Tandem TR 85.5

DISTRIBUTED COMPUTER SYSTEMS -- FOUR CASE STUDIES

Jim Gray, Mark Anderton

Revised February 1986

ABSTRACT

Distributed computer applications built from off-the-shelf hardware and software are increasingly common. This paper examines four such distributed systems with contrasting degrees of decentralized hardware, control, and redundancy. The first is a one-site system, the second is a node replicated at a remote site for disaster backup, the third is a multi-site system with central control, and the fourth is a multi-site system with node autonomy. The application, design rationale, and experience of each of these systems are briefly sketched.

This paper has been submitted for publication to the IEEE Transactions On Database Systems.

TABLE OF CONTENTS

INTRODUCTION

IBM's CICS Inter Systems Communications and Tandem's Encompass System have been widely available for ten years. Both these systems allow construction of distributed computer applications with general-purpose off-the-shelf hardware and software. Initially, building such systems was a pioneering adventure. But now, after hundreds of distributed computer systems have been built, it is fairly routine to build the next one -- not much harder than building a large centralized application.

These existing applications provide models and precedents for future designs. By generalizing from case studies we can see trends and deduce general principals of distributed application design.

Distributed systems have two sources: (1) the expansion of a single application, and (2) the integration of multiple existing applications. In both cases, the result is a large system. Hence, distributed systems have the characteristic problems of large systems -- complexity and manageability. It is important to separate these large system issues from the issues unique to distributed systems.

The main reasons for choosing a distributed system design are:

- **Reflect Organization Structure**: Increasingly, decentralized organizations are building systems so that each region, division or other organizational unit controls the equipment that performs its part of the application. This is especially true of applications that have grown together from multiple independent applications.

- **Modular Growth**: In a centralized system, the system is upgraded to a newer-faster-larger system as the application demand grows. The older-slower-smaller system is retired. In a distributed system, the system can grow in increments as the demand grows. The existing hardware is not retired -- rather it is augmented with additional hardware Most applications find it impossible to predict future demand for the system, so modular growth of hardware is a very attractive feature of distributed systems.

The extreme argument for modular growth applies when no single system is big enough to handle the whole problem. In such cases, one is forced to use a several cooperating systems.

Similar arguments apply to growing software. A distributed system using the modular technique of requestors and servers (see below) allows new applications to be added to the system without disrupting existing applications. In addition, it gives clean high-level interfaces between the components of an application so that a service can be reorganized without affecting its clients.

- **Availability:** System availability can be improved by placing data and applications close to the user.

- **Disaster Protection:** Geographic distribution is an essential aspect of disaster recovery. Distributing applications and data geographically limits the scope of a disaster and protects against natural disasters and sabotage.

- **Communication Delays:** Transcontinental and intercontinental transmission delays are sometimes unacceptable. These delays may be avoided by placing processing and data close to the users.

- **Price:** In some cases, many small systems are cheaper than one giant one, especially when the savings of modular growth vs upgrade are considered. In addition, some systems realize substantial savings in telecommunication costs by placing processing and data near the system users.

KEY CONCEPTS IN DISTRIBUTED SYSTEMS DESIGN

Many believe that the key to distributed systems is to have an integrated distributed database system that allows data to be moved and part at will among discs in a computer network.

Tandem has provided such a mechanism for over 10 years, and even today it is one of the few vendors to provide location transparency of data partitioned in the network.

Surprisingly, Tandem recommends against indiscriminate use of this unique system feature. Rather, we recommend using the distributed database only within the local network, and even then, only within application or administrative bounds. When crossing administrative boundaries or when using a long-haul network, we recommend using a **requestor-server design** in which an application makes high-level requests to servers at remote nodes to perform the desired functions at those nodes.

There are two reasons for this:

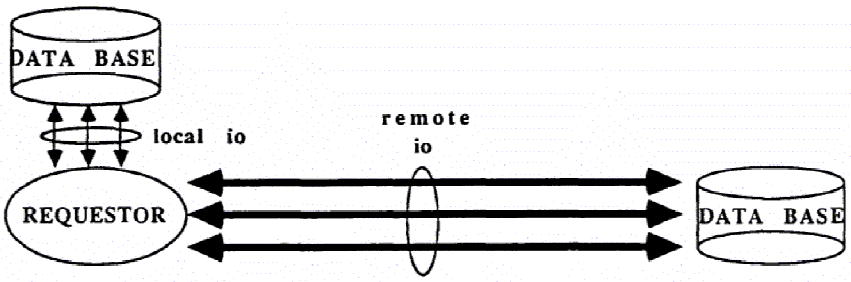
- **Performance:** Making many low—level database requests to a remote node is much slower than making one high-level request to a remote server process at that node which then does all the low-level database accesses locally. Hence, one should only distribute data and low-level database calls across a fast, reliable, and cheap network -- that is a local network.

