Communication Facilities for Distributed Transaction-Processing Systems

Enrique Mafla and Bharat Bhargava, Purdue University

Communication software is critical in distributed computing systems. This research identifies efficient mechanisms and paradigms for distributed transaction processing in a replicated database environment.

Distributed transaction-processing systems must manage such functions as concurrency, recovery, and replication. One way to improve their efficiency and reliability is to increase software modularity, which means the separate components should execute in separate address spaces to permit hardware-enforced separation. This structure offers advantages but demands efficient interprocess communication (IPC) services.

In our research at Purdue University, we are investigating mechanisms and paradigms for efficient communication support in conventional architectures, such as virtual-memory, single-processor machines with no special IPC hardware support. (Some mainframes have hardware assistance where more than one address space can be accessed at the same time.)

We are studying communication designs in the context of the Raid system, a robust and adaptable distributed database system for transaction processing.[1] Raid has been developed at Purdue on Sun workstations under the Unix operating system in a local area network.

In Raid, each major logical component is implemented as a server, which is a process in a separate address space. Servers interact with other processes through a high-level communication subsystem. Currently, Raid has six servers for transaction management: the user interface (UI), the action driver (AD), the access manager (AM), the atomicity controller (AC), the concurrency controller (CC), and the replication controller (RC). High-level name service is provided by a separate server, the oracle.

Raid's communication software, called Raidcomm, has evolved as a result of the knowledge we gained from other systems and from our own experiments, which are summarized in the following sections. The first version, Raidcomm V.1, was developed in 1986. Implemented on top of the SunOS socket-based IPC mechanism using UDP/IP (User Datagram Protocol/Internet Protocol), it provides a clean, location-independent interface between the servers.[2] To permit defining server interfaces in terms of arbitrary data structures, we used Sun's external data representation standard, XDR. We developed Raidcomm V.2 in 1990 to provide multicasting support for the AC and RC servers. We designed Raidcomm V.3 to

0018-9162/91/0800-0061$01.00 © 1991 IEEE

support transmission of complex database objects. It is based on the explicit control-passing mechanism and shared memory.

Related research work

Research in operating systems, multiprocessor systems, and computer networks has made useful contributions to the field of communication facilities:

- Carnegie Mellon's Camelot project[3,4] identified communication subsystem requirements.
- Remote procedure call (RPC)-based session services support the interaction among data servers and applications.
- Highly specialized datagram-based communication facilities increase the performance demanded by data servers and applications.
- Efficient local and remote IPC has been investigated in many distributed computing systems, for example, the V system,[5] Mach,[6] Amoeba,[7] Sprite,[8] and *x*-kernel.[9]

We classify the existing communication paradigms into three groups: local interprocess communication, remote interprocess communication, and communication protocols for both local area and wide area networks.

Local interprocess communication. To improve local machine performance, Bershad[10] introduced two new mechanisms: lightweight remote procedure call (LRPC) and user-level remote procedure call (URPC). LRPC takes advantage of the control transfer and communication model of capability-based systems and the address-space-based protection model of traditional IPC facilities. URPC, another cross-address-space communication facility, eliminates the role of the kernel as an IPC intermediary by including communication and thread management code in each user address space.

Both LRPC and URPC were implemented on a DEC SRC Firefly multiprocessor workstation running the Tao operating system.[10] A simple cross-address-space call using SRC RPC takes 464 microseconds on a single C-VAX processor. LRPC takes 157 microseconds for the same call, and using URPC reduces the call's latency to 93 microseconds.

We've found the ideas used in LRPC and URPC applicable in systems such as Raid.

Remote interprocess communication. Several message-based operating systems can reliably send messages to processes executing on any host in the network. The V system[5] implements address spaces, processes, and the interprocess communication protocol in the kernel to provide a high-performance message-passing facility. All high-level system services are implemented outside the kernel in separate processes. Mach[6] uses virtual-memory techniques to optimize local IPC. Remote communication goes through a user-level server process, which adds extra overhead. Amoeba[7] uses capabilities for access control and message addresses. It has a small kernel, and most features are in user processes.

