The Cost of Messages

Jim Gray

Technical Report 88.4
March 1988
Part Number: 14661

THE COST OF MESSAGES

Jim Gray

March 1988

Tandem Technical Report 88.4

Abstract: Distributed systems can be modeled as processes communicating via messages. This model abstracts the three degrees of distribution: shared memory, local network, and wide area network. Although these three forms of distribution are qualitatively the same, there are huge quantitative differences in their message transport costs and message transport reliability. This paper quantifies these differences for past, current, and future technologies.

Table of Contents

Introduction

Virtually all computers are structured as distributed systems. The computers in wristwatches and microwave ovens are still structured as VonNeumann computers: a single processing unit with dedicated memory and peripherals. But larger computers: workstations, minis, mainframes, and supers are structured as nonVonNeumann machines with multiple processors cooperating to perform a task. In a workstation, this distribution typically takes the form of functional specialization, one processor manages the display, another manages discs and other peripherals, and others run the operating system and applications. In a departmental system, work is distributed as a local network of workstations, servers, gateways, super-computers, and so on. Such local networks are typically part of a larger wide-area network, each node of the network manages a geographic partition of the global system's processing and data.

The resulting hierarchy of distributed systems emerges from the above observations:

- **Central:** A shared memory multi-processor.
- **Local:** A local network connecting several central nodes.
- **Wide-Area:** A long-haul network connecting several local networks.

This article sketches the commonly used execution model for distributed systems -- processes and messages. It documents the folklore that although distributed systems are qualitatively the same, there are huge quantitative differences among these three forms of distribution. These quantitative differences are:

- **Cost**: The time and equipment cost to transport messages rises by at least an order of magnitude at each degree of distribution.
- **Reliability**: The reliability of message transmission drops by at least an order of magnitude at each degree of distribution.

As a consequence, the message cost of a distributed algorithm is an important measure of its cost.

The paper's organization is as follows. First, a simple execution model for distributed systems is outlined. Then the model is refined to include message cost and faults. These attributes are quantified by reference to the performance and reliability of commercially available computing and communication equipment, and by forecasts of future equipment.

This development substantiates the claims that messages are expensive and wide-area messages are very expensive. It also shows that messages are sometimes lost, and that wide-area messages are the least reliable component of a distributed system.

An Execution Model

At the hardware level, the three forms of distributed system can be diagramed as in Figure 1.a.

