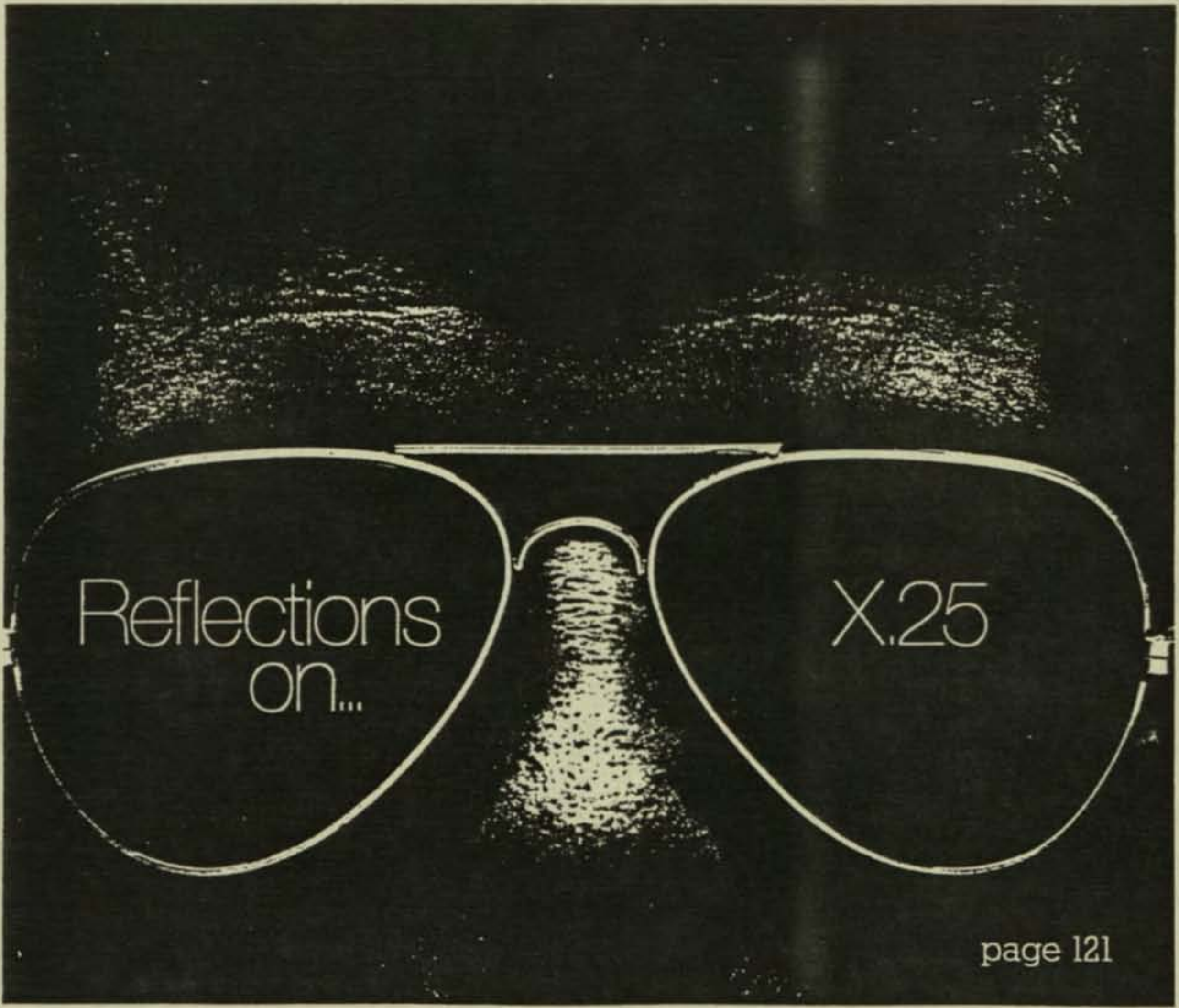
Yet another factor in evaluating a prospective vendor are offerings beyond SNA compatibility. It may be possible to springboard the investment into other areas, such as networking and database management. Seasoned users typically know that an integrated product, wherein the vendor has added SNA compatibility to other sought-after software or hardware, is generally quicker and easier to implement than trying to add SNA compatibility to existing non-IBM products themselves. ■

Five dollars

Data Communications

McGraw-Hill's Magazine of Distributed Data Processing and Information Networks

October 1983



Jim Gray and Sandy Metz, Tandem Computers Inc., Cupertino, Calif.

Solving the problems of distributed databases

True distributed databases—where dispersed records look to users as one unit, without centralized control—are now appearing. Here's how it is done.

Despite the increasing number of computers within many companies today, the full value of all this hardware—the potential return on investment—is often not realized because the diverse computing resources cannot share information. However, recent advances in the area of distributed databases (DDBs) are now making it possible for all corporate data to be accessible through a single resource.

Such schemes permit companies that have even widely dispersed data repositories to retain the advantages of locally controlled data. A true distributed database represents a decentralized scheme for data management wherein files are spread through a collection of autonomous nodes that communicate with one another via a common language. The purpose of such a decentralized database is to make all the data that is available to the corporation as a whole also conveniently available to individual users. This data availability can, for example, facilitate the local management of day-to-day tasks while also providing a basis at the corporate level for planning future strategies.

Though the nodes in a distributed database can exist in one room or building, these nodes are usually geographically separated. The DDB can therefore link a worldwide corporation into a single operating entity, with vital information available in a timely fashion wherever it is needed (Fig. 1). With a properly implemented distributed database, critical data can be stored, updated, and retrieved, independent of the location of either the data or the user.

The term "distributed" database has been used to describe some data management schemes that really offer only a subset of true distributed database capabilities. One example is a centralized database that is accessible from remote nodes. This can more precisely be called a shared database, which provides, in reality, only distributed access to centralized data. Another scheme features individual databases residing on computers that are linked in a network. While these are, in a literal sense, "distributed" databases, the data within each is still inherently centralized.

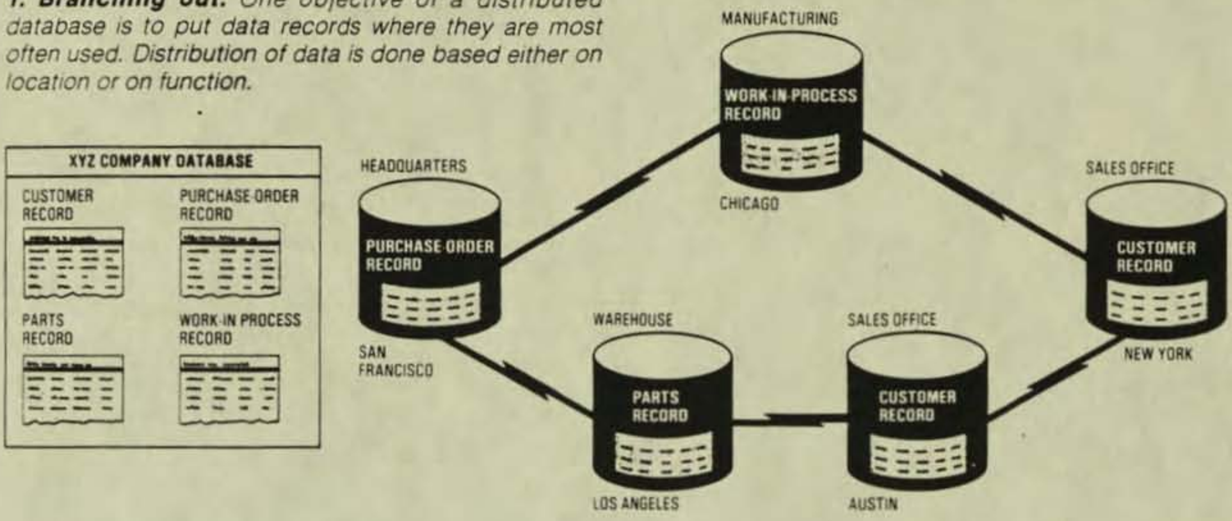
There are several technical considerations that make a truly distributed data management scheme attractive—the main one being the sheer size of many databases today. Linking diverse data files into a single resource often provides additional capacity that is increasingly hard to find with the single, centralized approach. A decentralized data management "network" could consist of literally hundreds of individual processors located around the world, with the data in each available to every node.

Only the data that is used on a daily basis need be kept at a local node; other useful information in the database is accessible remotely. In this way, data availability can be guaranteed by placing critical data at the local node. Naturally, placing data next to its most frequent users speeds response time in retrieving this data.

The autonomy of nodes in a distributed database allows each organizational entity to manage its information in its own way. And since each node is independent, and the data location transparent to the user community, the database configuration is modular and, therefore, flexible. Network nodes can be added, deleted, and rearranged without significantly affecting data access and usage.

From a management standpoint, linking data into a single resource provides a way to track the status of the corporation as a whole with convenient access to network-wide data. At the same time, control of local data resources can be kept at the divisional or depart-

1. Branching out. *One objective of a distributed database is to put data records where they are most often used. Distribution of data is done based either on location or on function.*



mental level. The existence of a corporate-wide database need not impact the efficiency of local data management and retrieval activity.

Another big plus to management is the flexibility provided by a decentralized data management scheme. The data distribution can be designed to reflect the changing needs of a business: When information needs change, the database can too.

Distribution design

Several non-issues with a centralized database, such as how the data will be distributed, become critically important in a decentralized environment. There are two main approaches to distributing data: Decentralize by function, or decentralize by location.

The selection of the best decentralization method is based on the particular application, or the way data will be used. If the data will typically be accessed repeatedly by the same users, then decentralization by function could be the more appropriate. Examples of this would be putting manufacturing materials lists at the appropriate manufacturing plants and customer information at sales locations.

Partitioning customer information on a node-per-region basis is an example of decentralizing by location. This method might be used for data pertaining specifically to a sales region or other geographically based entity within the corporation.

Another key issue that has to be resolved in evaluating the feasibility of a distributed database is the degree of decentralization. For example, function and maintenance of individual nodes can be decentralized while the operation and control of the collective database and network remains centralized. Or it may be preferable, depending on the situation, to further decentralize operation and control while keeping the design of the database and network architecture centralized. At the extreme, it may be desirable to decentralize everything, except the "global protocol" architecture.

An analogous example of maximum decentralization is the international telephone network. Each telephone company independently implements the common protocols of the international phone network (such as for dialing and billing), and the only centralized function is the architecture of these protocols. Within each company, design and architecture are typically centralized, while operation and control are delegated to the operating regions. These regions, in turn, delegate most operation and maintenance to the individual exchanges, which operate and maintain their own local hardware.

Searches

A major challenge in designing and managing a distributed database results from the inherent lack of centralized knowledge of the entire database. It is difficult and often undesirable to maintain information concerning the entire database in any one place, but this requirement seems inevitable in order to manage requests such as, "Where is file A?"

One solution to this dilemma involves the concepts of global, local, and semiglobal data. Global data is information that is common to and shared by all sites. Examples of global data are an item master file of parts that comprise a company's parts catalog and a bill-of-materials file that describes a product's structure.

Local data is information that is uniquely important to the individual site using it, although it is accessible to all sites. Examples of local data are items in stock and work in process. Local data retains the same format as corresponding data has at other sites.

Semiglobal data is used in internodal—and often intersite—transactions. This might be the case for, say, an interplant materials transfer. In this case, a request by one site for materials from another is placed, processed, and monitored. The process requires that all data and status information pertaining to the request be resident at both sites. But since this information is of no use to any third party, it is duplicated only at the two nodes that use it.

Information is made available to the entire network by partitioning or replicating the data files. Partitioning a data file means splitting it into records and then distributing the records so that each record resides at exactly one network node (Fig. 2A). Replication means duplicating data records at more than one node (Fig. 2B). Local data can be partitioned, but global data must be replicated.

Data is partitioned to put it close to the sites that use it. An example might be storing bank account data at the home branch of the bank customer. This has the effect of reducing message traffic and message delay, and of distributing work. In the case of an airlines reservation network, data is partitioned by corporation. Most transactions submitted by one airline deal only with that airline and therefore run on a single node. Transactions that deal with other airlines are routed to other airlines' nodes, as appropriate.

Replication also serves the purpose of bringing data closer to the user, and has long been used to improve data availability. If one copy of a file is lost, for whatever reason, another can be accessed at a remote node. Global data is replicated at all sites. In a geographically distributed database network, replication also provides the benefit of improving response time by eliminating long-haul message delays.

Updating

Partitioned data is most efficient when the data must be kept current, which generally means that it is updated frequently. The single copy of each data item makes updating an efficient process. However, nonlocal "read" operations are more expensive, making partitioning less efficient for data that is widely used but updated infrequently. In the Tandem scheme, a database record manager allows files to be partitioned among network nodes based on single field values within files, such as "part number" or "customer name."

Replicated data is most efficient when multiple reads of the data are expected, but updates are not as frequent. The data is duplicated at nodes where high-volume reads are expected, producing high availability and good response time. When replicated data must be updated, however, an update to a record at one node should cause an identical update at all other nodes where that record resides. If any one replica is unavailable, there could be problems.

A variety of schemes can be employed for updating replicated data, even though the copy of the record may be temporarily unavailable at one or more of the nodes. One technique requires that a majority of the replicas be read and updated as part of each transaction, though the definition of "majority" varies with the application. This scheme has the advantage of tolerating some nodal unavailability, but it is not practical for either very small or very large networks.

