Digital Technical Journal

INTERNET PROTOCOL V.6

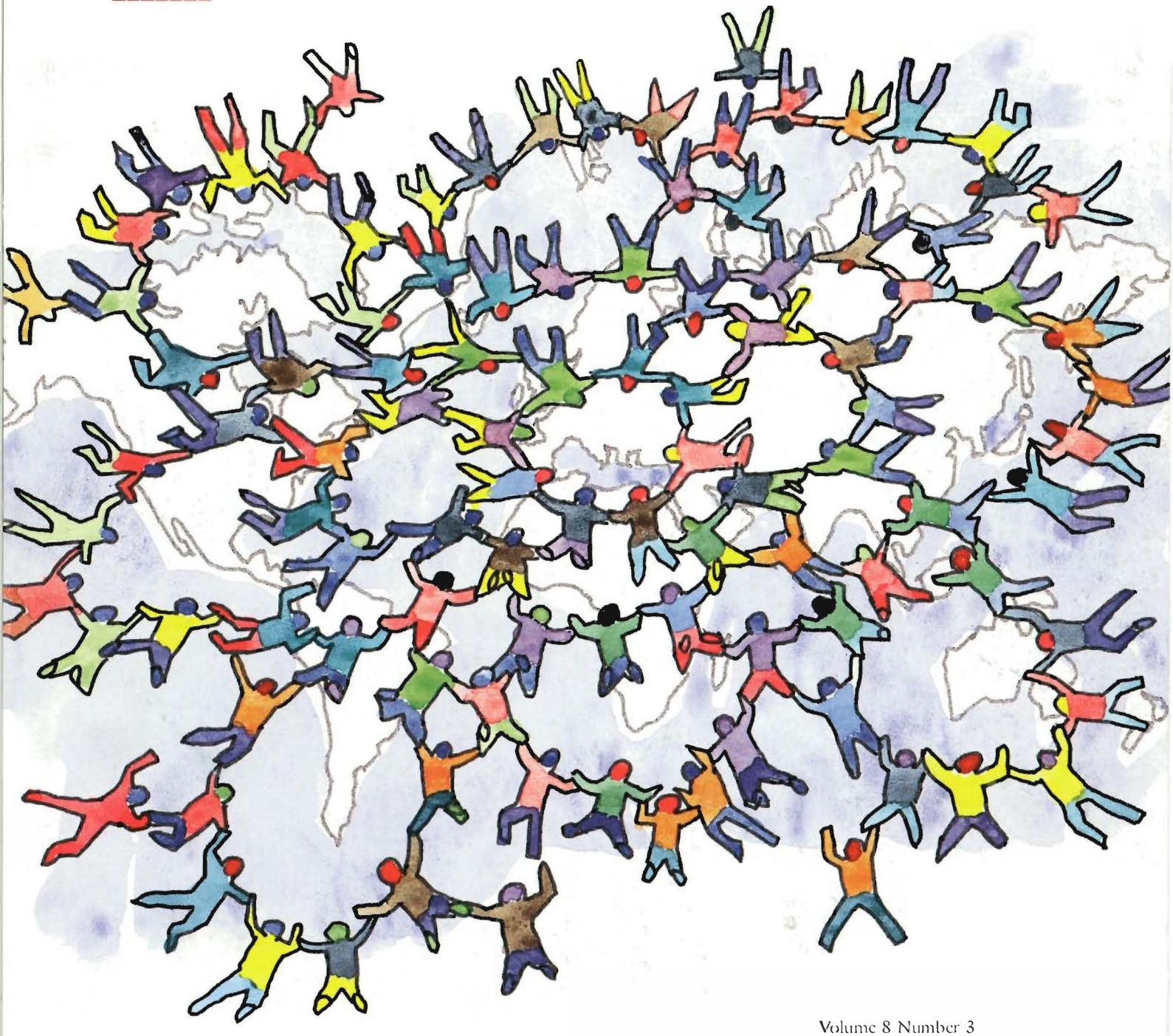
PRESERVATION OF HISTORICAL COMPUTER SYSTEMS

FORTRAN FOR PARALLEL COMPUTING

SERVER PERFORMANCE EVALUATION AND OPTIMIZATION

INTERNET COLLABORATION SOFTWARE

digital™



Volume 8 Number 3
1996

Cover Design
The function of the Internet is a simple one: Connect individuals through computer networks worldwide for the purpose of communication. The graphic on our cover symbolizes this worldwide connection of innumerable people in "cyberspace." Inside the issue, two papers address aspects of the complex work needed to make the connections, first, at the protocol level, Internet Protocol version 6, and at the user level, AltaVista Forum software for collaboration on the Internet.

The cover image is based on a photograph taken by Chuck Gillette of sky divers who set a record in October 1996 for the number of people (104) in a single formation. The cover design is by Lucinda O'Neill of Digital's Corporate Design Group.

The *Digital Technical Journal* is a refereed journal published quarterly by Digital Equipment Corporation, 50 Nagog Park, AKO2-3/B3, Acton, MA 01720-9843.

Subscriptions can be ordered by sending a check in U.S. funds (made payable to Digital Equipment Corporation) to the published-by address. General subscription rates are $40.00 (non-U.S. $60) for four issues and $75.00 (non-U.S. $115) for eight issues. University and college professors and Ph.D. students in the electrical engineering and computer science fields receive complimentary subscriptions upon request. Digital's customers may qualify for gift subscriptions and are encouraged to contact their account representatives.

Single copies and back issues are available for $16.00 (non-U.S. $18) each and can be ordered by sending the requested issue's volume and number and a check to the published-by address. See the Further Readings section in the back of this issue for a complete listing. Recent issues are also available on the Internet at **http://www.digital.com/info/dtj.**

Digital employees may order subscriptions through Readers Choice at URL http://webrc.das.dec.com or by entering VTX PROFILE at the system prompt.

Inquiries, address changes, and complimentary subscription orders can be sent to the *Digital Technical Journal* at the published-by address or the electronic mail address, dtj@digital.com. Inquiries can also be made by calling the *Journal* office at 508-486-2538.

Comments on the content of any paper are welcomed and may be sent to the managing editor at the published-by or electronic mail address.

ISSN 0898-901X

Documentation Number EC-N7285-18

Book production was done by Quantic Communications, Inc.

Contents

Editor's Introduction

This issue presents papers on diverse computing topics—the Internet, modern Fortran language extensions for parallel computing, and performance measurement of AlphaServer 64-bit RISC systems—each representing an area of engineering strength for Digital. Also in the issue is a thought-provoking paper on the preservation of historical computers.

The opening paper on the Internet Protocol version 6 examines the status of today's Internet and looks toward its future. Digital is one of several companies participating in the working groups of the Internet Engineering Task Force on the transition to a new protocol. Dan Harrington, Jim Bound, Jack McCann, and Matt Thomas report what they have learned from designing an IPv6 prototype, and compare and contrast the new version with the existing protocol, IPv4. The most important difference between the versions—one that will relieve the strain on the Internet—is the increase in IPv6 of address size from 32 bits to 128 bits. The authors conclude with a look at future work in such areas as security and data link interfaces for ATM.

Our next paper—an unusual one not only for the issue but for this *Journal*—temporarily moves the discussion from computing's future to its past. Max Burnet and Bob Supnik argue that an understanding of computing's past is vital to its future. The authors present two computer preservation techniques: restoration and simulation. To exemplify issues in restoration, they review the status of a project to restore a large UNIBUS-based PDP-11 system. The section on simulation describes the types and purposes of simulators and presents a case study of SIM, a simulator implemented in C for the study of historical computer architectures.

In a paper on modern Fortran, Bill Celmaster demonstrates that today's Fortran is a viable mainstream language for parallel computing. Since its development more than 40 years ago, Fortran has been extended by language designers to meet the needs of users, particularly the needs of scientific/technical users who require mathematical expressivity and code optimization. Bill reviews key features of Fortran 90, recent efforts to standardize parallel extensions to Fortran, and shared-memory parallelism. He includes three case studies that illustrate the data parallel and single-program–multiple-data styles of programming.

Two papers describe testing methodologies that resulted in leadership system performance under the TPC-C benchmark for a cluster system and for a single-node system. The first paper presents the evaluation of an AlphaServer 8400 5/350 TruCluster configuration supporting the Oracle Parallel Server database. Judy Piantedosi, Archana Sathaye, and John Shakshober discuss the system tuning and the record-setting results of their work. The second paper, by Tareef Kawaf, John Shakshober, and Dave Stanley, looks at two optimization techniques—locking intrinsics and OM profile-based optimization—applied to a large database program running in the very large memory (VLM) environment on an AlphaServer 8400 system. The results of these optimizations are significant increases in throughput and database-cache hit ratios.

The development of AltaVista Forum is the subject of our final paper. Unlike other groupware products, AltaVista Forum uses the World Wide Web as an infrastructure to facilitate the rapid development of collaboration applications for NT and UNIX systems. Dah Ming Chiu and Dave Griffin explain this design decision and share their experiences with usability studies, an interpretive language (Tcl) for building the toolkit, and the inclusion of an indexing and search engine.

The next issue of the *Journal* will feature the new AlphaServer 4100 high-performance midrange server system, a new implementation of MEMORY CHANNEL, and large-database technologies in the VLM environment.

Jane C. Blake
Managing Editor

Foreword

Alan G. Nemeth
Corporate Consultant
UNIX Architecture and Technology

"The Internet is dying." I feel quite confident you will regularly see articles with this message in the industry and general press over the next few years. The message won't be as new as the authors of the articles might believe, and the work to remove the most frequently identified problems was begun years ago within the Internet Engineering Task Force (IETF). Internet Protocol version 6 (IPv6) is a large family of protocols that form the basis of the IETF response to a set of problems identified in the early 1990s and for which the need is accelerated by the explosion of Internet usage.

One of the major concerns about the current Internet is the limited amount of address space. The underlying address for IP endpoints is 32 bits wide, permitting a total of 4 billion distinct addresses. Although this number seems large (and it seemed truly gigantic in the early 1970s when the width was selected), it is currently a real, practical barrier to current deployment patterns. Large users of Internet addresses can no longer get the address space they need for assignments. Because the Internet has run as a decentralized organization over the years, there is no effective central administration to support competition for scarce resources such as address space. Instead, the response of the community is to provide resources sufficient to keep allocation as a low-overhead activity. So IPv6 defines an address space of 128 bits. This currently seems like a gigantic number!

But limited address space is hard to build into a persuasive case for change. End users are much more likely to be concerned about the local problem of getting just "one more address," rather than the problems of keeping the Internet as a whole alive and functioning. So the IPv6 design deliberately incorporates a set of functionality improvements that provide attractive end-user capabilities. IPv6 includes much easier schemes for assigning addresses, which will reduce the administrative burden for users and their network managers. IPv6 provides a better framework for encryption and an expectation that it will be widely available and used. And IPv6 provides some systematic mechanisms for describing requests for specific quality levels in the service offered by the transport provider. These capabilities will address some very real, practical problems that do afflict individual end users of the Internet.

However, there is no expectation that it is acceptable to switch the set of Internet users to IPv6 either simultaneously or even over an extended time period. IPv6 must interoperate with the current installed IPv4 protocols for an indefinite period. This implies services that translate between the different addresses (and address assignment approaches that ease mechanical derivation of IPv6 addresses from IPv4 addresses), dual protocol stacks to permit communication with both protocols depending on the capabilities of the participants in the conversation, and schemes to accommodate security mechanisms and quality of service requests.

The entirety of IPv6 represents a large implementation effort to be undertaken by many different organizations. The Internet represents the largest example I know of a distributed computation that has survived for 27 years. (I date from 1969 when the first ARPANET [Advanced Research Projects Agency Network] nodes were installed.) With a few notable exceptions, this computation has run continually, despite constant changes in hardware, software, implementers, and operators. It has survived explosive growth far beyond the designs of its originators. It has done so with a volunteer organization driving the development direction. The community spirit has been crucial to making this work. IPv6 is an example of that community at work; no one organization can implement it all, either at a product level or at a deployment level.

The IPv6 paper in this issue describes the technical design needed to build an IPv6 implementation for the core protocols under the Digital UNIX operating system. Digital has been one of the leading prototype builders of the design specifications as they have evolved in the industry debates. At the time the Internet Protocol Next Generation (IPng) Directorate officially adopted

key elements of the protocol, Digital's implementation was the only one running to demonstrate that the design was indeed feasible. But we don't believe that we can implement all the pieces of IPv6 as a single company. Therefore we choose to share the implementation experience through this paper to aid others in their efforts to deal with the implementation problems. We also don't claim completeness; the full suite of specifications for IPv6 is evolving, and the software to implement it is large. We fully expect that portions of our ultimate product offerings will be developed by others in the industry.

The long-term evolution of the Internet captured in the IPv6 implementation paper is but one example in this issue of the extent to which computing now has a history that gives us much insight into the future. Certainly the paper by Supnik and Burnet is an explicit trip through computing history. The re-creation, both physical and logical, of computing systems of the past can only help remind us that the artifacts we create have a longer life than we anticipate. As our programmers write new code, or our hardware designers produce new architectural approaches, or our storage designers push the boundaries on new media technologies, do they consider the imponderables of running these systems 25 or more years in the future? The view of archivists trying to preserve this history reminds us of the difficulty of preservation after the fact and of the amazing duration of design decisions.

The paper on the evolution of Fortran is yet another example of the rich history of computing. Here we see clearly the evolution of a key language to accommodate the changing patterns of system architectural designs and parallel program concepts. The computer industry frequently develops commercially important programs by evolution—the 100,000-line program that 10 years later has become 10 million lines of code in an assortment of languages and computing styles. Here the venerable Fortran (first introduced in 1954!) adds support for some of the latest approaches to fast system interconnect represented by MEMORY CHANNEL and the parallel architectures of clusters of SMP systems.

MEMORY CHANNEL reappears in the paper about TPC-C performance on TruCluster systems. This paper, one of a pair on the issues of tuning a commercially important benchmark, presents an attractive model for the benefits in performance that can be derived from a very fast interconnect and software structures to match. The performance levels achieved shatter world records on a benchmark that has had extensive attention and work.

The other paper on TPC-C performance with very large memory (VLM) illustrates the truth of an old design maxim, "If memory is getting cheaper, use more of it!" When Digital first built a 2-gigabyte (GB) memory board, it took more than a million dollars' worth of DRAM chips to populate the initial instance. However, memory prices have continued to drop sharply, and today over 40 percent of the AlphaServer 8400 systems ship with 2 GB or more of memory. The memory prices will continue to come down, and the insights offered in this paper will help in understanding where additional memory can provide real benefits to customer workloads.

The final paper in the collection is on the AltaVista Forum approach to collaboration among groups exploiting the Internet and WWW technologies and brings us back around to the initial thoughts in this foreword. The ubiquitous nature of the Internet permits and encourages tools such as this that utilize computer systems in new ways. This approach builds on the fabric that we emphasized in the IPv6 paper but sees the Internet as a tool and a component of a larger solution and shows how to exploit these capabilities to allow new ways of working. Using imagination and building on the work of others are characteristic of the approach taken by those who are catalysts in the industry. The paper demonstrates how easy it is to build a system that would have been a major project just five years ago. This ease of construction is a benefit of the programming techniques and infrastructure investments and a spur to keep doing more of it.

Internet Protocol Version 6 and the Digital UNIX Implementation Experience

Daniel T. Harrington
James P. Bound
John J. McCann
Matt Thomas

In the early 1990s, the Internet community recognized that the current TCP/IP architecture was not capable of sustaining the explosive growth of the Internet. In July 1994, the Internet Protocol next generation (IPng) directorate responded to the problem with the Internet Protocol version 6 (IPv6) as the replacement network layer protocol. Working groups of the Internet Engineering Task Force (IETF) then began to build specifications that would address the needs for an expanded Internet address space, an increase in router table size, and new technology features. As a contributor to these efforts, Digital has implemented IPv6 on the Digital UNIX platform. The primary goal of Digital's efforts has been to evaluate the technical feasibility of the proposed architecture and provide critical feedback to the standards development process in the IETF. The secondary goal has been to evaluate system design alternatives to gain the experience needed to allow Digital to incorporate this new architecture into existing products.

As one of its ongoing advanced development efforts in networking technology, Digital has built an Internet Protocol version 6 (IPv6) prototype for the Digital UNIX operating system. In this paper, we describe the design of the Digital UNIX IPv6 prototype and its history relevant to the Internet Protocol next generation (IPng) effort in the Internet Engineering Task Force (IETF). We also compare its relationship with the existing Transmission Control Protocol/Internet Protocol (TCP/IP) suite. We emphasize techniques and technologies that were developed to accommodate particular aspects of the IPv6 architecture and issues that required further discussion in the IETF. In particular, we discuss the modifications to the transport layer modules to use two distinct network layer protocols, along with the implications to the UNIX socket layer and applications. In addition, we describe the new IPv6 and Internet Control Message Protocol (ICMP) network layer modules, including their interactions with both the data link layer and the IPv4 protocol. We review the new Neighbor Discovery Protocol and its algorithms and give details of its implementation.

To accommodate the dynamic nature of future networks, IPv6 includes mechanisms to do both stateless and stateful address configuration, as well as router discovery; we explain the design of a user-mode process that implements these functions. The paper includes a discussion of enhancements to well-known IPv4 services, such as dynamic updates to the domain naming service (DNS), as well as general techniques to support the transition of existing applications. The paper concludes with an overview of what we have learned in this project and summarizes our current status and future work, including efforts in nonbroadcast multiple access (NBMA) data link technologies such as asynchonous transfer mode (ATM) and resource reservation protocols.

Internet Protocol Next Generation

In the early 1990s, the members of the Internet community realized that the address space and certain aspects of the current TCP/IP architecture were not capable of sustaining the explosive growth of the

Internet. Within the IETF, several efforts were undertaken to both study and improve the use of the 32-bit Internet Protocol (IPv4) addresses, as well as to identify and replace protocols and services that would limit growth. The 32-bit addressing architecture in the network layer was quickly determined to be the crux of the problem, with both hardware and human limits approaching fundamental boundaries.[1] IPv4 addresses are unevenly allocated in blocks that are often too large or too small; they are also difficult to change within any existing network.

When the IETF called for replacement proposals, Digital participated in this industry-wide effort by submitting white papers outlining issues and by developing and evaluating prototypes of the various proposals. Digital also participated in the IETF working groups and in the IPng directorate, which had the responsibility for making the ultimate decision. In July 1994, the IPng directorate selected the Internet Protocol version 6 (IPv6) as the replacement network layer protocol, and IETF working groups began to build specifications. "The Recommendation for the IP Next Generation Protocol" summarizes the candidates and explains the selection of this protocol.[2]

Digital UNIX Prototype

The current Digital UNIX IPv6 prototype project is Digital's most recent addition to an ongoing effort to develop and evaluate the competing IPng proposals. This began with the Simple Internet Protocol (SIP), which used eight octet addresses. SIP was later melded with another early proposal and became known as Simple Internet Protocol Plus (SIPP), the direct antecedent of IPv6.[3] The primary goal of Digital's efforts has been to evaluate the technical feasibility of the proposed architecture and provide feedback to the IETF working groups. This is critical to the standards development process in the IETF, which requires multiple independent and interoperable implementations of a specification before it may become an Internet standard. An additional goal has been to evaluate system design alternatives to gain the experience needed to allow Digital to incorporate this new architecture into existing products. Digital has made the prototype available to researchers within the company as a source code distribution and more recently has begun to supply binary kits for early adopters and evaluators in the Internet community. As the IPv6 protocol and architecture matures, we have begun to focus on how to best integrate the code into the Digital UNIX product.

IPv6 Overview

To understand the system-wide impact of IPv6, we review some of its new features and contrast them with the IPv4 model. IPv6 is both a completely new network layer protocol and a major revision of the Internet architecture. At both levels, it builds upon and incorporates experiences gained with IPv4.

Figure 1 shows the evolution of the packet format into the new IPv6 header. It retains some fields (version, source, and destination address), clarifies the role of others (for example, the Time To Live [TTL] field is renamed the Hop Limit), and introduces new ones (such as Flow ID) with as yet untapped potential. The next header field allows for modular construction of complex packets: different header types can be chained together to provide specialized functionality, including security and source routing. Finally, all headers are structured to allow 64-bit alignment, which should allow optimal processing both at source and destination systems, as well as in transit.[4]

The most striking departure from IPv4 is the address size: it has increased from 32 bits to 128 bits. The IPv6 addressing architecture is rich, with prefixes for multicast addresses and predefined scopes for both unicast and multicast addresses. One special type of unicast address is the link-local address, which permits communications with only those systems directly connected on the same link. This allows a standard bootstrapping mechanism, so that systems can learn about neighbors and services before a routable address is assigned to an interface. Various address assignment options have been defined, including hierarchical models based upon regional registries and service provider identifiers.[5,6] In each case, care has been taken to ensure proper route aggregation, which will help yield more efficient backbone router performance.