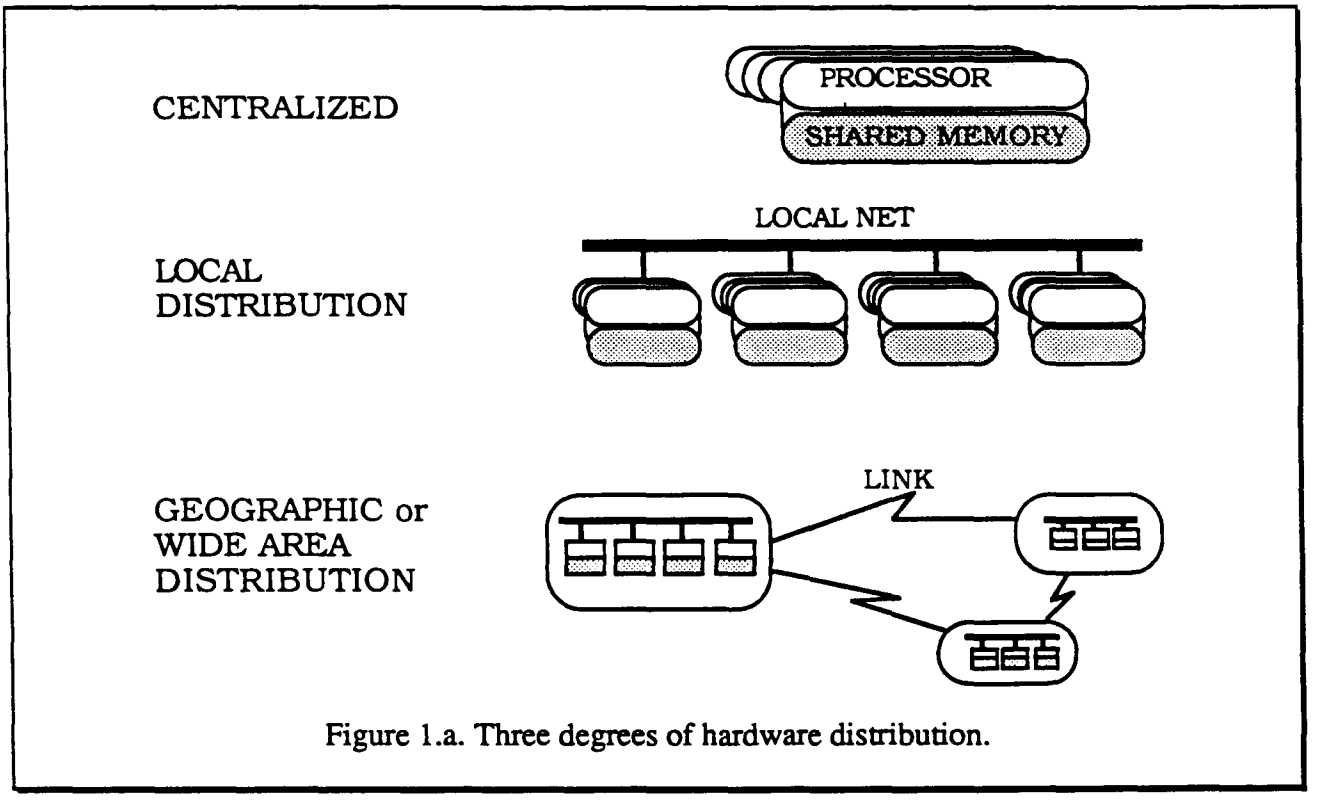

Figure 1.a. Three degrees of hardware distribution.

But at the software level, all three designs provide the common execution model shown in Figure 1.b. In this software model, there are only two abstractions: processes and messages. Processes can send messages to other processes in the network without being aware of their location or distance -- this ignorance of location is called location-transparency.

Figure 1.b. The software abstraction describing all three degrees of hardware distribution has only two abstractions: processes and messages.

In this simple execution model, each process has a unique name which is a surrogate for its location. The process executes sequential programs against private state variables. It runs at finite (non zero) speed and may perform the following actions:

- Execute a statement on its local state.
- Send or receive a message.
- Create another process.

The execution of:

SEND(*to*: process, *data*: message)

by a process called the sender, creates a *message* located *at* the sending process. Eventually the message is moved *to* the receiving process -- that is, eventually *message.at* changes to *message.to*. The delivery delay can be deterministic or probabilistic. The execution of:

RECEIVE () RETURNS (*message*: data)

by a process named P has one of two outcomes:

If there is some message *to* P *at* P: ($\exists$ *message: message.to* = *message.at* = P)
then return a copy of that *message.data* as the result and delete the message;
otherwise, return null.

Processes model executing programs, active devices, sensors, transducers, and people interacting with the system. Messages are the only mode of communication among processes. Attention is restricted to the process-message abstraction because it is the unique aspect of distributed systems and because, as shown below, message handling dominates both cost and failure statistics.

This simple model ignores broadcast, multi-cast, and does not count message hops because these features are second-order and complex. Similar conclusions emerge when these features are included in the model.

A Cost Model

Computer and communications prices are difficult to understand. For non-technical reasons, prices do not reflect true costs. For example, the cost of long-haul communications is artificially high today due to government regulations.

Fortunately, time cost is easier to measure and a definite pattern emerges from a cost model based on the delay to send and receive a message.

The time cost (delay) to send and receive a message can be computed as:

$$\text{Delay} = \text{Transmit\_Delay} + \frac{\text{Message\_Size}}{\text{Bandwidth}} + \text{CPU} \qquad (1)$$

Where Transmit_Delay is the speed of light delay to send one bit,
CPU is the processing time required to send and receive the message,
Message_Size is the size of the message in bits, and
Bandwidth is the speed (bits/second) of the communication media.

This model assumes no queueing delays. If network utilization is high (say 50%) then the total delay may double. The model also ignores transmit delays caused by passage through intermediate bridge, gateway and switch nodes. Such delays are common in wide-area communication networks. But interest here is in orders of magnitude and so queueing and switching delays are ignored.

Estimating the Transmit_Delay is easy: for short distances, it is dominated by setup times; for long distances it is dominated by the finite speed of light. Over public carriers, it is about a millisecond per hundred miles. Assuming a diameter of 1000 miles for the wide-area network the typical transmit delays for the various kinds of distributed systems is:

Speed of Light Delay in Network	
Shared Memory:	1μs
Local Net:	10μs
Wide-Area Net:	10,000μs

(2)

The delay in the public network can rise to 100ms when communicating via terrestrial lines between Tokyo and New York, or Los Angeles and London. The delay may rise to 300ms when satellite links are used. So 10ms is a very conservative estimate of public network delays.