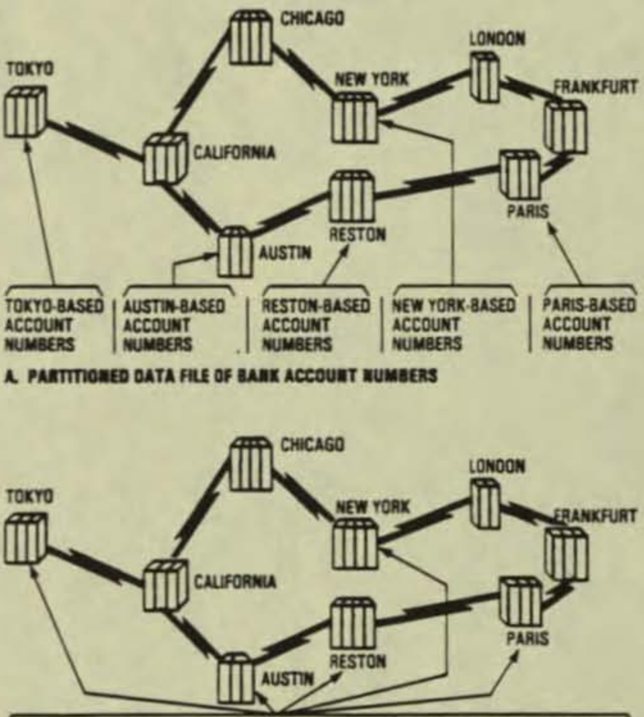
In a very small network of, say, two nodes, having either node unavailable prevents an update of a majority of the nodes. In larger networks, delays in completing the update transaction are proportional to network size: As the network grows, transactions will take longer to complete. One example of a file manager that uses a majority-update scheme is an experimental database network built at Xerox Research (see references for additional information).

Another method for updating replicas is the "as soon as possible" (ASAP) method. This technique involves designating one replica, the "master copy," on either a record-type or case-by-case basis, which ensures that the file at its node is updated. The updates are then asynchronously sent to the other replicas. This approach sacrifices consistency for availability and response time. Tandem's internal distributed database application, called Empact, is one that uses ASAP updates for frequently used data, and consistent updates for critical data.

A different method involves a time-based technique, in which there is a master copy of the data record, and its replicas (or slaves) are "snapshots" of the master as of a specific time. The slave copies are periodically updated, and each replica is "time-stamped" to indicate its degree of currency. This technique is appropriate for files that change very slowly and for which currency is not critical to business operations. IBM's experimental "R" System provides this time-stamping of replicas.

When retrieving the time-stamped replicas, the degree of currency can be specified in the query. It may

2. Replication. *Local data files may be partitioned (A) at the same site. Global files, on the other hand, are replicated in each network node (B).*


CHICAGO
LONDON
TOKYO
NEW YORK
FRANKFURT
CALIFORNIA
PARIS
RESTON
AUSTIN
TOKYO-BASED ACCOUNT NUMBERS
AUSTIN-BASED ACCOUNT NUMBERS
RESTON-BASED ACCOUNT NUMBERS
NEW YORK-BASED ACCOUNT NUMBERS
PARIS-BASED ACCOUNT NUMBERS
A. PARTITIONED DATA FILE OF BANK ACCOUNT NUMBERS
CHICAGO
LONDON
TOKYO
NEW YORK
FRANKFURT
CALIFORNIA
PARIS
RESTON
AUSTIN
TOKYO-BASED ACCOUNT NUMBERS
AUSTIN-BASED ACCOUNT NUMBERS
RESTON-BASED ACCOUNT NUMBERS
NEW YORK-BASED ACCOUNT NUMBERS
PARIS-BASED ACCOUNT NUMBERS
B. REPLICATED DATA FILE OF BANK ACCOUNT NUMBERS

not always be necessary to read the most current copy, so some time and communications costs might be saved by reading a copy that is physically closer, but with an older time-stamp.

Relational

With data distributed all about a network, the retrieval method must be convenient and fairly simple to the user. This means that the database manager must keep track of all data locations in a manner that is transparent to the user. This requirement, combined with the flexibility needed to move data from node to node as information requirements change, makes a relational model almost a necessity in a distributed database environment.

A relational database stores data in two-dimensional tables of rows and columns containing related information (Fig. 3). Information is entered into the database by creating the tables and filling them with pertinent data. Expanding the database is a matter of adding new tables or adding new entries to existing tables.

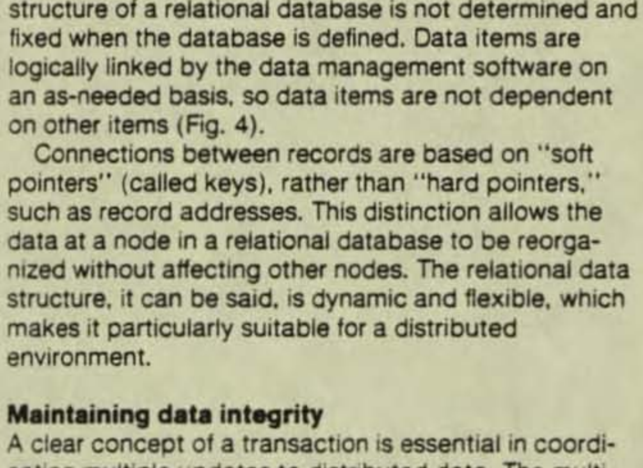
Unlike hierarchical and network databases, the structure of a relational database is not determined and fixed when the database is defined. Data items are logically linked by the data management software on an as-needed basis, so data items are not dependent on other items (Fig. 4).

Connections between records are based on "soft pointers" (called keys), rather than "hard pointers," such as record addresses. This distinction allows the data at a node in a relational database to be reorganized without affecting other nodes. The relational data structure, it can be said, is dynamic and flexible, which makes it particularly suitable for a distributed environment.

Maintaining data integrity

A clear concept of a transaction is essential in coordinating multiple updates to distributed data. The multiple nodes and multiple copies of data items can mean distributed chaos if transactions are not carefully imple-

3. Relational. *A relational database differs from the hierarchical database in that common elements in the file permit records to be logically connected.*



CUSTOMER

CUSTOMER NO.	NAME	CITY	CREDIT LIMIT
101	ACME	NY	500
102	SMITH	ATLANTA	1,000
218	JONES	SF	1,000

ORDER DETAILS

ORDER NO.	STOCK NO.	QTY.
123	A43	10
123	G04	100
124	E98	50

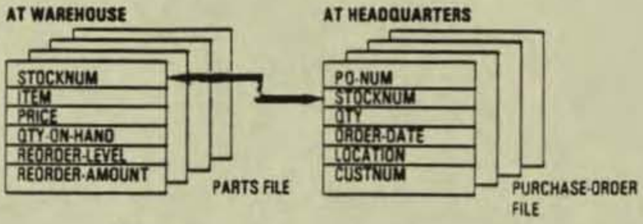
PURCHASE ORDERS

P.O. NO.	STOCK NO.	QTY.	ORDER DATE	CUST. NO.
27560	A43	10	1/10/82	102
13892	A76	43	1/30/82	101
42610	C61	2	2/2/82	216

PARTS

STOCK NO.	ITEM	PRICE	QTY. ON HAND	QTY. ON ORDER
A43	WIDGET	2.98	200	10
A76	TRIBBLE	4.38	[illegible]	500
C61	[illegible]	[illegible]	[illegible]	0

4. Linking up. *In this DDB transaction, a warehouse parts file is linked with a headquarters purchase order file to determine how many items are in a particular order.*



QUERY:

```
OPEN PARTS, PURCHASE ORDER
LINK PARTS TO PURCHASE ORDER VIA STOCKNUM
LIST BY STOCKNUM
     ITEM,
     QTY-ON-ORDER,
     ORDER-DATE,
     WHERE LOCATION = "SAN FRANCISCO";
```

RESULTS:

STOCKNUM	ITEM	QTY-ON-ORDER	ORDER-DATE
A76	TRIBBLE	500	05-19-82
G04	BLIVET	500	05-19-82
H73	WHATSIT	1,000	04-28-82
K88	BLURB	400	03-08-82
M36	PLAZZER	50	04-15-82
T05	KRUPUS	535	02-17-82

mented and monitored. A transaction is an operation in which application procedures, such as banking operations, are mapped into transformations (by executing programs) that invoke database actions. These include: Read the customer, account, and teller records; write the account, teller record, and a memorandum record; and send response messages to a terminal. The result of this process should be that the database is moved from one consistent state to another.

The key properties of a transaction are:

- Consistency—the transaction is a consistent transformation of the database state (for automated teller or banking transactions, that money is neither created nor destroyed)
- Atomicity (transactions are "atomic")—either all the actions invoked by the transaction occur, or else the entire transaction is nullified (in the banking case, that no account is left in a partially updated state)
- Durability—once a transaction is completed, its effects cannot be nullified without running a compensating transaction (funds removed from an account would have to be redeposited to be accessed again).

All of these criteria and requirements must be upheld uniformly across the network in order for a distributed database to work. Database management packages that consider a single database action to be a transaction, therefore, are unsuitable for a distributed environment.

There are several techniques available for maintaining consistency, atomicity, and durability in a centralized environment, including concurrency control and transaction backout (reversing the effect of a partially completed transaction). These techniques can also be applied in the distributed environment, but their management on a network-wide scale becomes much more complex due to the added communications considerations.

To ensure database integrity in a distributed transaction, all messages between nodes must arrive safely, and the sending node must be made aware that each message has in fact arrived. Both requirements can be met by using a "two-phase commit" protocol.

"Committed" transactions

A two-phase commit protocol uses a commit coordinator program to centralize the decision to commit or abort a transaction. The commit coordinator has a communications path to all the participants of each transaction. These participants, it should be noted, can be processes, autonomous components within a process, or both.

The commit coordinator asks all the participants to enter a "prepare" state, from which each participant can either commit or abort its part of the transaction. Once all participants are in the prepare state, each will transmit a message indicating this to the commit coordinator, which in turn can send a commit or abort message to all the participants (Fig. 5).

Once the commit coordinator sends the commit message, it waits for an acknowledgment from each participant before terminating the transaction. Use of this two-phase commit protocol helps ensure the integrity of a distributed transaction.

Distributed administration

Management of a worldwide database must be both distributed and centralized. Certain aspects of the database are common to the entire network and therefore must be designed and controlled by a central organization. A prime example of this is the global record format.

Local database functions can be controlled at the local node to provide site autonomy, which is one of the basic goals of a distributed database. An example of one such local function is a report format.

A hierarchy of control can therefore be imposed, with network-wide functions being managed by a central organization and control of other database activities being distributed in a hierarchical fashion. The key requirement, however, is that each level use the protocol of the global architecture for all its inputs and outputs. Each organizational level has an administrator, who publishes and controls the protocols of his component of the database network. And while a great degree of autonomy can be exercised, the structure of levels and control at this level should parallel the structure of the overall organization.

DDB selection

The choice of a distributed database management system is naturally dependent on the application requirements. However, care should also be taken to implement sufficient flexibility into whatever database network is constructed, to account for rapidly changing application requirements.

Requirements of a true distributed database include the ability to distribute data files between at least two computer nodes; to provide location transparency between data and users; to retain data file relationships (even when the files are located at separate network nodes); and to ensure transaction integrity in the distributed environment. The two commercially available distributed transaction management systems that most closely meet these requirements are IBM's CICS/ISC and Tandem Computers' Encompass. The ISC feature of IBM's CICS provides for distributed transactions and the ability to access remote files, but it does not transparently handle data partitioning or replication.

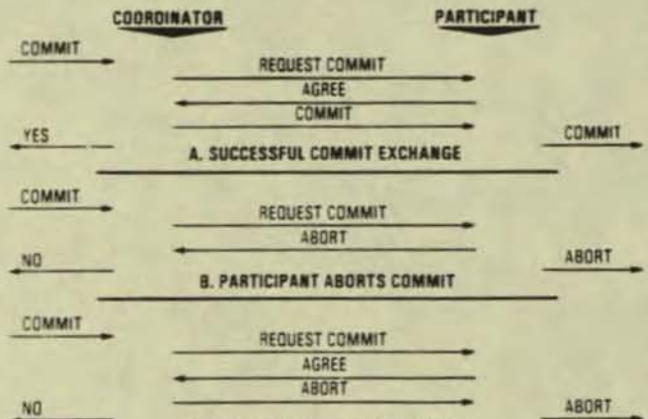
Data partitioning requires direct action by an operator with IBM's CICS/ISC, while this is done automatically—transparently to the operator—with Encompass. Manual intervention is also required with the IBM product for data replication, but Encompass requires manual intervention only for files resident on a remote node.

Another selection criterion is flexibility, since one of the purposes of a distributed database is to allow for the changing information needs of a corporation. The ability to add nodes, delete nodes, and reconfigure the distribution of data without changing application programs is a requirement.

Inherent in all of these requirements are a reliable data communications and networking capability, and the use of a relational database model. Without this base on which to build, no distributed data management network can be successful.

Beyond these basic requirements are some features that will enhance the usefulness of the database throughout a lifetime of changing requirements. One way of achieving this goal is through the use of highly reliable hardware and network software. Even though the database must be designed so that a failure at one node cannot prevent access to critical data, the distributed network will be much more efficient if extraneous hardware and software failures can be kept to a

5. Commitments. *A dialog between the commit coordinator and a participant (A) ensures that transactions will be completed. The commit coordinator has a path to all participants, any of which may abort (B and C).*



minimum. The ideal, of course, is to maintain data availability in the face of component failures or temporary inaccessibility of some network nodes.