Multiple means of acquiring addresses have been defined for IPv6 addressing, with the goals of allowing flexibility through different administrative policies

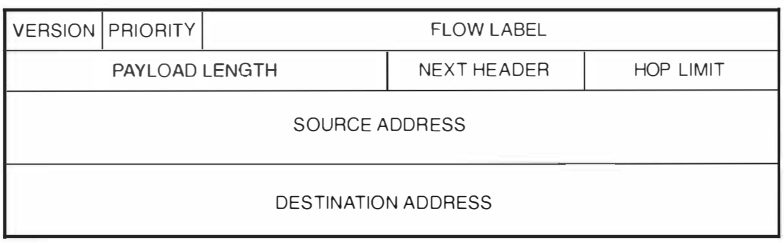


Figure 1
IPv6 Header

and, perhaps more important, of demanding that network address reassignment be supported throughout the architecture. The two new addressing services are Stateless Address Autoconfiguration and the stateful, transaction-based Dynamic Host Configuration Protocol version 6 (DHCPv6).[7,8] In the stateless model, address prefixes are learned by listening for router advertisement packets. Addresses are formed by combining the prefix with a link-specific token such as the 48-bit Ethernet hardware address. In the stateful procedure, hosts may request addresses, configuration information, and services from dedicated configuration servers, with routers potentially serving as relay stations during the initial phase. In both cases, the resulting addresses have associated lifetimes, and systems must be prepared to both learn new addresses and release expired addresses. Combined with the ability to register updated address information with DNS servers, these mechanisms provide a path toward network renumbering, a goal that has proved difficult to achieve in the IPv4 world.

Finally, the Internet Control Message Protocol version 6 (ICMPv6) was developed.[9] This specification aimed to merge the functions of two distinct IPv4 protocols for reporting errors and status, ICMP for unicast packet transmission and the Internet Group Message Protocol (IGMP) for multicast traffic.

The messages defined in this protocol are categorized as either error or informational, with a family of messages in the second group used to provide the Neighbor Discovery Protocol.[10] Neighbor discovery serves multiple purposes with the overall theme of providing a system with topological and environmental hints. For example, link-layer address resolution, router discovery, destination address redirection, and address autoconfiguration mechanisms are all specified using neighbor discovery packet types.

Although the network layer did experience the largest amount of change, Figure 2 shows that the effects of this work touch nearly all aspects of the Digital UNIX system. We point out examples of decisions made due to our fundamental design philosophy, which is based upon integration with the UNIX system framework, modular and extensible software, support for multiple operational policies, and a desire to take advantage of the Alpha platform without compromising portability.

In the following sections, we study these topics in depth, beginning with the network layer, then covering the transport layer modifications and the new neighbor discovery algorithms. After that, we discuss address autoconfiguration mechanisms and their effects upon the system. We conclude with services that will be affected by the transition from IPv4 to IPv6 such as the socket application programming interface (API) and DNS.

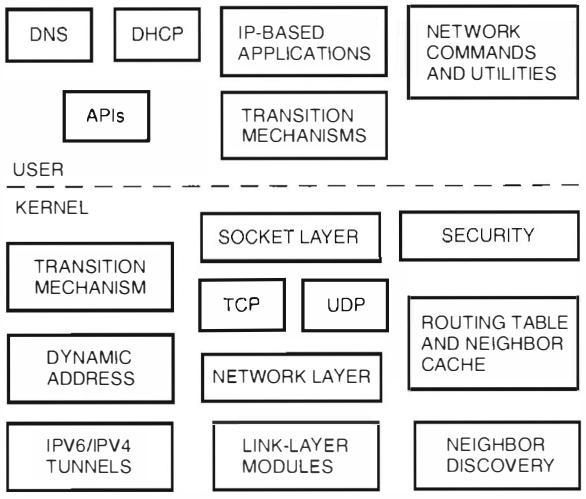


Figure 2
Base Platform Changes

Network Layer

In this section, we review the processing requirements of the IPv6 modules, including ICMPv6, extension header options, and fragmentation. An early design decision was made to base the networking subsystem on the Berkeley Standard Distribution (BSD) 4.4 model and code base, which allows great flexibility in dealing with multiple network layers.[11] This also has the advantage of providing support for variable-bit-length netmasks (also known as CIDR-style netmasks, from Classless Inter-Domain Routing), which are appropriate to both IPv4 and IPv6.[12] We have also tried to take maximum advantage of the 64-bit Alpha architecture when implementing IPv6, while making certain that this implementation would run on 32-bit CPUs as well. For example, the checksum routines operate on 32-bit quantities (allowing the carry to overflow into the upper 32 bits of a 64-bit register). The checksum routine is also designed to allow it to be issued to multiple Alpha execution units, which remains a topic for further investigation.

Adaptations to Existing IP and ICMP Routines

The IPv6 and ICMPv6 routines are completely independent of the corresponding IPv4 and ICMPv4 routines, and the processing styles have distinct differences. In IPv6, the incoming packet is treated as being read-only, while the BSD IPv4 code manipulates fields within the IPv4 header. We also avoid unnecessary use of the m_pullup routine (which consolidates chained memory buffers into a single large buffer) because this could cause the packet to be needlessly lost. Finally, instead of passing numerous arguments when calling from function to function, a common data structure is

used to store necessary data and pointers; for most function calls, it is only necessary to pass a pointer to this structure. This reduces the stack overhead and also yields modular and easily extensible subroutines.

IPv6 has a dedicated interrupt processing thread, and received IPv6 packets are placed onto their own interface input queue (ifqueue). When an IPv6 packet is taken off the ifqueue, basic validity tests are done; only after passing them is the packet tested to see if it is directed to a unicast or a multicast address.

If the packet is to a multicast address, the destination is compared to the enabled IPv6 multicasts for the interface over which the packet was received. If the destination matches, the packet is passed up to normal packet processing; otherwise, a copy of the packet is passed to the multicast forwarder.

Similarly, unicast packets are checked to see that the destination matches one of the system's addresses. In the special case of the packet being targeted to a link-local address, only the link-local address for the receiving interface is compared. If there is an exact match, the packet is processed normally; otherwise, it is passed to the unicast packet forwarding routine.

Header Processing

After a packet has been matched to a local address, the IPv6 headers must be processed, independently of whether the packet is multicast or unicast. This processing is done in a common routine that handles all types of IPv6 headers. A number of actions may result from the verification and analysis phase, including an ICMPv6 packet being sent back to the source, the packet being silently dropped, or being forwarded to another node due to a source route. If none of these possibilities occurs, the next IPv6 header in the packet is processed.

If the header is a known IPv6 header type, the appropriate routine is called. If not, this packet is probably destined for another protocol module such as TCP, the User Datagram Protocol (UDP), or ICMPv6. The header type is looked up in the list of active protocols and passed to the matching protocol input routine. If no entry is found, an ICMPv6 error may be sent back.

Header Options

Since the hop-by-hop and destination node headers have the same format, a common routine processes both types. As the routine processes each option, it validates the option. If this fails, it checks whether an ICMPv6 parameter problem error should be sent, whether the packet should be discarded, or the option ignored.

ICMPv6 Processing and Checksums

Upon receipt of an ICMPv6 packet from a node in the network reporting an error or other information, it is first validated for correct packet format and checksum. The packet is then further processed based upon its ICMPv6 type value. If it has an ICMPv6 error type (i.e., type value less than 128), the appropriate notifications are sent to the affected protocol. Neighbor discovery packets, which are all informational, have a number of additional consistency checks, and the packet is dropped if it fails them. After the ICMPv6 packet has been processed, it is also sent to any ICMPv6 raw sockets that have requested reception of that type. The exception to this rule is an ICMPv6 echo request packet, which is not copied to the raw sockets.

When an ICMPv6 echo request is received and validated, the ICMPv6 echo response packet is prepared. In the typical case, it is identical to the echo request except for the ICMPv6 type and checksum value. The exception would be an echo request sent to a multicast address, in which case a source address must also be selected. Rather than computing the checksum on the new packet, the received checksum is simply adjusted down by 1, since the sole difference between the two packets is the value of the ICMPv6 type fields, and ICMPv6 echo request and echo response types differ by 1.

IPv6 requires all nodes to support multicasting, specifically level 2 as defined in "Host Extensions for IP Multicasting."[13] Although this was written for IPv4, the same general algorithms are used for IPv6. One notable exception to this is that the multicast addresses used for neighbor solicitions and the predefined link-local multicasts such as all-nodes and all-routers do not require periodic status reports.

Path Maximum Transmission Unit Discovery

One of the significant differences between IPv4 and IPv6 concerns fragmentation. In IPv6, fragmentation may be done only by the node from which a packet originates. Forwarders, which may be routers or hosts acting upon source routing headers, are not permitted to fragment packets. The burden is on the originating node to send packets that are small enough to fit through all the links along the paths to their destinations, where each link type may have a different maximum transmission unit (MTU). To ease this burden, IPv6 defines a minimum link MTU of 576 bytes. A node may use this as the upper limit on packet size and be assured that its packets are sufficiently small to reach their destinations.

The minimum MTU of all the links in a path between two nodes is referred to as the path MTU.[14] In many cases, the path MTU will exceed 576 bytes, and it is desirable to send the largest possible packets. IPv6 provides a mechanism by which a node may discover a path's MTU.[15] When a forwarder cannot forward a packet because the packet is too large for the next hop's link MTU, it sends an ICMPv6 Packet Too Big (PTB) message back to the source of the packet. The PTB

message contains the MTU of the constricting link. The source node adjusts its packet size to fit through this link.

Path MTU information is kept on a per-destination basis and is stored in the routing table entry for a given destination. Packets sent on that route will be sized according to the path MTU value. When a PTB message is received, the appropriate route is updated to contain the new path MTU value as reported in the PTB message, and a timer is started. When the timer expires, the path MTU value is increased to the (known) MTU of the first hop link. This allows the node to detect increases in the path MTU.

Switches are provided to disable path MTU discovery system-wide, on a per-destination basis and on a per-socket basis. When path MTU discovery is disabled, packets are limited to 576 bytes.

Fragmentation

A packet that is larger than the MTU of the path on which it is to be sent must be fragmented. Unlike IPv4, the IPv6 header contains no fields to carry fragmentation information. Instead, this information is carried in a specialized extension header, called the fragment header. As shown in Figure 3, the fields in the fragment header include an offset, in eight octet units, and an identifier common to all fragments of the original packet. The M (managed) flag is used to indicate intermediate fragments; the terminal fragment has the bit cleared. Note that the amount of data in a fragment packet is derived from the total packet length.

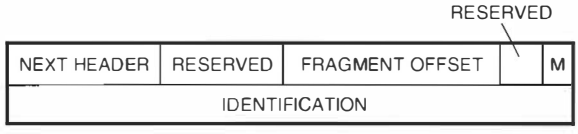


Figure 3
Fragment Header

The first step in the fragmentation process is to identify the fragmentable and unfragmentable parts of the original packet (see Figure 4). The unfragmentable part of the packet consists of the IPv6 header and any extension headers that must be processed by each node traversed by the packet (e.g., hop-by-hop header, routing header). The fragment header is appended to the unfragmentable part. The rest of the packet is divided into fragments, and each fragment is appended to a copy of the unfragmentable part plus fragment header.

When the fragment header is appended to the unfragmentable part, two fields in the unfragmentable part must be updated. First, the payload length field in the IPv6 header must be updated to reflect the length of the fragment packet. Second, the next header field in the last header of the unfragmentable part must be changed to indicate that a fragment header follows.

A copy of the unfragmentable part is created for each fragment packet. As an optimization, Digital UNIX allows portions of a packet to be shared among copies of the packet, to avoid an actual data copy. As with IPv4, care must be taken to ensure that fields being updated are not contained in shared buffers. This is typically accomplished by copying the portions that must be updated into a private memory buffer (mbuf). Unlike IPv4, the unfragmentable part may not fit in a single mbuf, and the IPv6 fragmentation code must be capable of handling this case.

To reduce the possibility of fragment loss at the source node, all the fragment packets are built before any is passed to the data link for transmission.

A question that arises here is how big should the fragment packets be? Should they be sized according to the path MTU, or should they be limited to 576 bytes? The former yields the desirable larger

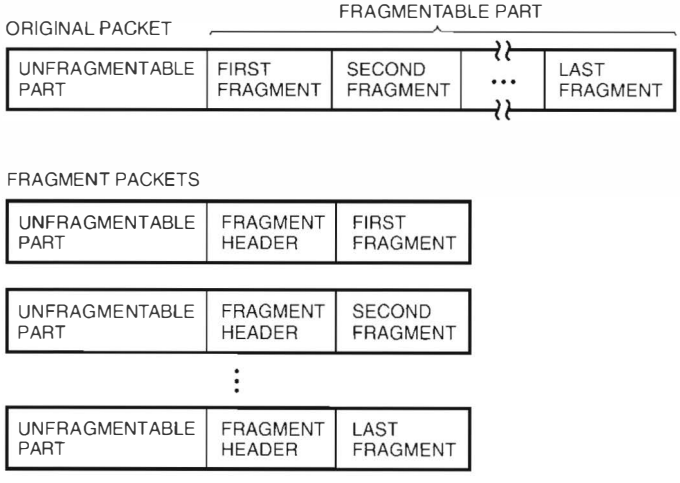


Figure 4
Fragmentation

packets, while the latter avoids undesirable fragment loss (due to the fragment packet being too big). The Digital UNIX IPv6 prototype supports either choice on a system-wide, per-destination, or per-socket basis. This is an example of separation of mechanism from policy, a basic guideline being used across this project.

Reassembly

The reassembly process reconstructs the original packet from fragment packets. Fragments belonging to the same packet are identified by a combination of source IP address, next header type (first header of the fragmentable part) and fragment identifier. Individual fragments are queued within the network layer until the original packet can be completely reassembled, at which point it is passed to the appropriate protocol module.

When all fragments have arrived, the original packet can be reassembled. A single copy of the unfragmentable part is kept, and the data from each fragment packet is appended. The payload length field of the IPv6 header is updated to reflect the length of the reassembled packet, and the next header field of the last header of the unfragmentable part is restored to reflect the first header in the fragmentable part.

As with the fragmentation code, care must be taken so that fields being updated are not in buffers shared with other copies of the packet.

When the first fragment of a packet arrives, a timer is started. If the timer expires before that packet is complete, the fragments are discarded. If the offset zero fragment has been received, an ICMPv6 error message is sent.

Forwarding and Routing

If a received packet does not match one of the system's addresses and the system is not acting as a router, the packet is silently dropped. Otherwise, an attempt is made to forward the packet. The first step in forwarding is to do a lookup in the routing table; the type of lookup depends on whether the packet contains a nonzero flow label. If it does, the lookup is based on both the source address and the flow label; otherwise the destination address is used. If the lookup succeeds and the length of the packet fits within the MTU of the resultant route and interface, the packet is transmitted to the next hop as indicated by the route. Otherwise an appropriate ICMPv6 error is sent back to the originating node.

Tunnels

Tunneling is a mechanism that allows packets of one network type to be encapsulated and forwarded within a network layer packet of the same or a different type. IPv6 packets can be tunneled over either IPv4 or IPv6 networks, as may IPv4 packets.[16,17] The tunneling routine takes as input a packet, prepends the appropriate IP header for the network over which the packet will be tunneled, and transmits the resultant packet over that network. Tunnels are unidirectional; there need not be a corresponding tunnel in the reverse direction.

Rather than having multiple tunnel interfaces (one for each possible combination of protocol Y over protocol X), the Digital UNIX implementation uses a single tunnel interface. This method was the suggestion of Keith Sklower of the University of California at Berkeley.[18] When the interface is initialized, only automatic tunneling of IPv6 over IPv4 is enabled.[19] To configure a static tunnel, where fixed end points are used, a static route is added to the routing tables with the proper destination and gateway (tunnel end point) addresses.

When a packet is presented to the tunnel interface, it looks up the route entry of the destination address. The route contents tells the tunneling routine how the packet is to be encapsulated and forwarded. The route's gateway address indicates what underlying network to use, and the route's destination address indicates what type of packet is being tunneled.

When a tunneled packet is received, the initial header is stripped and the resulting packet is placed on the appropriate IPv6 or IPv4 ifqueue.

Transports

One of the strengths of the IPng effort was the commitment to preserve the well-understood transports, TCP and UDP, upon which a wealth of applications have been built.

The IPv6 specification calls for three particular requirements of upper-layer protocols:

1. The pseudoheader checksum must accommodate larger addresses.
2. The maximum packet lifetime is no longer computed.
3. The larger IPv6 header(s) must be taken into account when computing the maximum payload size (e.g., TCP's maximum segment size [MSS]).[4]

In addition to these mandated modifications, we had to make a fundamental design choice. With two different network layer protocols in the system, each using a different size address, our design choice could have been to use two independent transport modules, one for each network layer. Figures 5 and 6 show the independent versus the integrated transport design options.

Although the independent model offers an element of design simplicity, it wastes memory by duplicating each transport layer function. In the Digital UNIX implementation, these modules are implemented in the kernel, and duplication would be expensive. Also, the design and use of a single programming interface to access both sets of services would be complicated.

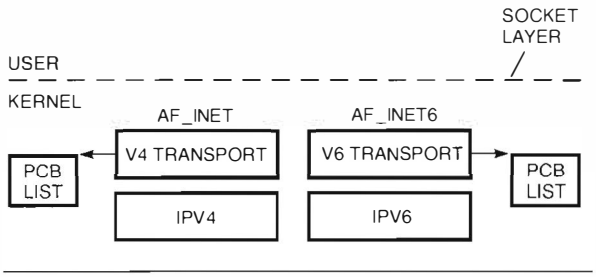


Figure 5
Independent Transport Implementation

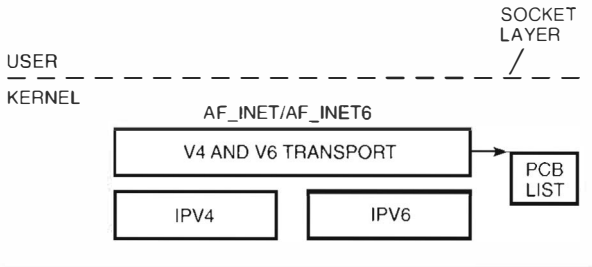


Figure 6
Integrated Transport Implementation

The ability to maintain, let alone extend, the code base would also suffer. Fortunately, due to the fact that IPv4 addresses are a well-defined subset of the entire IPv6 address space, it is relatively straightforward to implement the transports so that a single set of modules can be used over both network layers.[20] To accomplish this, we increased the storage space allocated for addresses and separated those functions that are dependent upon a particular network layer. We discuss each of these issues in this section.

Storing Large Addresses

Two specific data structures must be modified to accommodate addresses larger than the 32-bit IPv4 type. The first of these is the sockaddr struct, which is used when dealing with the BSD socket layer and passed along to user applications. The second is the Internet Protocol Control Block (PCB) data structure, the in_pcb. In this section, we review the modifications to each structure.

A program that uses a transport does so by means of the BSD sockets interface and passes addressing information in a sockaddr structure. For IPv6, this is a sockaddr_in6. Internally, the structure is defined so that 64-bit alignment is preserved; however, it has the following public definition:

```
struct sockaddr_in6 {
        u_char  sin6_len;
        u_char  sin6_family;
        u_short sin6_port;
        u_int   sin6_flowlabel;
        struct  in6_addr sin6_addr;
};
```

Although the concept of a sockaddr is generic in the BSD architecture, the flow label and in6_addr members of this structure are unique to IPv6 and would be used only in the AF_INET6 address family. The details of this are specified in Reference 21.

The in_pcb data structure is created for each socket using TCP, UDP, or other clients of the network layer. In addition to storing the source and destination addresses, various other pieces of information required for proper communication are stored here, including the port numbers, options and flags, a pointer to the socket receiving the data, a header template, and a pointer to the routing entry for the given destination. For IPv6, this basic model has been retained, and additional information is stored. This information includes local and remote flow labels and indicators of which address family the application is using and which network layer the transport communication is using. Finally, a partial checksum of the transport pseudo-header is stored here as well; its use is described in the following section.

In addition to the explicit storage of the network layer and address family, the fundamental technique that facilitates the use of a common transport is the storage of IPv4 addresses in an IPv6 format. This is known as an IPv4-mapped address and is described in "IP Version 6 Addressing Architecture."[20] This address format is explicitly reserved to store addresses of systems that are capable of using only the IPv4 protocol, and thus is an appropriate form of storage in the PCB for communications that will be sent using the IPv4 protocol, as opposed to IPv4-compatible addresses, which are sent using IPv6 packets. These mapped addresses are of the following form:

```
0000:0000:0000:0000:0000:FFFF:204.123.2.75
```

These addresses are manipulated within the IPv4 TCP and UDP protocols by means of macros that allow the IPv4 addresses to be inserted, extracted, or compared while in an IPv6 address structure (in6_addr). As an example, the code fragment in Figure 7 shows an address being extracted for use in evaluating a configurable IPv4 socket option.

Special Transport/Network Layer Interactions

Within the integrated transport layers, the transport protocol is treated independently of the particular network layer being used, and network-layer-specific functions are used to interface to either IPv4 or IPv6.

There are two particular instances in which the transport layer has interactions with the IPv6 network layer over and above the exchange of data packets for input or output. These are the notification and update of path MTU, which is required in IPv6, and the potential to refresh the neighbor discovery cache based on forward progress; i.e., if the transport knows that data is reaching its destination, it can validate the

```
/*
 * Test address for IPv4 characteristic
 */
if (inp->inp_netlayer == AF_INET) {
        struct in_addr tmp;

        tmp.s_addr = IN6_EXTRACT_V4ADDR(inp->inp_faddr);
        if (!in_localaddr(tmp))
            :
            :
}
```

Figure 7
Code Fragment of a IPv4-mapped Address

current network layer path. We investigate each of these issues in turn.

Path MTU discovery, as previously described, is triggered by ICMP messages processed in the network layer, with learned information stored in the routing table. In the course of processing a PTB message, the transport layer is notified through its control input (ctlinput) path. This is required because the reception of such an ICMP message indicates that the packet in transit has been discarded, thus the protocol may need to take appropriate action. In the case of TCP, it is necessary to recompute the maximum segment size and retransmit the affected data. Although this is not required for UDP, which is a pure datagram service, this knowledge can be made available to the corresponding socket owner.