However, not all the systems use the small-kernel approach with remote IPC outside the kernel. In Sprite,[8] the IPC is through a pseudodevice mechanism. Sprite kernel communication is through Sprite kernel-to-kernel RPC. RPC in *x*-kernel[9] is also implemented at the kernel level. Table 1 shows the performance of various RPCs over Ethernet.

Communication protocols. Communication protocols provide a standard way to communicate between hosts connected by a network. Datagram protocols such as IP are inexpensive but unreliable. However, more reliable protocols, such as virtual-circuit and request-response protocols, can be built on top of datagram protocols.

The versatile message transaction protocol (VMTP) is a transport-level protocol that supports the intrasystem model of distributed processing.[5] Page-level file access, remote procedure calls, real-time datagrams, and multicasting dominate the communication activities. VMTP provides two facilities, stable addressing and message transactions, useful for implementing conversations at higher levels. A stable address can be used in multiple message transactions, as long as it remains valid. A message transaction is a reliable request-response interaction between addressable network entities (ports, processes, procedure invocations). Multicasting, datagrams, and forwarding services are provided as variants of the message transaction mechanism.

Using virtual protocols and layered protocols, the *x*-kernel implements general-purpose yet efficient RPCs.[9] Virtual protocols are demultiplexers that route the messages to appropriate lower level protocols. For example, in an Internet environment, a virtual protocol will bypass the Internet Protocol for messages originating and ending in the same network. The support of atomic broadcasting and failure detection within the communication subsystem simplifies transaction-processing software and optimizes network broadcasting capabilities.[11] For example, a two-phase commit protocol can be implemented by atomic broadcasting.

Table 1. Performance data for remote procedure calls. (The Sun 3/75 is a 2-MIPS machine, and the Sun 3/60 is a 3-MIPS machine.)

System	RPC Type	Architecture	Latency	Latency by MIPS
V	User level	Sun 3/75	2.50 ms	5.0 ms
Mach	User level	Sun 3/60	11.00 ms	33.0 ms
Amoeba	User level	Sun 3/60	1.10 ms	3.3 ms
Sprite	Kernel level	Sun 3/75	2.45 ms	4.9 ms
x-kernel	Kernel level	Sun 3/75	1.70 ms	3.4 ms

Studies and enhancements

We have conducted a series of experiments on the performance of the facilities available for building the Raid communication software.[2,12]

Experimental measurements and observations. The measurements were done on Sun 3/50s (1-MIPS machines) that use the SunOS 4.0 operating system. We configured one workstation

Charles F. Goldfarb, IBM Almaden Research Center

HyTime is being developed as an ment, plus the large capital and orga- and SPDL. Some industry standards

with a special microsecond resolution clock to measure elapsed times. (This timer board, developed by Peter Danzig and Steve Melvin, uses Advanced Micro Devices' AM9513A timer chip. The timer has a resolution of up to four ticks per microsecond, and the overhead to read a time stamp is approximately 20 microseconds.)

Our experimental work focused on general-purpose interprocess communication facilities, multicasting, and the impact of interprocess communication on distributed transaction-processing performance.

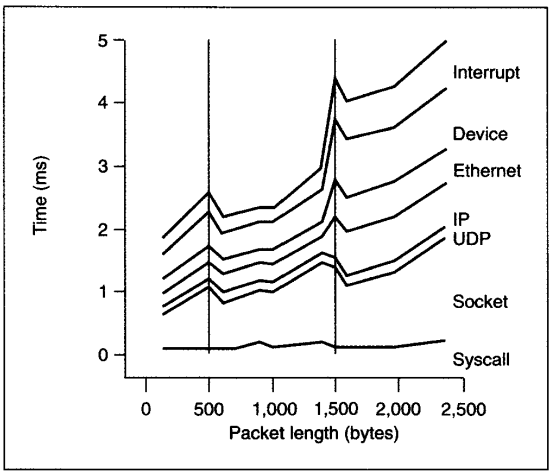


Figure 1. Timing for a User Datagram Protocol send.

Communication. We measured the overhead introduced by the layers of the socket-based interprocess communication model for datagram communication (UDP). These layers include the system call mechanism, the socket abstraction, communication protocols (UDP, IP, and Ethernet), and interrupt processing.