Evolution

The linking of highly reliable computers into a single distributed database is not easy, but progress in this area and the availability of proven products is making this once blue-sky objective possible to achieve. That computer networks will move in this direction is inevitable.

Information management schemes, now computer-based, are replacing traditionally paper-based ones. But these earlier operations were not totally inefficient—the paper was invariably located at the point where it was most often used. The move to centralized data management procedures changed all that, though it came about more from a need to optimize expensive computing resources in the earlier days of computer technology than from the desire to centralize information resources.

With the cost of hardware rapidly decreasing and the reliability of data communications steadily increasing, the time has come to return to an information management operation that puts the data back where it is needed, as long as it can be done without sacrificing the advantages of a centralized database. Distributed databases are therefore the logical continuation in the evolution of computer usage for information management. And this evolution has been considerable: from compact data storage, to early database management systems, to the on-line access of centralized data, to remote data processing, and now, finally, to the distributed database management system, which promises to provide accurate and consistent data to all users, acceptable response time, and availability—even through otherwise catastrophic communications and hardware failures. ■

■

Additional reading/references:

Gifford, D.K., "Weighted voting for replicated data," *ACM Operating Systems Review*, 13.5, December 1979, pp. 150-162.

"Information storage in a decentralized computer system," Xerox Publication CLS-81-8, Palo Alto, Calif., 1981, pp. 21-75.

Houston, George B., "Tightening up software on a distributed database," *Data Communications*, April 1983, p. 133.

AN ARCHITECTURE FOR HIGH VOLUME TRANSACTION PROCESSING

Robert W. Horst
Timothy C. K. Chou

Reprinted from
IEEE PROCEEDINGS OF THE 12TH ANNUAL INTERNATIONAL
SYMPOSIUM ON COMPUTER ARCHITECTURE,
BOSTON, MASSACHUSETTS, June 17-19, 1985

AN ARCHITECTURE FOR HIGH VOLUME TRANSACTION PROCESSING

Robert W. Horst - Timothy C.K. Chou

Tandem Computers Incorporated - 19333 Vallco Parkway - Cupertino, CA USA

ABSTRACT

This paper describes a commercially available multiple processor system that delivers mainframe class performance for high volume transaction processing. Multiple systems of up to sixteen processors each are configured in a ring structure using fiber optics. This structure allows from two to over two hundred processors to be applied to a single on-line application. We discuss the tradeoffs in selecting a large number of small processors versus a small number of large processors. Benchmark results are presented to demonstrate the linearity in system performance as processors are added.

1.0 INTRODUCTION

In recent years, the market for high volume transaction processing has increased rapidly. In the 60's, only large airlines required large on-line transaction processing (OLTP) systems. These requirements were filled by centralized mainframe computers running specialized, highly tuned applications. They suffered from limited expandability, a costly applications environment, and the requirement to program in assembly language [5].

Today, many other industries are taking advantage of OLTP systems. Some of these applications include on-line banking, credit authorization, debit cards, teletext, medical information systems and paperless factories. These markets have similar systems requirements: the system must continue to operate despite hardware failures, it must be expandable, and it must be capable of high transaction throughput.

One example of a high volume transaction processing environment is the TWA Reservation System (PARS) [5]. This system is composed of a high performance mainframe connected to over 10,000 terminals. The system supports a peak throughput in excess of 170 transactions/second while maintaining an average response time of 1.5 seconds.

In 1976 Tandem introduced a new system architecture which was specifically designed to address the problems of OLTP. Designated the NonStop I, this system consisted of from two to sixteen loosely coupled processors which communicated with each other over dual high speed busses [2,10]. The Tandem system architecture is illustrated in Figure 1. The loosely coupled architecture has proven to be quite effective for transaction processing. It supports incremental expansion, high availability, and high performance [8]. The loose coupling does not limit performance since transaction processing, unlike most scientific processing, is easily partitioned into multiple relatively independent processes.

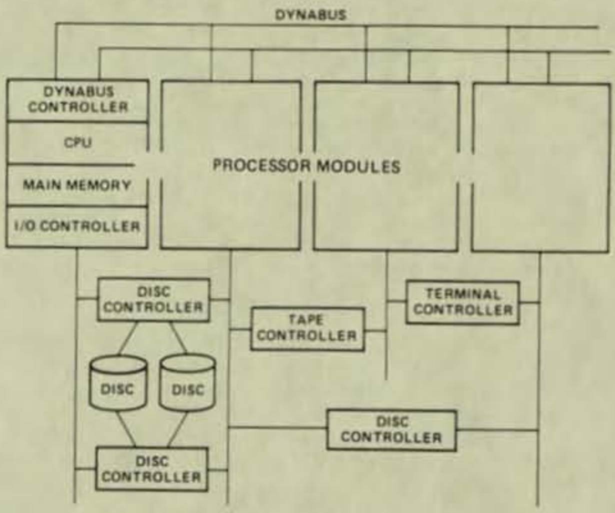


Figure 1. System Architecture

In 1981, the NonStop II was introduced to remove addressing limitations of the 16-bit NonStop I. In the intervening five years, many new features have been added to the basic offering. The system was expanded into a network of up to 255 systems, a relational

0149-7111/85/0000/0240$01.00 © 1985 IEEE

data base management system as well as transaction management software [1,3,4] were also added. These products allowed users to develop distributed fault tolerant OLTP applications without worrying about the underlying system implementation..

The system as it stood in 1981 solved all of the requirements for OLTP, but the performance required for some large applications was beyond the reach of a 16-processor system. The network provided a way to apply more than one system to a single application without reprogramming; however, the relatively slow speed of data communications lines and software overheads of long-haul communication protocols proved to be a bottleneck in high volume applications. It was apparent that there was a market need for systems of larger than 16 processors. The need was addressed by starting projects both to expand the number of processors which could be applied to a single application, and to develop a new higher performance processor.

2.0 FIBER OPTIC EXTENSION (FOX)

The most obvious way to increase the number of processors in a system is to extend the high-speed interprocessor busses to handle, say 32 or 64 processors. While this is not difficult from a hardware design standpoint, there are drawbacks. All processors would be required to be in close physical proximity in order to keep from degrading the bus performance. This would cause a physical space problem in some computer rooms. In addition, allowing a single system to expand to more processors does not help existing customers that already have several systems requiring higher bandwidth communications.

An alternative approach to adding processors in the same system is to use a high speed connection between systems. This effectively adds another level of interconnection hierarchy between the 26 Mbytes/sec inter-CPU links, and the 56 Kbyte/sec network links. Figure 2 illustrates the Tandem solution which uses fiber-optics to link up to fourteen systems. Each node can accept or send up to 4 Mbytes/sec. With this additional bandwidth, a cluster of up to 224 CPU's can be configured to handle a single on-line transaction processing application.

Fiber optic links were chosen both to solve technical and practical problems in configuring large clusters. Since

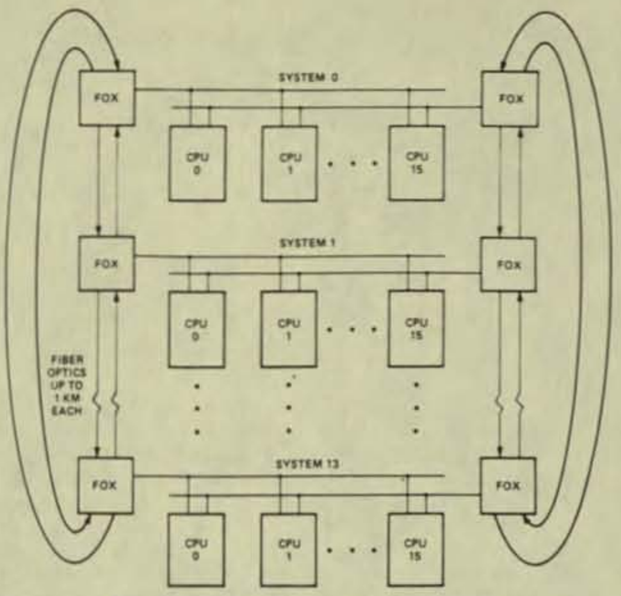


Figure 2. FOX Architecture

fiber optics are not susceptible to electromagnetic interference, they provide a reliable connection even in noisy environments. They also provide high bandwidth communications over fairly large distances (1 km). This eases the congestion in the computer room and allows many computers in the same or nearby buildings to be linked. Fiber optic cables are also flexible and of small diameter, thus easing installation.

The topology of the FOX connection is a store-and-forward ring structure. Four fibers are connected between a system and each of its neighbors. Each interprocessor bus is extended by a pair of fibers, which allows messages to be sent in either direction around the ring. The four paths provided between any pair of systems assures that communication is not lost in the event that one entire system is disabled (due to a power-fail perhaps) or in the event an entire four fiber bundle is severed. The ring topology also has advantages over a star in that there is no central switch which could be a single point of failure. In addition, cable routing is easier with a ring than with a star.

In a ring structure, bandwidth increases as additional nodes are added. The total bandwidth available in a FOX network depends on the amount of passthrough traffic. In a 14 node FOX ring if each node sends to all other nodes with equal probability the network has a usable bandwidth of 10 Mbytes/sec. With no passthrough traffic the bandwidth increases to 24 Mbytes/sec. Theoretically an application generating 3K bytes of traffic per transaction ,at 1,000 transactions per second, would require a FOX ring bandwidth of only 3 Mbytes/second. This would put total utilization of the FOX network at less than 30% of the total available bandwidth.

4.0 PROCESSOR SIZE

Once a system architecture can apply a large number of processors to a single application, there is still a question of what the characteristics of each processor should be. The possibilities span a large range of performance levels. At the low end of the performance range are microprocessor based designs. These may be based on one chip microprocessors such as the Motorola 680xx family, the Intel 8086 family or the National 32000. In the midrange are designs based on medium-scale integration or gate arrays. Such designs are typified by mini/supermini computers such as the Digital VAX series, Hewlett-Packard 3000 and the IBM 43xx series. At the high end are designs based on the most aggressive IC, packaging, and cooling technologies. Examples of this type of design are mainframes such as the IBM 308x, Amdahl 470 and 580, and high end machines from Sperry, Burroughs, CDC and Cray.

Many factors need to be analyzed in order to decide whether the micro, mini, or mainframe type design should be preferred for the processors in a high volume transaction processing system. These factors include cost-performance, granularity of fault tolerance, granularity of adding performance, and ease of managing the system.

4.1 COST-PERFORMANCE

In a system which is modularly expandable, cost-performance is the driving force in the development of a new processor. Improving cost-performance by lowering costs is difficult. Even using a microprocessor, which may be nearly free, may not significantly reduce costs over a minicomputer-style design due to the many fixed costs which make up a system. The cost of main memory, packaging, power and cabling are not reduced in proportion as the CPU cost is reduced.

It is easier to improve cost-performance by increasing the performance of the processor. This would seem to favor a mainframe as the best choice for a multiprocessor system. There are several reasons why this is not the case. The complexity of a mainframe design requires a much larger development cost and longer development time. In addition, in trying to improve uniprocessor performance some nonlinearities are encountered. For instance, the jump from air to liquid cooling constitutes a large cost increment.

4.2 OTHER FACTORS

Other factors influence the choice of processors. If the processor is too small, the number of processors required to perform a large application can become so large as to be unmanageable. In addition, if the system is not carefully designed, performance improvements can cease to be linear beyond some number of processors.

On the other hand, if the processor is so powerful that the application can be handled by a single processor, fault tolerance can suffer. Today the only fault tolerant configurations of mainframes require an expensive duplicated hot standby. Even if these machines were incorporated into a Tandem-like structure which would allow the second machine to do useful work, the failure of one of the mainframes would remove half the computing power from the system. In extremely critical applications, it may not be tolerable to degrade performance while a hardware failure is being repaired. If the application requires only one processor to handle the peak load, a second processor is needed in case of failure, for a 100% overhead. In contrast, if four less powerful processors are used to handle the same workload, only one extra processor is needed in case of failure for a 25% overhead.

5.0 NONSTOP TXP

Taking the above reasons into consideration, Tandem developed the NonStop TXP processor, which was introduced in 1983. Its primary design objectives were to improve both cost-performance and absolute performance over the NonStop II [7]. The initial pricing of the NonStop TXP offered a 50% improvement in price-performance over the NonStop II at about 2.5 times its performance.

In the NonStop TXP design, the emphasis on cost-performance extended all the way to the component level. One of the first decisions to be made was the selection of static RAM's to be used in the cache memory and control store. The most advanced RAM's at that time were organized as 4Kx4 and 16Kx1 bits with access times of 45ns. These were four times the density of and 10 ns faster than the RAM's used in the NonStop II.

To implement logic functions, advanced MSI Schottky technology and programmable array logic were chosen. An extensive analysis of LSI alternatives, including many different gate arrays, showed that a gate array machine would have been about the same performance level, higher cost, and would have required a much longer development cycle.

Once a technology was chosen, the next challenge was to develop an architecture which could utilize the improved components to improve performance and cost-performance. One such area was the cache memory design. Although extensive academic research in cache memories was available during the NonStop TXP design [11], most of the studies did not anticipate the impact of large RAM's on cache organizations. Using 16K static RAM's, a 64K byte cache requires only 32 components (not including parity or the tag comparison RAM's or logic). This makes it much more economical to design a large "dumb" cache (direct mapped) than a smaller "smart" cache (set associative). After performing some real time emulation of different cache organizations, the final cache design for the NonStop TXP was chosen. It is a 64K byte direct mapped virtual cache with hashing of some of the address bits. Hit ratios for the cache have been measured between 96% and 99% while performing real transaction processing workloads.