- **Modularity:** If an application externalizes its database design to other applications, it is stuck with that database design forever. Other applications will become dependent on the database design. Database system views do not eliminate this problem -- views are generally not updatable [Date, pp. 128—129]. Rather, information hiding requires that a application externalize a high-level request interface rather than a low-level database design.

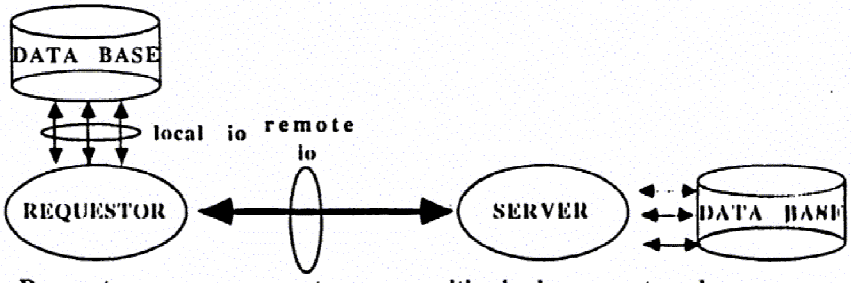
To give a concrete example of the difference between remote IO and remote servers we preview the first case study -- a bank with a retail service and a currency exchange service. The currency exchange service sends a DEBIT or CREDIT request to the retail banking application. The retail banking server checks the validity of the request, then reads and rewrites the account, adds a journal entry in the general ledger, and replies. The currency exchange application is unaware of the database design or accounting rules followed by the retail bank. Contrast this to the integrated distributed database design where the currency exchange application would check the validity of its own request, and then proceed to read and write the retail banking database directly across the network.

This direct access to the data is much less manageable and involves many more messages.

Requestors and servers provide a way to decompose a large system into manageable pieces. Put bluntly, distributed databases have been oversold. Distributed databases are essential in a local network but are inappropriate to a long-haul network. Geographically or administratively distributed applications do not need an integrated distributed database. They need the tools for distributed execution -- variously called requestors and servers [Pathway], Advanced Program to Program Communication [CICS], or remote procedure calls [Nelson]. The case studies below demonstrate this controversial thesis.



1.a. Requestor directly accesses remote data via multiple low level database read and write requests.



1.b. Requestor accesses remote server with single request and server accesses local database with multiple local database requests. This has better performance if long-haul messages are slow or expensive. In addition, the remote server hides remote database design from the requestor.

Tandem's Pathway system allows requestors to transparently call programs in other processes (in other processors). These calls look like ordinary procedure calls. At the applications level, there is no additional complexity. Calls to remote servers look like subroutine calls. Designing the servers is equivalent to designing the common subroutines of the application. Pathway manages the creation of server processes and the associated load balancing issues [Pathway]. CICS and SNA LU6.2 introduce similar mechanisms via Advanced Program to Program Communication (APPC) [Gray].

All but one of the case studies below use requestors and servers outside the local network. Even the "centralized" distributed system case study does not allow "local" IO across organizational boundaries (e.g. from retail to commercial banking), because such an "integrated" database would pose management problems when one part of the bank wants to reorganize its data. The designers of the one application that uses Tandem's transparent distributed database (remote IO rather than remote servers), agree this is a mistake, but cannot yet justify the system redesign.

Transactions are the second key concept for distributed applications. The failure modes of distributed systems are bewildering. They require a simple application programming interface that hides the Complexities of failing nodes, failing telecommunications lines and even failing applications. Transactions provide this simple execution model.

A transaction manager allows the application programmer to group the set of actions, requests, messages, and computations into a single operation that is "all or nothing" -- it either happens or is automatically aborted by the system. The programmer is provided with COMMIT and ABORT verbs that declare the outcome of the transaction. Transactions provide the ACID property (Atomicity, Consistency, Isolation, and Durability) [Hearder and Reuter].

Both CICS and Encompass provide a distributed transaction mechanism. The transaction mechanism is critical to all the case studies cited below.

In summary, the three key concepts for building distributed systems are:

- Transparent data distribution within a local network.
- The requestor-server model to allow distributed execution.
- The transaction mechanism to simplify error handling of distributed execution.

The case studies below exemplify these features. They also demonstrate a design spectrum from geographically centralized to almost completely decentralized applications.

Before discussing the case studies, we sketch the Tandem system architecture which allows these applications to be built with off-the-shelf hardware and software.

OVERVIEW OF TANDEM SYSTEM FEATURES

Tandem produces a fault tolerant distributed transaction processing system. A Tandem node consists of from 2 to 16 processors loosely coupled via dual 12 Mbyte per second local networks. The network may be extended via a fiber optic LAN to over 200 processors and via long-haul lines to over 4000 processors [Horst].

All applications, including the operating system, are broken into processes executing on independent processors. Requests are made via remote procedure calls that send messages to the appropriate server processes. This design allows distribution of both execution and data.