The other interaction between an upper layer and the IP layer occurs when the upper layer, specifically the TCP transport, wishes to indicate that communications with a destination host has made forward progress, for the purpose of resetting the timer in the neighbor discovery cache. This positive feedback mechanism is described in the neighbor unreachability detection portion of the "Neighbor Discovery for IP Version 6" specification and prevents unnecessary probing of the current path.[10] When acknowledgments to previously sent data have been received, the TCP updates the routing table entry by means of an RTM_CONFIRM message. This call is handled by the neighbor discovery module, which resets the internal neighbor discovery state for appropriate route entries, as described later in this section.

Source Address Selection

Many applications do not specify a particular source address to use when initiating communications with a remote host but instead use the symbol INADDR_ANY, which allows the transport to select a source address (and corresponding interface) to use. For most IPv4 systems, this is a trivial exercise if only a single address on a single interface exists. However, multiple addresses per interface will be a common occurrence on IPv6 hosts, and so the process of choosing a source address to use becomes important. The source address selection is typically done when the PCB is bound to the application's socket, but this function is also available to users of raw sockets and to other network-layer users such as ICMPv6.

The source address selection function takes as arguments a destination address and an optional interface pointer. The latter is used when known, but in the case of initiating a transport connection it is null. The destination address is first checked against the list of current prefixes that have been advertised on the host's links, which would indicate which interface to use. (It also indicates that the destination is a potential neighbor, but this knowledge is not used at this point.) Next, the address is tested for multicast versus unicast, and then the scope (link-local, site-local, organization-local, and global) is evaluated. Finally, a local address of equivalent (or greater) scope than the destination with the longest prefix match is returned. The scope must be taken into consideration to ensure that the destination system will be able to successfully respond to the communication. The longest prefix match is an attempt to ensure a reasonable routing path between the two systems, which could involve a complex mix of service providers.

Checksum Optimization

Although the IPv6 header itself does not contain a checksum, the TCP, UDP, and ICMPv6 protocols do require a 16-bit one's complement checksum calculation to validate the integrity of transmitted and received data. Performing this checksum can be an expensive operation. While this prototype was being developed, some alternative mechanisms were investigated, such as varying the size of the sum variables and operand units and tight versus expanded loops. The selected algorithm offered the best performance for the Alpha processors, while retaining the ability to be used on 32-bit processors.

At the point where the PCB is established for transport communications, a partial checksum is calculated

for the IPv6 pseudoheader, which consists of the source and destination addresses, the packet payload length, and the next header value. This partial checksum, with the exception of the payload length (which varies per packet), is then stored in the PCB, to be passed along with the pointer to user data within the memory buffer to the checksum function. The initial checksum calculations are done using 32-bit values in 64-bit registers, and later are collapsed to the final 16-bit sum. This is coded as one large C statement, adding the various pseudoheader components in piecemeal fashion. This allows the compiler to schedule the instructions for optimal performance. The final packet checksum can then be computed by adding the partial checksums for the pseudoheader with the checksum values for the data itself, plus the payload length.

Neighbor Discovery Overview

The Neighbor Discovery specification describes several important aspects of an IPv6 node's behavior in relation to other IPv6 nodes connected to a common link. IPv6 nodes on the same link use neighbor discovery to discover each other's presence, to determine each other's link-layer addresses, to find routers, and to maintain reachability information about the paths to active neighbors and remote destinations.[19] These functions are performed with several ICMPv6 messages and options, as shown in Figure 8. The same messages are also used for address autoconfiguration and duplicate address detection, as described in "IPv6 Stateless Address Autoconfiguration."[7]

Interface Initialization

When an interface is initialized for use with IPv6, a link-local address may be created without any external configuration, allowing the system to begin transmitting and receiving packets to nodes sharing a common link. This is performed by appending an interface token to the predefined link-local address prefix, FE80::. The length and content of the interface token is link specific. For example, the address token for an Ethernet interface is the interface's built-in 48-bit IEEE 802 address, resulting in a link-local address such as FE80::0800:2BBE:6260.[22]

Duplicate Address Detection

Before a unicast address can be assigned to an interface, a process known as duplicate address detection (DAD) must be performed.[7] This process provides a degree of assurance that two nodes do not use the same address on the same link. The basic mechanism involves sending an ICMPv6 neighbor solicitation (NS), where the target address is the address being tested. If another node is using the address, it will respond with a neighbor advertisement (NA). Multicast is used for both NS and NA packets, so DAD can be performed for all unicast addresses, including the first address assigned to the interface.

While an address is undergoing DAD, it is said to be a tentative address. It is not used to receive packets, nor is it used in outbound packets. The LA6_TENTATIVE flag in the in6_localaddr structure identifies addresses in this state. When a duplicate address is detected, the error is logged and the LA6_DADFAILED flag is set in the in6_localaddr structure. If a duplicate address is not detected, the LA6_TENTATIVE flag is cleared, making the address available for use on the interface.

Address Resolution

In IPv6, the function of mapping unicast IPv6 addresses into link-layer addresses is performed using ICMPv6 messages. This is a departure from IPv4, which relied on separate protocols (e.g., Address Resolution Protocol [ARP]) to perform this function.[23] IPv6 unicast address resolution is defined in a generic manner and can be run over any link layer that provides a link-layer multicast service (this includes point-to-point and broadcast links, special cases of multicast). This protocol may also be used for nonmulticast-capable media (e.g., nonbroadcast multiple access [NBMA] media such as ATM), provided that the link layer provides the necessary services. The function of mapping multicast IPv6 addresses into link-layer addresses is specific to each link type.

The unicast address resolution function uses two ICMPv6 message types: the NS and the NA. When a node needs to resolve the unicast IPv6 address of a neighbor to a link-layer address, it builds an NS containing the IPv6 address to be resolved (target) and sends it to the solicited-node multicast address corresponding to the target address. As an optimization, the node includes its own link-layer address as an option in the NS message.

When an address is assigned to an interface, a node is required to join the solicited-node multicast group corresponding to that address, so a node will receive NSs sent to its solicited-node multicast address. Upon receipt of an NS, the target node builds an NA containing its link-layer address. The NA also contains the IPv6 target address, so that the soliciting node can associate the response with the corresponding request. The NA is then sent back to the soliciting node.

Upon receipt of an NA, the soliciting node can map the target IPv6 address to the corresponding link-layer address and send any packets that were queued awaiting address resolution. Once a node has resolved an IPv6 address, the link-layer address is cached until it must be replaced or deleted. A different link-layer address may be received in a subsequent NA packet, with the O-bit (override flag) set to indicate a new value. If the neighbor unreachability detection algorithm (explained in the next section) detects that the cached

ROUTER SOLICITATION

TYPE	CODE	CHECKSUM
RESERVED		
OPTIONS ...		

ROUTER ADVERTISEMENT

TYPE	CODE			CHECKSUM
CURRENT HOP LIMIT	M	O	RESERVED	ROUTER LIFETIME
REACHABLE TIME				
RETRANSMIT TIMER				
OPTIONS ...				

NEIGHBOR SOLICITATION

TYPE	CODE	CHECKSUM
RESERVED		
TARGET ADDRESS		
OPTIONS ...		

NEIGHBOR ADVERTISEMENT

TYPE	CODE	CHECKSUM
R S O	RESERVED	
TARGET ADDRESS		
OPTIONS ...		

REDIRECT

TYPE	CODE	CHECKSUM
RESERVED		
TARGET ADDRESS		
DESTINATION ADDRESS		
OPTIONS ...		

SOURCE/TARGET LINK-LAYER ADDRESS OPTION

TYPE	LENGTH	LINK-LAYER ADDRESS ...

PREFIX INFORMATION OPTION

TYPE	LENGTH	PREFIX LENGTH	L	A	RESERVED1
VALID LIFETIME					
PREFERRED LIFETIME					
RESERVED2					
PREFIX					

REDIRECTED HEADER OPTION

TYPE	LENGTH	RESERVED
RESERVED		
IP HEADER AND DATA		

MTU OPTION

TYPE	LENGTH	RESERVED
MTU		

Figure 8
Neighbor Discovery Packets

value is not reachable, the mapping will be deleted.

The address resolution process has several implications for the implementation. Outbound packets must be queued pending link-layer address resolution, and link-layer addresses must be stored somewhere. The "Neighbor Discovery for IP Version 6" specification describes a conceptual neighbor cache to hold this information.[10] The Digital UNIX IPv6 prototype uses several data structures to implement the neighbor cache. An nd6_llinfo structure keeps track of each entry in the neighbor cache. This structure contains the queue header for packets awaiting link-layer-address resolution. The link-layer address is stored in the routing table, in a host route entry for the destination IPv6 address. The RTF_LLINFO flag in the route entry indicates the presence of link-layer information. Each nd6_llinfo structure contains a pointer to the corresponding routing table entry, and the routing table entry points back to the nd6_llinfo structure.

The use of routing table entries to hold the link-layer-address information is an optimization. A routing table entry is associated with the majority of packets transmitted for reasons other than address resolution. Storing the link-layer address in the routing table entry avoids the overhead of a separate link-layer-address table. This approach is modeled after the BSD 4.4 system's ARP implementation.

Neighbor Unreachability Detection

Neighbor unreachability detection (NUD) has its roots in the dead gateway detection in IPv4 but has been generalized in IPv6 to include all neighboring nodes (not just gateways).[24] Unlike IPv4, the mechanisms supporting NUD are an integral part of IPv6. IPv6 nodes monitor the reachability of neighboring nodes to which packets are being sent. An IPv6 node relies on reachability confirmations to determine the reachability state of a neighbor. In the absence of any reachability indications, an IPv6 node will periodically use an NS to actively probe the reachability of a neighbor. An NA sent in response to an NS provides reachability confirmation. The S (solicited) flag in the NA is provided specifically for this purpose. If neither method succeeds within a given period of time, a neighbor is considered unreachable. Figure 9 shows the neighbor unreachability states.

A reachability confirmation may take several different forms. Any packet received from a neighbor can be viewed as a reachability confirmation, provided that the packet would only have been sent by the neighbor in response to a packet sent from the local node. A TCP acknowledgment is one example: receipt of a TCP ACK indicates that a packet sent to the neighbor did in fact reach it. Another example is an ICMPv6 redirect message. Receipt of a redirect message indicates that the neighboring router received a packet from the local node.

In the Digital UNIX IPv6 prototype, the nd6_llinfo structure holds NUD state and retransmit count information. A field in the routing table entry is used for NUD timers. The RTF_LLVALID flag in the route entry is used to indicate that the neighbor is reachable. A new routing message type (RTM_CONFIRM) was defined to pass reachability confirmations to the neighbor cache. This mechanism is used by TCP upon receipt of new acknowledgments.

Autoconfiguration

One of the goals of IPv6 is to work properly in a dynamic network environment without the need for manual intervention on each system attached to the

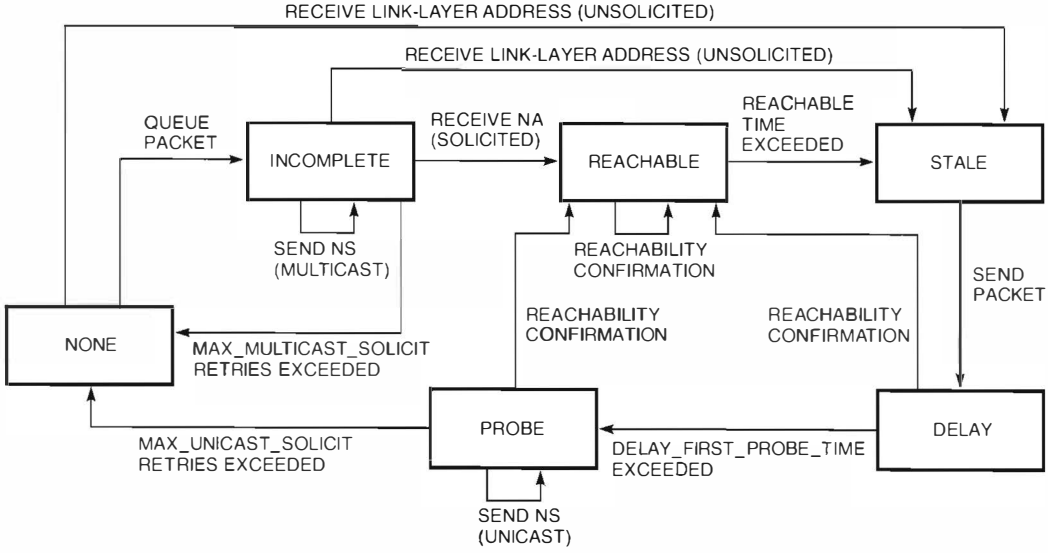


Figure 9
Neighbor Unreachability States

network. The solution is to allow important pieces of information to be learned and the system to autoconfigure itself using this data. IPv6 autoconfiguration encompasses the following items:

- Router discovery
- On-link prefix discovery
- Interface attribute configuration
- Stateless address configuration
- Stateful address configuration

The mechanism for delivering this information to the hosts is the router advertisement (RA) packet of the Neighbor Discovery Protocol. In the following sections, we describe the methods we developed to process these packets and update the system.

Host Autoconfiguration Daemon

To process these RAs, we designed a host daemon called nd6hostd, which resides in the application space of the Digital UNIX operating system. We determined that a user-mode daemon was the most efficient way to implement IPv6 autoconfiguration for the following reasons:

- A user-mode daemon would avoid kernel bloat.
- Maintenance and extensibility would be easier.
- The function is not performance critical.

The autoconfiguration processing is implemented as a single executable image, as a cohesive set of tightly coupled modules. The daemon currently is designed as a single-threaded application that uses a dispatch mechanism to call each specialized function module in turn. We will examine the idea of having this daemon run as a multithreaded application in the future.

The nd6hostd daemon communicates with the network subsystem in the kernel through multiple techniques. Figure 10 shows the autoconfiguration processing modules. The raw socket interface is used to receive RAs, and I/O control messages (ioctls) are used to manipulate kernel data structures. Also, the routing table is updated as necessary, by means of a raw socket interface to the PF_ROUTE protocol family.

We designed the IPv6 raw socket's interface with the ability to pass only specific ICMPv6 messages to a user and to filter extraneous packets or protocols. The nd6hostd daemon sets a socket option to receive only neighbor discovery RAs. It then executes a dispatch routine that polls the raw socket, awaiting packets. When data is available on the socket, the daemon determines the characteristics of the message, creates a data structure to contain it, and calls the necessary functions to perform autoconfiguration. The dispatch module, in addition to polling socket descriptors, supports necessary timer management functions such as creation, deletion, and expiration. Figure 11 shows the application daemon design center.

Kernel Interface Data Structures

In many ways, the data link interface is the focus of IPv6 autoconfiguration support. The kernel data structures for IPv4 interfaces are not sufficient to implement the necessary IPv6 functions. We designed and implemented new interface data structures that encapsulated the existing IPv4 structures. This allowed us to avoid a recompilation of the existing data link drivers on the Digital UNIX operating system. In the future, we will attempt a design in which the interface structures for IPv4 and IPv6 are completely integrated.

As shown in Figure 12, we designed an in6_ifnet structure to support each data link type (e.g., Ethernet, PPP, loopback) and used the existing ifnet structures to point to those link interfaces. The in6_ifnet has its own in6_ifaddr structure for each IPv6 address configured in the data structure in6_localaddr. We also defined the in6_router structure to support each router available for the implementation. The in6_router structure specifies the interface of the router, neighbor cache route, and the IPv6 address of the router.

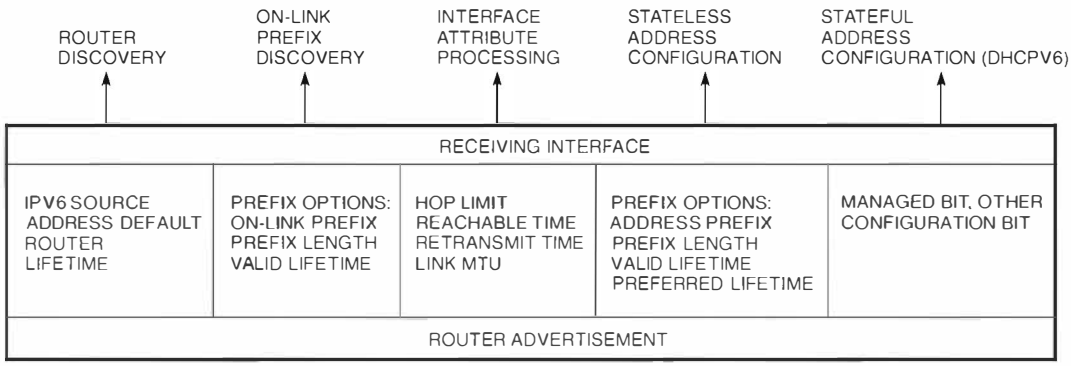


Figure 10
Autoconfiguration Processing Modules

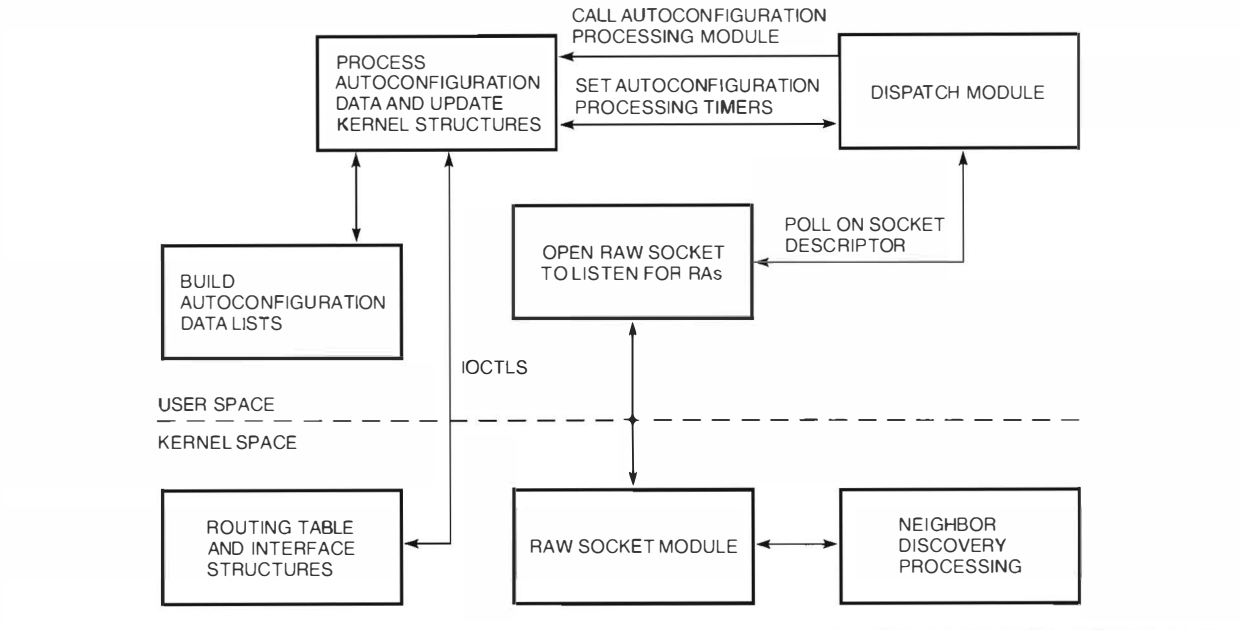


Figure 11
Application Daemon Design Center

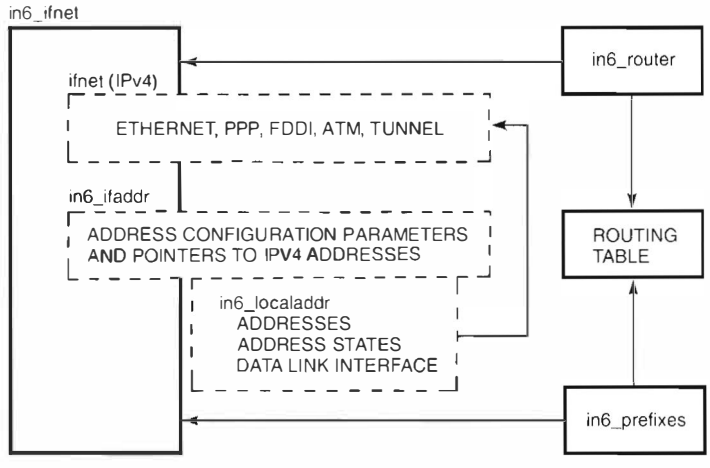


Figure 12
Autoconfiguration Interface Structures and Relationships

Interface Attribute Autoconfiguration

To autoconfigure the interfaces for IPv6, we created new ioctl functions to create, delete, update, and access the interfaces. In addition to their use by the nd6hostd daemon, these ioctls may be used by any future modules that need to access or manipulate the interfaces. This might include specialized configuration utilities, Simple Network Management Protocol (SNMP) management functions, security tools, or other services.

The interface module to update and maintain interface structures for nd6hostd serves two purposes: to update data link attributes provided by the RA, and to maintain the data structures as a set of linked lists for router discovery, on-link prefixes, and address configuration. Figure 13 shows the interface attribute updates.