Figure 1 shows each layer's contribution to the total time of the send operation of user-level messages. We found that the socket abstraction, which included copying the message between user and kernel spaces, was expensive. Starting the physical device required approximately 20 percent of the total send time. The peaks are due to the SunOS's special memory allocation policy.[12]

We investigated several mechanisms, including message queues, named pipes, shared memory synchronized by two semaphores, and UDP sockets in both the Internet and Unix domains. We measured the round-trip time for a null message for each of these methods. We measured message queues at 1.7 milliseconds, named pipes at 1.8 milliseconds, shared memory with semaphores at 2.5 milliseconds, the Internet domain UDP socket at 4.4 milliseconds, and the Unix domain UDP socket at 3.6 milliseconds.

Multicasting. We studied several approaches to multicasting. The first two alternatives are based on the Simple Ethernet (SE), a suite of protocols that provide low-level access to the Ethernet.[12] The user-level SE multicast utility is implemented on top of the SE device driver, which provides point-to-point Ethernet communication. The kernel-level SE multicast utility uses the multi-SE device driver. Finally, we experimented with physical multicasting. Although physical multicasting minimizes bandwidth, it demands a priori knowledge of the multicast address by all group members. This requirement may incur extra messages to set up the address.

Impact. To observe the impact of Raidcomm V.1 IPC on Raid's transaction-processing performance, we ran the DebitCredit benchmark.[12] (Known as TP1 or ET1, DebitCredit is a simple yet realistic transaction-processing benchmark that uses a small banking database of three relations and 100-byte tuples.) The ratio of user times to system times for different servers is 2:1. Most of the system times are incurred by the communication activities.

Results. Details on the setup, procedures, and analysis of our experiments can be found elsewhere.[2,12] Below, we summarize only the major lessons and observations.

• *Expensive.* General-purpose communication facilities are too expensive,[10,12] even though many abstractions and mechanisms are useful to support a variety of applications and users. Messages have to go through several unnecessary layers of the communication subsystem. To overcome these problems, we recommend using a simple IPC memory management mechanism. Virtual and/or layered protocols in *x*-kernel and VMTP provide support to avoid such overhead.

• *Communication intensive.* Transaction-processing systems are communication intensive,[4] and most of the communication is local rather than remote.[10] If the local communication is handled as a particular instance of the remote case, the operating system kernel becomes the system bottleneck because of the high message traffic and the high cost to process messages. Communication facilities specialized for the local case can be simpler and more efficient.

• *Communication support.* Some operating systems do not provide enough communication support for distributed transaction processing. The transaction-processing system implementer has to supply these services. It is desirable to define high-level interfaces between the modules. For communication, the modules use typed messages rather than simple buffers of bytes supported by the operating system. To be sent, a message has to be marshaled into kernel buffers. The receiving side must perform the inverse operation.

• *Multicasting.* General-purpose multicasting mechanisms require group initialization and maintenance. In distributed transaction processing, multicasting groups are typically dynamic and short lived. In this case, the overhead of group initialization can obliterate the performance advantages of multicasting. Our experiment shows that simulating multicasting inside the kernel reduces CPU overhead.[12] Cheriton[5] has proposed multicasting for many applications. We suggest that the group (multicasting) addresses used during commitment time be established as a function of the unique transaction ID. This eliminates the need for extra messages to set up group addresses.

• *Name resolution.* Name resolution can become an expensive and complicated process. In general, we can have three different name spaces: application name space, interprocess communication name space, and network name

space. The Raid system uses a special protocol to map Raid names into interprocess communication addresses (UDP/IP addresses). These addresses have to be mapped into network addresses (for example, Ethernet addresses) via a second address resolution protocol. For a local area network, a straightforward correspondence between logical and physical communication addresses can be established.