The system supports a relational database. Files may be replicated for availability. Files may be partitioned among discs at a node or among discs in the network. This replication and partitioning is transparent above the file system interface. The system also transparently supports distributed data, allowing remote IO directly against a local or remote file partition. As stated above, users are encouraged to use the requestor-server approach to access geographically remote data. Sending a request to a server at a remote node reduces message delays and gives a more manageable interface among nodes.

The system is programmed in a variety of languages. Cobol is most popular, but Fortran, Pascal, C, Basic and others are supported. The system has a data dictionary, a relational query language and report writer, and a simple application generator.

Transactions may do work and access data at multiple processors within a node, and at multiple network nodes. An automatic logging and locking mechanism makes a transaction's updates to data atomic. This design protects against failures of transactions, applications, processors, devices and nodes. Transaction management also guards against dual media failures by using archived data and a transaction audit trail to reconstruct a consistent database state [Borr].

Also supported are communications protocols, ranging from x.25 to much of SNA and OSI, as well as presentation services such as virtual terminals, document interchange, and facsimile.

A minimal node consists of two processors, so all Tandem systems could be considered distributed. But, a multi-kilometer LAN and long-haul distribution provide "real" distribution of processing and data. About 68% of the customers take advantage of these sophisticated networking features.

The most common system consists of one node of up to 16 processors. The next most common is two nodes. The largest system is the Tandem corporate network (over 200 nodes and over 1000 processors). The largest customer system has over 200 processors and over 50 sites. Several customers have systems with over 10 nodes and over 50 processors.

This paper examines some of these larger systems to see what approaches they have taken to distributing data and processing power.

A DISTRIBUTED SYSTEM IN A ROOM

The most common distributed system is a one-site system. A single site avoids many of the operational and organizational problems of distribution because it is run just like a centralized system. Some of the advantages of geographic centralization of a distributed computer system are:

- LANs give high-speed, low-cost, reliable connections among nodes.
- Only one operations staff is required.
- Hardware and software maintenance are concentrated in one location, and are hands-on rather than thin-wire remote.
- It is easier to manage the physical security of premises and data.

A one-site distributed system differs from a classical centralized system. It consists of many loosely-coupled processors rather than one giant processor. The arguments for choosing a distributed architecture rather than a centralized one are:

- **Capacity**: No single processor is powerful enough for the job.
- **Modular Growth**: Processing power can be added in small increments.
- **Availability**: Loose coupling gives excellent fault containment.
- **Price**: The distributed system is cheaper than the alternatives.
- **Reflect Organization Structure**: Each functional unit has one or more nodes that perform its function.

IBM'S JES, TPF-HPO, IMS Data Sharing, and DEC's VAX Cluster are examples of using loosely-coupled processors to build a large system-in-a-room out of modular components.

A European bank gives a good example of a Tandem customer using this architecture. The bank began by supporting their 100 Automated Tellers, 2500 human tellers, inter-bank switch, and administrative services for retail banking [Sammer].

The initial system supported a classic memo-post application: Data, captured during the day from tellers and ATMs, is processed by nightly-batch runs to produce the new master

file, monthly statements to customers, inter-bank settlements, and management reports. The peak online load is 50 transactions per second.

In addition to designing the user (teller/customer) interface, the bank took considerable care in designing the operator interfaces and setting up procedures so that the system is simple to operate. Special emphasis was put on monitoring, diagnosing, and repairing communications lines and remote equipment. The application and device drivers maintain a detailed status and history of each terminal, line and device. This information is kept in an online relational database. Another application formats and displays this information to the operator, thus helping him to understand the situation and to execute and interpret diagnostics.

Recurring operator activities (batch runs) are managed automatically by the system. Batch work against online data is broken into many mini-batch jobs so that, at any one time, most of the data is available for online access.

The initial system consisted of 32 processors with 64 disc spindles. The processors were divided into three production nodes and two small development nodes connected via Tandem's FOX fiber optic ring [Horst].

Once retail banking was operational, the next application to be implemented was cash management. It allows customers to buy or sell foreign currency, and to transfer assets among accounts in different currencies. This is a separate part of the banking business, so it has an "arms-length" relationship to the retail banking system. Although it runs as part of the bank's distributed system, the cash management system does not directly read and write the retail banking databases. Rather, it sends messages to the retail banking system server processes that debit and credit accounts. These retail banking servers access the retail database using the procedures implemented by the retail banking organization. Money transfers between the two bank departments must be atomic -- it would be improper to debit a savings account and not credit a foreign currency account. The transaction mechanism (locks and logs) automatically makes such cross-organization multi-server transactions atomic.