Router Discovery

An RA packet has mandatory and optional parts. Before a default router is added to the routing table, the following interface attributes must be determined:

1. Receiving interface
2. Current hop limit
3. Reachable and retransmit times for use in NUD

The link-local address from the source link-layer option of the RA is then added to the routing table,

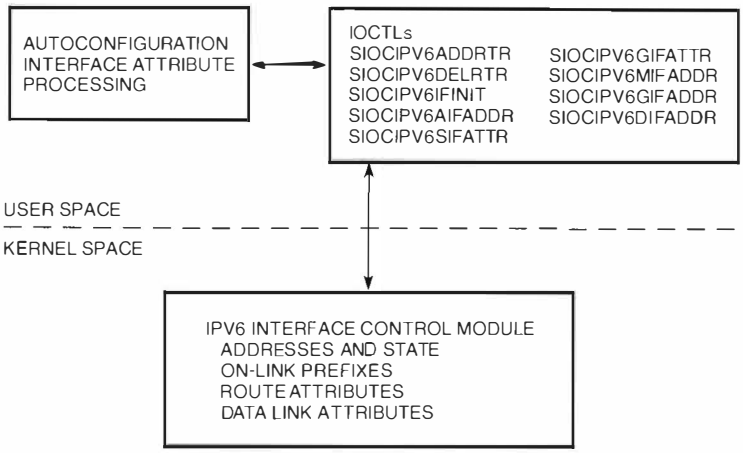


Figure 13
Interface Attribute Updates

and the kernel data structures for router information are updated. The router lifetime field in the RA defines how long this router may be used as a default router.

The nd6hostd daemon first updates the interface attributes. A timer is set using the appropriate routine from the dispatch module. When the timer expires, the delete default router routine is called, and the router is deleted from the routing table. The daemon must also be able to delete the router if it receives an RA with a zero lifetime value, which can occur when a node is acting as a router but is reset to be a host.

On-link Prefixes

An on-link prefix in IPv6 defines a subnet and is typically configured on a router for a specific link by the network administrator. The router then advertises this prefix to all nodes connected to that link as a prefix option, appended to an RA. A prefix option defines a single prefix only, but an RA may contain more than one such option. As shown in Figure 8, the prefix option provides the following information:

- Prefix length
- Link- or L-bit, which is set if the prefix is directly readable on link (i.e., a neighbor)
- Autonomous- or A-bit, which is set if the prefix can be used for stateless address configuration
- The length of time the prefix is valid

The daemon adds the prefix to the routing table. Then a timer routine is called from the dispatch module and is set for the time the prefix is valid. When the dispatch routine calls the delete on-link prefix module, the prefix is deleted from the routing table. A prefix can also be deleted when a new RA presents the prefix with a lifetime of zero. In that case, the on-link prefix module will stop the timer routine and delete the prefix from the routing table.

Address Configuration

Address configuration is one of the new paradigms that must be supported in IPv6. Two configuration methods, stateless and stateful, are provided to autoconfigure addresses for a host. The M-bit flag in an RA message determines which method to use and informs a host. In addition, the other-bit (O-bit) flag is provided to configure other network parameters required for the host's operation on the network when the stateful configuration is used.

Address autoconfiguration in IPv6 supports the ability to dynamically renumber a link or a complete network through the use of lifetimes specified in the RA message. The valid lifetime is the time the address has before expiration. When the timer expires, all connections using that address are dropped by the implementation, and no new connections are permitted. The preferred lifetime is provided to inform an implementation that an address is about to expire; it typically is set to a lower value than the valid lifetime. When this timer expires, the address is said to enter the deprecated state, at which point an implementation is permitted (as a configuration option) to prevent new communications using this address as a source or destination. This model is designed to provide network administrators with control over the use of network addresses without manual intervention of each host on the network. The stateless model is intended for users who do not need tight control over address configuration; stateful mechanisms will be used where the administrators want to delegate addresses based on a client/server method. Figure 14 shows the address autoconfiguration diagram.

When the daemon receives an RA, and the A-bit is set, the daemon can use the prefixes provided to perform stateless address configuration. The daemon uses the on-link prefix(es) provided in the RA to configure addresses for an interface. Addresses are created,

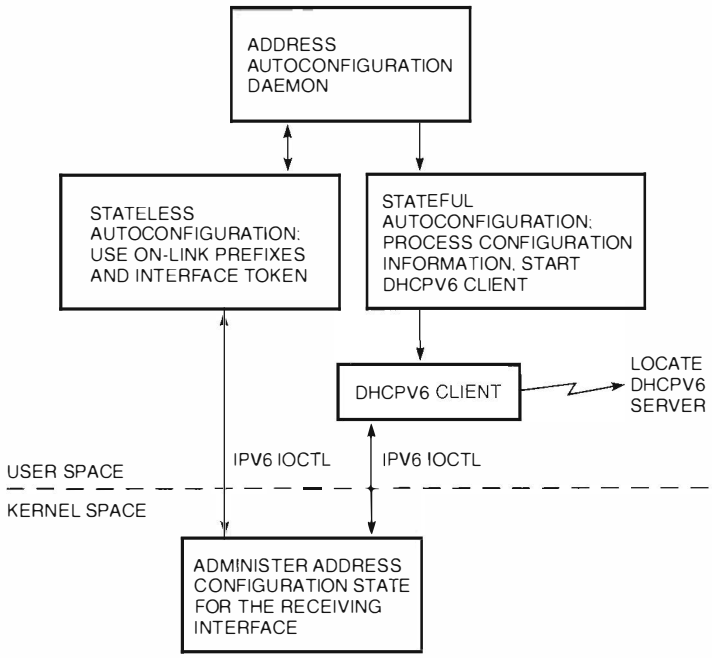


Figure 14
Address Autoconfiguration

deleted, or updated on the interface based on the prefixes and lifetimes received in the RA packet.

Interactions with Stateful Address Configuration

When the daemon receives an RA, and the managed bit flag is set, the host can use stateful address configuration, using DHCPv6. DHCPv6 is implemented as a separate daemon process in our prototype. DHCPv6 defines a complete new model from the existing DHCPv4 implementations in the industry to dynamically configure addresses. The use of link-local addresses, multicast, address configuration, and inherent support for dynamic renumbering of hosts in IPv6 caused a new architecture and design in the DHCPv6 specification. A comparison of the architectural changes between DHCPv4 and DHCPv6 can be found in the DHCPv6 specification.[8]

Application Services

Most TCP and UDP applications can be used with IPv6 with relatively minor modifications. The primary issue is the larger address size, both for internal storage needs in the application and for address transfer across system interfaces. In this section, we review these issues and others.

API

Any API currently in use for IPv4 could be modified for IPv6, but only the BSD sockets API is being investigated within the IETF for two reasons.[24,25] First, large numbers of applications use the sockets interface for IPv4, which represents a very large investment and a potential pool of IPv6 applications. Second, this API is perhaps in the most widespread use in the industry and is available on a wide variety of platforms: the benefits of standardization are compelling.

DNS AAAA Support

DNS provides support for mapping names to IP addresses and mapping IP addresses back to their corresponding names.[26] The type A resource record is used to hold an IPv4 address. Since its size is fixed at 4 bytes, a new resource record type, AAAA, was defined to hold IPv6 addresses.[27] The Digital UNIX IPv6 prototype includes a widely used implementation of the DNS known as Berkeley Internet Name Domain (BIND), which has been modified to support AAAA records.

Address Manipulation Routines

A typical IP implementation provides several library routines for manipulating IP addresses. These include routines for converting addresses between binary and textual representations and routines for translating names to addresses and addresses to names. New routines had to be provided to perform these functions for IPv6 addresses. The Digital UNIX IPv6 prototype provides the routines described in "Basic Socket Interface Extensions for IPv6."[21]

inetd Daemon

The inetd daemon creates sockets on behalf of applications, invoking the applications only when needed and

passing the open sockets to them. With the advent of the AF_INET6 socket type, inetd was modified to accept a new application configuration option in its configuration file. The keyword inet6 is used to indicate an application that wants to use AF_INET6 sockets. The keyword inet (or the absence of a keyword) indicates use of AF_INET sockets.

Applications

A typical application needs only minor modification to use the AF_INET6 address family. Applications that use addresses as part of their design or protocol, such as the File Transfer Protocol (FTP), require more extensive modification. The Digital UNIX IPv6 prototype includes several basic applications that have been modified to support IPv6, including Telnet and FTP. These programs were modified to use IPv6 sockets, address structures, and library routines. Note that the IPv6 sockets also support communications over IPv4, so that applications need not maintain separate sockets for IPv4 and IPv6, and a single executable image can interoperate with both types of remote system.

Future Work

Future implementation efforts will include security, routing, stateful address configuration, dynamic updates to DNS, IPv6 over PPP and ATM, resource reservation, and service location. In addition, we will review elements of our existing design and implementation architecture to increase performance and to ease the transition from IPv4 to IPv6. We will continue to participate in the IPv6 industry multivendor interoperability events, which is a practical and concentrated effort to debug the specifications and the code base.

IPv6 security supports both the authentication and the encryption of IPv6 packets end-to-end.[28] The module for these functions will reside in the kernel and most likely will be called at the point where the IPv6 network layer packet is processed. A key management framework is being developed to support both authentication and encryption. To access the key management interface, a sockets API extension will be provided to supply the keying criteria for the security modules.

To test the interoperability and robustness of the IPv6 implementations, a test network known as the 6BONE has been created on the Internet. This nascent test bed is currently being built with statically defined tunnels connecting IPv6 networks. Our next step in IPv6 development will be to implement routing protocols, starting with Routing Information Protocol version 6 (RIPv6) for unicast routing. Subsequent goals will be to support Open Shortest Path First version 6 (OSPFv6) and to provide multicast routing.

Stateful address configuration will be implemented as specified in DHCPv6 and will contain a client, a server, and a relay-agent. This work will be tightly coupled with dynamic updates to DNS to provide autoconfiguration in conjunction with autoregistration in the directory service. Even for networks that use stateless address autoconfiguration, DHCPv6 will be available to configure other parameters for the host and to add, delete, and update name information associated with addresses in DNS.

Additional data link interfaces will be supported for PPP and ATM. These nonbroadcast architectures will require some design analysis to implement in order to support neighbor discovery, autoconfiguration, and the routing models for IPv6. Digital has been active within the IETF working groups that are defining the ATM solutions.

IPv6 now supports flow information in the IPv6 header and in the IPv6 BSD socket API structure. This inherent quality-of-service (QOS) mechanism in IPv6 meshes well with efforts to support reserve resources on a network as specified in the Resource Reservation Protocol (RSVP).[29] Using RSVP over broadcast and nonbroadcast data links will encompass a design center that supports a wide range of resource reservation parameters to maintain a consistent performance model for video- and audio-related applications across a network path.

Service location is an emerging technology that will permit a host to query the network about the location of different services (e.g., NFS, security key management, directory services).[30] Currently in development for IPv4, service location holds promise for IPv6 and may benefit from the greater level of support for basic technologies, such as security and multicast capabilities.

Summary

Digital has designed a prototype of IPv6 on the Digital UNIX operating system. Techniques and technologies have been developed to accommodate aspects of the IPv6 architecture; in particular, the transport layer modules were modified to use two distinct network-layer protocols. The new Neighbor Discovery Protocol and algorithms have also been implemented in the prototype. IPv6 includes mechanisms to do both stateless and stateful address configuration as well as router discovery. The Digital UNIX IPv6 prototype contains a user-mode process that implements these functions. In addition, enhancements have been made to IPv4 services, and techniques have been developed to support the transition of existing applications.

References

1. P. Gross and P. Almquist, "IESG Deliberations on Routing and Addressing," RFC1380 (November 1992).

2. S. Bradner and A. Mankin, "The Recommendation for the IP Next Generation Protocol," RFC1752 (January 1995).

3. R. Hinden, "Simple Internet Protocol Plus White Paper," RFC1710 (October 1994).

4. S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC1883 (January 1996).

5. Y. Rekhter and T. Li, "An Architecture for IPv6 Unicast Address Allocation," RFC1887 (January 1996).

6. R. Hinden and J. Postel, "IPv6 Testing Address Allocation," RFC1897 (January 1996).

7. S. Thomson and T. Narten, "IPv6 Stateless Address Autoconfiguration," RFC1971 (August 1996).

8. J. Bound and C. Perkins, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)," Work in progress (August 1996).

9. A. Conta and S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6)," RFC1885 (January 1996).

10. T. Narten, E. Nordmark, and W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)," RFC1970 (August 1996).

11. M. McKusick et al., *The Design and Implementation of the 4.4 BSD Operating System,* (Reading, Mass.: Addison-Wesley, ISBN: 0-201-54979-4, 1996).

12. V. Fuller et al., "Classless Inter-Domain Routing (CIDR): An Address Assignment and Aggregation Strategy," RFC1519 (September 1993).

13. S. Deering, "Host Extensions for IP Multicasting," RFC1112 (August 1989).

14. J. Mogul and S. Deering, "Path MTU Discovery," RFC1191 (November 1990).

15. J. McCann et al., "Path MTU Discovery for IP Version 6," RFC1981 (August 1996).

16. W. Simpson, "IP in IP Tunneling," RFC1853 (October 1995).

17. A. Conta and S. Deering, "Generic Packet Tunneling in IPv6 Specification," Work in progress (October 1996).

18. K. Sklower, private communication to Matt Thomas, September 1995.

19. R. Gilligan and E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers," RFC1933 (April 1996).

20. S. Deering and R. Hinden, "IP Version 6 Addressing Architecture," RFC1884 (January 1996).

21. R. Gilligan et al., "Basic Socket Interface Extensions for IPv6," (Work in progress, April 1996).

22. M. Crawford, "A Method for the Transmission of IPv6 Packets over Ethernet Networks," RFC1972 (August 1996).

23. D. Plummer, "An Ethernet Address Resolution Protocol," RFC826 (November 1982).

24. D. Clark, "Fault Isolation and Recovery," RFC816 (July 1982).

25. W. Stevens and M. Thomas, "Advanced Sockets API for IPv6," Work in progress (October 1996).

26. P. Mockapetris, "Domain Names—Concepts and Facilities," RFC1034 (November 1987).

27. S. Thomson and C. Huitema, "DNS Extensions to Support IP Version 6," RFC1886 (December 1995).

28. R. Atkinson, "Security Architecture for the Internet Protocol," (Work in progress, June 1996).

29. "Resource ReSerVation Protocol (RSVP)—Version 1 Functional Specification," (Work in progress, August 1996).

30. J. Veizades et al., "Service Location Protocol," (Work in progress, June 1996).

General References

S. Bradner and A. Mankin, eds., *IPng—Internet Protocol Next Generation* (Reading, Mass.: Addison-Wesley, ISBN: 0-201-63395-7, 1996).

S. Thomas, *IPng and the TCP/IP Protocols* (New York: John Wiley & Sons, Inc., ISBN: 0-471-13088-5, 1996).

R. Braden, "Requirements for Internet Hosts—Communication Layers," RFC1122 (October 1989).

G. Wright and R. Stevens, *TCP/IP Illustrated, Volume 2—The Implementation* (Reading, Mass.: Addison-Wesley, ISBN: 0-201-63354-X, 1995).

Biographies

Daniel T. Harrington

As a principal software engineer in Digital's IPv6 Program Office, Dan Harrington participated in the Digital UNIX IPv6 prototype effort. Prior to this work, he helped develop the DECnet/OSI products on the ULTRIX and the Digital UNIX platforms. After joining Digital in 1982, Dan worked in performance analysis, field support, and software development. He received a B.S. in mathematics from Rensselaer Polytechnic Institute. Dan is currently with Lucent Technologies.

James P. Bound
Jim Bound is a consulting software engineer and the technical director for IPv6 within the IPv6 Program Office. Jim is responsible for the overall advanced development architecture and reference Alpha Digital UNIX code base, which verifies that the IPv6 specifications are implementable. He is also Digital's IETF IPv6 technical leader and one of the IPv6 advanced development engineers on Alpha Digital UNIX. In 1993, Jim began his participation in the IETF to work on the IPng and the advanced development IPng prototype. As a member of the IETF's IPng Directorate, Jim helped determine the requirements and core architecture for IPng's Internet protocol and related functionality to support IPng. The result was the selection of a proposal, now known as the Internet Protocol version 6 (IPv6). Jim has an A.S. in business management and an A.S. in computer science. He is a coauthor of several IPv6 specifications and a contributing author to the book *IPng: Internet Protocol Next Generation*. He is a member of the IEEE and the Internet Society.

John J. McCann
Jack McCann is a principal software engineer in the UNIX Engineering Group and a member of the IPv6 project team. He contributed to the design and implementation of the Digital UNIX IPv6 prototype, including router discovery, autoconfiguration, fragmentation, reassembly, path MTU discovery, forwarding, and the IPv6 API. He participates in several IETF working groups and is a coauthor of Internet RFC 1981, "Path MTU Discovery for IP version 6." Jack joined Digital in 1988 to become a member of the Distributed Systems Technical Evaluation Group. He also worked in the DECnet/OSI for OpenVMS Engineering Group before taking his current position. He received a B.S. in computer science (magna cum laude) from the University of Lowell in 1988 and an M.S. in computer science from Boston University in 1995.

Matt Thomas
Matt Thomas joined Digital in 1983 with Software Services in California. Although he is a principal software engineer in the OpenVMS Systems Software Group, Matt has spent the last eight years as a developer of networking products for the Digital UNIX and ULTRIX systems. In addition to his ongoing involvement with Digital UNIX IPv6 efforts, he is responsible for adding IP security to the Digital UNIX operating system. Matt is an active participant in various IETF working groups and is a coauthor of several Internet Drafts.

Preserving Computing's Past: Restoration and Simulation

Maxwell M. Burnet
Robert M. Supnik

Restoration and simulation are two techniques for preserving computing systems of historical interest. In computer restoration, historical systems are returned to working condition through repair of broken electrical and mechanical subsystems, if necessary substituting current parts for the original ones. In computer simulation, historical systems are re-created as software programs on current computer systems. In each case, the operating environment of the original system is presented to a modern user for inspection or analysis. This differs with computer conservation, which preserves historical systems in their current state, usually one of disrepair. The authors argue that an understanding of computing's past is vital to understanding its future, and thus that restoration, rather than just conservation, of historic systems is an important activity for computer technologists.

The Computing Past

The continuous improvements in computing technology cause the rapid obsolescence of computer systems, architectures, media, and devices. Since old computing systems are rarely perceived to have any value, the danger of losing portions of the computing record is significant. When a computing architecture becomes extinct, its software, data, and written and oral records often disappear with it.

Older computer systems embody major investments in software, the value of which may persist long after the systems have lost their technical relevancy. For example, the PDP-11 computer has not been a leading-edge architecture since the introduction of 32-bit systems in the late 1970s and has not received a new hardware implementation since 1984. Nonetheless, PDP-11 systems continue to be used worldwide, particularly in real-time and control applications. The unavailability of suitable replacements of worn-out original parts is a serious issue for PDP-11 systems still in use.

Another area of potential loss is data. In recent years, archival storage media have undergone rapid technologic evolution, and the industry standards of computing's first 30 years, such as 0.5-inch magnetic tape, are now antiques. Salvaging data from original media is an industry-wide problem and has generated a small cottage industry of specialists in data recovery. This problem will only proliferate, as transitions in media types accelerate. Ten years from now, the large-diameter optical disks used for today's archives will look as quaint as DECtape and magnetic tape storage systems do to current computer users.

Finally, the disappearance of older equipment typically entails loss of information: not only design sketches, blueprints, and documentation but also the folklore about these systems. The absence of systematic archiving, as well as the absence of a perceived value of the archived data, causes continual information decay about design and operational details.

This paper describes two techniques for preserving computing systems of historical interest. The first section of the paper discusses the restoration of old computers to working order. It also includes a description of the Australian Museum collection and the

process of restoring a particular PDP-11 minicomputer. The second section discusses the simulation of old computers on modern systems. It describes a simulation framework called SIM, which has been used to implement simulators for the PDP-8, PDP-11, PDP-4/7/9/15, and Nova minicomputers.

Restoring Old Computers

Since the computer became a mass-produced item in the late 1960s, its typical life cycle has consisted of initial installation, rental or depreciation for about five years, retention and use for a few more years (just in case), and then retirement and a trip to the refuse dump. There is only a brief window of opportunity to collect old computers at the end of their working life. Once that window is closed, the computers are gone forever.

The Australian Museum Collection

In Sydney, Australia, this window of opportunity first became apparent in 1971, when the early PDP systems reached the ends of their life cycles. Digital's Australian subsidiary began collecting systems by a creative program of trade-ins for new equipment.[1] It was especially urgent to obtain examples of the 12-bit, 18-bit, and 36-bit PDP series, as they were relatively few in number. Table 1 lists the percentage of available units that have been collected. The status of each is given as

- Static—can never be made to work for various reasons
- Restorable—could be made to work with enough care, patience, time, and effort
- Working—running its operating system the last time it was turned on

Once a representative sample of the early PDP systems had been collected, the urgency abated. Hundreds of PDP-11 and VAX systems were then brought to Australia; the window of opportunity for collecting them is still open.

The collection has grown significantly during the last 25 years. At the present time, we have in Sydney a comprehensive collection of most early Digital machines, including hardware, manuals, software, and spares (see Table 2). The collection is catalogued in a 6,000-line database that resides, appropriately, on a MicroVAX I computer, running the first version of the MicroVMS operating system. Figure 1 shows an example from the collection, a PDP-8/E computer system with peripheral equipment.

The goals of the collection are varied and are summarized in Table 3. Apart from the academic challenge of keeping all old data media running, there is the responsibility to ensure that they can be kept alive and available. The extensive variety of media types offered by Digital alone in only 30 years is summarized in Table 4. The evolving status of the collection has been reported at several Australian DECUS Symposia.[2,3] The restoration of the Australian collection will probably ensure a retirement job for the curator for the next 30 years!