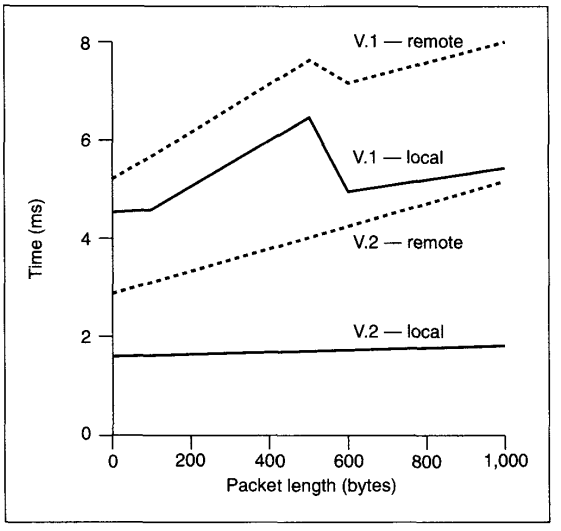


Figure 2. Round-trip times in milliseconds.

Enhancements. The Raidcomm V.2 implementation for local area networks employs low overhead, simple naming, and transaction-oriented multicast support. Some of these ideas are based on LRPC and URPC.[10] Below, we briefly discuss ports, naming, multicasting schemes, and communication primitives. (Details can be found elsewhere.[12])

Ports. Processes communicate through ports, which are the basic communication abstraction. These ports reside in a memory segment shared by the process and kernel address spaces. Thus, data can be exchanged without copying. This method reduces copying by 50 percent compared with other kernel-based IPC methods. The mapped memory segment contains a transmission buffer and a set of receiving buffers. The number and length of these buffers are specified by a process at the time it opens a port. The receiving buffers form a circular queue, which is coherently managed by the kernel and the process according to the conventional producer/consumer paradigm. Associated with the transmission buffer and each of the receiving buffers is an integer, which specifies the actual length of the message. In addition, there is a counter for the number of active messages (messages that have arrived, but which the server has not yet processed).

Naming. Within a given node, ports are uniquely identified by the triplet <Raid instance number, server type, server instance>. The other component of a Raid address, the site number or the transaction ID, determines the address of the physical node for monocast or the addresses of the group of nodes for multicast. In the case of Ethernet, we use only multicasting addresses for link-level communication. Site numbers or transaction IDs are used to build multicasting addresses by copying them into the four more-significant bytes of the Ethernet address.

Multicasting. In physical multicasting for processing a given transaction's data requests, each participant site sets a multicasting address using the transaction ID as its four more-significant bytes. At commitment time, the coordinator uses this address to multicast messages to all participant sites. This avoids the overhead of other multicasting methods. Currently, multicast addresses are added or deleted by Raid servers. The RC adds a new multicasting address for a transaction when it receives the first operation request for that transaction. Under normal conditions, the AC deletes the multicasting address once the transaction is committed or aborted. In the presence of failures, the CC does this job as part of its cleanup procedure. In the future, we plan to manage the multicasting addresses in the communication subsystem.

Communication primitives. System calls are provided to open and close a port, to send a message, and to add or delete a multicasting address. There is no need for an explicit receive system call. If idle, a receiving process must wait for a signal (and the corresponding message) to arrive. To send a message, a process writes it into the transmission buffer and passes control to the kernel. If the message is local, it is copied into a receiving buffer of the target port, and the port's owner is signaled (the owning process' ID is stored in the port's data structure). We use the Unix Sigio signal for this purpose. Otherwise, one of the existing network device drivers sends the message to its destination. The send operation will be aborted if there are not enough receiving buffers. The destination address is constructed as described above, and the message is enqueued into the device's output queue. When a message arrives over the network, it is demultiplexed to its corresponding port. Again, a signal alerts the receiving process about the incoming message. All this is done at interrupt time, so there is no need to schedule further software interrupts.

Performance of the communication primitives. We measured the latency of a user-to-user round trip of local interprocess communication in Raidcomm V.1 and V.2. In version 2 it is 1.4 milliseconds, compared with the 4.4 milliseconds of the UDP socket used in version 1. In version 1 the round-trip time increases by a 1-millisecond average for each kilobyte of data; in version 2 the round-trip time increases by 0.34 millisecond for each additional kilobyte. The latency of a user-to-user round-trip remote communication reduces from 5.1 milliseconds in version 1 to 2.7 milliseconds in version 2. The round-trip time increase for each kilobyte of data also reduces from about 3.0 milliseconds to 2.5 milliseconds. (See Figure 2.)