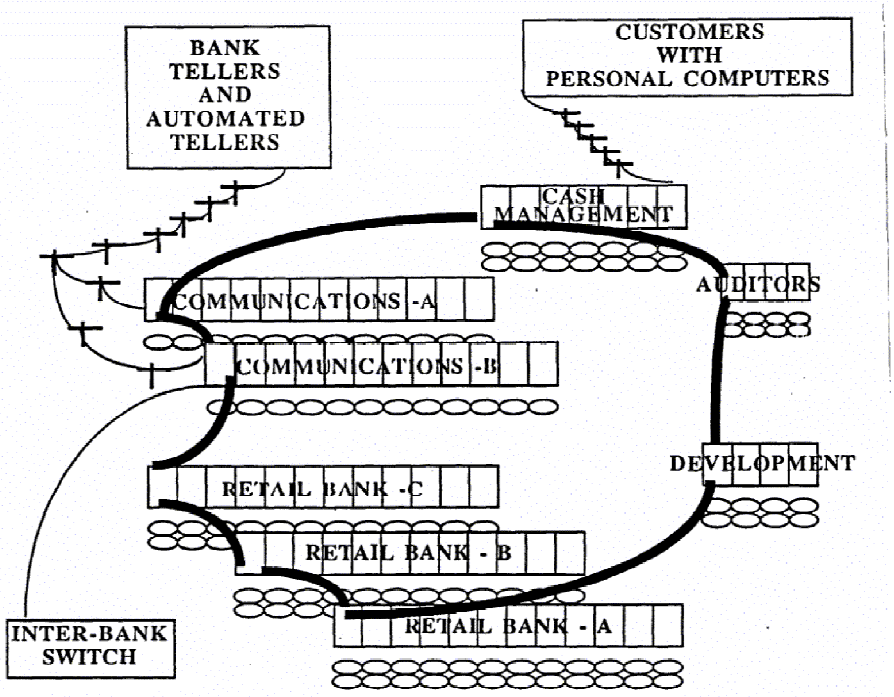
This interaction between retail and cash-management banking exemplifies the case for the requestor-server architecture. Even though the data was "local", the designers elected to go through a server so that the two parts of the application could be independently managed and reorganized.

Later, electronic mail was added to the system. The bank is now going completely online, eliminating the problems implicit in a memo-post nightly-batch system.

The bank has demonstrated linear growth -- it can add processors and discs to the system and get a proportional growth in throughput. In addition, the bank's development staff has been able to add applications which build on the existing application servers. The system consists of 68 processors (over 100 mips) with over 200 megabytes of main memory and 90 spindles of duplexed disc. The processors are divided into 10 nodes: each node consists of two to sixteen processors. Particular nodes are functionally specialized: two handle the long-haul network, three others do retail banking, one does cash management and one does electronic mail, while others are reserved for auditors or for developers. This functional division of nodes simplifies system tuning and gives each department control over its resources. Although many processors and discs are involved, programs have a "single-system-image;" that is, all files, processes and terminals appear to be local. This software illusion is made a reality by connecting all the processors together with high speed (100mbit/sec) local networks. (Figure 2).

All management and operations staff is in one place, only the terminals are remote. This is easier to manage than supervising multiple computer rooms and staff spread over a broad geographic area.

This system could have been a large (100 mip, 200Mb, 100 disc) centralized computer system if such a computer were available. The additional desire for modular growth, availability and low cost, each forced the choice of a distributed hardware and software architecture.



2. A one site system of 9 nodes connected via a local network (dark line). The nodes are functionally specialized. Some functions are large enough to require multiple nodes.

SITE REPLICATION FOR DISASTER PROTECTION

Disaster protection is an emerging application for distributed systems. Traditionally, disaster backup has been provided by taking periodic archive copies of the data. This data is kept off site. In case of disaster, the archive data is moved to a prearranged regional disaster backup facility and the work run from there until the primary facility is repaired. It generally takes several hours to reach the disaster facility and then several more hours to get the application up and running. This approach to disaster protection is acceptable for batch transaction processing, but is not appropriate for online transaction processing systems because:

- **Lost Data**: The scenario of going back to an "old master" and working from there implies losing days or weeks of transactions. This is not acceptable for many businesses where each transaction is of significant value.
- **Delay**: Traditional disaster backup facilities need several hours notice and preparation. For practical and legal reasons, many businesses cannot tolerate outages of that duration. In addition, a disaster that affects a wide geographic area may swamp the capacity of a shared recovery center.
- **Communications**: Commercial disaster recovery centers cannot afford to support the large number and variety of communications lines, network hardware and network software required for online applications. Hence, there is no disaster backup for the communications network. This means that a network of online terminals cannot be protected by a conventional disaster center.
- **Credibility**: If the disaster facility is not an exact replica of the primary site, disaster recovery plans cannot be adequately rehearsed or audited.