General Issues in Restoration

Restoration is a painstaking and time-consuming process. The goal of restoration is to return a system to a state where it will reliably run a major operating system and offer as many media conversion facilities of the vintage as possible. Fortunately, computers do not deteriorate greatly in storage, provided the storage area is dry. (One item that does decay dramatically is the black foam used to line side panels and to separate

Table 1
Early Digital CPUs in Australia

Model Name	Number Brought to Australia	Number in Museum Collection	Condition
PDP-5	1	1	Restorable
PDP-6	1	1	Some items
PDP-7	1	1	Static
PDP-8	28	3	Working
PDP-8/S	20	2	Static
LINC-8	2	2	Restorable
PDP-9	7	1	Restorable
PDP-10	8	1	Some items
PDP-12	2	2	Restorable
PDP-8/I	24	2	Restorable
PDP-8/L	21	2	Restorable
PDP-15	10	1	Static
PDP-8/E	90	4	Working

Table 2
The Digital Australian Collection (chronological order)

Year	Item	Description	Status
1958	138	A/D converter	Static
1960	ASR-33	Teletype reader/punch, 110 baud	Working
1962	KSR-35	Heavy-duty Teletype	Working
1963	PDP-6	Modules of first Digital computer in Australia	Parts
1963	PDP-5	First minicomputer in Australia	Working
1967	PDP-7	Third Digital computer in Australia	Static
1965	PDP-8	Classic, table-top model	Working
1965	PDP-8	Cabinet model	Restorable
1965	PDP-8	Typesetting system	Static
1965	PDP-8	Cabinet model, first in New Zealand	Restorable
1965	COPE-45	Remote batch (OEM PDP-8)	Restorable
1966	PDP-9	18-bit computer	Static
1966	KA10	Console of PDP-10 mainframe	Static
1966	Linc-8	Early medical computer	Working
1967	PDP-8/S	Serial, under $10,000, CPU	Static
1967	PDP-8/S	Serial computer	Static
1967	DF32	Digital's first disk, 1/16 Mb	Static
1967	PDP-9/L	Last transistor logic, 18-bit	Static
1968	PDP-8/I	Digital's first IC minicomputer	Working
1968	PDP-8/L	OEM version of PDP-8/I	Static
1969	PDP-12	Laboratory computer	Working
1969	PDP-12	Laboratory computer	Static
1969	PDP-15	Last of 18-bit family	Static
1969	KI10	Console of DECsystem-10	Static
1970	PDP-8/E	Pinnacle of PDP-8 development	Working
1970	PDP-8/E	Full LAB 8 configuration	Working
1970	PDP-11/20	The first PDP-11	Working
1970	CR11	Card reader, 285 cpm	Working
1971	PDP-8/F	Small PDP-8/E	Working
1971	VT05	Digital's first video terminal	Working
1971	LA30P	Digital's first hard-copy terminal	Working
1971	PDP-11/45	Last PDP-11	Static
1972	GT40	Graphics workstation	Broken
1972	PDP-11/10	Small PDP-11	Static
1973	PDP-11E10	First packaged system	Working
1973	PDP-11/35	Mid-range PDP-11	Static
1973	PDP-8/A	Last non-chip PDP-8	Working
1974	PDP-11/40	Mid-range, end-user PDP-11	Restorable
1975	VT50	Video terminal	Working
1975	LA36	DECwriter II printer	Working
1975	DS310	Desk-based commercial system	Working
1975	PDP-11/70	Largest PDP-11	Restorable
1976	PDP-11/34	Mid-range PDP-11	Working
1977	PRS01	Portable paper tape reader	Working
1977	LS120	DECwriter printer	Working
1977	WS78	Word processor, 8-inch floppy disks	Working
1978	LA120	DECwriter III printer, 180 cps	Working
1978	VAX-11/780	Original unit of 1 VAX-11/780	Restorable

continued on next page

Table 2 (continued)

Year	Item	Description	Status
1979	VT100	Famous video terminal	Working
1980	MINC	LSI-11 lab unit with RT-11	Working
1980	VAX-11/750	Mid-range VAX system	Restorable
1980	PDT-150	Table-top LSI-11 with RX01 drives	Working
1981	GIGI	Low-cost terminal for schools	Working
1982	VT125	Video terminal with graphics	Working
1982	WS278	DECmate I word processor	Restorable
1982	VAX-11/730	Low-performance VAX system	Working
1982	LA12	Portable hard-copy terminal	Static
1982	LQP03	Letter-quality printer	Working
1982	DECmate II	Word processor on mobile stand	Working
1982	DECmate II	Word processor	Working
1982	Rainbow	Personal computer	Working
1982	PRO350	Professional PC	Working
1983	VT241	Graphics color terminal	Working
1983	MicroVAX I	Smallest VAX .3 VUP	Working
1983	VAX-11/725	Lowest cabinet VAX .3 VUP	Working
1984	LN03	Laser printer	Working
1985	MicroVAX II	Famous MicroVAX II	Working
1986	VAXmate	286-based PC with RX33 drive	Working
1986	DECmate III	Small word processor	Working
1987	MicroVAX III	3-VUP MicroVAX II system	Working
1987	VAX 8250	Dual VAX CPU, BI-based	Restorable
1989	VAX 9000	Chip set	Static
1990	DS3100	Mips UNIX workstation	Restorable

ribbon cables. After 20 years, it turns into a sticky, gooey mess. It should be removed as soon as possible; otherwise, it falls into the modules and backplane. Replacing it with a modern equivalent can be done but is not essential.)

The first step in restoration is to collect hardware, software, and documentation.

- Collect the hardware, if possible two or ideally three items of each example. This provides a system to work on and a spare, as well as the ability to make comparisons between units.
- Collect diagnostic and operating software on original bootstrap media. Sources are very useful, particularly for diagnostics.
- Collect hardware manuals and schematics.

There is a network of enthusiasts around the world who can help at this stage.

Once the "ingredients" have been collected, the steps needed to restore a 1960s or 1970s vintage machine are as follows:

- Inspect the hardware for physical safety, particularly the heavy drawers and slide mechanisms.
- Physically assemble the hardware, checking module allocations, cabling, etc.
- Carefully inspect the power system, high-voltage sources can kill. Although most of the power wiring material appears to stand the test of time, the early machines often had rather thin coverings on terminals. Safety-first is a principal criterion in restoration, since someday nontechnical people may open the back door.
- Assemble a minimal system of CPU, memory, and console switch register for initial tests.
- Power up the computer, checking supply voltages, fans, and front console for signs of life.
- Use simple routines at the switch register to check for elementary operation.
- Fit a serial line unit so that a VT or a Teletype console can be used.
- Get the keyboard echoing to the screen or printer with simple routines.
- If they are available, run the internal tests of the read-only memory (ROM).

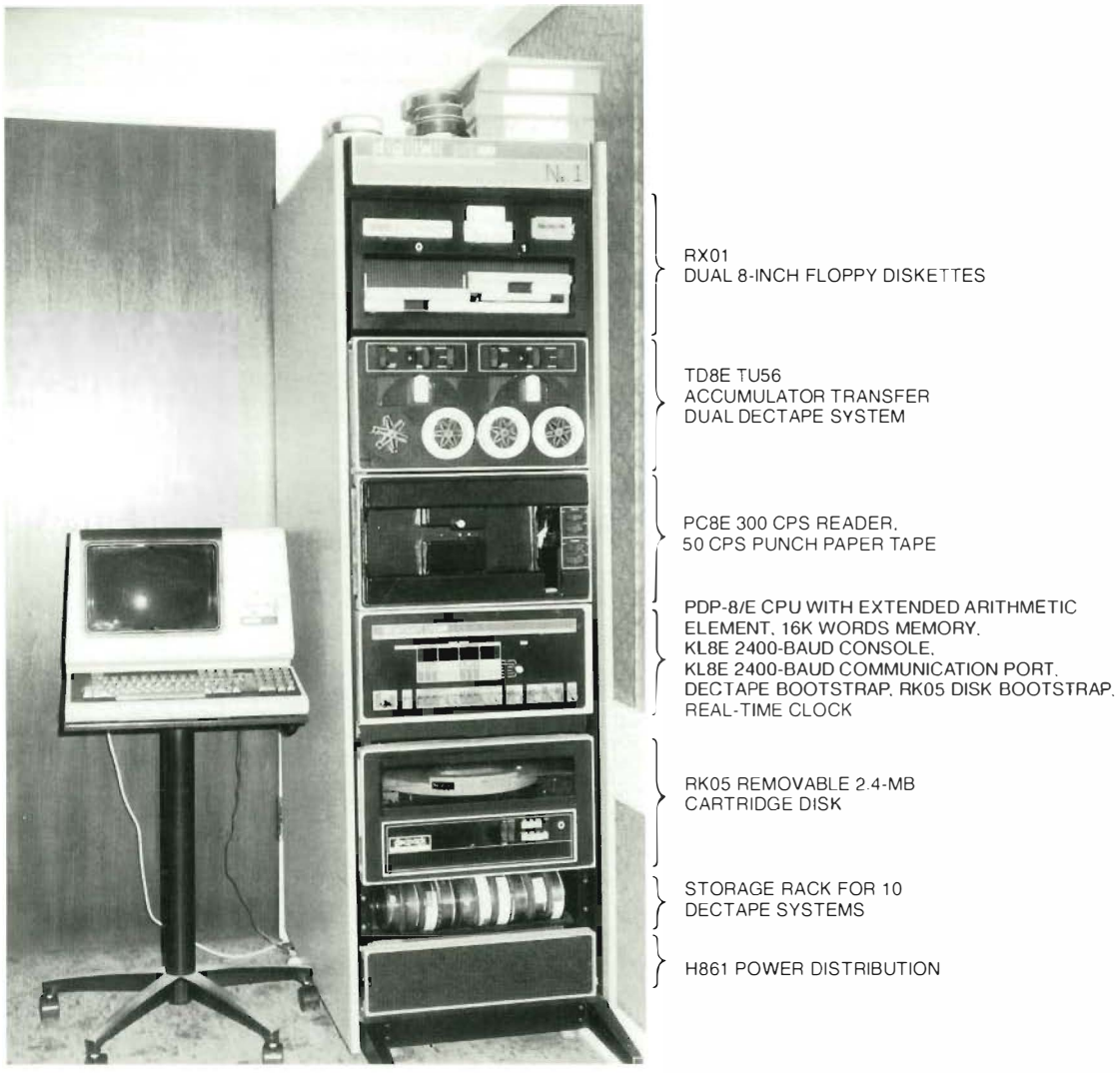


Figure 1
PDP-8/E Computer System

Conventional wisdom would now advise that all the diagnostic routines be run. However, diagnostics were (philosophically) always used to find bugs in a previously good machine; they are too complex when huge chunks of the machine might still be missing. The most practical next step is to get mass storage on-line. Depending on the manufacturer, the target device may be a floppy disk drive, a cartridge hard disk drive, or some form of magnetic tape. With a working mass storage device and a bootstrap routine, it becomes possible to boot a simple operating system (like OS/8 or RT-11 for Digital's systems). This quickly shows whether the machine is working or not.

If a mass storage device is not available, the next best thing is paper tape. This can be either the system's rack-mounted reader and punch or the paper tape reader on an ASR33 or ASR35 console. The reliability is questionable, however, and the procedure is tedious. Many diagnostics were on paper tape, but usually the quickest test is to load a complete paper system (such as FOCAL for Digital's systems). If the diagnostics run, the system is probably functional.

Once the CPU, console, and memory are verified, additional peripherals can be added, one at a time. It pays to take the time and effort to research bus addresses, interrupt vectors, power supply loading, and module placement, and to keep a log book with configuration diagrams and results. In general, if the configuration rules are followed, the items will work. There are few electronic failures, even in 20- or 30-year-old modules. When a problem arises, it is usually address vector strapping, physical damage, or missing cables. Corrosion of board contacts can be a problem; they should be cleaned with a clean cloth or cardboard

Table 3
Goals of the Australian Digital Museum

To preserve one of each model of Digital's computers
To keep each major Digital operating system working
To have a working unit of each Digital terminal, console, and PC
To provide conversion and archival facilities for old media
To preserve significant Digital literature and manuals
To preserve a VAX-11/780 computer as the original unit of 1 VUP
To disseminate instructive and educational material
To educate and amuse our staff, our customers, and the public
To support the DECUS NOP (nostalgic obsolete product) Special Interest Group
To preserve spares, tools, test gear, and documentation to keep the collection working
To preserve and protect these treasures for future generations

Table 4
Digital Data Media from 1960 to 1996

Paper tape
80-column punched and mark sense cards
7-track, half-inch magnetic tape
9-track, half-inch magnetic tape
DECtape and LINCtape systems
Audiocassette
DECtape II cartridge (TU58)
CompacTape (TK50, etc.)
Quarter-inch cartridge tape
Digital audio tape
8-inch floppy disk
5.25-inch floppy disk
3.5-inch floppy disk
RK05 removable disk
RK06, RK07 removable disk
RL01, RL02 removable disk
RP01...RP06 removable disk
RM03, RM05 removable disk
RC25 removable disk

(for example, a business card), not with a pencil eraser, which leaves residues. Silicon components appear to be very stable and a tribute to the conservative design principles of early computer engineers.

The main components that seem to age are power supply capacitors, fans, and lights. The filter capacitors across the high-voltage sources can short, and reference electrolytic capacitors in power supply regulators can dry out. Although the large capacitors in power supply RC filters have proven to be reliable, some restorers replace them as a matter of course for safety reasons. Small rotary fans may seize if they have logged many hours. Incandescent panel lamps are always failing and can be replaced by modern light-emitting diodes (LEDs) if required. The irony is that the panel lamps are needed only during initial checkout; once the operating system is running, they are rarely used.

Once restored, are old units reliable? Experience proves that they are. A classic PDP-8 system restored in 1988 still turns on happily (untouched) eight years later. A fully configured PDP-8/E system is still working four years after restoration.

Restoring a Minicomputer: A Case Study

An ongoing project is the restoration of a large, UNIBUS-based PDP-11 system with many UNIBUS peripherals attached to it. The project was started using the original PDP-11/20 CPU. Since many PDP-11 peripherals were designed long after the PDP-11/20 CPU, it could not cope with single-board direct memory access (DMA) devices, metal-oxide semiconductor (MOS) memory, and other later inventions. The project refocused on the mid-range PDP-11/34, which in retrospect has proved wise. The PDP-11/34 supports MOS memory, has an LED and push-button console, and represents a mature implementation of the PDP-11 instruction set. It has an optional cache, battery backup, floating-point operation, and the extended instruction set (EIS).

The current configuration occupies three large cabinets in what used to be the dining room of Max Burnet's house. The virtues of the UNIBUS are many; in particular, it allows modular connection of I/O devices and other components. However, I/O devices of the era often weigh 100 pounds and are mounted in 10-inch drawers; their sheer physical size and weight are disincentives to reconfiguration.

The project currently uses the RT-11 operating system because of its simplicity and extensive device drivers. Eventually, it may be possible to run the RSX-11M and the RSTS/E systems, but there is little to gain from a media conversion point of view, because RT-11 includes utilities for dealing with foreign file formats.

The main difficulties encountered have been associated with the power supply: the DC low signal threads its way through every peripheral. The absence of UNIBUS grant continuity cards can create havoc. Since this PDP-11 system is very large, it is straining the design rules concerning floating vectors, current loading, and bus loads.

The CPU and memory are relatively easy to check out. Due to the versatility of the UNIBUS, however, checking out the I/O system is very laborious. Starting with programmed I/O tests works best, followed by interrupt tests, and finally DMA or nonprocessor reference (NPR) tests. Experience shows that tests need to be rerun whenever a new peripheral is added.

The system currently runs the RT-11 version 5.04 operating system on a configuration comprising

- RT-11/34 CPU with real-time clock and bootstraps
- 256 kilobits of MOS memory
- RX01 and RX02 floppy disks
- Dual RL02 disks
- TU56 dual DECtape storage system
- TU58 DECtape II storage system
- Serial line units for console and serial printer
- CM11 mark sense and CR11 punched card reader
- TU60 cassette
- PC11 paper tape reader and punch

Although the following peripherals are available, they await installation time and effort:

- LPS-40 analog-to-digital (A/D) converter
- TU10 magnetic tape
- TSV03 magnetic tape
- Cache and commercial instruction set
- Battery backup kit

The eventual goal is to keep "the last great (UNIBUS) PDP-11" running with almost every UNIBUS peripheral ever made.[4] Time will tell.

Simulating Old Computers

A simulator is a computer program operating on one computer system (known as the host system) which mimics the behavior of another computer system (known as the target system). The simulator's data is the state of the target computer system—registers, memory, timed events, and so on. The simulator operates on presented state and transforms it, usually by sequential evaluation, in the same manner as would the target computer system.

Simulators typically consist of an execution engine, which performs the state transformations; a simple timed-event mechanism, which supports deferred and asynchronous events such as I/O completions; and a control panel, which provides user access to simulated state. The execution engine is responsible for decoding instructions in simulated memory and performing the specified alterations of simulated machine state. The execution engine keeps track of simulated time in arbitrary units, which may be precise representations of the execution time of the target system, or simple representations of advancing time, such as the number of instructions executed. The event mechanism provides a way to schedule events, such as I/O completion, for later evaluation. It can also implement other time-based mechanisms such as keyboard polling. Finally, the control panel provides access to simulated state as well as basic control commands such as start and stop. It may also provide more elaborate facilities to support performance instrumentation or debugging.

Historically, simulators have been used for many purposes, including the following:

- Design of new systems. The simulator mimics the behavior of a future chip or computer system and is used to understand and debug the behavior of the proposed design. For example, prior to fabrication, all modern microprocessors are extensively simulated, first as abstract performance models and then at increasing levels of detail. [5,6]
- Debugging for embedded systems. If the simulator contains facilities for program debugging, it becomes a useful tool for debugging programs that run in highly constrained environments such as embedded systems. Simulators can capture more state and provide a wider range of facilities than in situ debuggers. For example, simulators can implement program counter (PC) change queues, data access breakpoints, or precise traps on errors.
- Replicable event tracing. Most simulators are fully deterministic. Asynchronous events are scheduled based on simple, nonrandom algorithms, such as fixed time-out or calculated seek time. As a result, simulators allow for straightforward replication or playback of complicated sequences, removing the randomness factor that often plagues the debugging of asynchronous software on real systems.
- Preservation of past software. Simulators can provide migration assistance in the transition from older to newer architectures. Many transitional computer systems have provided simulators for older architectures, typically at the microcode level, to assist customers and developers in preserving their investments in the previous architecture. Examples include the early IBM System/360 series, which had models that simulated the 1401, 1410, 7070, and 7090 families, and the early Digital VAX systems, which included a PDP-11 compatibility mode.[10,11]

Simulation Levels

Simulators can be written at various levels of detail and thus various levels of fidelity to the target system. Three common levels of simulation are register transfer level (RTL), instruction, and software specific.

An RTL simulator attempts to mimic the major hardware blocks of the target system and to implement its actual logic equations. The goal is absolute

fidelity, the test of which is that no piece of software running on the simulator should behave differently than it would on the target hardware. In practice, such perfect mimicry is difficult to achieve, as it requires a painstaking re-creation of timing detail (for example, the actual acceleration curve of a DECtape storage system) and access to implementation documentation that has often vanished. Nonetheless, some simulators have achieved results very close to this goal: MIMIC, a DECsystem-10 simulator written at Applied Data Research, was able to run CPU- and device-specific diagnostics. (As testimony to the vulnerability of computing's past, all machine-readable copies of the MIMIC sources appear to have been lost.)

An instruction simulator steps back from the RTL level and tries to simulate at the functional or the behavioral level. System elements are treated as functions that transform state according to the abstract definitions of the system architecture, rather than as logic blocks that transform state based on implementation equations. Instruction simulators sacrifice absolute fidelity to the idiosyncrasies of a particular implementation and focus on the intentions of the architecture specification. As a result, instruction simulators can usually run systems software and applications but can rarely fool diagnostics.

Finally, a software-specific simulation further abstracts the functions of the target system to only those needed by a particular piece of target system software. For example, the OS/8 operating system on the PDP-8 computer does not use program interrupts; a simulator aimed at running only the OS/8 operating system would not need to implement interrupts or even queued events. A recent PDP-11 simulator designed to run the 2.9 BSD UNIX operating system abstracted parts of the PDP-11 system's interrupt model and could not run other PDP-11 operating systems.[12]

Simulating Minicomputers: A Case Study

SIM is a portable instruction-level minicomputer simulator implemented in C. Its objectives are to facilitate the study and use of historic computer architectures by making simulated implementations and historic software available to anyone who has a 32-bit computer. It supports the following target architectures

- PDP-8
- PDP-11
- Nova
- 18-bit PDP series (PDP-4, PDP-7, PDP-9, PDP-15)

and has been successfully ported to the VAX VMS, the Alpha OpenVMS, the Digital UNIX, and the Linux architectures. Ports to the Windows NT and the Windows 95 architectures and to an IBM 1401 simulator are under way.

General Design Considerations The design of an instruction-level simulator is not technically complicated; indeed, simulating a PDP-8 system is a common problem in undergraduate computer science courses. SIM follows the processor-memory-switch (PMS) structure proposed by Bell and Newell and implemented in MIMIC and countless other simulators since.[10,13] The simulated system is a collection of devices, one of which has special properties (the CPU). Each device has state (registers) and one or more units. Each unit has state and fixed- or variable-sized storage. In the CPU device, the storage is main memory. In an I/O device, the storage is the device media. The CPU is distinguished from other devices by having the master routine for instruction execution. This routine is responsible for the sequential evaluation of instructions and for the state transformations that represent simulated execution. The CPU also provides a few systemwide routines, such as symbolic disassembly and input and a binary loader.

The devices interface to a control panel that provides access to simulated state and control over execution. The available commands in SIM are listed in Table 5.