The socket-based IPC in version 1 and the new communication facility in version 2 provide the same functionality in a local area network environment, and both are equally affected by significant network device-driver overhead. Despite this fact, the new communication facility achieves improvements of up to 50 percent. For multicasting, the performance advantages of the new com-

Charles F. Goldfarb, IBM Almaden Research Center

HyTime is being developed as an ment, plus the large capital and orga- and SPDL. Some industry standards

munication facility become even more significant. The sending time does not depend on the number of destinations. On the other hand, the multicasting time for the socket IPC method grows linearly with the number of destinations.

Socket-based IPC does not optimize for the local case. Local round-trip costs are 68 to 88 percent of those for remote round trips. In the new communication subsystem, local round-trip times are only 35 to 50 percent of the corresponding remote round-trip times.

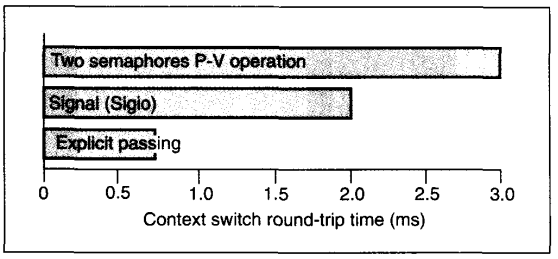


Figure 3. Performance of a context switch on the Sun 3/50 running SunOS 4.0.

Impact on transaction processing. We ran the DebitCredit benchmark on a single-site and a five-site system. For the five-site system, we used the ROWA (read-once, write-all) replication method. This limited remote communication to the AC server. The benchmark contained 115 transactions that had write operations and required access to remote sites in the two-phase commit protocol. Using Raidcomm V.2 reduced the system time for transaction processing by an average of 62 percent, bringing the user-time to system-time ratio to 3:1.[12]

Other issues. In Raidcomm V.3, we are addressing some problems with XDR, scheduling, and context switching.

XDR. New applications require the underlying communication subsystem to provide cheap transportation for complex data objects. The transmitted data structures are usually bounded linear buffers. We can build our local communication channel in shared memory to avoid multiple encoding/decoding. Unlike lower level buffer-based communication resources, which are one dimensional and usually accessible only as a whole, the shared memory segments are multidimensional, randomly addressed storage resources just like the main memory. These schemes can eliminate the overhead of XDR in local communication.

Scheduling. Scheduling policies should consider high-level relationships that may exist among a group of processes. Conflicts may exist between the optimization criteria at the operating system and application levels. The application should have some way to partially control CPU allocation. The sender and the receiver process can collaborate under some high-level mechanism provided by the operating system to transfer the thread of control. In Unix, for example, the receiver process often goes to sleep to lower its priority. It is then put in a list and waits for the event (I/O, signals) that will grant it CPU time.

Context switching. We also conducted a series of experiments on context switching, based on the idea of explicit thread control passing. This is similar to the hand-off scheduling in Mach, but we've extended it to schedule ordinary processes in Unix. Figure 3 shows round-trip times for a context switch between two processes. Raidcomm V.3 uses an explicit control-passing mechanism and shared memory. This reduces the latency of sending a high-level monocast Raid message to 0.68 millisecond. It was 2.4 milliseconds in Raidcomm V.1, and 1.1 millisecond in Raidcomm V.2.

Overall, we have identified several communication services and mechanisms that can make Raid efficient. Separate address spaces can be used to structure a DTP system. High interaction among servers also triggers costly context-switching activity in the system. Increasing availability through distribution and replication of data demands special-purpose multicasting mechanisms. The idea of shared memory between the kernel and user processes is appealing, since it reduces context-switching activity. The identification of sites involved in transaction processing for accessing replicated data can be used in physical multicasting.

Our work has studied Unix-like operating systems. Obviously, Unix is not the only system on which commercial systems might be built, but it does provide a good benchmark for experimental study of new ideas in an academic setting. Our work has been conducted in a local area network environment. Similar studies must be undertaken to identify and reduce the overhead in wide area networks.