Computing the other components of message delay requires estimates of communications bandwidth, software overhead for various protocols, and processor speeds. These numbers are in flux. The following are conservative assumptions about the capabilities of processors, local networks, and public wide-area networks commonly available in the years 1980, 1990, and 2000.

Technology Forecast

Year	1980	1990	2000	
CPU Speed	1mip (vax)	10mip (sparc)	100mip (sparc)	(3)
Local Net Bandwidth	10mb/s (Ethernet)	100mb/s (many)	1gb/s (fiber)	(4)
Wide-Area NET Bandwidth	5kb/s (voice)	50kb/s (ISDN)	1mb/s (fiber)	(5)

The CPU speed predictions are conservative. For example SUN Microsystems predicts a 32mip workstation in 1990, and a 16,000mip workstation in the year 2000 (Joy's law: mips = $2^{year-1985}$). The bandwidth predictions are also conservative; various phone companies are looking for applications to justify building a 100Mb public network based on high-bandwidth fiber optics.

The message transmission time for the three forms of network during various decades is computed by (Message_Size/Bandwidth). To simplify the presentation, assume that messages are about 100 bytes long. This assumption is wrong for file transfer operations, but is typical of distributed computations based on remote procedure call. Message headers are typically 32 bytes, and message bodies are typically a bundle of a few parameters. The local and wide-area network transmission times are computed by dividing the message length by the bandwidth (from tables (4) and (5)) and then rounding. The shared memory message transmission time is computed by estimating the time to copy the message from the sender's buffer to the receiver's buffer. The resulting message transmission times are:

Message Transmission Time (100 bytes)

Year	1980	1990	2000	
Shared Memory	10 μs	1 μs	.1μs	
Local Net	100 μs	10 μs	1μs	(6)
Wide-Area Net	200,000 μs	20,000 μs	1,000μs	

Processing time is the last component of delay in formula (1). Surprisingly, there is a definite pattern for the cost of sending messages over various communication media.

In a shared memory system, sending and receiving a message involves creating the message, dispatching the receiver process which reads the message, and then deallocating the message. A good implementation will take several hundred instructions to implement this. A typical implementation may take several thousand instructions. Assume a good implementation uses about 250 instructions.

In a local network, the cost of sending a message ranges upward from 2,500 instructions [Cheriton], [Watson], [Uren]. The increased cost, compared to the shared memory case, comes from the need to frame and checksum the message, do packet assembly and disassembly, execute standard protocols, and pass through operating system layer's overhead to get to the network. You might think that some smart programming could reduce the CPU cost of local messages. Indeed, low level messages can be sent in 500 instruction times [Kronenberg, Uren], but these interfaces are unprotected and very limited in function. Most algorithms are not allowed to use this raw interface to the hardware. Unfortunately, sending plus receiving a message in 2,500 instruction times is considered a very good performance at the operating system level (VMS or Guardian for the referenced systems). Systems with that performance have been very carefully designed and implemented. It is easy to find implementations that cost five times that [Watson]. The best hope of reducing the CPU cost is to build special co-processors to perform the local message protocols -- but this just hides the cost "outside" the processor and is likely to have the same delays. So the only "real" hope is much faster processors.

Exchanging messages over a wide-area network typically involves sending a message to a local gateway process, which then sends the message to a remote gateway process, which in turn passes the message on to the destination process. This structure costs at least a factor of three in message passing. In addition, the wide-area network protocols are typically more elaborate than the local network protocols. Consequently, wide-area protocols have larger instruction times. One implementation of X.25 consumes about 12,000 instructions to send and receive a message [X.25], another implementation based on SNA consumes about 15,000 instructions [CICS]. Measuring wide-area network processing costs is subtle, since "outboard" communications processors may perform much of the work. These communications processors are often slow and so may contribute significantly to delay. Hence, the estimate of 12,000 instructions per message is conservative for wide-area network protocols.

These measurements are summarized in the CPU column of table (7). Subsequent columns of Table 7 combine the send-receive instruction counts with CPU speeds (table (3) above) to compute the processing delay for a message transmission.