Other tradeoffs were also made in the interest of cost performance. In order for the NonStop TXP to be plug-compatable with the NonStop II processor, the CPU was required to fit on four circuit boards. Had it overflowed those boards, a large jump in cost would have occurred. For this reason, the NonStop TXP relies on microcode to perform some of the functions done in hardware on many machines. For instance, after every cache access, the microcode must perform a conditional branch to test for a cache miss. If a miss occurred, the microcode fetches the block from memory and refills the cache block. Performance could have been improved a few percent by providing additional control and data path hardware to perform the cache refill directly. However, since this hardware would have required an additional logic board, it would have adversely affected cost-performance.

Many of the tradeoffs made in the NonStop TXP design were based on detailed measurements of the NonStop II performance. A complex performance analyzer, named XPLOR, was designed and built solely for that purpose. XPLOR was used to perform the cache emulation experiments. In addition, it provided data on instruction frequencies, percent time spent in each instruction, and the memory reference activity of each instruction. This allowed hardware to be saved in the support of less frequent instructions and applied to accelerating the more frequent instructions. XPLOR also provided data which enabled the microcode to be optimized for the more frequent paths through complex instructions.

The final result (see Appendix A) is a processor which has a 83.3 ns microcycle time and can execute simple instructions in only two cycles. In typical applications, the NonStop TXP executes about 2 million native instructions per second. This new processor has not uncovered any bottlenecks in the I/O or message system; hence, the improvements in CPU performance have been directly translated into transaction processing performance.

6.0 BENCHMARKS

Recently several customer benchmarks have been run which demonstrate the capability of this system architecture to support high volume, high performance on-line transaction processing. Two of these benchmarks are described below.

6.1 BANKING BENCHMARK

In the summer of 1984 a major European bank benchmarked a Tandem system to support a bank card and electronic funds transfer application. The data base for the application consisted of the following files:

Card	: 1.2 million records
Memo Balance	: 2.4 million records
Log	: Sequential log of all updates to the Memo Balance File
Customer Product	: Info about accounts

The application was described by five major types of transactions. Two of the more frequently executed transactions were the DEBIT and LOOKUP transactions.

- DEBIT Transaction Profile:

  Message in;
  Read random Card File;
  Read random Memo Balance File;
  Message out;
  Message in;
  Read random with lock same
  record as before from Memo
  Balance File;
  Rewrite record to Memo Balance
  File;
  Sequential write to LOG file of
  Memo Balance File update;
  Message out.

- LOOKUP Transaction Profile:

  Message in;
  Read random Card File;
  Read random Memo Balance File;
  Message out;
  Message in;
  Read random Customer Product
  File;
  Message out.

In this benchmark the transactions occurred with the following frequency:

Transaction Name	Frequency
DEBIT	52.5%
LOOKUP	39.5%
OTHER	8.0%

The benchmark was done on a 4, 8, 12, and 16 processor single system as well as 1,2,3 and 4 4-CPU systems FOXed together. The results are shown in Figure 3.

Benchmark	#Systems	#TXPs/System	#TXPs	TPS	TPS/TXP
Banking	1	4	4	16	4
Banking	1	8	8	30	3.8
Banking	2	4	8	28.5	3.6
Banking	1	12	12	47	3.9
Banking	3	4	12	46	3.8
Banking	1	16	16	62.5	3.9
Banking	4	4	16	59	3.7
Retail	2	8	16	61	3.8
Retail	2	16	32	124	3.9

TPS = Transactions Per Second

Figure 3. Benchmark Results

6.2 RETAILING BENCHMARK

In the fall of 1984 a major American retailer benchmarked a retail credit authorization application. The requirements were for three individual sites, each site providing 100+ authorization transactions per second for its area. Each site will be connected through a large SNA network and can communicate directly with the other two when necessary for processing out-of-area authorizations. The data base for the application consisted of the following files:

Authorization	: 20 million records
Bankcard Negative	: 10 million records
Out-of-Area Index	: 10 million records

The application was composed largely of one transaction, MAIN, which was nearly 75% of all transactions.

- MAIN Transaction Profile:

  Message in;
  Four in-memory table lookups;
  Read of indexed file;
  Read of indexed file;
  Update of indexed file;
  Write to sequential file;

6.3 SUMMARY

Both of these benchmarks demonstrate the capability for this system architecture to provide high volume transaction processing. The actual transaction throughputs are shown in Figure 3. Response times in both benchmarks averaged less than 2 seconds. Figure 4 graphically illustrates both the linearity and processing power in the region between 2 and 32 processors.

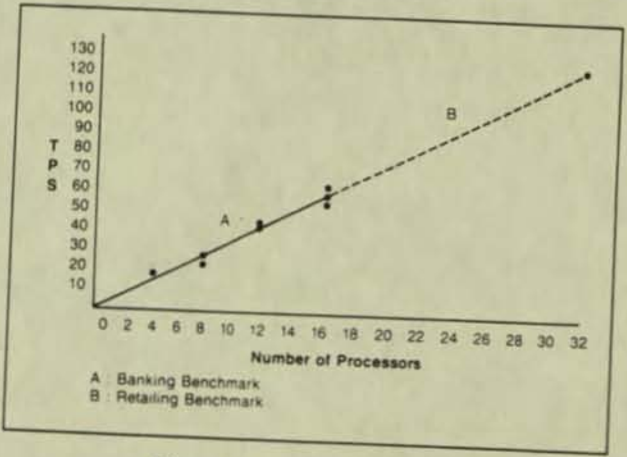


Figure 4. High Volume Transaction Processing

While these graphs do not contain a wealth of data points it should be noted that each experiment is expensive in both time and equipment. However, they do represent experimental results as opposed to theoretical modeling results.

There is a great temptation to extrapolate the curve shown in Figure 4 to 224 processors (the total number in a FOX ring). Assuming linear growth the transaction rate that hypothetically could be supported is somewhere over 800 transaction per second. In reality this may or may not be achievable; however, we hope at some point in time to be able to ascertain this experimentally. Once again, it should be noted that building a 224 processor system and benchmarking it is extremely costly in both time and equipment.

7.0 CONCLUSIONS

For a number of years there has been academic interest and hypotheses [9] that a number of small processors could be tied together in some way and provide the computing power of a larger machine. While this may not be true in general this paper illustrates that it is possible in on-line transaction processing.

Ordinary OLTP systems bottleneck at 50 transactions per second while high performance OLTP systems achieve 200 transactions per second. Beyond these rates, users have been forced to build their applications using ad hoc techniques rather than general purpose systems. Even using these ad hoc techniques, the highest transaction throughput claimed by any mainframe manufacturer is somewhere near 800 transactions per second [6]. Our experimental results to date indicate that the Tandem architecture ,a general purpose multiprocessor, is capable of supporting high volume transaction processing.

ACKNOWLEDGEMENTS

The authors would like to thank Jim Gray, Harald Sammer, Eric Chow, Joan Zimmerman, and Hoke Johnson. for providing many helpful remarks and suggestions.

TM

Tandem, NonStop, NonStop II, NonStop TXP and FOX are trademarks of Tandem Computers Incorporated.

REFERENCES

[1] Arceneaux, G. et al., "A Close Look at Pathway," Tandem Computers, SEDS-003, June 1982.

[2] Bartlett, J.,"A NonStop Kernel," Proceedings of the Eighth Symposium on Operating System Principles, pp. 22-29, December 1981.

[3] Borr, A., "Transaction Monitoring in ENCOMPASS," Proceedings of the Seventh International Conference on Very Large Databases, September 1981. Also available as Tandem Report TR 81.2.

[4] Borr, A., "Robustness to Crash in a Distributed Database: A Non Shared-Memory Multi-processor Approach," Proceedings of the Tenth International Conference on Very Large Databases, September 1984.

[5] Gifford, D., Spector, A., "The TWA Reservation System," Communications of the ACM, vol. 27, no. 7, pp. 650-665, July 1984.

[6] Gray, J. et al., "One Thousand Transactions Per Second",To appear in the Proceedings of the IEEE COMPCON-85, San Francisco.

[7] Horst, R. and Metz, S.,"New System Manages Hundreds of Transactions/Second," Electronics, pp. 147-151, April 19, 1984.

[8] Kim, W., "Highly Available Systems for Database Applications," Computing Surveys, vol. 16, no. 1, pp. 71-98, March 1984.

[9] Satyanarayanan, M., "Multiprocessing: An Annotated Bibliography," Computer, pp. 101-116, May 1980.

[10] Siewiorek, D., Bell, C.G., Newell, A., Computer Structures: Principles and Examples. McGraw Hill Book Company, 1982.

[11] Smith, A. J., "Cache Memories," Computing Surveys, vol. 14, no. 3, pp. 473-530, Sept. 1984.

Architecture Determines Cost and Reliability of Fault-Tolerant Design

Different architectures are available to achieve fault-tolerant computing. Cost-effectiveness and reliability are now available for commercial environments.

by **Dennis McEvoy and Sandra Metz,**
Tandem Computers Inc.

Fault tolerance, or the ability of computers to recover automatically from failures and continue processing, is becoming a critical element in systems used in an on-line, interactive mode. There are several fault-tolerant system architectures and configurations available today, each optimized for the needs of different high-reliability markets. This article will discuss several ways to achieve fault tolerance in computing systems, and the relative merits of each.

A classical system architecture typically consists of a centralized processor, an I/O channel, and controllers to manage such peripherals as terminals and disks. Because a failure in any one of these components can take the entire system off-line, this configuration is acceptable only for applications that don't require continuous system availability.

Applications that demand continuous availability typically require some kind of fault-tolerant computing system. Such a design must strategically duplicate components in an efficient and cost-effective manner. The methods of duplication—and the resulting levels of fault tolerance—vary dramatically and are suitable for a wide range of applications.

SWITCHED BACKUP

A very basic approach to fault-tolerant computing is to duplicate the central-processing unit. Fig 1 shows a dual-processor system in which one processor acts as a backup for the other. The processors are connected through a switch, and either one can communicate with the other components to control the entire system.

This standby configuration is known as switched backup and provides the benefit of fault tolerance in the case of a single processor failure. Switched backup is one of the most commonly used configurations for large on-line mainframe applications.

Several problems are associated with switched backup. One involves every company's biggest concern—dollars. Since the second processor is a backup, it remains idle until the first processor fails. If the backup happens to be a $2 million mainframe, a fairly expensive piece of hardware will be idle until a failure occurs in the on-line system.

However, to gain better return on your investment, you can use the backup system for program development while the on-line system is still functional. In that case, however, when the on-line processor fails, the development work on the backup processor must be halted so that the backup system can be switched on-line.

Another problem with a switched backup is that it provides fault tolerance for only one situation—a single processor failure. An I/O-channel failure between the bus switch and disk controller, for example, would quickly turn both $2 million mainframes castors up. The bus switch itself is another single point of system failure.

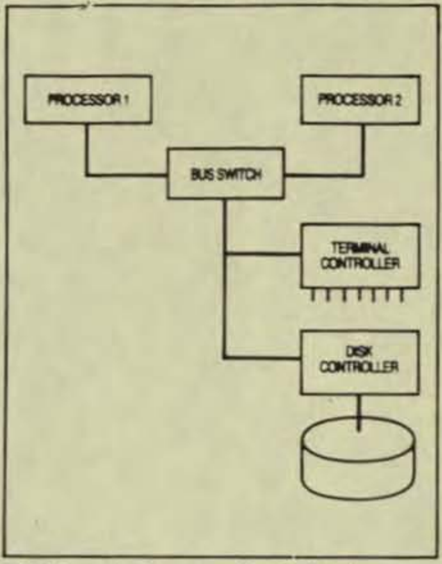


Fig 1 A very basic approach to fault-tolerant computing is switched backup, which duplicates the central-processing unit. A switch connects two processors so that either can communicate with the other system components and control the computer as a whole.

TRIPLE MODULAR REDUNDANCY

A higher level of fault tolerance is achieved with a configuration known as triple modular redundancy. Like switched backup, triple modular redundancy is a hardware-only approach, but it goes a step further by providing three copies of every piece of system hardware (Fig 2).

The three processors are lock-stepped together and simultaneously run the same instruction stream. A piece of hardware called a voter receives and compares the output from all three processors. If one output doesn't match the others, the voting hardware accepts the output from the two matching processors and passes it along to the appropriate interface.

This approach has been used successfully for many years in such areas as space research and nuclear power plants, where computer failure is unacceptable. In the commercial environment, however, triple modular redundancy poses two major problems: tripled cost and complex design. A single clock must run the entire system, and the clock must be implemented so that it can't possibly fail. This is a technically complex requirement.

These limitations weren't a stumbling block in applications like the U.S. space program, where literally billions of dollars were riding on the fact that the computer would be available 100% of the time.

COMPARISON LOGIC

A similar approach is to have four processors running the same instruction stream (Fig 3). The processor outputs are paired with each other and compared. If a mismatch occurs, the pair containing the mismatch is shut down. The system continues running with the remaining pair of processors.

Although similar to triple modular redundancy, such comparison logic has the advantage of being easier to implement than voter logic, and therefore is less expensive to design and maintain. However, the financial problems of redundant hardware all running a single instruction stream and performing identical operations still remain.