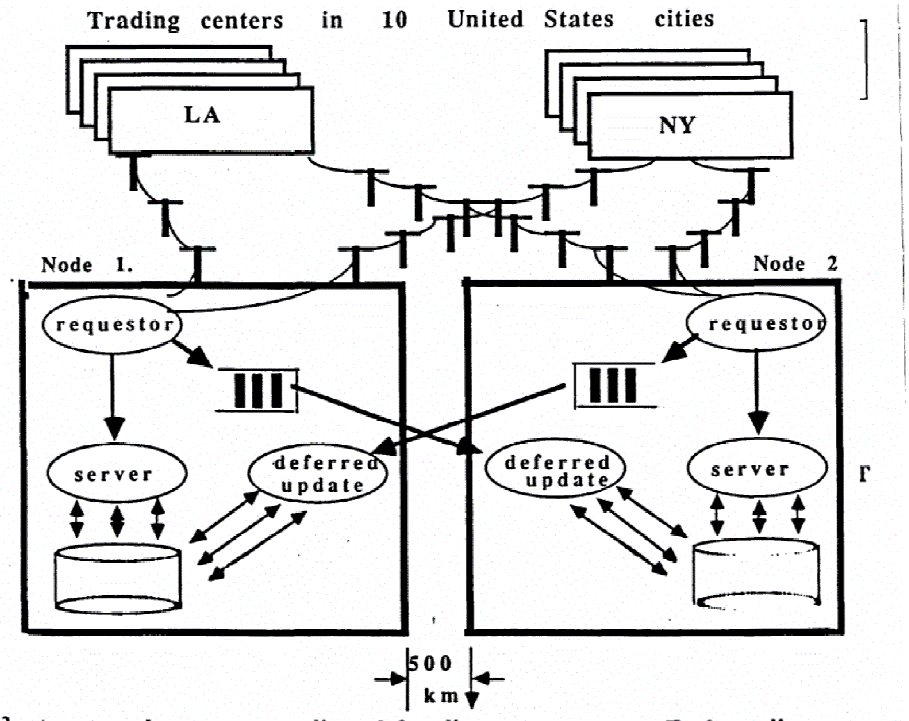
In short, the traditional approach to disaster protection does not apply to online transaction processing.

An alternate approach to disaster protection is to have two or more sites actively back one another up. During normal operation, each site stores a replica of the data and each carries part of the telecommunications and computational load. In an emergency, all work is shifted to the surviving sites.

For failsafe protection, the sites must be independent of each other's failures. They should be geographically separate, part of different power grids, switching stations, and so on. Geographic diversity gives protection from fires, natural disasters, and sabotage.

A finance company gives a good example of this approach. The company has traders of notes, loans, and bonds located in ten American cities. To allow delivery to the banks for processing on a "same-day" basis, deals must be consummated by 1PM. Traders do not start making deals until about 11AM because interest rates can change several times during the morning. In a three-hour period the traders move about a billion dollars of commercial paper [Martino].

An outage during this three-hour window would be costly -- over 300K$ in interest alone. Losing transactions is unacceptable due to the large dollar value of each one.



3. A two node system replicated for disaster recovery. Each trading floor is connected to both nodes. Each request is first executed at one node as a single transaction. Part of the transaction queues a request to the other node to apply the update as a second transaction.

To guard against lost data and lost work, two identical data centers were installed, each a six-processor system with 10 spindles of disc. The data centers are about 500 km apart. Each center supports half the terminals, although each trading floor has a direct path to both centers. Each data center stores the whole database. As is standard with Tandem, each data center duplexes its discs so the database is stored on four sets of discs -- they call this the "quad" database architecture. Hence, there is redundancy of communications, processing, and data.

The 75 traders generate a peak rate of two transactions per second. The transactions are processed as follows: A trader is signed on to one of the data centers. Using a terminal, he fills out a form that requests an inquiry or update transaction. Most inquiry tasks can be done locally and so only the one node is involved. Updates are more complex [Martino]. When a trader at Node2 makes an update request:

1. A transaction is initiated at Node2.
2. A request message is sent to a server at Node2.
3. The server reads and writes the Node2 database as follows:
   3.1 For updates to aggregate data, it uses "delta" transformations (e.g. "add 1M$" rather than "set from 990,000,000$ to 991,000,000$").
   3.2 For updates to individual records, it checks that the timestamp of the record's remote replica agrees with the timestamp on the local copy of the record (the local copy is locked at this point). This logically serializes all updates to that record. The record is then updated and its timestamp advanced.
4. The server replies to the requestor.
5. The requestor queues a work request to Node1.
6. The requestor commits the transaction and replies to the trader's terminal.

If the transaction aborts, then the request queued in step 5 is deleted. But if the transaction commits, then the request queued in step 5 is asynchronously delivered Node1 where a server applies the update to its version of the database. This delivery and application is a second atomic transaction. Thus each successful transaction is structured as two transactions, a local one and an asynchronous distributed transaction.

Of course, all this processing complexity is invisible to the trader. He merely fills out a form on his screen. The system replies with an acknowledgment, and prints a contract to be delivered to a bank.

This is just one approach to managing site replication. The manufacturing system to be described later uses a slightly different scheme. Yet a third design spools the transaction log to the remote site. The remote site applies the log to a replica of the database so replication is completely transparent to application programs. Many other schemes are possible.

Since the nodes and the communications net are duplexed, no single fault can cause a node or trading center failure. If there is a double fault, a disaster, it is unlikely to affect both nodes or more than one trading center. So the "up" nodes and trading centers can continue operation. When the "down" system returns to service, the "up" system sends it the queued transactions to bring its database up to date.