SEND+RECEIVE Processing Cost (CPU instructions, time)

	CPU	1980	1990	2000	
Procedure Call	25ins	25 µs	2 µs	.2 µs	
Shared Memory	250ins	250µs	25 µs	2 µs	
Local Net	2,500ins	2,000 µs	250 µs	25 µs	(7)
Wide-Area Net	12,000ins	12,000 µs	1,000 µs	120 µs	

Now the delay defined by equation (1) can be displayed by combining tables (2), (6), and (7).

Delay to SEND & RECEIVE 100 Bytes

	1980	1990	2000	
Procedure Call	25 µs	2 µs	.2 µs	
Shared Memory	261 µs	27 µs	3 µs	
Local Net	2,000 µs	270 µs	36 µs	(8)
Wide-Area Net	222,000 µs	31,000 µs	11,000 µs	

Table (9) displays the dominant delay term for each entry of table (8). The bottleneck in local networks is processing time (protocol); bandwidth or the speed-of-light delay are not the dominant delays in local networks. In contrast, the bottleneck in wide-area networks is currently bandwidth, but the speed-of-light delay will be the bottleneck in the future. As the speed-of-light delay begins to dominate, the gap between the performance of local and wide-area nets will grow.

Dominant Message Delay Factor

	1980	1990	2000	
Shared Memory	CPU	CPU	CPU	
Local Net	CPU	CPU	CPU	(9)
Wide Area Net	bandwidth	bandwidth	light	

Summarizing, the patterns that emerge from (8) and (9) are:

- Messages in a shared memory system cost ten times more than procedure calls.
- Local network messages cost ten times more than shared memory messages.
- Wide-area network messages cost one hundred times more than local messages.
- The gap between local and wide-area message costs is increasing.
- CPU is the bottleneck in local nets, bandwidth or propagation delay is the bottleneck in wide-area networks.

What about Broadcast?

The execution and cost models ignore broadcast communication. This convenient simplification troubled several reviewers. One reason for the omission is that the topic is controversial. I have tried to document folklore while avoiding controversy. But there is considerable interest in broadcast because it figures prominently in several basic algorithms -- for example most Byzantine agreement algorithms use several broadcast rounds.

The execution model of broadcast is that one process SENDs a message addressed to all other processes, and eventually the message is delivered and RECEIVEd by all other processes. The consequent cost model (ignoring queueing) is similar to the simple SEND-RECEIVE model. The delay is approximately the average message delay time. The processing cost for N receivers is approximately $1+\frac{N}{2}$ times the processing cost of a simple message.

Bus and ring local networks offer a broadcast media. Gateways and bridges among such nets can propagate broadcast messages. Wide area networks are generally point-to-point rather than broadcast. Broadcast channels can be bought using satellite technology, but such channels have high transmission delay (.3sec/hop) and are relatively expensive. The use of fiber optics in wide area networks will provide point-to-point, not broadcast, transmission. So here is another fundamental difference between local and wide area networks -- broadcast "fits" in local network technology, but not in wide-area technology.

Practitioners have avoided heavy dependence on broadcast because it implies algorithms with N^2 cost. That is, if each of N nodes is using broadcast then total message traffic and message processing overhead rises as N^2. Such overhead limits the maximum size of a network, since each processor must handle ~N broadcast messages per second. Based on Table (7), a network of 1000 ten mip processors broadcasting once a second would be saturated. Practitioners want architectures which scale to arbitrary size networks. As a consequence they limit attention to *multicast:* broadcast to a "small" group of processes [Cheriton]. Multicast scales to very large networks; that is, multicast to sets of bounded size induces a constant load on each node as the network grows. Mulitcast to a set of M processes has delay similar to a single message, and processing cost $\sim\frac{M}{2}$.

A Fault Model

Lampson and Sturgis [Lampson] postulated that there are three forms of behavior:

Correct:	The object behaves as specified.
Fault:	The object misbehaves in an expected way.
Disaster:	The object misbehaves in an unexpected way.

Designing N-fault tolerant algorithms is a major focus of current research. Such algorithms deliver correct results if there are at most "N" faults in a specified time interval. More than N faults in the interval is classed as a disaster. The algorithms do not tolerate disasters. In disaster cases the algorithms give unspecified behaviors. Disasters are declared to be very rare and are ignored.

Process behaviors are:

Correct:	Process eventually properly executes next sequential instruction.
Fault:	Process resets to start state and eventually executes first instruction.
Disaster:	Program is incorrect.
	Process incorrectly executes next sequential instruction.