Also, comparison logic causes a good processor to be taken off-line along with the problem unit, thus wasting a resource that could be working with the system. And, again, the clock is the weakest link in the system, a single point at which the entire system could fail.

A MULTICOMPUTER SYSTEM

A different approach to fault-tolerant computing, involving a unique, integrated hardware/software architecture, was taken by Tandem Computers. Tandem's NonStop system was designed specifically for the on-line transaction-processing marketplace. This arena requires a system architecture that's not only fault-tolerant and easily expandable, but also maintains a price/performance ratio suitable for commercial transaction-processing applications.

This architecture combines hardware and software so that there's no single point of failure (Fig 4). Components are duplicated but are not redundant. Each processor runs its own instruction stream, so no back-up components sit idle until a failure occurs.

Fault tolerance is achieved by keeping each processor in constant communication with the others in the system via a dual high-speed interprocessor system bus. If one processor fails, its workload will automatically be absorbed by the remaining processors.

Each processor has its own I/O channel and each I/O controller is dual ported. Thus, if one processor fails another can still use that controller. Since disks are physically duplicated with identical data stored on each even a disk crash can't bring down the system. The duplicate disks pay for themselves by ensuring complete system fault tolerance and improved performance. One disk performs seeks on the inner portion of its surface while the other disk performs the same seeks on its outer portion. This provides a significant performance improvement in seek times.

The software architecture is based on independent processors and independent programs within them all of which communicate through a message system that is an integral part of the operating system. This operating system sees all programs and data transfers as communications distributed over several processors.

Programs can access any device anywhere in the system, even those not physically connected to the processor running the program. Conversely each program is unaffected by the processor on which it runs. The operating system sees all physical resources as logical files. Only the message-routing part of the operating

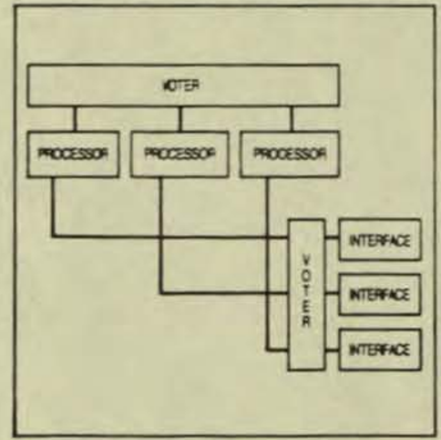


Fig 2 A higher level of fault tolerance is achieved with triple modular redundancy, which provides three copies of every piece of system hardware. The three processors are lock-stepped together and voter logic receives and compares the output from all three.

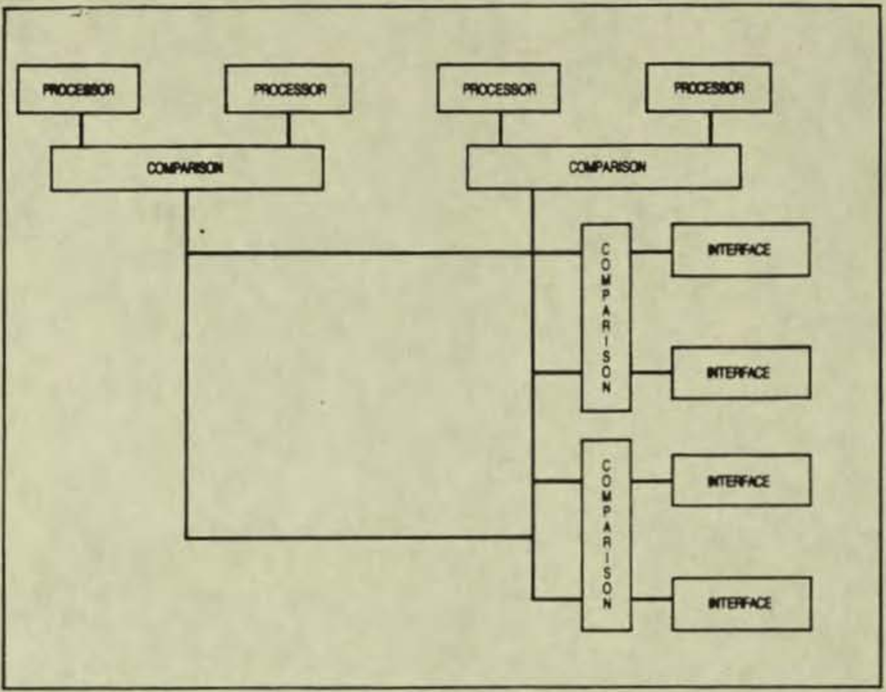


Fig 3 An approach similar to triple modular redundancy is to have four processors running the same instruction stream with their output paired and compared. If a mismatch occurs, the pair containing the mismatch is shut down and the remaining pair continues to run the system.

system knows the geographic locations of resources, so data can be rerouted and resources dynamically reallocated during a failure.

Software fault tolerance is improved by distributing the operating system across all the processors in the system. If one processor fails, its job will be picked up by those remaining. In a system where three or four copies of the hardware are lock-stepped together, a severe software bug could crash all the processors simultaneously. Past experience shows that software bugs severe enough to crash a system are typically related either to timing or to the placement of data in memory. In a NonStop system, however, a software failure will probably not crash more than one processor because the timing circumstances and exact placement of data in memory are unique to each processor.

SYSTEM EXPANSION

On-line transaction processing is characterized by the need for easy system expansion. For example, if a banking application initially involves control of 50 automated teller machines and the application is successful, the bank will typically wish to expand the automated-teller network.

The interprocessor bus system is the key to easy hardware expansion because each processor module has its own memory, I/O channel, and bus controller. To expand this system, additional processor modules are simply added to the bus. The message system keeps track of all system resources, so the system software load is automatically allocated to the appropriate hardware. The system can then be fine tuned for optimum performance.

It's possible to have a modularly expandable multiprocessor system where the processors are linked through a shared memory. A shared-memory architecture provides fast data access to all the processors and provides additional processing power. But the additional processing power obtained doesn't increase linearly.

For example, a second processor doubles your investment and provides only 1.7 to 1.8 times the power of a single CPU because of contention for the shared memory and associated shared resources. A third unit provides approximately 2.4 times the power for three times the cost. By the time the fourth processor is added, it's possible to reach a state of diminishing returns.

One way to get around the contention problem is to locate a cache memory in front of each CPU. Each

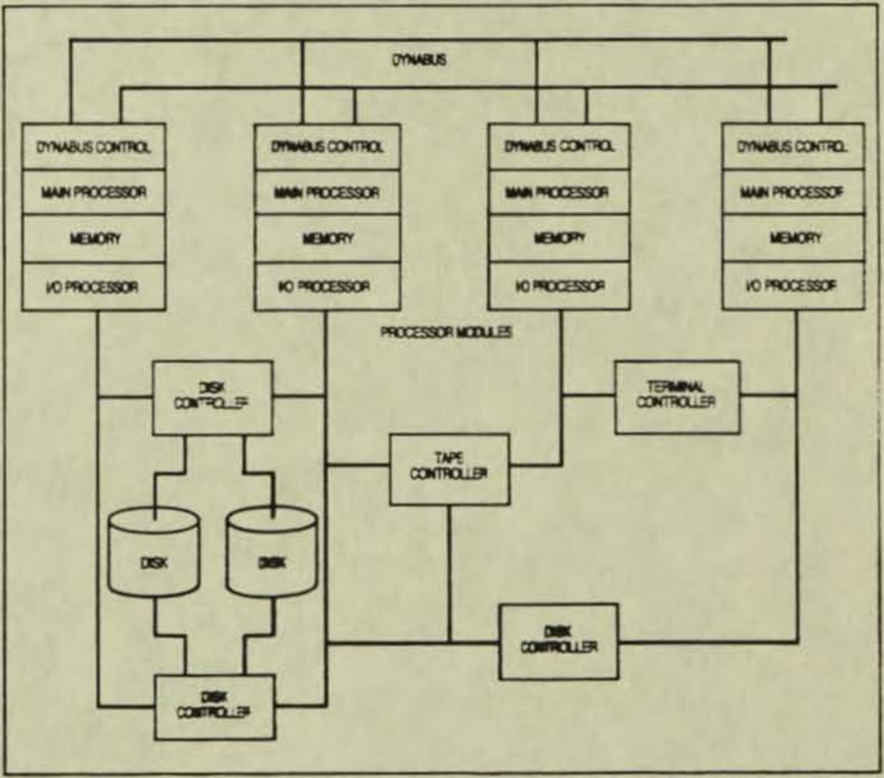


Fig 4 In a multicomputer system architecture, components are duplicated but not redundant. Each processor runs its own instruction stream and none of the backup components sits idle waiting for a failure.

cache still has to fetch instructions from memory though, so that while the problem may be minimized, the limits of the shared memory will be reached eventually.

A multicomputer software architecture eliminates such system bottlenecks as memory contention because the operating system runs in each processor, and the only communication between CPUs occurs when a user program requests I/O from another processor. This communication is done over the system bus, which runs at an aggregate rate of 26 Mbytes/s. Since the processor memory runs at 5 Mbytes/s, bus contention doesn't cause a problem.

Like the shared-memory approach, multicomputer architecture is modularly expandable. But because interprocessor communication occurs in the system bus, memory contention is eliminated along with the potential single-point failure of a shared memory. Therefore, a dual-processor system yields a full two processors' worth of computing power, three units yield three processors' worth of power, and so forth. There's virtually no performance performance degradation associated with expandability.

Another advantage of multicomputer architecture is the networking scheme inherent in a single system. Since the message system already keeps track of the physical location of all resources, a configuration can be expanded into a network of systems without any changes in existing software. The message system keeps track of each system as easily as it keeps track of each processor, and the user doesn't require any additional programming.

This networking scheme also contains a distributed database capability, so data files can be distributed among a network of systems, while the message system automatically keeps track of their locations. The user doesn't need to know where a specific piece of data resides in order to access it.

CHOOSING A FAULT-TOLERANT SYSTEM

The choice of a fault-tolerant system should depend on the application. Some require 100% availability, but only during critical time periods. An automated betting system at a race track, for instance, must be available on race days, but could be serviced at night. In contrast, a system that monitors medical equipment in a hospital must be available continuously, even during periods of maintenance and repair.

You can determine the degree of fault tolerance required by analyzing the cost of failure for a given application. For example, suppose a system advertised a guaranteed up-time of over 99%. That may sound fairly reliable, but 1% downtime translates into approximately a third of a day each month. If the application involved is control of automated teller terminals and the downtime occurs during a Friday lunch hour, the 99% uptime won't soothe the angry customers. The cost of failure in this case requires 100% system availability.

Other considerations include the intial system cost, ease and cost of expansion and modification, and the vendor-supplied system software. Also, if a database management system is needed for the application, you should choose a vendor who can supply one because it's difficult to add features like networking and database management to a fault-tolerant system unless they're designed into it at the start. ■

Dennis L. McEvoy *is vice president of software development at Tandem. He joined in 1974 as a member of the original software development team and project manager for the development of the firm's NonStop operating system.*

Sandra Metz *is a technical public relations specialist at Tandem. She joined the firm in 1982 after working as a writer for Hewlett-Packard and holds a BS in technical writing from Carnegie-Mellon Univ.*

file copy

Tandem Business
Information Center

Kent Madsen and David Foley, Tandem Computers Inc., Sunnyvale, Calif.

How multiprocessor nodes can become more sociable

This month's continuing look at Tandem's corporate network and its nodes shows a company increasingly dependent on distributed network applications like electronic mail.

At Tandem Computers Inc., electronic mail began as an ad hoc program allowing employees to send messages through existing machines used primarily for development, marketing, manufacturing, and order processing. However, today it is the most heavily used application in a worldwide internal network, and its importance is growing. During a recent 18-month period, network traffic doubled, but mail volume tripled.

Through electronic mail, Tandem salespeople from Singapore to Stockholm now collaborate and share information on a daily basis. Analysts from Montreal to Melbourne help each other respond to customer problems and to queries from prospects. Hardware repair personnel at distant sites communicate with manufacturing workers to resolve customer equipment problems. Managers at all levels use the network to keep in touch with employees and colleagues throughout the world. Moreover, electronic mail helps employees establish and maintain personal relationships across geographic or organizational boundaries. Thus, it contributes to a sense of community and teamwork.

One way of understanding the impact of electronic mail on the organization as a whole is to follow a fictitious employee through a typical day, noting how extensively this corporate resource is used. John is the manager of a technical support group of 40 or 50 people affiliated with the headquarters' marketing organization. The first thing he does when he arrives at the office in the morning is to log on to the local computer to scan his electronic mail, that is, to view a list of all messages in his electronic in-box.

Each item in the list of incoming mail gives the sender's name, the message type (original, forward, reply, and so on), and a subject line. Having scanned his mail, John can then select the most important messages to read first. Currently, there are four messages waiting. The first invites him to a strategy meeting that morning with the software development group. Another message contains minutes of a meeting he attended last week. John responds immediately to the invitation and files the minutes of last week's meeting (on disk) in electronic folders bearing names that he has specified.

John's third memo is a request from the vice president of marketing asking him to provide people to help with a design review for a large customer's application. John uses the electronic-mail editor to compose a short addendum and to add an enclosure to the vice president's request, providing more specific instructions. John then forwards the whole package to a subordinate, who will follow through for him.

He notices among his messages a request from a technical analyst in Australia for performance information on a new hardware product. John doesn't have the performance information needed, but he knows of someone at the performance research center in Germany who might. He forwards the message there and replies to the analyst in Australia, indicating what he has done, then heads out the door for his meeting.