We believe communications hardware and media technology is advancing at a rapid pace. Our research, along with related work in industry and academia, is intended to promote software advances. ■

Acknowledgments

Many students working on the Raid project have contributed to the system's communications facilities. Tom Muller, John Riedl, and Brad Sauder contributed to the earlier versions, and Yongguang Zhang is helping us with the design of Raidcomm V.3. Zhang has also helped in collecting data from the designers of the various systems discussed in the section on related work and in revising the article to meet the page limit.

We thank each referee for giving us detailed guidelines that helped improve the article.

This research is supported by NASA and AIRMICS under grant NAG-1-676, NSF Grant IRI-8821398, and AT&T.

References

1. B. Bhargava and J. Riedl, "The Raid Distributed Database System," *IEEE Trans. Software Eng.*, Vol. SE-15, No. 6, June 1989, pp. 726-736.

2. B. Bhargava, E. Mafla, and J. Riedl, "Communication in the Raid Distributed Database System," *Computer Networks and ISDN Systems*, Journal of the ICCC, Vol. 21, 1991, pp. 81-92.

3. A.Z. Spector, "Communication Support in Operating Systems for Distributed Transactions," in *Networking in Open Systems*, G. Müller and R.P. Blanc, eds., Springer Verlag, New York, 1986, pp. 313-324.

4. D. Duchamp, "Analysis of Transaction Management Performance," *Proc. 12th ACM Symp. Operating Systems Principles*, ACM, New York, 1989, pp. 177-190.

5. D.R. Cheriton, "The V Distributed System," *Comm. ACM*, Vol. 31, No. 3, Mar. 1988, pp. 314-333.

6. R.F. Rashid, "Threads of a New System," *Unix Review*, Vol. 4, No. 8, Aug. 1986, pp. 37-49.

7. A.S. Tanenbaum et al., "Experiences with the Amoeba Distributed Operating System," *Comm. ACM*, Vol. 33, No. 12, Dec. 1990, pp. 46-63.

8. J.K. Ousterhout et al., "The Sprite Network Operating System," *Computer*, Vol. 21, No. 2, Feb. 1988, pp. 23-36.

9. L. Peterson et al., "The *x*-kernel: A Platform for Accessing Internet Resources," *Computer*, Vol. 23, No. 5, May 1990, pp. 23-33.

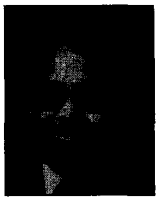
10. B.N. Bershad, "High-Performance Cross-Address Space Communication," PhD thesis, Tech. Report 90-06-02, University of Washington, Seattle, 1990.

11. K.P. Birman and T.A. Joseph, "Reliable Communication in the Presence of Failures," *ACM Trans. Computer Systems*, Vol. 5, No. 1, Feb. 1987, pp. 47-76.

12. E. Mafla and B. Bhargava, "Implementation and Performance of a Communication Facility for Distributed Transaction Processing," *Proc. Usenix Symp. Experiences with Distributed and Multiprocessor Systems*, Usenix Assoc., Berkeley, Calif., Mar. 1991, pp. 69-85.

Enrique Mafla is a faculty member at Escuela Politécnica Nacional, Casilla, Ecuador. His research interests include distributed systems, database systems, communication, and computer networks.

He received the BS degree in meteorology from the Odessa Institute of Meteorology, Odessa, USSR, in 1981 and the MS and PhD degrees in computer science from Purdue University, West Lafayette, in 1988 and 1990, respectively.



Bharat Bhargava is a professor in the Department of Computer Science at Purdue University. His research involves both theoretical and experimental studies in distributed database systems. He is working on adaptability in distributed systems, replication management, and new paradigms in communications for high-performance transaction processing. He is the editor of *Concurrency Control and Reliability in Distributed Systems* (Van Nostrand and Reinhold, 1987) and a recipient, with John Riedl, of the best paper award at the 1988 IEEE Data Engineering Conference.

Bhargava is on the editorial board of *IEEE Transactions on Knowledge and Data Engineering* and active in the IEEE Computer Society's Technical Committee on Distributed Processing.

Questions regarding this article can be addressed to Bhargava at the Department of Computer Science, Purdue University, West Lafayette, IN 47907.

Charles F. Goldfarb, IBM Almaden Research Center

HyTime is being developed as an ment, plus the large capital and orga- and SPDL. Some industry standards