This is the FailFast model [Gray]. If failed processors are never repaired then it is called the FailStop model [Shicterling].

Message behaviors are:

Correct:	Message is created by a process.
	Message is delivered to destination.
	Message text is received as sent.
Fault:	Message is lost.
	Message is detectably corrupted.
	Message delivery is delayed forever.
	Message is delivered multiple times.
Disaster:	Message is undetectably corrupted.
	A message is spontaneously created.

As shown by Lampson and Sturgis, the use of sequence numbers, checksums, and timeout can convert all message faults to lost-message faults. So, ignoring disasters the fault model is:

Process:

Correct:	Process eventually properly executes next sequential instruction.
Fault:	Process resets to start state and eventually executes first instruction.

Messages:

Correct:	Message is created by a process.
	Message is delivered to destination.
	Message text is received as sent.
Fault:	Message is lost.

Lampson and Sturgis have shown how to build single-fault tolerant processes and messages from faulty ones. They use pairs of processes with independent failure modes to mask process faults, and use message retransmission to mask lost messages. They assume that there is at most one fault within the repair window, and that messages are usually quickly delivered. These ideas are quite old and have been implicitly used for many years in fault-tolerant computers -- the contribution of Lampson and Sturgis was to define the failure model and the general techniques.

The generalization of these ideas to tolerate multiple faults is obvious, but multi-fault tolerance is rarely used in practice since single-fault tolerance typically gives theoretical mean-times to failure measured in centuries. Multiple faults are much less likely than a disaster like a program bugs or operator error. To be specific, if the mean time to module failure is one year and the mean time repair is one day, then duplexing gives a 365 year mtbf while triplexing gives a 100,000 year mtbf. Operations, software, and environment have fault rates higher than duplexed modules.

In discussing faults, it is useful to distinguish between transient faults and hard faults. A transient fault is one that will not be repeated if the operation is immediately retried; a hard fault will require time to correct. By duplexing processes and communications links, one can convert (mask) most hard faults to transients by resending messages and by using a backup process if a primary process fails.

Fault rates are quantified as follows. Fault rates for processors are well understood. The hardware is rated at years, but due to environmental, software, and operations problems, processors are assumed to have a mean times to failure measured in weeks or months and a mean times to repair measured in minutes. Duplexing a processor and installing emergency power can convert most hard processor faults to transient processor faults. Hiding these transient faults

creates a node with mean time to failure measured in decades. For such nodes, environment, software, and operations faults (disasters) are the main sources of outage [Gray].

Well engineered local networks have bit error rates of 10^{-10}. This translates into about one fault (corrupted message) per hour. By using timeout and retransmission, along with duplexed local networks, transient local network faults can be hidden. If this is done, environmental problems, software, or operations may disable the local net every few decades.

Wide-area networks promise bit error rates of 10^{-6} [ATT]. This in turn translates to one damaged message per second for a 1Mb/s communications line. Again, these errors are transient and clustered -- in fact ATT's specifications for T1 lines promises 95% error free seconds. So once every twenty seconds, there may be a burst of errors. T1 lines also promise 99.7% availability -- that is they may be out of service for 4 minutes per day. Again, duplexing may help here but experience suggests that you should expect some short but visible outage of communications every few years.

A measure of the ARPA Network lines [Kleinrock] showed that links had widely varying fault rates, some lost a packet every 4 seconds, others never lost a packet. The average fault rate was 40 seconds.

Summarizing these fault rates:

Distributed System Component Fault Rates

	fault/sec	mtbf	duplexed	
Processor:	~ 10^{-8}	weeks	decades	(10)
Local Net:	~ 10^{-5}	hours	decades	
Wide-Area Net:	~10^{-1}	seconds	months	

Table (10) shows a familiar pattern: centralized systems are more reliable than local nets, local nets are much more reliable than wide-area networks. There are orders of magnitude differences between the raw error rates, and there is a huge difference between the corrected mtbf of a duplexed local net and that of a duplexed wide-area net.

In Defense of Distributed Systems

So far the case against distributed systems seems pretty clear: wide-area networks are slow and unreliable -- not to mention fabulously expensive. Local networks are thousands of times faster, more reliable, and cheaper -- but in comparison to centralized systems, local networks waste instructions and time.

This narrow focus ignores the benefits of distributed systems -- they allow many processors to be applied to a problem in parallel. In addition distributed systems offer high availability through fault tolerance. They also provide modular growth, and geographic distribution of data and processing next to the primary consumers of the data and processing.