The control panel also includes routines that are needed by most simulators, such as event queue maintenance and character-by-character terminal I/O. Different simulators need not use the same time base, but all the SIM-based implementations to date use the number of instructions executed as the time base.

Note that the control panel provides for starting simulation, but termination is determined entirely by the simulated CPU. By convention, the CPU returns control to the control panel under the following conditions:

1. If a HALT instruction is executed
2. If a fatal exception is detected
3. If a fatal I/O error is detected
4. If a special character is typed at the controlling terminal

Likewise, the control panel does not implement any debugging facilities beyond state examination and modification and instruction stepping. To facilitate debugging with operating systems, CPUs provide a simple instruction breakpoint capability and a one-level PC trace facility.

Implementation The implementation of a particular simulator begins with collecting reference manuals, maintenance manuals, design documents, folklore, and prior simulator implementations for the target system. This is nontrivial. In the early days of computing, companies did not systematically collect and archive design documentation. In addition, collected material is subject to information decay, as noted

Table 5
Commands Available in SIM

Command	Definition
attach <unit> <file>	Associate file with unit's media.
detach <unit> | ALL	Disassociate unit's (all units) media from any file.
reset <device> | ALL	Reset device (all devices).
load <file>	Load binary program from file.
boot <unit>	Reset all devices and bootstrap from unit.
run {<new PC>}	Reset all devices and resume execution at the current PC (or new PC).
go {<new PC>}	Resume execution at the current PC (or new PC).
cont	Resume execution at the current PC.
step {<number>}	Execute one instruction (or number instructions).
examine <list>	Display contents of list of memory locations or registers.
iexamine <list>	Display contents of list of memory locations or registers and allow interactive modification.
deposit <list> <value>	Store value in list of memory locations or registers.
ideposit <list>	Interactively modify list of memory locations or registers.
save <file>	Save simulator state in file.
restore <file>	Restore simulator state from file.
show queue	Display the simulator's event queue.
show configuration	Display the simulator's configuration.
show time	Display the simulated time counter.
show <device>	Show device's configuration options.
set <device> <option>	Set a device configuration option.
help	Display a terse help message.
exit | quit | bye	Leave the simulator.

earlier. Lastly, the material is likely to be contradictory, embodying differing revisions or versions of the architecture, as well as errors that have crept in during the documentation process.

For Digital's 12-bit and 16-bit minicomputers, the typical hierarchy of documentation was the following:

- Processor Handbook. Providing an all-inclusive summary of the instruction set architecture, peripherals, bus interface, and software, these paperback-size books are the most common form of system documentation but also the least accurate.
- Subsystem Reference Manual. As the programmer's reference manual for a particular subsystem, such as the CPU or the disk drive, these manuals describe the registers and functions accurately but omit maintenance-level features and other fine points.
- Subsystem Maintenance Manual. As the maintenance engineer's manual for a particular subsystem, these manuals describe the registers and functions at the hardware implementation level, often including substantial abstracts from the print set. Because of the level of detail, the maintenance manuals have proven to be the most useful references for simulator implementation.
- Design documents. For systems that do not have very large-scale integration (VLSI), the only extant design documents are the logic prints and the binary microcode ROM listings. The prints are essential for RTL simulation: they provide the only documentation of implementation quirks. For VLSI systems, there are chip-level design specifications as well as human-readable microprogram listings.
- Folklore. During the useful lifetime of a system, its users exchange information and create an informal record, both written and verbal, of shared experiences (folklore) regarding the fine points of operations, hardware/software interfaces, system "personality," and other factors. Folklore is subject to rapid information decay, particularly once the target system becomes obsolete.
- Prior implementations. Prior simulator implementations can provide useful information, but it must be used cautiously. Unless the prior implementation is an RTL model, it embodies simplifications and abstractions that are not explicitly documented. The MIMIC sources (which are fragmentary and available only on paper) proved trustworthy, but others did not: for example, the 1970s PDP-11 simulator in the DECUS archives is highly misleading about interrupts, condition codes, and other details.

An important consideration is that much of the documentation, all the folklore, and most working systems are in the hands of individual collectors. The Internet plays a vital role in locating material held by enthusiasts, through news conferences such as alt.folklore.computers, alt.sys.pdp8, alt.sys.pdp11, and comp.emulators.misc, and more recently, through World Wide Web sites devoted to historic systems.[14–16] The sources for each simulator in SIM are listed in Table 6.

The last step in implementation is collecting software to run on the simulator. Software collection immediately raises the problem of media translation. Software for historic systems resides on paper tapes, DECtape storage systems, 200/556/800 bits-per-inch magnetic tapes, disk cartridges, 8-inch floppy disks, and so on. Few if any modern systems have these peripherals; and few if any historic systems have modern network interconnects. Thus, media translation usually entails linking a working version of the target system to a modern system by means of a serial line. KERMIT or some other simple protocol allows for a byte-by-byte network copy from the original media to a file on a modern system.

Once the software has been located and moved to a file, the next issue is sources. Without sources, diagnostics and other test programs are useless; detected errors cannot be traced back to causes without manual decode of the binary program. The absence of sources was a principal reason for including symbolic disassembly and input in SIM.

The final issue in software is licensing. Even though the target systems are obsolete and often no longer manufactured, the operating system software may be protected by copyrights and licenses. Most PDP-8 software is in the public domain; however, the PDP-11 and Nova operating systems are still licensed, as are all versions of UNIX. Corporate licensing policies rarely accommodate hobbyists; this limits operating system distribution to legitimate (that is, business) users. Table 7 lists the software found for each simulator in SIM.

Debug The debug path for a simulator depends on the available software. Ideally, the simulator would be debugged with the same software tests used to debug the target hardware, but this software is rarely archived. Diagnostics can provide low-level checking, but diagnostics typically check for broken parts in a correct implementation, rather than an incorrect implementation. Even when diagnostics do check architecture rather than implementation (as in the basic instruction diagnostics on the PDP-11 system), the absence of sources limits their utility. Consequently, the simulators were debugged mostly with simple hand tests and then with the operating systems.

Operating systems are both exacting and imprecise tests of implementation correctness. Unless an operating system takes a deliberately restrictive view of hardware (for example, OS/8 does not use the PDP-8 interrupt system, and RT-11 does not use any

Table 6
Sources for Simulators in SIM

Architecture	Documents	Location
PDP-8	Minicomputer Handbook	Private collection
	Reference manuals	Digital archive
	Maintenance manuals	Digital Australia collection
	Print sets	Digital Australia collection
	Prior implementations	Public archive[17]
		Public archive[18]
		MIMIC, private collection
PDP-11	Minicomputer Handbook	Private collection
	Reference manuals	Digital archive
	Maintenance manuals	Digital Australia collection
	Chip specifications	Private collection
	Microcode listings	Private collection
	Prior implementations	Public archive[19]
		MIMIC, private collection
Nova	System Reference Manual	Private collection
	Reference manuals	Data General archive
	Maintenance manuals	Private collection
	Prior implementations	MIMIC, private collection
18-bit PDP	Reference manuals	Digital archive
	Maintenance manuals	Digital archive
	Print sets	Digital archive

Table 7
Software for Simulators in SIM

Architecture	Software	Location
PDP-8	Basic instruction tests 1 and 2	Digital Australia collection
	Memory management test	Digital Australia collection
	FOCAL69	Digital Australia collection
	OS/8 system disk	Public archive[18]
PDP-11	RT-11	Transcribed from real system
	RSX-11M	Transcribed from real system
	RSTS/E	Transcribed from real system
	UNIX V5, V6, V7, 2.9 BSD	PDP UNIX Preservation Society (PUPS) archive[20]
	2.11 BSD	Private collection
Nova	RDOS	Private collection
18-bit PDP	No software to date	

optional PDP-11 instructions), the operating system will be sensitive to every error in implementation. For example, Digital's second-generation PDP-11 systems—the PDP-11/05, 11/40, and 11/45—were debugged with DOS-11 and RSTS after diagnostics failed to detect certain subtle implementation errors. Unfortunately, in an operating system, the distance in time and space between the error and the symptom may be enormous, and the traceable path may be lengthy and complicated. Artifacts in the software can also complicate debug: the OS/8 disk image on the Internet contains a copy of BASIC that is broken.

Results SIM implements four minicomputer architectures: PDP-8, PDP-11, Nova, and 18-bit PDP. Each simulator includes a particular CPU; basic peripherals such as terminal, paper tape, clock, and printer; and a selection of mass storage peripherals (see Table 8).

The PDP-8 simulator has run the FOCAL69 and the OS/8 operating systems. The PDP-11 simulator has run the following operating systems: RT-11 V4 and V5; RSX-11M V4; RSTS/E V8; UNIX V5, V6, and V7; and BSD V2.9 and V2.11. The Nova simulator has run the RDOS V7.5 operating system. No system software for the 18-bit PDP systems has been found. The simulators were exercised on an AlphaStation 3000/600 workstation (approximately 120 SPECint92); the performance is given in Table 9.

Figures 2, 3, and 4 show screen shots from the various simulators running their principal operating systems.

In Defense of Computing's History

As professional engineers who have been lucky enough to witness the computer revolution, the authors believe that the industry has a duty to keep early machines alive. There are practical reasons, such as preservation of software and data; beyond that, there is an obligation to future generations. In 100 years, the systems from computing's early history will appear to be absolute dinosaurs of the past. Yet their educational and sociological value will be considerable. A computer is a machine with a soul, and it must be kept alive with its operating environment to show its abilities and the contemporary state of the art.

Acknowledgments

Max Burnet: I would like to thank Digital Equipment Corporation Australia Pty Ltd for tolerating my eccentricity and for supporting the Australian Digital museum collection. Also the DECUS Australia NOP (nostalgic obsolete product) SIG members for help, encouragement, knowledge, good humor, and camaraderie on the last Wednesday of the month. My thanks to my coauthor Bob Supnik for his continued inspiration; it is great to see a V.P. who can cut code with the best of them. My thanks also to the contributors to the Digital Notes files, a great source of folklore. Therein lies a treasure trove of solutions from people who are helping each other solve the same problems.

Bob Supnik: The design, implementation, and debug of SIM was made possible by the generous help of many people. Craig St. Clair and Deb Toivonen of the Digital archives located rare manuals and documents on Digital's 12-bit, 16-bit, and 18-bit systems. Tom West and Don Lewine of Data General Corporation provided documentation and support on the Nova. Carl Friend's private collection of Data General hardware and software was a crucial source of documentation and software for the Nova and the RDOS operating system. Doug Jones, Bill Haygood, and John Wilson allowed me to use the sources to their simulators and freely answered arcane questions about

Table 8
Architectures Implemented by SIM

	PDP-8	PDP-11	Nova
CPU	PDP-8/E	J-11, Q-bus	Nova 820
Options	KE8E EAE, KM8E memory extension	Integral FP11	Multiply/divide
Memory	4–32K words	16 KB–4 MB	4–32K words
Terminal	KL8E	DL11	KSR-33, Dasher
Paper tape	PC8E	PC11	Yes
Clock	DK8E	KW11L	Yes
Printer	LE8E	LP11	Yes
Storage	RX8E/RX01 RK8E/RK05 RF08/RS08	RX11/RX01 RK11/RK05 RLV11/RL01,2	4019 4046/4047, 4048, 4057, 4234
Magnetic tape	TM8E/TU10	TM11/TU10	6026

	PDP-4	PDP-7	PDP-9	PDP-15
CPU	PDP-4	PDP-7	PDP-9	PDP-15/30
Options		T177 EAE, T148 memory extension	KE09A EAE, KX09A memory protection KP09A power	KE15 EAE, KM15 memory protection KP15 power
Memory	4–8K words	4–32K words	4–32K words	4–128K words
Terminal	KSR-28	KSR-33	KSR-33	KSR-35
Paper tape	Integral T75 punch	T444 reader T75 punch	PC09A reader- punch	PC15 reader- punch
Clock	Yes	Yes	Yes	Yes
Printer	T62	T647	T647E	LP15
Storage		T24 drum	RF09/RS09	RF15/RS09 RP15/RP02
Magnetic tape			TC59/TU10	TC59/TU10

the hardware. In addition, Bill provided a working OS/8 system disk, and John copied several PDP-11 operating system disks off a working PDP-11/34. Megan Gentry was an important source of PDP-11 folklore, debugged some of the subtlest problems, created the Makefile, and provided the first and most frequently used distribution site. Ben Thomas provided the character-by-character I/O routines for VMS. Chris Suddick helped debug the PDP-11 floating-point code. Warren Toomey and the enthusiasts at PUPS (the PDP UNIX Preservation Society) in Australia allowed me access to their archive of early UNIX releases. Leendert Van Doorn debugged the PDP-11 simulator with UNIX V6, and Franc Grootjen with 2.11 BSD. Larry Stewart provided the initial impetus to the project, and Ken Harrenstein made an important contribution to preservation by implementing a DECsystem-10 simulator. Last, but not least, Max Burnet generously provided documentation and software from the Digital Australia collection, answered questions based on his 30 years of experience with Digital's systems, and made connections with and introductions to the worldwide community of historic machine hobbyists and enthusiasts.

References and Notes

1. As managing director of Digital's Australian subsidiary from 1975 to 1982, Max Burnet created and operated the PDP trade-in program.
2. M. Burnet, "An Update on the Museum Treasures," *DECUS Australia Symposium Proceedings*, August 1993.
3. M. Burnet, "The '94 Update on the Museum Treasures," *DECUS Australia Symposium Proceedings*, August 1994.
4. M. Burnet, "The Last Great PDP-11," *DECUS Australia Symposium Proceedings*, August 1995.

Table 9
Simulator Performance

Simulator	Simulated Instructions per Second	Real Instructions per Second	Ratio
PDP-8	1,800,000	400,000	4.5:1
PDP-11	440,000	500,000	.88:1
Nova	1,700,000	750,000	2.26:1

```
ucoder> pdp8

PDP-8 simulator V2.2b
sim> att rk0 os8.dsk
sim> boot rk0

.DA 08-APR-96

.DIR

 08-Apr-96

COPYIT.SV    2 09-Mar-93    PASS2 .SV   20 11-Oct-92    FORT3 .LD    3 06-Jul-93
DIRECT.SV    7 11-Oct-92    PASS20.SV    5 11-Oct-92    CLOSE .SV    2 10-Jul-93
CCLX  .SV   24 25-Feb-93    PASS3 .SV    8 11-Oct-92    FORT4 .FT    1 11-Jul-93
PIP   .SV   11 11-Oct-92    RALF  .SV   19 11-Oct-92    FORT4 .LD    2 04-Aug-93
FOTP  .SV    8 11-Oct-92    RESORC.SV   10 11-Oct-92    FORT6 .LD    2 09-Aug-93
ABSLDR.SV    5 11-Oct-92    RUNOFF.SV   24 11-Oct-92    FORT5 .FT    1 09-Aug-93
BASIC .SV   11 11-Oct-92    SABR  .SV   24 11-Oct-92    FORT5 .LD    2 09-Aug-93
BATCH .SV   10 11-Oct-92    SCROLL.SV   17 11-Oct-92    FORT6 .FT    1 09-Aug-93
BCOMP .SV   26 11-Oct-92    SET   .SV   20 11-Oct-92    METSC .SV   10 11-Aug-93
BITMAP.SV    5 11-Oct-92    SRCCOM.SV    5 11-Oct-92    METSC2.SV   10 11-Aug-93
BLOAD .SV   10 11-Oct-92    TECO  .SV   32 11-Oct-92    EMAT  .SV    9 11-Aug-93
BOOT  .SV    5 11-Oct-92    VERSN3.SV   10 11-Oct-92    EMDCT .SV   14 11-Aug-93
BRTS  .SV   24 11-Oct-92    BUILD .SV   33 11-Oct-92    EMTST .SV   10 11-Aug-93
CHEKMO.SV   15 11-Oct-92    BASIC .OV   16 11-Oct-92    SINST1.SV   14 11-Aug-93
COMPAF.SV    5 11-Oct-92    BUILD6.SV   33 11-Oct-92    ADDER .SV   13 11-Aug-93
CREF  .SV   13 11-Oct-92    BUILT .SV   33 12-Oct-92    FORT7 .FT    1 30-Aug-93
EDIT  .SV   10 11-Oct-92    HELP  .HE    1 18-Oct-92    CLEAR .LS    2 13-Jan-94
EDITS .SV    6 11-Oct-92    HELP  .HL   72 18-Oct-92    CLEAR .CF    2 13-Jan-94
EPIC  .SV   14 11-Oct-92    HELP  .OC    4 18-Oct-92    CLEAR .SV    2 13-Jan-94
F4    .SV   20 11-Oct-92    FORT7 .LD    2 07-Sep-93    CLEAR .PA    1 13-Jan-94
FRTS  .SV   26 11-Oct-92    JMPTST.SV    3 18-Oct-92    CLEAR .BN    2 13-Jan-94
FUTIL .SV   26 11-Oct-92    JMPJMS.SV    3 18-Oct-92    DEMO  .      28 21-Mar-95
HELP  .SV    5 11-Oct-92    RK8ENS.BN    1 30-Oct-92    DOS   .PA    4 25-Jan-94
LIBRA .SV   11 11-Oct-92    INST1 .SV   14 01-Dec-92    DOS   .BN    1 25-Jan-94
LIBSET.SV    5 11-Oct-92    INST2 .SV   11 01-Dec-92    DOS   .LS   10 25-Jan-94
LOAD  .SV   16 11-Oct-92    FORT  .FT    1 17-Jun-93    SHELL .PA    1 25-Jan-94
LOADER.SV   12 11-Oct-92    FORT  .LD    2 09-Jul-93    SHELL .BN    1 25-Jan-94
MATST .SV    9 11-Aug-93    FORT2 .LD    2 09-Jul-93    SHELL .LS    2 25-Jan-94
MDTST .SV   14 11-Aug-93    FORT2 .FT    1 22-Jun-93    BASIC .WS    1 10-Mar-94
OCOMP .SV    8 11-Oct-92    DOS   .SV    2 25-Jan-94    FOO   .PA    1 31-Mar-94
OPTF4 .SV   13 11-Oct-92    SHELL .SV    2 25-Jan-94    FOO   .BN    1 31-Mar-94
PAL8  .SV   19 11-Oct-92    FORT3 .FT    1 26-Jun-93

  95 Files In  980 Blocks - 2212 Free Blocks

.
Simulation stopped, PC: 01207 (KSF)
sim>
```

Figure 2
PDP-8 Simulator Running OS/8

```
ucoder> nova

NOVA simulator V2.2b
sim> att dp0 rdos.dsk
sim> set tti dasher
sim> boot dp0

Filename?

NOVA RDOS Rev 7.50
Date (m/d/y) ? 4 8 96
Time (h:m:s) ? 16 26 0

R
list/e sys-.-
SYS5.LB          17216  D        05/24/77 13:18  05/31/85  [001017]      0
SYS.SV           56320  SD       12/14/95 16:21  12/14/95  [005057]      0
SYS.LB           20240  D        04/30/85 14:49  05/31/85  [000746]      0
SYS.OL           30720  C        12/14/95 16:21  12/14/95  [005272]      0
SYSGEN.SV        23040  SD       05/02/85 22:20  05/31/85  [001401]      0
R
disk
LEFT: 2158   USED: 2706   MAX. CONTIGUOUS: 2054
R

Simulation stopped, PC: 41740 (LDA 1,4,3)
sim>
```

Figure 3
Nova Simulator Running RDOS

5. A. Ahi, G. Burroughs, A. Gore, S. LaMar, C.-Y. Lin, and A. Wiemann, "Design Verification of the HP 9000 Series 700 PA-RISC Workstations," *Hewlett-Packard Journal*, vol. 43, no. 4 (1992).

6. W. Anderson, "Logical Verification of the NVAX CPU Chip Design," *Digital Technical Journal*, vol. 4, no. 3 (1992): 38–46.

7. R. Calcagni and W. Sherwood, "VAX 6000 Model 400 CPU Chip Set Functional Design Verification," *Digital Technical Journal*, vol. 2, no. 2 (1990): 64–72.

8. A. Hutchings, "The Evolution of the Custom CAD Suite Used on the MicroVAX II System," *Digital Technical Journal*, vol. 1, no. 2 (1986): 48–55.

9. M. Kantrowitz and L. Noack, "Functional Verification of a Multiple-issue, Pipelined, Superscalar Alpha Processor—the Alpha 21164 CPU Chip," *Digital Technical Journal*, vol. 7, no. 1 (1995): 136–144.

10. D. Siewiorek, C. Bell, and A. Newell, *Computer Structures: Principles and Examples*, "The IBM System/360, System/370, 3030, and 4300: A Series of Planned Machines That Span a Wide Performance Range," and "PMS Notation" (New York: McGraw-Hill, 1982).

11. R. Brunner, ed., *VAX Architecture Reference Manual*, chapter 9, "Compatibility Mode" (Bedford, Mass.: Digital Press, 1991).

12. This simulator has since been withdrawn from the network.

13. R. Rustin, ed., *Debugging Techniques in Large Systems*. R. Supnik, "Debugging Under Simulation" (Englewood Cliffs, N. J.: Prentice-Hall, 1971).

14. For information on and pictures of Data General minicomputers, see C. Friend's web page at http://www.ultranet.com/~engelbrt/carl/museum/index.html.

15. For information on and pictures of many historic computers, see J. Jaeger's web page at http://www.msn.fullfeed.com/~cube/collect.htm.

16. For information on and pictures of many historic computers, see P. Pierce's web page at http://www.teleport.com/~prp/collect/index.html.

17. For documentation and relevant links, see D. Jones's web page at www.cs.uiowa.edu/~jones/pdp8/. For his simulator, cross assembler, and core images, see ftp://ftp.cs.uiowa.edu/pub/jones/pdp8.