All of this has been accomplished in less than five minutes. The messages John has sent are delivered almost immediately (or within the time frame he has specified). Thus, problems can be resolved and answers obtained very quickly, and John can be responsive without carrying details around in his head all day (or all week).

After returning from his strategy meeting, John again reviews his in-box. This keeps him in touch with people and problems from hour to hour, without being interrupted and without interrupting others. He typically checks his mail each time he finishes one of the day's major projects (a meeting, an interview, a lunch engagement, or a trip to the corporate library).

By using the in-house electronic mail, John avoids the frustration and inefficiency of "telephone tag" and cuts down on his long-distance telephone costs. He can leave lengthy messages on complex technical topics for someone who may not be available at the moment, and be assured that the message will not be garbled by an intermediary. In addition, John uses electronic mail to communicate easily with people on the other side of the world, whose business day may not overlap his own at all.

Letting digits do the walking . . .

John (and every other employee) is identified by a correspondent name. Electronic mail includes a way to look up the correspondent names of people on local and remote nodes. These names are listed in a directory application of the "public database," which contains such information as employee office locations, telephone numbers, correspondent names, and department affiliations.

It is easier to look up the correspondent name via computer than it is to find a number in a conventional telephone directory, because the computer does the searching. The correspondent database is updated weekly to incorporate all new hires, internal transfers, changes of address, and changes of status. With a query program, John searches the database for correspondent names by office location, node location, surname, or partial spelling of the surname.

John uses electronic mail to broadcast messages to large numbers of people via distribution lists. For example, when he wants to call a group meeting, he invokes a mailing list containing the correspondent names of everyone in his group. When he wants to announce some change in support policies, he invokes a much larger distribution list to send a message to everyone in the company. Thus, it is no more difficult to address many than it is to communicate with a few—or with one.

. . . and putting heads together

An interesting and well-used offshoot of electronic mail is an archive of technical information. Much of the mail that comes across the network is in the form of technical queries, broadcast to all employees. Workers send such queries when they need information on competitors, product performance, how to link particular devices, and so on. Answers are not hard to come by. Chances are that, out of the 5,200 Tandem employees who receive these messages, at least one will have the required experience or information.

At first, broadcast technical queries typically provoked dozens of secondary queries from people wanting to know what the original questioner had learned. To avoid having to answer these secondary queries individually, authors of technical query messages began to adopt the practice of identifying a file (accessible over the network) in which any replies would be stored. This allowed other interested parties to benefit from the exchange simply by accessing the reply file a day or two after the initial query was sent out.

The whole process has now been taken one step further. An administrative employee systematically reads and files (on disk) all technical queries related to a particular topic. After a week or so, the employee copies all the reply fields to a central node, stores them on disk (with the relevant query), and adds appropriate entries to a subject index.

This centralized repository, referred to as the "archive," is equipped with search software to facilitate information retrieval. Many employees may not be interested in particular informal exchanges of information at the time they occur. However, they can locate and access a stored record of these discussions whenever the need arises. The archive virtually eliminates the need for duplicate queries and provides an extremely valuable information resource, reachable from any network node.

There is a vast reservoir of information and insight within the organization itself, many times larger and more valuable than most formal, structured databases. The combined on-the-job experience of the 5,200 Tandem employees probably exceeds 30,000 years, and their combined college and university experience may amount to 20,000 years or more. Electronic mail and the archive allow people to share insights easily and store them for the use of others. These tools are instrumental in tapping a wealth of information and human experience.

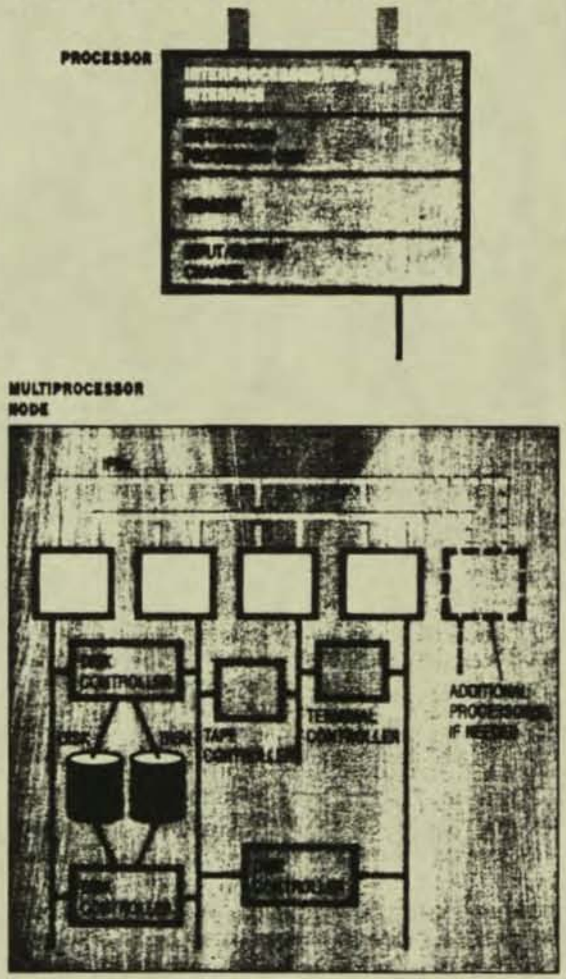
Electronic mail helps to preserve small-company interpersonal communications in the face of rapid growth. It also gives employees easy access to information resources within the company, regardless of where those resources may be. The archive and the electronic-mail network eliminate much duplication of effort and energy.

Overcoming hardware failures

While other applications are growing in use, electronic mail has become the lifeblood of the company's operations. It succeeds basically because users have come to trust the network to deliver their messages. This trust can be credited to one fundamental design principle, which underlies the network architecture and the hardware and software architecture at each node. The principle is that of "fault tolerance."

A fault-tolerant computer is one that can ensure continuous operation and data integrity in the face of any single component's failure. In addition, it must allow hardware service personnel to replace components and activate them without shutting the machine down. Such performance is crucial to computers in a network as large as Tandem's because the more computers there are, the more likely it is that a disabling failure will occur somewhere in the network.

A hypothetical computer running 24 hours a day and capable of offering 99 percent availability might have a mean time between failures of about two weeks. A two-node network composed of such machines would therefore average one failed node a week. With 200 nodes, a network would experience a node failure about every two hours. If we assume that communica-



1. Hardware. *Each node contains several processors (at least two) that communicate with a high-speed bus and dual-ported device controllers.*

tions lines go down as frequently as nodes and that the mean time to repair a line or a node is three or four hours, it is likely that a network that large would never be completely operational. Clearly, much higher standards of machine availability are required for large networks.

The overall availability of Tandem's corporate network rests on the continued functioning of its constituent hardware, as shown in Figure 1. Each node (including the backbone nodes) contains multiple processors linked by dual high-speed buses. Any one of these interprocessor buses (IPBs) can carry traffic at up to 13 Mbyte/s.

Each processor has its own IPB interface, instruction processing unit, memory, and input/output (I/O) channel (as well as its own power supply). However, the Guardian operating system distributed over these processors makes them appear to local or remote users as a single computer. It also allows them to cooperate with one another in processing individual transactions, to share the work load equitably, and to back one another up in the event of a failure.

The IPB uses a multiplexed, packet-interleaved protocol for high-speed interprocessor communications at the local node with a minimum of CPU interruption. However, it would be a mistake to view it as an ordinary communications link. From a logical standpoint, it is more like an internal bus in a conventional computer, since it ties cooperating elements of the local machine together and makes them appear as one.

As Figure 1 shows, designing for fault tolerance meant using multiple hardware components within a single computer. It also implied that techniques would have to be found for detecting failures, disabling problem components, and allowing for their repair and replacement without bringing down the rest of the computer.

Accordingly, hardware and software designers devised rigorous internal consistency checks for each processor. They believed that it was important to detect problems rapidly and halt a failed processor before it had a chance to contaminate data or disrupt the operations of the other processors. In keeping with this philosophy, if a processor finds that it cannot pass its own internal consistency checks, it will simply halt itself, allowing another processor to take over control of its peripheral devices.

If, by some fluke, a processor with a problem manages to pass its own internal checks anyway, there is yet another mechanism provided by which the processor can be restrained. Designers of the operating system decided that, once every second, each processor within a given node would send status messages over the IPB to all others indicating that it is alive and well. Also, every two seconds, each processor would check to make sure that it had received such a message from all the others.

When operational processors detect that one of the others is not following this established protocol, they can effectively quarantine the offender by declaring it "down." Control of its peripheral devices is then transferred automatically to the backup processor. In addition, backup applications program modules running in the other processors are activated to take over the work formerly assigned to the failed processor. When a processor is declared down in this manner, one of the other processors will also take corrective action to clean up any outstanding messages.

To allow the transfer of control of peripheral devices, hardware designers built dual-ported device controllers that can be connected to the I/O channels of two different processors. The controller is owned by only one processor at a time. However, if there is a problem either with the owning processor or with its I/O channel, operating system procedures switch ownership of the device and controller to the other processor (and its I/O channel). In this way, any device can be accessed even if the controlling processor or I/O channel fails. To ensure continuous operation even in the face of disk-controller failures, the disks themselves are dual-ported as well and can be connected to two different controllers, as shown in Figure 1.

The hardware designers also made provisions for the attachment of "mirrored disks" so that failure of a disk drive or its storage medium does not require that the computer be shut down. Mirrored disks are pairs of physically independent disk drives. Writes are performed on the disks in both drives; reads are taken from whichever disk has the shortest seek time. If one becomes inoperable, processing can continue with reads and updates directed to the healthy disk. When the failed disk is repaired, it can be restored to operation and its contents brought to a recent, consistent state. Then all updates performed on the other disk in the interim are transferred to it automatically.

Clearly, fault tolerance could not be achieved without the duplication of hardware components within a single computer. However, software runs on the hardware, and, therefore, if one processor is to take over for another, software components must be duplicated as well. This is accomplished through the use of "process pairs." (A process is a program module running in a particular processor.)

The operating system allows a "primary" process running in one processor to send periodic checkpoint messages to a "backup" process running in another processor. Checkpoint messages, usually sent before the primary process performs a critical task, such as I/O or updating a database, contain all the information that the backup process would need to take over for the primary one in the event of a processor crash.

If a processor goes down, backup processes running in other processors are activated so that they can continue the activities of the primary processes that were running in the now-failed processor. Because the backup process does not duplicate the activities of the primary one while the primary is still functional, it places only minimal demands on the processor in which it resides. Thus, processors can host backup processes as well as primary ones and do almost as much useful work as they would if the backup processes were not present.

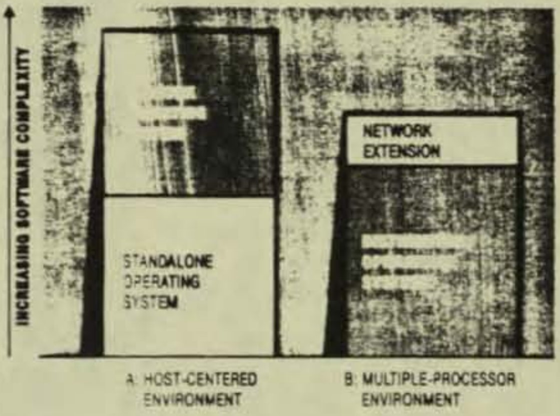
The operating system itself is protected by the implementation of process pairs. Early users of Tandem machines had to program their own primary and backup application processes (and devise effective checkpointing strategies) to get full protection. Now, however, a standard Tandem application development environment obviates the need for user programming of process pairs.

Sociable vs. self-centered computers

Hardware and software reliability are critical in a large network. However, there is an equally important need for machines that communicate and cooperate with their peers. Over the years, Tandem has come to believe that successful networking begins with the design of such "sociable" computers. A number of complex problems, which many people associate with networking itself, actually stem from the architectures of the machines being used as nodes. In most cases, such machines were originally designed with no thought for networking.

The economic need to preserve these architectures has forced many computer vendors to adopt an "add-on" approach to networking (Fig. 2). According to this approach, at each end of the communications line is a computer with an operating system designed for standalone processing. This operating system conditions the computer to think of itself as the main (if not the only) intelligent entity and to view anything attached to it as a peripheral.

2. Overhead of the loner. *To get a typical computer to interact takes a lot more code than if the machine already considers itself a network.*



Layered on top of each node's operating system is a very substantial body of code, which must correct that myopic view. By clever and complex ruses, this "network operating system" overcomes the ingrained reclusiveness and egocentricity of the standalone computer, making it possible for that machine to converse with other computers in a network. Such communications is limited, though, and subject to rigid and somewhat arbitrary constraints, because each computer must be made to believe that it is still only talking to peripherals.

Inherent in this approach is a heavy communications processing burden that falls on the computers themselves (Fig. 2A), stemming from the fact that, in essence, the network operating system has to work against the local operating system. In addition, this approach entails a formidable burden of mounting software complexity. This burden is like an irrevocable tax on every network user, manager, and application, and it has a substantial negative impact on productivity for as long as the network exists.

This complexity increases exponentially as more (and different) computers are added to the network. And even if the linked computers are identical, networking is almost always a strange new world, with remote access procedures and syntax rules vastly different from those used at the local level.

Intranode communications

By contrast, the software that supports Tandem's corporate network does not present a new world to users and programs, but rather functions as an extension of the local environment. In a very real sense, networking is not layered on top of the nodal operating system, but built into it. No separate network operating system is needed.