Distributed systems offer good peak performance through parallelism, and good price performance by using inexpensive components. Often it is simply not possible to construct a shared memory system with comparable power.

It is a paradox that one must build on faulty components to get high availability through fault-tolerance. The fault-tolerant designs mentioned in the previous sections depend on a distributed hardware and software base to mask processor faults by switching to a backup processor. Shared-memory multi-processors can provide high reliability through fail fast designs, but they cannot provide high availability unless there is a fallback system with good fault isolation to give "instant" mean time to repair. Such backup systems are structured as distributed systems. The trend is to geographically distribute (replicate) processing and data to protect against common mode failures (power failure, fire, weather, sabotage, operator error,...) [Lyon].

Properly designed distributed systems are often more economic and more reliable than their centralized analogs. In part this is due to the use of many inexpensive hardware components rather than a few very high-performance but very expensive ones, and in part it is due to the distributed system reflecting the structure of the user's organization and tasks.

Conclusions

The main reason for emphasizing the comparatively high cost and low reliability of messages is that one occasionally reads of algorithms which are message intensive and which assume that processors are less reliable than message delivery. For example, Byzantine agreement algorithms often have this flavor -- they typically send a number of message rounds, each round consisting of n^2 messages; and, they typically treat a damaged message as a processor failure. This high message density is expensive, and actually increases the chance of failure by heavy dependence on message transport which is the least reliable part of the system [Babaoglu].

The obvious conclusion from the tables of message cost (8) and message failure rates (10) is that local and wide-area networks are fundamentally different -- and the difference is increasing. It is probably the case that different algorithms are appropriate for them. Algorithms which work well in a local network, may be disastrous in a wide-area net.

The delay cost to transport a message -- for local nets it is in the range of 10^3 instruction times, for wide-area messages it is in the range of 10^5 instruction times and rising. These are significant numbers of instructions when compared to the logic of most distributed algorithms. So messages are expensive enough to merit careful counting when evaluating the performance of a distributed algorithm.

Lastly, fault tolerant algorithms should be aware of the relatively high fault rate of wide-area message transport -- it is the least reliable component of a distributed system.

Acknowledgments

Critiques by Andrea Borr, David Cheriton, and Michael Merritt improved this paper.

References

[ATT] "High Capacity Terrestrial Digital Service", ATT Pub. 41451, Jan 1983

[Babaoglu] Babaoglu, O., "On The Reliability of Consensus Based Fault Tolerant Distributed Computer Systems", ACM TOCS, V. 5.4, 1987.

[Cheriton] Cheriton, C.R., "The V Distributed System", CACM, V31.3, 1988.

[CICS] *CICS/VS Performance Data*, IBM Form# SC33-0212-1, Armonk, N.Y., 1986.

[Gray] Gray, J., "Why Do Computers Stop?", Tandem Computers P# 87614, Cupertino, CA, 1985.

[Kleinrock] Kleinrock,L., Naylor, R., "On the Measured Behavior of the ARPA Network", AFIPS, V43, 1974.

[Lampson 81] Lampson, B.W., "Atomic Transactions", *Distributed Systems -- Architecture and Implementation: An Advanced Course*. Chapter 11.Lecture Notes in Computer Science #105, Springer-Verlag, 1981.

[Lyon] Lyon, J. "Design Considerations in Replicated Database Systems for Disaster Protection", CompCon 88 Digest of Papers, IEEE Press, 1988.

[Shicterling] Shicterling, R, Schneider, F., "Fail-Stop Processors, An Approach to Designing Fault Tolerant Computer Systems", ACM TOCS, V1.3, 1983.

[X.25] *X25 Access Method -- X25AM*. Tandem Computers P# 82434, Cupertino, CA, 1985

[Kronenberg] Kronenberg, N., Levy, H., Strecker, W., "VaxCluster: A Closely-Coupled Distributed System", ACM TOCS, V4.2, 1986.

[Uren] Uren, S., "Message System Performance Tests", Tandem Systems Review, V3.4, Dec. 1986.

[Watson] Watson, R.W., & S.A. Mamrak, "Gaining Efficiency in Transport Services", ACM TOCS, V5.2, May 1987.

Distributed by
TANDEM
Corporate Information Center
10400 N. Tantau Ave., LOC 248-07
Cupertino, CA 95014-0708