This distributed system is operated as a centralized system. The design, control, and operations of the system are centralized at one site. A backup operations staff and operations facility is available at the other site. Normally, the second site runs unattended.

GEOGRAPHICALLY DISTRIBUTED NODES, CENTRALIZED DEVELOPMENT AND CONTROL

The next step in distribution is to geographically distribute the database but keep system control and development centralized. This avoids much of the organizational stress associated with distribution because most of the key people are still in one place. The remote nodes run unattended, only semi-skilled operators are needed there for occasional tape handling. Of course, the operations staff does site audits on a regular basis and the, vendor must visit the site for preventative maintenance.

The rationale for choosing a multi-site distributed architecture rather than a single site is:

- **Reflect Organization Structure**: Each site supports some region of a large functional unit. It is a local facility which is centrally managed.
- **Communications**: If there is strong geographic locality of reference, putting the data close to the users reduces communications time and costs.
- **Availability**: A disaster is unlikely to affect more than one site. Failures that isolate a node still allow users to access local data.

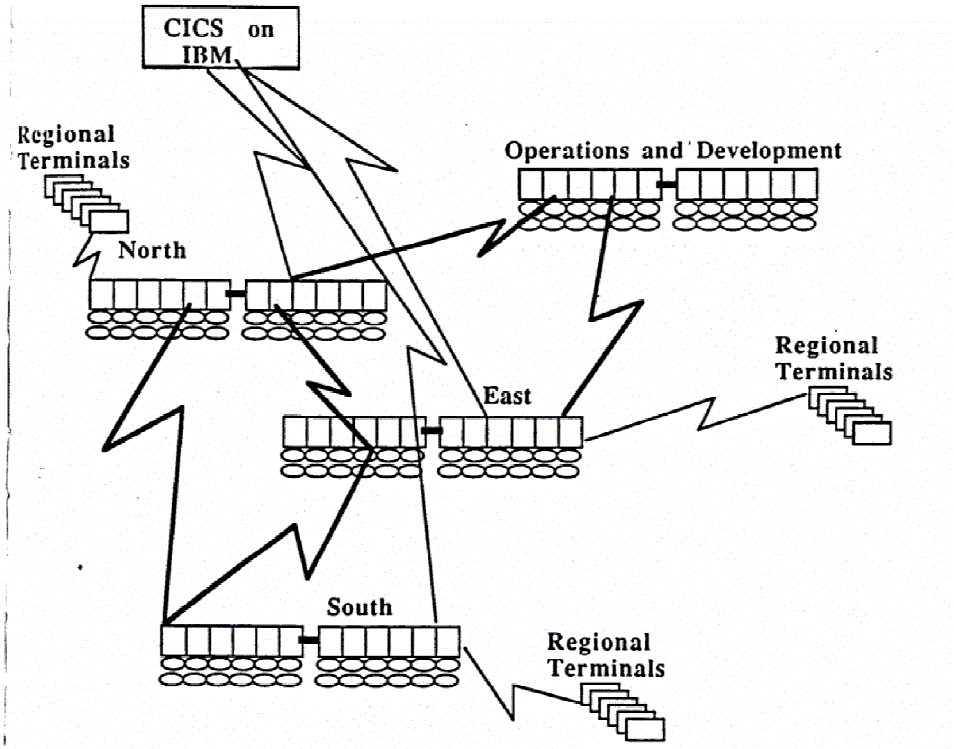
There are several examples of such systems. The best one we have seen is a telephone company network to support two major applications: Telephone sales and terminal pass-through to other systems.

The telephone sales application works as follows. A customer visits a phone store or telephones to arrange for service. The salesperson converses with the customer and fills out a multi-screen form to arrange for the service. The transaction produces a work order to connect the customer at the central exchange, allocates a trunk line, and schedules linemen to attach the trunk to the feeder and the feeder to the house. In addition, the customer may be given the handset and the inventory updated. The transaction also produces various billing and accounting records that are passed to the financial systems. This transaction typically executes 30 million instructions and does over 250 disc IOs -- the most complex online transaction we have seen.

The transaction may access records at several nodes of the network because a customer may want service at some other region. All the transaction's updates are coordinated with the transaction mechanism (logging and locking). The transactions do remote IO rather than sending requests to remote servers (see Figure 1). Remote IO was chosen because it was simple to implement -- the Tandem file system implements it. In retrospect, the designers agree that a requestor server approach would have been more

manageable for database requests across the long-haul network. But, they cannot now justify changing to that approach -- remote IO works.

The phone store sales transaction is about 50% of the application. Other transactions update the database to indicate that the various work orders have been completed, to indicate that new lines or trunks have been added, to produce management reports, and so on.



4. A multi-site system with centralized control. Each node carries part of the load. Some of the work is "pass-through" traffic to IBM hosts. The whole system covers two states and is managed from a remote operations center.