Nodes in the corporate network routinely engage in "internal dialogues." Since each computer is, in and of itself, a local network composed of multiple independent processors, a machine's processing consists of conversations between its constituent parts. (Artificial intelligence theorists suggest that people may also think this way.) The same communications mechanisms that the operating system uses to blur processor boundaries on the local level are effective in blurring node boundaries within a network as well. When coupled with basic internode communications protocols, these local mechanisms (built into the operating system) contribute greatly to network operations.

The most important of these mechanisms is a "message manager," upon which the entire local operating system is based. Messages are important in the Tandem computing environment because the operating system itself is not a single monolithic program but a collection of "interrupt handlers" and processes. Each process has particular areas of responsibility and must communicate and cooperate with others (through messages) to get work done. These messages must flow, not only within a single processor, but also among the various processors (none of which shares memory).

Copies of the most basic operating system processes (such as the "monitor" and the "memory manager") run in every processor. However, some functions, most notably input and output, must be handled by the particular processors to which the I/O devices are physically attached. This posed a problem for the early designers because all the hardware and software entities in a multiprocessor computer needed access to these I/O resources.

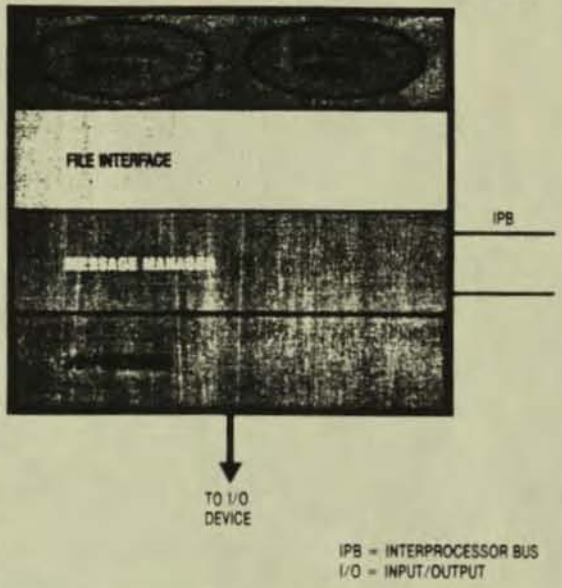
To resolve the problem, the designers developed the message manager, which allows a process to communicate with another process anywhere in the local machine simply by providing the destination entity's symbolic name. The message manager takes full responsibility for locating the named process and getting the message to it regardless of where that process may be running. Disks, tape drives, and terminals are all associated with processes. Thus, the message manager provides a way of addressing and accessing such devices from any location at all in the multiprocessor machine.

A message, as defined in the operating system, is bidirectional. It consists of a "request" for service and a "reply." Several such messages may be required to carry out a given operating system function. For example, a monitor process may be asked to create a new process. To do so, it must do some work and then make requests of several other operating system processes to gather the resources needed.

One of the requests would be to a "disk process," asking that space be allocated on disk as virtual memory for the new process. When the disk process has allocated the space, it replies, indicating that the work has been done. (This reply completes the message.) Other requests are made as well, and when the monitor process has seen to it that all of the necessary resources are in place, it replies to the entity that made the request, indicating that the process has been created.

It is easy to underestimate the value and uniqueness of the message manager. There is certainly nothing special about the concept of program modules passing information to one another. That happens all the time in conventional computing environments that lack a formal messaging scheme. Usually, one program module places information at a specified location in memory, and another picks it up. By contrast, the message manager is a general-purpose mechanism for getting a message between any two processes in a multiprocessor machine. It does not assume that communicating program modules will inevitably be running in the same processor or that memory is shared between the processors involved.

3. Software. *Segmented code allows the operating system to span processors over an IPB and hide details of the hardware from user programs.*

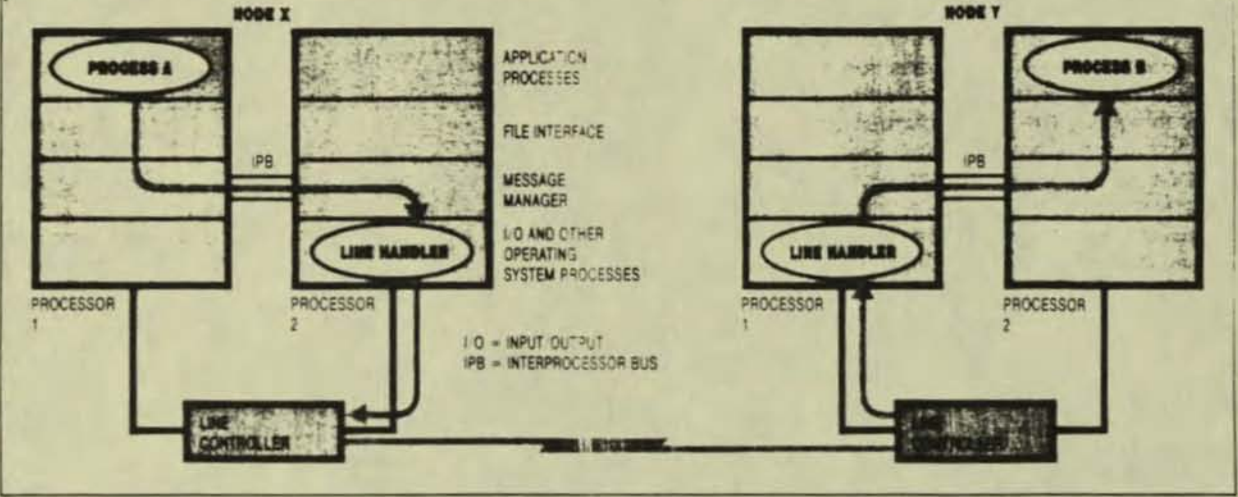


Accessing resources within nodes . . .
Applications processes are not allowed to communicate directly with the message manager or with basic operating system processes. However, the processes can make use of the operating system through a "file interface," which ensures that such interactions do not accidentally create problems for the user or for the machine (Fig. 3).

The file interface works with the message manager to allow application processes to communicate with entities such as other processes, files, and I/O devices, by a single set of calls. That is, such resources can all be referenced by means of pre-assigned symbolic file names. Application processes do not need to know physical locations since the file interface can access operating system tables that keep track of the entities.

To distribute the load over multiple processors (as

4. Network travel. *Application process A need only know the unique network name of process B, and the operating system handles the trip for any request from A. Thus, process A can access remote files, programs, and devices (through processes like B) almost as easily as it can reach local ones.*



well as for other reasons), application programs typically have a structure similar to the operating system in terms of requester and server processes. They also use the message manager's basic request/reply protocol.

For example, an application-type requester process running in one processor might be programmed to control a group of terminals and validate input from them, while an application server, running in another processor, might be programmed to formulate database queries. As a particular transaction is received, the requester validates the initial input and then sends a request to the server (via the file interface) that indicates what work needs to be done. The server process accesses the database (again via the file interface), retrieves the desired information (or updates the appropriate records), and then replies with the requested information or a confirmation that the work has been completed.

. . . and throughout the network

Since the computers that compose the network already consisted of multiple, communicating entities, it was far easier to interconnect them than it would otherwise have been. A group of four or five people spent only about a year developing the required networking software. This software, known as Expand, allows the local file interface and message manager to address and communicate with processes, files, and devices anywhere in a Tandem network.

The networking software consists of line handlers, a proprietary protocol for guaranteed data integrity, and a network control program (NCP). The NCP, which runs at every node, monitors and logs changes in network status.

Routing responsibility is distributed. The NCP maintains a copy of the network routing table (NRT) in each processor. The NRT lists the location of all of the other network nodes. NRTs are used to determine the best path to other nodes and to establish the communications link. (Thus there is no centralized routing that can fail and paralyze the entire network.)

The file interface bridges the network by allowing local processes to access files, processes, and devices anywhere in the network through a single set of calls. The addition of a node specifier to the symbolic name uniquely identifies these resources throughout the network.

Listening in on network dialogue

The networking software is basically an extension of the services provided by the local message manager. Just as the message manager software allows one process to send messages to others within the local machine, its networking extension allows a local process to send messages to other processes running at remote nodes in the network.

As mentioned last month, operations people and other users of the corporate network can, with proper security authorization, log on to a network node in California (or anywhere else) and do work on remote nodes. For example, with successive two- or three-word commands, they can start a program running on a machine in New York, instruct that program to access a file on a disk volume in Atlanta, and print out the results for another employee on a device attached to a computer in Chicago. Also noted was the fact that the command structures by which these operations are carried out are identical to those that would be used locally for similar operations except that, in each case, the program, file, or device name must be further qualified by a node name.

This sequence of events can be used to illustrate how the operating system entities work together. In fact, the different operations were achieved through one mechanism: messages. Process-to-process messages pass first through the local file interface and message manager, next through local and remote line handlers, and finally through the remote message manager and file interface software layers.

Consider process A running on node X within the corporate network (Fig. 4). If process A (an application process) needs to communicate with a process running at the local node, it gives the message, along with the name of the destination process, to the file interface. The file interface checks the message, makes sure that it is legitimate, consults operating system tables to find out where the destination process is running, and hands the message to the message manager.

The message manager takes responsibility for delivering the message and for returning the reply from the destination process. If the destination process is an operating system process, it replies directly through the message manager. If, on the other hand, it is a user application process, it must use the file interface to pass the reply to the message manager.

When process A wants to communicate with process B located at remote node Y, it proceeds the same way, giving a message and destination process name to the local file interface. However, this time the name consists of the process name appended to a node name (for instance, "nodeY.processB"). The file interface, discerning that this resource is not available locally, accesses the network routing table.

From the information contained in the table, the file interface determines that the message must go through a specific network line handler to reach its destination. It therefore preserves the name of the destination process but tells the message manager to deliver the message to that line handler. (In the event that there are two or more line handlers leading to the remote node, the routing table will indicate which path is best, based on the speed of the communications lines and the number of intervening nodes.)

When it receives the message, the local line handler compresses and packetizes the data and sends it over a communications line to another line handler at node Y. This line handler reassembles and decompresses the data and strips off the node portion of the destination process name. It then hands the message to node Y's message manager, which uses operating system tables to locate process B and present the message to it. Process B does whatever it was instructed to do and then replies. The message manager takes responsibility for seeing to it that the reply retraces the path of the request back to process A.

Through this basic mechanism, a user in California can log on to a local machine and enter a command that will start a program running on a machine in New York (Fig. 5A). First the user accesses a local "command interpreter" process, which reads input from the terminal through a terminal I/O process. In response to this input, the command interpreter (acting as a requester process, analogous to process A in Figure 4) sends a message to an operating system process, the "monitor" (a server, analogous to process B), running in the New York machine. The monitor process starts up the program and replies.

The new program started in New York then requests input from the user terminal in California (by sending a message to the terminal I/O process). In response, the user instructs the program to access a disk file attached to a machine in Atlanta. As shown in Figure 5B, the New York program (now analogous to process A) sends a message to the disk process in Atlanta (analogous to process B). The disk process does the necessary work and replies with the requested information. The program in New York then responds to the California terminal and requests further input.

In response, the user in California instructs the program in New York to print out the results of the disk access on a machine in Chicago. As shown in Figure 5C, the New York program sends a message to a line-printer process in Chicago. The line-printer process sees to it that the information gets printed, and replies.

In all these cases, the destination process does not have to be aware of the origin of requests. From its point of view, the message might just as well have come from a local process. This is because all the destination process has to do is to hand the reply to the message manager (or file interface), which will see that the reply finds its way back to the source process, following essentially the same path as the request.

Because Tandem chose to use standard software products at all nodes in the network, the syntax in the above operations is the same as would be used to perform similar operations locally. The only change needed is that the file, process, or device name would have to be qualified by a node name.