18. For information on his simulator and OS/8 disk image, see W. Haygood's web page at ftp://sunsite.unc.edu/pub/academic/computer-science/history/pdp-8/emulators/haygood.

19. For more information on J. Wilson's simulator (executable only), see his web page at ftp://ftp.update.uu.se/pub/ibmpc/emulators.

20. For more information on the PDP-11 UNIX archive, see the PUPS home page at http://minnie.cs.adfa.oz.au/PUPS/index.html.

```
ucoder> pdp11

PDP-11 simulator V2.2b
sim> att rk0 rtrk.dsk
sim> boot rk0

RT-11SJ (S) V05.04

.

.da 8-apr-96

.dir
 08-Apr-96
NL     .SYS      2  18-Sep-89       RT11FB.SYS     94  18-Sep-89
RT11SJ.SYS     80  18-Sep-89       SPOOL .REL     11  14-Apr-87
PTESTX.MAC     23  27-Jan-94       GVI   .SAV      5  18-Apr-90
BINCOM.SAV     24  27-Sep-88       DUP   .SAV     49  27-Sep-88
DIR    .SAV     19  27-Sep-88       IND   .SAV     58  27-Sep-88
LIBR   .SAV     24  27-Sep-88       MACRO .SAV     61  27-Sep-88
LINK   .SAV     49  27-Sep-88       RESORC.SAV     25  27-Sep-88
FORMAT.SAV     24  27-Sep-88       ODT   .SAV      8  05-Oct-89
PBCOPY.SAV      2  16-Feb-89       SYSLIB.OBJ    55P 05-Oct-89
ODT    .OBJ      8  05-Oct-89       SYSMAC.SML     61  16-Mar-89
SIPP   .SAV     21  27-Sep-88       DATE  .SAV      3  02-Feb-89
IOP    .SAV     11  24-Apr-89       SWAP  .SYS     27  27-Sep-88
TT     .SYS      2  18-Sep-89       DL    .SYS      4  18-Sep-89
DM     .SYS      5  18-Sep-89       DP    .SYS      3  18-Sep-89
DX     .SYS      4  18-Sep-89       RK    .SYS      3  18-Sep-89
LS     .SYS      5  05-Oct-89       MT    .SYS      9  18-Sep-89
LP     .SYS      2  18-Sep-89       SP    .SYS      6  18-Sep-89
PIP    .SAV     30  27-Sep-88       HANDLE.SAV      7  16-Feb-89
LD     .SYS      8  26-Dec-90       MAC   .SAV     61  27-Sep-88
LC     .SYS      2  01-Jan-80       UCL   .SAV     13  22-Dec-89
UCL    .CCL      4  07-Oct-90       STARTS.COM      1  19-Jan-94
MTPIP  .SAV     28  27-Feb-87       MTROL .SAV     17  27-Feb-87
MLIB   .SYS    300  20-Dec-90       HELP  .SAV    132  20-Dec-90
XPC    .SAV     16  25-Jun-91       DESS  .SAV     18  09-Mar-88
PTESTX.OBJ      8
 49 Files, 1432 Blocks
 3330 Free blocks

.sho dev

Device     Status          CSR      Vector(s)
------     ------          ---      ---------
  NL       Installed      000000    000
  TT       Installed      000000    000
  DL       Installed      174400    160
  DM       Not installed  177440    210
  DP       Not installed  176710    254
  DX       Installed      177170    264
  RK       Resident       177400    220
  LS      -Not installed  176500    470 474 300 304
  MT       Installed      172520    224
  LP       Installed      177514    200
  SP       Installed      000000    110
  LD       Installed      000000    000
  LC       Installed      177514    200

.
Simulation stopped, PC: 146506 (ASR R5)
sim>
```

Figure 4
PDP-11 Simulator Running RT-11

Biographies

Maxwell M. Burnet
Max Burnet has been with Digital in Australia for 29 years. During that time, he has sold, serviced, or marketed all the machines in the collection. He managed the Digital Australia subsidiary for seven years. He was a salesman in Boston during 1971 and managed to replace an IBM 1620 at Tufts University with a PDP-10. He is currently the oldest surviving "techie" in the Sydney office and makes many corporate presentations in Australia. He manages the Australian DECUS Society, the Subsidiary's local content and export obligations with the Australian Government, and the local Product Assurance Group. He has collected a museum of early Digital machines and is known around Sydney as "Museum Max." He received a B.Sc. (honours) from Melbourne University.

Robert M. Supnik
Bob Supnik has been with Digital in the United States for 19 years. He joined the Mass Storage Group and then moved into Semiconductor Engineering, where he successively managed the last PDP-11 implementation (the J-11), Advanced Development, the first single-chip VAX implementation (the MicroVAX chip), and the VAX Microprocessor Group. He also wrote or contributed to the microcode of every single-chip VAX microprocessor. In 1988, he started the Alpha program, which he managed through launch of the first products in 1992. He then became technical director, first of Engineering and then of the Computer Systems Division. In 1996, he became vice president of Research and Advanced Development. He has B.A. degrees in mathematics and in history from MIT, and an M.A. in history from Brandeis University.

William N. Celmaster

Modern Fortran Revived as the Language of Scientific Parallel Computing

New features of Fortran are changing the way in which scientists are writing and maintaining large analytic codes. Further, a number of these new features make it easier for compilers to generate highly optimized architecture-specific codes. Among the most exciting kinds of architecture-specific optimizations are those having to do with parallelism. This paper describes Fortran 90 and the standardized language extensions for both shared-memory and distributed-memory parallelism. In particular, three case studies are examined, showing how the distributed-memory extensions (High Performance Fortran) are used both for data parallel algorithms and for single-program–multiple-data algorithms.

A Brief History of Fortran

The Fortran (FORmula TRANslating) computer language was the result of a project begun by John Backus at IBM in 1954. The goal of this project was to provide a way for programmers to express mathematical formulas through a formalism that computers could translate into machine instructions. Initially there was a great deal of skepticism about the efficacy of such a scheme. "How," the scientists asked, "would anyone be able to tolerate the inefficiencies that would result from compiled code?" But, as it turned out, the first compilers were surprisingly good, and programmers were able, for the first time, to express mathematics in a high-level computer language.

Fortran has evolved continually over the years in response to the needs of users, particularly in the areas of mathematical expressivity, program maintainability, hardware control (such as I/O), and, of course, code optimizations. In the meantime, other languages such as C and C++ have been designed to better meet the nonmathematical aspects of software design, such as graphical interfaces and complex logical layouts. These languages have caught on and have gradually begun to erode the scientific/engineering Fortran code base.

By the 1980s, pronouncements of the "death of Fortran" prompted language designers to propose extensions to Fortran that would incorporate the best features of other high-level languages and, in addition, provide new levels of mathematical expressivity popular on supercomputers such as the CYBER 205 and the CRAY systems. This language became standardized as Fortran 90 (ISO/IEC 1539: 1991; ANSI X3.198-1992). At the present time, Fortran 95, which includes many of the parallelization features of High Performance Fortran discussed later in this paper, is in the final stages of standardization. It is not yet clear whether the modernization of Fortran can, of itself, stem the C tide. However, I will demonstrate in this paper that modern Fortran is a viable mainstream language for parallelism. It is true that parallelism is not yet part of the scientific programming mainstream. However, it seems likely that, with the scientists' never-ending thirst for affordable performance, parallelism will become much more common—especially

now that appropriate standards have evolved. Just as early Fortran enabled average scientists and engineers to program the computers of the 1960s, modern Fortran may enable average scientists and engineers to program parallel computers of the next decade.

An Introduction to Fortran 90

Fortran 90 introduces some important capabilities in mathematical expressivity through a wealth of natural constructs for manipulating arrays.[1] In addition, Fortran 90 incorporates modern control constructs and up-to-date features for data abstraction and data hiding. Some of these constructs, for example, DO WHILE, although not part of FORTRAN 77, are already part of the de facto Fortran standard as provided, for example, with DEC Fortran.

Among the key new features of Fortran 90 are the following:

- Inclusion of all of FORTRAN 77, so users can compile their FORTRAN 77 codes without modification
- Permissibility of free-form source code, so programmers can use long (i.e., meaningful) variable names and are not restricted to begin statements in column 7
- Modern control structures like CASE and DO WHILE, so programmers can take advantage of structured programming constructs
- Extended control of numeric precision, for architecture independence
- Array processing extensions, for more easily expressing array operations and also for expressing independence of element operations
- Pointers, for more flexible control of data placement
- Data structures, for data abstraction
- User-defined types and operators, for data abstraction
- Procedures and modules, to help programmers write reusable code
- Stream character-oriented input/output features
- New intrinsic functions

With these new features, a modern Fortran programmer can not only successfully compile and execute previous standards-compliant Fortran codes but also design better codes with

- Dramatically simplified ways of doing dynamic memory management
- Dynamic memory allocation and deallocation for memory management
- Better modularity and therefore reusability
- Better readability
- Easier program maintenance

Additionally, of course, programmers have the assurance of complete portability between platforms andarchitectures.

The following code fragment illustrates the simplicity of dynamic memory allocation with Fortran 90. It also includes some of the new syntax for declaring variables, some examples of array manipulations, and an example of how to use the new intrinsic matrix multiplication function. In addition, the exclamation mark, which is used to begin comment statements, is a new Fortran 90 feature that was widely used in the past as an extension to FORTRAN 77.

```
  REAL, DIMENSION(:,:,:),      ! NEW DECLARATION SYNTAX
&  ALLOCATABLE :: GRID         ! DYNAMIC STORAGE
  REAL*8 A(4,4),B(4,4),C(4,4) ! OLD DECLARATION SYNTAX
  READ *, N                    ! READ IN THE DIMENSION
  ALLOCATE(GRID(N+2,N+2,2))    ! ALLOCATE THE STORAGE
  GRID(:,:,1) = 1.0            ! ASSIGN PART OF ARRAY
  GRID(:,:,2) = 2.0            ! ASSIGN REST OF ARRAY
  A = GRID(1:4,1:4,1)          ! ASSIGNMENT
  B = GRID(2:5,1:4,2)          ! ASSIGNMENT
  C = MATMUL(A,B)              ! MATRIX MULTIPLICATION
```

Some of the new features of Fortran 90 were introduced not only for simplified programming but also to permit better hardware-specific optimizations. For example, in Fortran 90, one can write the array assignment

```
A = B + C
```

which in FORTRAN 77 would be written as

```
      DO 100 J = 1,N
        DO 200 I = 1,M
              A(I,J) = B(I,J) + C(I,J)
200     END DO
100   END DO
```

The Fortran 90 array assignment not only is more elegant but also permits the compiler to easily recognize that the individual element assignments are independent of one another. If the compiler were targeting a vector or parallel computer, it could generate code that exploits the architecture by taking advantage of this independence between iterations.

Of course, the particular DO loop shown above is simple enough that many compilers would recognize the independence of iterations and could therefore perform the architecture-specific optimizations without the aid of Fortran 90's new array constructs. But in general, many of the new features of Fortran 90 help compilers to perform architecture-specific optimizations. More important, these features help programmers express basic numerical algorithms in ways inherently more amenable to optimizations that take advantage of multiple arithmetic units.

A Brief History of Parallel Fortran: PCF and HPF

During the past ten years, two significant efforts have been undertaken to standardize parallel extensions to Fortran. The first of these was under the auspices of the Parallel Computing Forum (PCF) and targeted global-shared-memory architectures. The PCF effort was directed to control parallelism, with little attention to language features for managing data locality. The 1991 PCF standard established an approach to shared-memory extensions of Fortran and also established an interim syntax. These extensions were later somewhat modified and incorporated in the standard extensions now known as ANSI X3H5.

At about the time the ANSI X3H5 standard was adopted, another standardization committee began work on extending Fortran 90 for distributed-memory architectures, with the goal of providing a language suitable for scalable computing. This committee became known as the High Performance Fortran Forum and produced in 1993 the High Performance Fortran (HPF) language specification.[3] The HPF programming-model target was data parallelism, and many data placement directives are provided for the programmer to optimize data locality. In addition, HPF includes ways to specify a more general style of single-program–multiple-data (SPMD) execution in which separate processors can independently work on different parts of the code. This SPMD specification is formalized in such a way as to make the resulting code far more maintainable than previous message-passing-library ways of specifying SPMD distributed parallelism.

Can HPF and PCF extensions be used together in the same Fortran 90 code? Sure. But the PCF specification has lots of "user-beware" warnings about the correct usage of the PARALLEL REGION construct, and the HPF specification has lots of warnings about the correct usage of the EXTRINSIC(HPF_LOCAL) construct. So as you can see, there are times when a programmer had better be very knowledgeable if she or he wants to write a mixed HPF/PCF code. Digital's products support both the PCF and HPF extensions. The HPF extensions are supported as part of the DEC Fortran 90 compiler, and the PCF extensions are supported through Digital's KAP Fortran optimizer.[3,4]

Shared Memory Fortran Parallelism

The traditional discussions of parallel computing focus rather heavily on what is known as *control parallelism.* Namely, the application is analyzed in terms of the opportunities for parallel execution of various threads of control. The canonical example is a DO loop in which the individual iterations operate on independent data. Each iteration could, in principle, be executed simultaneously (provided of course that the hardware allows simultaneous access to instructions and data). Technology has evolved to the point at which compilers are often able to detect these kinds of parallelization opportunities and automatically decompose codes. Even when the compiler is not able to make this analysis, the programmer often is able to do so, perhaps after performing a few algorithmic modifications. It is then relatively easy to provide language constructs that the user can add to the program as parallelization hints to the compiler.

This kind of analysis is all well and good, provided that data can be accessed democratically and quickly by all processors. With modern hardware clocked at about 300 megahertz, this amounts to saying that memory latencies are lower than 100 nanoseconds, and memory bandwidths are greater than 100 megabytes per second. This characterizes today's single and symmetric multiprocessing (SMP) computers such as Digital's AlphaServer 8400 system, which comes with twelve 600-megaflop processors on a backplane with a bandwidth of close to 2 gigabytes per second.

In summary, the beauty of shared-memory parallelism is that the programmer does not need to worry too much about where the data is and can concentrate instead on the easier problem of control parallelism. In the simplest cases, the compiler can automatically decompose the problem without requiring any code modifications. For example, automatic decomposition for SMP systems of a code called, for example, cfd.f, can be done trivially with Digital's KAP optimizer by using the command line

```
kf90 -fkapargs='-conc' cfd.f -o cfd.exe
```

As an example of guided automatic decomposition, the following shows how a KAP parallelization assertion can be included in the code. (Actually, the code segment below is so simple that the compiler can automatically detect the parallelism without the help of the assertion.)

```
C*$*    ASSERT DO (CONCURRENT)
        DO 100 I = 4,N
           A(I) = B(I) + C(I)
        END DO
```

For explicit control of the parallelism, PCF directives can be used. In the example that follows, the KAP preprocessor form of the PCF directives are used to parallelize a loop.

```
C*KAP*PARALLEL REGION
C*KAP*&SHARED(A,B,C) LOCAL(I)
C*KAP*PARALLEL DO
      DO 10 I = 1,N
         A(I) = B(I) + C(I)
10    CONTINUE
C*KAP*END PARALLEL REGION
```

Cluster Fortran Parallelism

High Performance Fortran V1.1 is currently the only language standard for distributed-memory parallel computing. The most significant way in which HPF extends Fortran 90 is through a rich family of data placement directives. There are also library routines and some extensions for control parallelism. HPF is the simplest way of parallelizing data-parallel applications on clusters (also known as "farms") of workstations and servers. Other methods of cluster parallelism, such as message passing, require more bookkeeping and are therefore less easy to express and less easy to maintain. In addition, during the past year, HPF has become widely available and is supported on the platforms of all major vendors.

HPF is often considered to be a *data parallel* language. That is, it facilitates parallelization of array-based algorithms in which the instruction stream can be described as a sequence of array manipulations, each of which is inherently parallel. What is less well known is that HPF also provides a powerful way of expressing the more general SPMD parallelism mentioned earlier. This kind of parallelism, often expressed with message-passing libraries such as MPI,[5] is one in which individual processors can operate simultaneously on independent instruction streams and generally exchange data either by explicitly sharing memory or by exchanging messages. Three case studies follow which illustrate the data parallel and the SPMD styles of programming.

A One-dimensional Finite-difference Algorithm

Consider a simple one-dimensional grid problem—the most mind-bogglingly simple illustration of HPF in action—in which each grid value is updated as a linear combination of its (previous) nearest neighbors.

For each interior grid index i, the update algorithm is

$$Y(i) = X(i-1) + X(i+1) - 2 \times X(i)$$

In Fortran 90, the resulting DO loop can be expressed as a single array assignment. How would this be parallelized? The simplest way to imagine parallelization would be to partition the X and Y arrays into equal-size chunks, with one chunk on each processor. Each iteration could proceed simultaneously, and at the chunk boundaries, some communication would occur between processors. The HPF implementation of this idea is simply to add the Fortran 90 code to two data placement statements. One of these declares that the X array should be distributed into chunks, or blocks. The other declares that the Y array should be distributed such that the elements align to the same processors as the corresponding elements of the X array. The resultant code for arrays with 1,000 elements is as follows:

```
!HPF$ DISTRIBUTE X(BLOCK)
!HPF$ ALIGN Y WITH X
      REAL*8 X(1000), Y(1000)

      <initialize x>

      Y(2:999) = X(1:998) + X(3:1000) - 2 * X(2:999)

      <check the answer>
      END
```

The HPF compiler is responsible for generating all of the boundary-element communication code. The compiler is also responsible for determining the most even distribution of arrays. (If, for example, there were 13 processors, some chunks would be bigger than others.)

This simple example is useful not only as an illustration of the power of HPF but also as a way of pointing to one of the hazards of parallel algorithm development. Each of the element-updates involves three floating-point operations—an addition, a subtraction, and a multiplication. So, as an example, on a four-processor system, each processor would operate on 250 elements with 750 floating-point operations. In addition, each processor would be required to communicate one word of data for each of the two chunk boundaries. The time that each of these communications takes is known as the communications latency. Typical transmission control protocol/internet protocol (TCP/IP) network latencies are twenty thousand times (or more) longer than the time it typically takes a high-performance system to perform a floating-point operation. Thus even 750 floating-point operations are negligible compared with the time taken to communicate. In the above example, network parallelism would be a net loss, since the total execution time would be totally swamped by the network latency.

Of course, some communication mechanisms are of lower latency than TCP/IP networks. As an example, Digital's implementation of MEMORY CHANNEL cluster interconnect reduces the latency to less than 1000 floating-point operations (relative to the performance of, say, Digital's AlphaStation 600 5/300 system). For SMP, the latency is even smaller. In both cases, there may be a benefit to parallelism.

A Three-dimensional Red–Black Poisson Equation Solver

The example of a one-dimensional algorithm in the previous section can be easily generalized to a more realistic three-dimensional algorithm for solving the Poisson equation using a relaxation technique commonly known as the red–black method. The grid is partitioned into two colors, following a two-dimensional checkerboard arrangement. Each red grid element is updated based on the values of neighboring black elements. A similar array assignment can

be written as in the previous example or, as shown in the partial code segment below, alternatively can use the HPF FORALL construct to express the assignments in a style similar to that for serial DO loops.

```
!HPF$ DISTRIBUTE(*,BLOCK,BLOCK) :: U,V
      <other stuff>
      FORALL (I=2:NX-1,J=2:NY-1:2,K=2:NZ-1:2)
         U(I,J,K) = FACTOR*(HSQ*F(I,J,K) +     &
                U(I-1,J,K) + U(I+1,J,K)   +     &
```

The distribution directive lays out the array so that the first dimension is completely contained within a processor, with the other two dimensions block-distributed across processors in rectangular chunks. The red–black checkerboarding is performed along the second and third dimensions. Note also the Fortran 90 free-form syntax employed here, in which the ampersand is used as an end-of- line continuation statement.

In this example, the parallelism is similar to that of the one-dimensional finite-difference example. However, communication now occurs along the two-dimensional boundaries between blocks. The HPF compiler is responsible for these communications. Digital's Fortran 90 compiler performs several optimizations of those communications. First, it packages up all of the data that must be communicated into long vectors so that the start-up latency is effectively hidden. Second, the compiler creates so-called shadow edges (processor-local copies of nonlocal boundary edges) for the local arrays so as to minimize the effect of buffering of neighbor values. These kinds of optimizations can be extremely tedious to message-passing programmers, and one of the virtues of a high-level language like HPF is that the compiler can take care of the bookkeeping. Also, since the compiler can reliably do buffer-management bookkeeping (for example, ensuring that communication buffers do not overflow), the communications runtime library can be optimized to a far greater extent than one would normally expect from a user-safe message library. Indeed, Digital's HPF communications are performed using a proprietary optimized communications library, Digital's Parallel Software Environment.[6]

Communications and SPMD Programming with HPF

Since HPF can be used to place data, it stands to reason that communication can be forced between processors. The beauty of HPF is that all of this can be done in the context of mathematics rather than in the context of distributed parallel programming. The code fragment in Figure 1 illustrates how this is done.

On two processors, the two columns of the U and V arrays are each on different processors; thus the array assignment causes one of those columns to be moved to the other processor. This kind of an operation begins to provide programmers with explicit ways to control data communication and therefore to more explicitly manage the association of data and operations to processors. Notice that the programmer need not be explicit about the parallelism. In fact, scientists and engineers rarely want to express parallelism. In typical message-passing programs, the messages often express communication of vector and array information.