The second major function of the system is to tie the terminals into several CICS applications on IBM hosts. Terminals connected to the Tandem can masquerade as terminals attached to the IBM hosts. Multiple high-speed lines connect the Tandem and IBM systems for this pass-through traffic.

The system has four sites. There are three production sites -- all remote from the operations center. Each site consists of two nodes. The developers have two small nodes that they use for developing and testing new software. The developers and operations staff are located on the same floor of the same building. The operations staff occasionally calls on the developers for support.

For fault containment reasons, no node is larger than 12 processors. The total system has about 50 processors, 100 duplexed disc spindles, and over 2500 terminals. All terminals in a region are attached to their local node.

At the remote nodes, unskilled local personnel mount and demount tapes for archive backups each night. Otherwise, the nodes are in "darkrooms" managed from the central site. The nodes are fault tolerant and so require relatively little attention -- when a component fails, the system continues to operate until the scheduled preventive maintenance. The ability to defer maintenance is essential to the economics of remote operations and maintenance.

Power and communications failures due to bad weather are an occasional problem -- but they affect only one of the four nodes at a time. A few years ago, dust from a volcanic eruption caused one of the sites to have above average disc failures, but the geographic distribution of the system isolated that problem to a single site.

The bulk of the system's problems are associated with communications lines. Many tools have been installed to ease the diagnosis of terminal and line errors.

Coordinating changes in the IBM system and their consequent impact on the Tandem system is a major management problem. The two systems must have matching protocols. So each time CICS, MVS, VTAM or NCP changes on the IBM side, corresponding changes may be required on the Tandem side. Gradually, the organization is migrating to standard protocols and static configurations for logical units within the IBM systems so that effects of release-to-release changes are minimized.

The system has been operational since 1981 and has been cloned by several other phone companies.

GEOGRAPHICALLY DISTRIBUTED AUTONOMOUS NODES

The next step in distribution is to make each node autonomous so that they form a federation. Only the system design and network management are centralized functions. This allows each organizational unit to manage its computing resources locally.

The rationale for choosing node autonomy rather than centralized control is:

- **Reflect Organization Structure:** Each site supports some region of a larger functional unit. It locally tailored and managed to serve local needs.

A manufacturing system gives a good example of such a federation of nodes. The application performs inventory control, work planning, scheduling, and tracking, and provides forecasts for materials resource planning. It supports a paperless factory [Norman].

A central group designed the databases and the server interfaces to the databases. Databases are either global (replicated at each node) or local (partitioned among nodes). Server programs, the parts list, and bill of materials are examples of global files. Inventory and work-in-process are examples of local files.

A master-record scheme with ASAP updates of the slave-records is used to manage the replication. Each global record is "owned" by some node, its "master", and replicated at other nodes. For example, the European node controls the unique parts and bills-of-material for 220 volt power supplies and cables because the engineers who designed these parts reside there. Several sites manufacture or use these parts, hence the data is global. When such a global record is updated, the update is first applied to the master record, at Europe in this example, and then queued for propagation and application to the slave records at other nodes [Norman]. The update at the master node is one transaction. Each update to a slave node is another transaction.

If the nodes and communications lines are functioning, remote updates lag a few seconds behind the update to the master record. Even if all the lines are lost, each node can function in isolation: it can read the global data and write any of its local data. In addition, it can update all the global records it owns. When communication is reestablished, these global updates will be delivered in sequence to the slave copies of the data.

The system is installed at nine sites. It consists of about 50 processors, and 100 discs supporting about 500 terminals. It has a peak demand of ten local transactions per second and a global update every few minutes. These nodes do more than just manufacturing; they are part of the larger corporate network. Hence, they participate in electronic mail, text processing, and other corporate functions.

Each site tailors the user interface to its needs. One is driven by bar-code readers and another uses screens in German. The only thing in common with ALL the systems is that they all support the same language (server interface) when accessing each other's data. They do not necessarily all have the same database design.

New software and new database designs are released on a node-by-node basis. When a bug is discovered, the fix is first tested on a development system, then on a local production system, and then made available to all the other nodes. The requestor-server design allows gradual migration to new software and new database designs, rather than requiring all nodes to convert simultaneously.

The system has been operating for about 5 years now (ASAP updates were installed three years ago). In that time it has operated smoothly. The major problem from the view of the central operations is the turnover of remote staff: as soon as one understands his job, he gets promoted. New staff tends to make mistakes or does not understand procedures. So the central organization has had to provide a 24-hour hotline, build fool-proof procedures, and document functions very carefully.

The biggest complaint of the remote staff is that changes come too slowly, and that the central organization seems to take a long time to implement a seemingly trivial request. Fortunately, node autonomy allows each node to tailor its system to get most of the features they want and still stay within the corporate guidelines governing the consistency of data and procedures.

Organizationally, a remote operations staff has dual responsibility: one to its local user community, and one to the network at large. This sometimes puts them in a difficult position.

These conflicts and benefits are just the ones we would expect from a decentralized organization.