Managing growth and applications

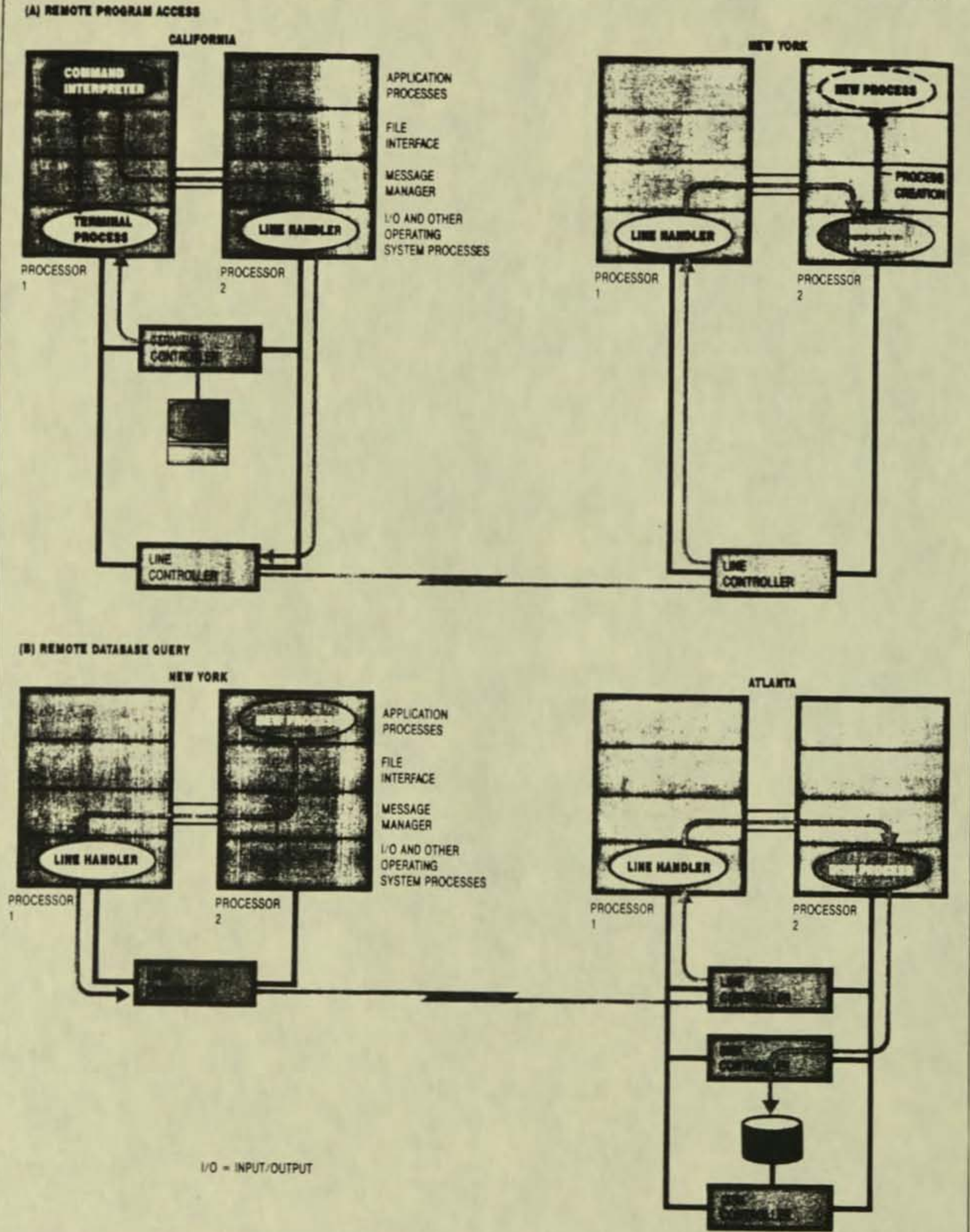
All applications, even those running on a single node, are designed in terms of the requester/server concept. As a result, these applications can be distributed easily when the need arises.

For example, a particular order-entry application started out with all orders called in to a central point and stored on disk. This application became distributed when people in the branch offices were able to key in the information themselves. However, there was still a need to keep a copy on a central disk. Since the application was written using requester and server processes, it was simple to move the requester portions to the regional nodes while the servers remained at the central site to update the database and to return acknowledgments to the requester.

Growth can affect user applications almost as drastically as distribution. Most environments can only add processing power by purchasing a larger machine, often with a different architecture and operating system. Such a move almost inevitably entails a significant software conversion effort. Even if manageable locally, a capacity upgrade can become difficult in a network setting. Dozens of applications and remote nodes that regularly access the newly configured local node must then also be changed.

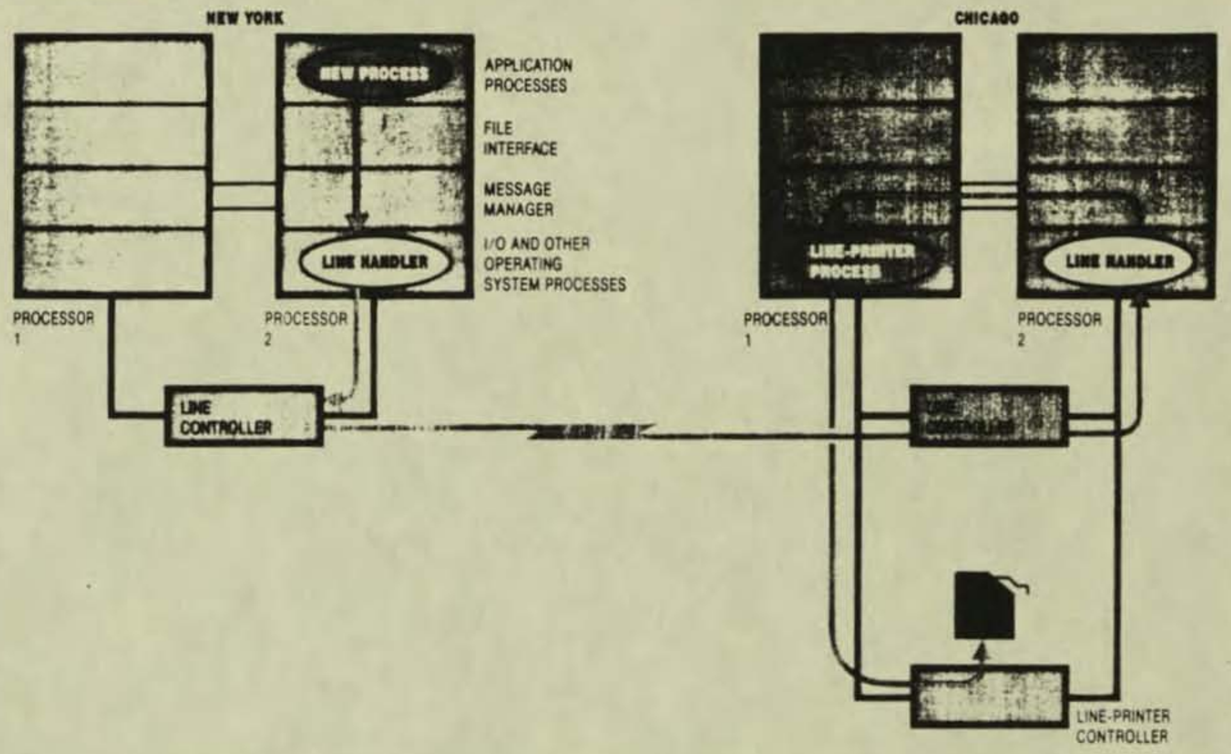
In Tandem's corporate network, however, increased data processing demands at local nodes have been met without producing waves locally and throughout the network. Nodes can be expanded from two to 16 processors, and duplicate copies of requester and server processes can be run in the new processors. Thus, the applications can handle twice the work load,

5. The network as single virtual machine. *Authorized users can log on to a network node in California (or anywhere else) and, with single commands, start a program on a node in New York (A), instruct that program to read a disk file in Atlanta (B), and have it print the file on a device in Chicago (C). Processes that be-*



gin as or are created by servers (such as the new process in New York) can turn around and make requests of other processes. They can also use the same techniques in communicating with remote nodes. User programs use the file interface, while operating system processes go directly via the message manager.

(C) REMOTE PRINTING



but programmers do not have to change a single line of code.

The network is remarkably homogeneous. All working nodes have the same architecture and run identical versions of the nodal operating system. Along with the local expandability of each node, this homogeneity greatly facilitates network management from a logical standpoint. It is much easier not having to deal with new operating systems as the network grows. The ability to reuse the same box also means that, physically, remote sites are more readily upgraded. Even at the central computer center, expanding a node is far less disruptive than replacing it.

Expandability also aids in the management of distributed applications. When the use of such applications grows, certain components such as disks, controllers, or CPUs may become bottlenecks. Modules can be added to replicate these components. In the example considered earlier, an order-entry application was replicated to the regional nodes. When a region grows, the requester program can be duplicated and run on a different CPU. The application then doubles its capacity without being rewritten, simply by having another copy of itself spawned.

New processors are often added to overworked nodes to relieve the burden on the existing processors. If several application processes are then moved from their original processors to the new ones to redistribute the load, the application programs do not have to be changed. They will run just as they did under the old configuration except that, overall, response times will be better.

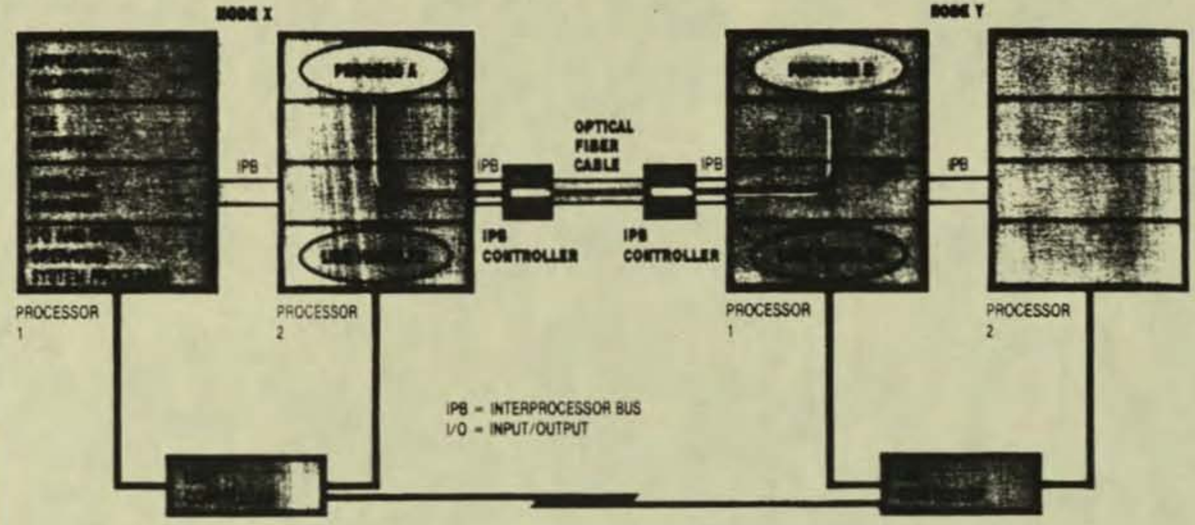
Looking to the future

In any structure as large and multifaceted as the Tandem corporate network, change is a constant. Use of the network's long-haul circuits has grown enormously in the recent past. For example, the bandwidth required on the backbone link between the two U. S. coasts began at 10 kbit/s in the middle of 1982. It doubled the following year. By 1984 it was up to 56 kbit/s, and by the end of this year it will have doubled again. Projections for the fourth quarter of next year show a need for another 56 kbit/s, a full T1 (1.544 Mbit/s) by the end of the decade.

These projections are for the network's data traffic only. The introduction of facsimile transmission capability to the network, which is currently taking place, will undoubtedly increase the rate of growth. Facsimile applications expected soon are a tie-in to electronic mail, store-and-forward facsimile switching, and dialing in or out to distribute and archive facsimile images on disk. The manufacturing division is starting with 40 to 80 pages a day, and the number is expected to increase.

To meet future needs for data transmission band-

6. Lightwave 'telepathy.' *Two nodes joined in a lightwave cluster exchange messages CPU-to-CPU, bypassing the traditional computer "talk-paths" (communications line, line handlers, and controllers). Transfers such as reads, writes, and protocol matters take place at the level of the message manager.*



width, the network support group is making plans to install satellite links between selected backbone nodes. These links will not replace existing terrestrial lines, which are needed for interactive traffic because of their low propagation delay. They will, however, provide the bandwidth needed to carry large volumes of mail and other transmissions for which rapid response times are not important.

Local traffic (particularly at the company's headquarters) is increasing roughly 50 percent faster than long-haul traffic. Clusters of machines linked by the company's lightwave product will therefore play an increasingly important role in the corporate network. Up to 14 computers can be linked in a ring via double-circuit optical fiber. The entire subnetwork thus created may contain up to 224 processors, each capable of processing 4 million instructions per second. The main headquarters subnetwork will initially contain over 100 processors.

A lightwave subnetwork is very much like a single, large, powerful node for two reasons. First of all, the transmission medium offers the speed and bandwidth needed to ensure that response times are essentially the same whether processing tasks involve communications within or between individual computers. Each of the four fibers (two full-duplex circuits) carries data at 10 Mbit/s, for an aggregate data transfer rate of 40 Mbit/s (the theoretical optimum; actual user throughput depends upon the application).

Secondly, the message manager allows users and executing programs to communicate with or access any other executing program, peripheral, or file in the corporate network simply by supplying its name and the relevant node name. As Figure 6 shows, the lightwave ring is designed to transport messages between processes. It sends them directly over the interprocessor bus, without using I/O channels or controllers. Clustering nodes on a lightwave ring takes advantage of higher-speed hardware. This method consumes up to 80 percent less CPU time than does the conventional way of handling data traffic, in which data leaves a node via a line handler. It also provides much faster response times. ■

(This is the second part of a two-part article.)

Kent Madsen is the editor of the Tandem Application Monograph Series, produced by the company's field productivity programs group. David Foley is the technical manager of the Tandem network. He is responsible for architectural and strategic planning, analysis, and operations support.

The authors gratefully acknowledge substantial contributions to this article by Denis Winn, operating system specialist with Tandem's software education department.

Further reading

Bartlett, J. F. "A NonStop Kernel." *ACM Operating Systems Review*, vol. 15, no. 5, December 1981, pp. 22-29.

Forsdick, H. C., Schantz, R. E., and Thomas, R. H. "Operating Systems for Computer Networks." *Computer*, vol. 11, no. 1, 1978, pp. 48-57.

Gray, J. and Metz, S. "Solving the Problems of Distributed Databases." *Data Communications*, October 1983, p. 183.

Holden, J. B. "Experiences of an Electronic Mail Vendor." *Proceedings, National Computer Conference*, AFIPS Press, 1980, pp. 493-497.

Holland, R. "Distributed Databases: Decisions and Implementations." *Data Communications*, May 1982, pp. 97-111.