SUMMARY

These systems exhibit a spectrum of decentralization. Each demonstrates some benefits of a distributed system architecture, but each has taken a different approach to designing and managing the system.

The common reasons for adopting a distributed architecture are capacity, modular growth, availability and price. The disaster recovery system has the additional need to replicate data at two sites. The multi-site system with central control chose to distribute processing and storage to reduce communications cost, improve local availability, and minimize the scope of the failures. The federated system with node autonomy decentralized to reflect the structure of its user community.

All these applications use transactions as the unit of work and unit of recovery. The transaction mechanism coordinates updates and protects data integrity in case of a software or hardware failure.

One of the systems uses long-haul remote-IO: one node directly accesses records from a file at a geographically remote node. By consensus, this is less desirable than the requestor-server approach where all remote data requests are serviced by processes local to the data. All the other systems use remote servers when crossing geographic or organizational boundaries.

All the systems were built with off-the-shelf hardware and software. They demonstrate that the programming of such systems is not hard; all are programmed in conventional Cobol.

Everyone agrees that managing a distributed system is the real challenge. In part, this is the large-system effect. Distributed systems tend to be large. Large systems are hard to manage. But having remote hardware, and with unskilled staff, present new and difficult design problems.

It is easier to centrally manage a distributed system than it is to manage a collection of autonomous nodes, each staffed by a separate operations group. Centralization of operations reduces organization problems.

Strategies for tolerating low skill levels at remote nodes include the use of fault-tolerant hardware and software so that maintenance can be periodic. In addition, all operator

interfaces should be designed to be done from the central operations site. Operations tasks designed for hands-on operation must be redesigned to allow remote operation. If this is not possible, then the remote operator interface must be fool-proof to compensate for lack of skill at the remote sites. In short, reduce or eliminate remote operations tasks.

Monitoring and diagnosing communications lines, remote devices, and remote nodes should be given special attention in design and operations.

Two of the systems fit into a larger network. The bank interfaces to a shared-ATM network and to an inter-bank clearing house. The phone store application also provides pass-through to a variety of CICS hosts running on IBM equipment. We have not emphasized the issues of heterogeneous distributed systems primarily because this is an area that still requires some pioneering work. We are optimistic that the emerging standards, notably SNA LU6.2 and OSI levels 6 and 7, will provide transaction and requestor-server protocols across heterogeneous systems. Certainly that is their intent. But it will take several years to see whether they are successful.

To summarize, building a distributed application is no longer a pioneering effort -- such systems can be built with off-the-shelf hardware and software. Distributed databases are not the key to distributed applications. Rather, distributed execution is the key to constructing efficient and manageable distributed applications. The application is structured as requestors and servers which communicate via requests and replies. A transaction mechanism coordinates the activities of requestor and server processes so that programmers can treat the servers as subroutines and have a simple execution model.

ACKNOWLEDGNENTS

We would like to thank those who consulted with us to explain their systems. In particular Andrea Borr, Tom Eastep, Roberta Henderson, and Charlie Linburg of XRT, Harald Sammer of Tandem, and Rocco Martino of XRT. Flaviu Cristian and several anonymous reviewers helped structure the presentation.

REFERENCES

[Borr] Borr, A., "Transaction Monitoring in Encompass: Reliable Distributed Transaction Processing", Proc. 7th Int. Conf. Very Large Databases, IEEE N.Y., Sept 1982, pp. 155-165.

[Date] Date, C.J., *A Guide to DB2*, Addison Wesley, 1984.

[CICS] CICS/VS Version 1.6, General Information Manual, GC33-1055, IBM Corp., Armonk, N.Y., 1983.

[Gray] Gray, J.N., Hansen, P.J., Homan, P., Lerner, M.A., Pozefsky, M., "Advanced Program to Program Communication in SNA", IBM Systems Journal, Vol. 22, No. 4, Oct. 1983, pp. 298—318

[Haeder] Haeder, T., Reuter, A., "Principals of Transaction-Oriented Database Recovery", A Computing Surveys, Vol. 15, No. 4, Dec. 1983.

[Horst] Horst, R., Chou, T., "The Hardware Architecture and Linear Expansion of Tandem NonStop Systems" Proceedings of 12th International Symposium on Computer Architecture, June 1985; or Tandem Technical Report 85.3

[Nelson] "Remote Procedure Call", PhD Thesis, Department of Computer Science, Carnegie Mellon University, 1981

[Norman] Norman, A.D., Anderton M., "Empact, a Distributed Database Application", Proceedings of AFIPS National Computer Conference, Vol. 52, pp. 203-217, 1983.

[Sammer] Sammer, Harald, "The UBF system", Tandem, P0. 560214, 6000 Frankfurt/Main 56, West Germany

[Martino] Martino, R.L., "The XRT Solution", XRT Inc., 18 Great Valley Parkway, Malvern, Penn. 19355.

[Pathway] "Introduction to Pathway", Tandem Computers Inc., Part No: 82339-A00, Cupertino, CA. 95014-2599, June 1985.