However, despite the fervent hopes of programmers, there are times when a parallel algorithm can be expressed most simply as a collection of individual instruction streams operating on local data. This SPMD style of programming can be expressed in HPF with the EXTRINSIC(HPF_LOCAL) declaration, as illustrated by continuing the above code segment as shown in Figure 2.

Because the subroutine CFD is declared to be EXTRINSIC(HPF_LOCAL), the HPF compiler executes that routine independently on each processor (or more generally, the execution is done once per *peer* process), operating on routine-local data. As for the array argument, V, which is passed to the CFD routine, each processor operates only on its local slice of that array. In the specific example above on two processors, the first one operates on the first column of V and the second one operates on the second column of V.

It is important to mention here that, although HPF permits—and even encourages—SPMD programming, the more popular method at this time is the message-passing technique embodied in, for example, the PVM[7] and MPI[8] libraries. These libraries can be invoked from Fortran, and can also be used in conjunction with EXTRINSIC(HPF_LOCAL) subroutines.

```
!HPF$ DISTRIBUTE(*,BLOCK) :: U
!HPF$ ALIGN V WITH U
      REAL*8 U(N,2),V(N,2)
      <initialize arrays>
      V(:,1) = U(:,2)        ! MOVE A VECTOR BETWEEN PROCESSORS
```

Figure 1
Code Example Showing Control of Data Communication without Expression of Parallelism

```
      CALL CFD(V)       ! DO LOCAL WORK ON THE LOCAL PART OF V
      <finish the main program>

      EXTRINSIC(HPF_LOCAL) SUBROUTINE CFD(VLOCAL)
      REAL*8, DIMENSION(:,:)   :: VLOCAL
!HPF$ DISTRIBUTE *(*,BLOCK)    :: VLOCAL
      <do arbitrarily complex work with vlocal>
      END
```

Figure 2
Code Example of Parallel Algorithm Expressed as Collection of Instruction Streams

Clusters of SMP Systems

During these last few years of the second millennium, we are witnessing the emergence of systems that consist of clusters of shared-memory SMP computers. This exciting development is the logical result of the exponential increase in performance of mid-priced ($100K to $1000K) systems.

There are two natural ways of writing parallel Fortran programs for clusters of SMP systems. The easiest way is to use HPF and to target the total number of processors. So, for example, if there were two SMP systems, each with four processors, one would compile the HPF program for eight processors (more generally, for eight peers). If the program contained, for instance, block-distribution directives, the affected arrays would be split up into eight chunks of contiguous array sections.

The second way of writing parallel Fortran programs for clustered SMP systems is to use HPF to target the total number of SMP machines and then to use PCF (or more generally, shared-memory extensions) to achieve parallelism locally on each of the SMP machines. For example, one might write

```
!HPF$ DISTRIBUTE (*,BLOCK) :: V
      <stuff>
      EXTRINSIC(HPF_LOCAL) SUBROUTINE CFD(V)
      <stuff>
C*KAP*PARALLEL REGION
```

If the target system consisted of two SMP systems, each with four processors, and the above program was compiled for two peers, then the V array would be distributed into two chunks of columns—one chunk per SMP system. Then the routine, CFD, would be executed once per SMP system; and the PCF directives would, on each system, cause parallelism on four threads of execution.

It is unclear at this time whether there would ever be a practical reason for using a mix of HPF and PCF extensions. It might be tempting to think that there would be performance advantages associated with the local use of shared-memory parallelism. However, experience has shown that program performance tends to be restricted by the weakest link in the performance chain (an observation that has been enshrined as "Amdahl's Law"). In the case of clustered SMP systems, the weak link would be the inter-SMP communication and not the intra-SMP (shared-memory) communication. This casts some doubt on the worth of local communications optimizations. Experimentation will be necessary.

Whatever else one might say about parallelism, one thing is certain: The future will not be boring.

Summary

Fortran was developed and has continued to evolve as a computer language that is particularly suited to expressing mathematical formulas. Among the recent extensions to Fortran are a variety of constructs for the high-level manipulation of arrays. These constructs are especially amenable to parallel optimization. In addition, there are extensions (PCF) for explicit shared-memory parallelization and also data-parallel extensions (HPF) for cluster parallelism. The Digital Fortran compiler performs many interesting optimizations of codes written using HPF. These HPF codes are able to hide—without sacrificing performance—much of the tedium that otherwise accompanies cluster programming. Today, the most exciting frontier for Fortran is that of SMP clusters and other nonuniform-memory-access (NUMA) systems.

References

1. J. Adams et al., *Fortran 90 Handbook* (New York: McGraw-Hill, Inc., 1992).
2. "High Performance Fortran language specification," *Scientific Programming*, vol. 2: 1–170 (New York: John Wiley and Sons, Inc., 1993), and C. Koelbel et al., *The High Performance Fortran Handbook* (Boston: MIT Press, 1994).
3. J. Harris et al., "Compiling High Performance Fortran for Distributed-memory Systems," *Digital Technical Journal*, vol. 7, no. 3 (1995): 5–23.
4. R. Kuhn, B. Leasure, and S. Shah, "The KAP Parallelizer for DEC Fortran and DEC C Programs," *Digital Technical Journal*, vol. 6, no. 3 (1994): 57–70.

5. For example see *Proceedings of Supercomputing '93* (IEEE, November 1993): 878–883, and W. Gropp, E. Lusk, and A. Skjellum, *Using MPI* (Boston: MIT Press, 1994).

6. E. Benson et al., "Design of Digital's Parallel Software Environment," *Digital Technical Journal*, vol. 7, no. 3 (1995): 24–38.

7. For example see A. Geist et al., *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Network Parallel Computing* (Boston: MIT Press, 1994).

Biography

William N. Celmaster

Bill Celmaster has long been involved with high-performance computing, both as a scientist and as a computing consultant. Joining Digital from BBN in 1991, Bill managed the porting of major scientific and engineering applications to the DECmpp 12000 system. Now a member of Digital's High Performance Computing Expertise Center, he is responsible for parallel software demonstrations and performance characterization of Digital's high-performance systems. He has published numerous papers on parallel computing methods, as well as on topics in the field of physics. Bill received a bachelor of science degree in mathematics and physics from the University of British Columbia and a Ph.D. in physics from Harvard University.

Performance Measurement of TruCluster Systems under the TPC-C Benchmark

Judith A. Piantedosi
Archana S. Sathaye
D. John Shakshober

Digital Equipment Corporation and Oracle Corporation have announced a new TPC-C performance record in the competitive market for database applications and UNIX servers on the AlphaServer 8400 5/350 four-node TruCluster system. A performance evaluation strategy enabled Digital to achieve record-setting performance for this TruCluster configuration supporting the Oracle Parallel Server database application under the TPC-C workload. The system performance in this environment is a result of tuning the system under test and taking advantage of TruCluster features such as the MEMORY CHANNEL interconnect and Digital's distributed lock manager and distributed raw disk service.

Current industry trends have moved from centralized computing offered by uniprocessors and symmetric multiprocessing (SMP) systems to multinode, highly available and scalable systems, called clusters. The TruCluster multicomputer system for the Digital UNIX environment is the latest cluster product from Digital Equipment Corporation.[1] In this paper, we discuss our test and results on a four-node AlphaServer 8400 5/350 TruCluster configuration supporting the Oracle Parallel Server database application. We evaluate this system under the Transaction Processing Performance Council's TPC-C benchmark to provide performance results in the competitive market for database applications.

The TPC-C benchmark is a medium-complexity, on-line transaction processing (OLTP) workload.[2,3] It is based on an order-entry workload, with different transaction types ranging from simple transactions to medium-complexity transactions that have 2 to 50 times the number of calls of a simple transaction.[4] To run the TPC-C benchmark on a clustered system, the operating system and the database engine must present a single database to the benchmark client. Thus the TruCluster system running the Oracle Parallel Server differs greatly from a network-based cluster system by two significant features. First, the Digital UNIX distributed raw disk (DRD) service enables the distributed Oracle Parallel Server to access all raw disk volumes regardless of their physical location in the cluster. Second, the Oracle Parallel Server uses Digital's distributed lock manager (DLM) to synchronize all access to shared resources (such as in memory cache blocks or disk blocks) across a TruCluster system.

In tuning the system under test, we used the DRD and the DLM services to balance the database across the TruCluster multicomputer system. The configuration includes a specialized peripheral component interconnect (PCI) known as the MEMORY CHANNEL interconnect to greatly improve the bandwidth and latency between two or more member nodes.[5] We tuned the system under test to attain the peak bandwidth of 100 megabytes per second (MB/s) for heavy internode communication during checkpointing by using a dedicated PCI bus for the MEMORY CHANNEL interconnect. We also tuned

the system under test to use the very large memory technology and trade off memory for the database cache with memory for DLM locks to improve the throughput. (For a discussion of this technology, see the section Performance Evaluation Methodology.) We measured the maximum throughput, the 90th percentile response time for each transaction type, and the keying and think times. Finally, we compared our measured throughput and price/performance with competitive vendors like Tandem Computers and Hewlett-Packard Company.

The rest of the paper is organized as follows. In the next section, we provide a synopsis of the TruCluster technology and introduce the Oracle Parallel Server, an optional Oracle product that enables the user to use TruCluster technology with the Oracle relational database management system. Following that, we give an overview of the TPC-C benchmark. Next, we describe the system under test and our performance evaluation methodology. Then we discuss our performance measurement results and compare them with competitive vendor results. Finally, we present our concluding remarks and discuss our future work.

TruCluster Clustering Technology

Digital's TruCluster configuration consists of interconnected computers (uniprocessors or SMPs) and external disks connected to one or more shared, small computer systems interface (SCSI) buses providing services to clients.[6] It presents a single raw volume namespace to a client with better application availability than a single system and better scalability than an SMP. A TruCluster configuration supports highly parallelized database managers, such as the Oracle Parallel Server, to provide incremental performance scaling of at least 80 percent for transaction processing applications. The underlying technology to provide this incremental growth includes a PCI-based MEMORY CHANNEL interconnect for communication between cluster members.[6] The MEMORY CHANNEL interconnect provides a 100-MB/s, memory-mapped connection between cluster members.[7] The cluster members map transfers from the MEMORY CHANNEL interconnect into their memory using standard memory access instructions. The use of memory store instructions rather than special I/O instructions provides low latency (two microseconds) and low overhead for a transfer of any length.[7]

The TruCluster for Digital UNIX product supports up to eight (four for commercial DLM/DRD-based applications) cluster members connected to a common cluster interconnect. The computer systems supported within a cluster are AlphaServer systems of varying processor speed and number of processors. The member systems run applications (for example, user applications), as well as monitor the state of each member system, each shared disk, the MEMORY CHANNEL interconnect, and the network. These cluster members communicate over the MEMORY CHANNEL interconnect.[6,8] A MEMORY CHANNEL configuration consists of a MEMORY CHANNEL adapter installed in a PCI slot and link cables to connect the adapters. In a configuration with more than two members, the MEMORY CHANNEL adapters are connected to a MEMORY CHANNEL hub. A typical TruCluster configuration with a MEMORY CHANNEL hub is shown in Figure 1.

Applications can attain high availability by connecting two or more member systems to one or more

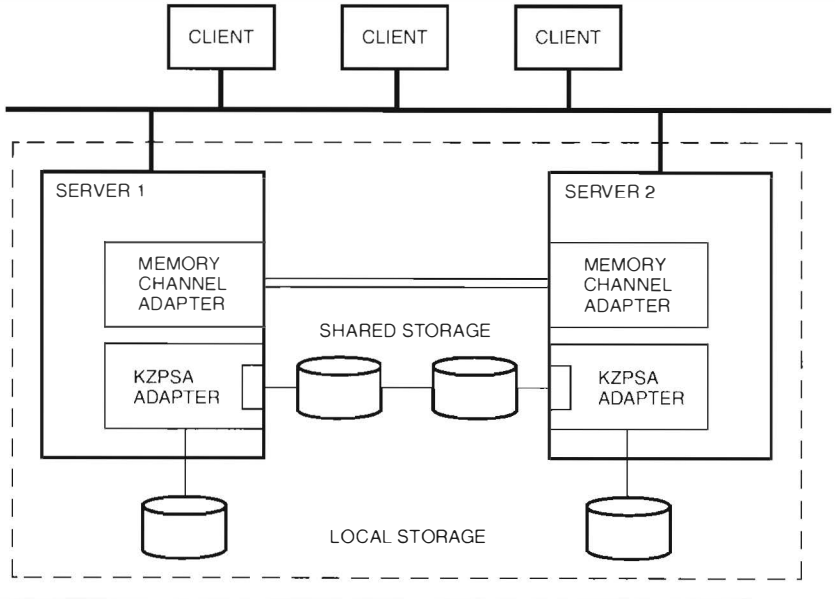


Figure 1
A TruCluster Configuration with MEMORY CHANNEL Hub

shared SCSI buses, thus constructing an Available Server Environment (ASE). A shared SCSI bus is required only for two-member configurations that do not have a MEMORY CHANNEL hub. Although MEMORY CHANNEL is the only supported cluster interconnect, Ethernet and fiber distributed data interface (FDDI) are supported for connecting clients to cluster members. Disks are connected either locally (i.e., nonshared) to a SCSI bus or to a shared SCSI bus between two or more member systems. A single node in the cluster is used to serve the disk to other cluster members. Disks on local buses obviously become unavailable upon failure of the server node. The SCSI controller supported in this configuration is the PCI disk adapter, KZPSA.

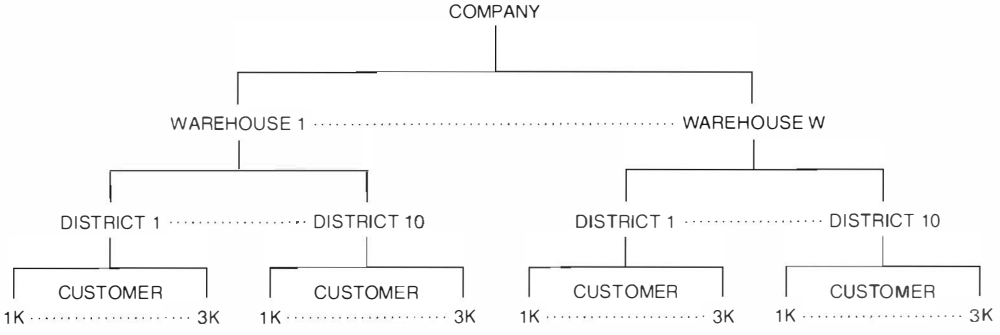
The distinguishing feature of the TruCluster software is its support of the MEMORY CHANNEL as a cluster interconnect, thus providing industry-leadership performance to intracluster communication.[9] The TruCluster software includes the following components: the DLM, the connection manager, the DRD, and the cluster communication service. The DLM facilitates synchronization to shared resources to all member systems in a cluster by means of a run-time library. Cooperating processes use the DLM to synchronize access to a shared resource, a DRD device, a file, or a program. The DLM service is primarily used by the Oracle Parallel Server to coordinate access to the cache and shared disks that have the database installed.[6] The connection manager maintains information about the cluster configuration and maintains a communication path between each cluster member for use by the DLM. The DLM uses this configuration data and other connection manager services to maintain a distributed lock database. The DRD allows the exporting of clusterwide raw devices. This allows disk-based user-level applications to run within the cluster, regardless of where in the cluster the actual physical storage resides. Therefore a DRD service allows the Oracle Parallel Server parallel access to storage media from multiple cluster members. The cluster communication service is used to allocate the MEMORY CHANNEL address space and map it to the processor main memory.

TPC-C Benchmark

The TPC-C benchmark depicts the activity of a generic wholesale supplier company. The hierarchy in the TPC-C business environment is shown in Figure 2. The company consists of a number of geographically distributed sales districts and associated warehouses. Further, there are 10 districts under each warehouse with each district serving 3,000 (3K) customers. All the warehouses maintain a stock of 10,000 items sold by the company. As the company grows, new warehouses and associated sales districts are created. The business activity consists of customer calls to place new orders or request the status of existing orders, payment entries, processing orders for delivery, and stock-level examination. The orders on an average are composed of 10 order lines (i.e., line items). Ninety-nine percent of all orders can be met by a local warehouse, and only one percent of them need to be sold by a remote warehouse.

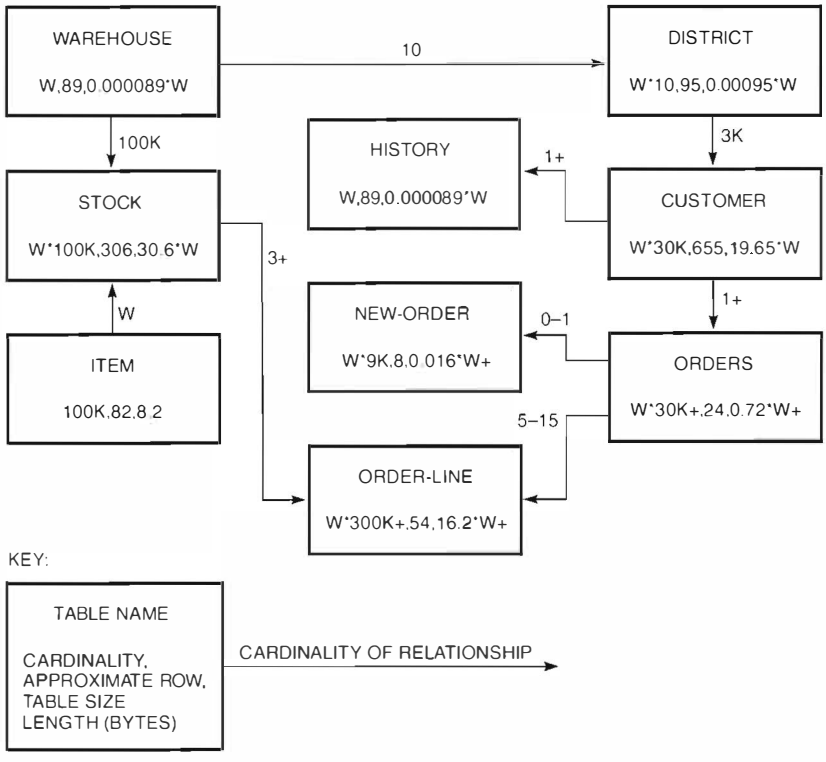
The TPC-C logical database components consist of nine tables.[3] Figure 3 shows the relationship between these tables, the cardinality of the tables (i.e., the number of rows), and the cardinality of the relationships. The figure also shows the approximate row length in bytes for each table and the table size in megabytes. The cardinality of all the tables, except the item table, grows with the number of warehouses. The order, order-line, and history tables grow indefinitely as the orders are processed.

The five types of TPC-C transactions are listed in Table 1.[3] The new-order transaction places an order (of 10 order lines) from a warehouse through a single database transaction; it inserts the order and updates the corresponding stock level for each item. Ninety-nine percent of the time the supplying warehouse is



Source: Transaction Processing Performance Council, *TPC Benchmark C Standard Specification,* Revision 3.0, February 1995.

Figure 2
Hierarchical Relationship in the TPC-C Business Environment



Note: + implies variations over measurement interval as rows are deleted or added.

Figure 3
TPC-C Database Tables Relationship

Table 1
TPC-C Requirements for Percentage in Mix, Keying Time, Response Time, and Think Time[a]

Transaction Type	Minimum Percentage in Mix	Minimum Keying Time (Seconds)	90th Percentile Response Time Constraint (Seconds)	Minimum Mean Think Time Distribution (Seconds)
New order	N/A[b]	18	5	12
Payment	43	3	5	12
Order status	4	2	5	10
Delivery	4	2	5	5
Stock level	4	2	5	5

Notes

[a] Table 1 is published in the Transaction Processing Performance Council's *TPC Benchmark C Standard Specification, Revision 3.0,* February 1995.

[b] Not applicable (N/A) because the measured rate is the reported throughput, though it is desirable to set it as high as possible (45%).

the local warehouse, and only one percent of the time is it a remote warehouse. The payment transaction processes a payment for a customer, updates the customer's balance, and reflects the payment in the district and warehouse sales statistics. The customer resident warehouse is the local warehouse 85 percent of the time and is the remote warehouse 15 percent of the time. The order-status transaction returns the status of a customer order. The customer order is selected 60 percent of the time by last name and 40 percent of the time by identification number. The delivery transaction processes orders corresponding to 10 pending orders, 1 for each district with 10 items per order. The corresponding entry in the new-order table is also deleted. The delivery transaction is intended to be executed in deferred mode through a queuing mechanism, rather than being executed interactively; there is no terminal response indicating the transaction completion. The stock-level transaction examines the quantity of stock for the items ordered by each of the last 20 orders in a district and determines the items that have a stock level below a specified threshold.

The TPC-C performance metric measures the total number of new orders completed per minute, with a 90th percentile response-time constraint of 5 seconds. This metric measures the business throughput rather than the transaction execution rate.[3] It is expressed in transactions-per-minute C (tpmC). The metric implicitly takes into account all the transaction types as their individual throughputs are controlled by the mix percentage given in Table 1. The tpmC is also driven by the activity of emulated users and the frequency of checkpointing.[10] The cycle for generating a TPC-C transaction by an emulated user is shown in Figure 4.

The transactions are generated uniformly and at random while maintaining a minimum percentage in mix for each transaction type. Table 1 gives the minimum mix percentage for each transaction type, the minimum keying time, the maximum 90th percentile response-time constraint, and the minimum think time defined by the TPC-C specification.

The delivery transaction, unlike the other transactions, must be executed in a deferred mode.[3] The response time in Table 1 is the terminal response acknowledging that the transaction has been queued and not that the delivery transaction itself has been executed. Further, at least 90 percent of the deferred delivery transactions must complete within 80 seconds of their being queued for execution. The performance tuning for the system under test determines the number of checkpoints done in the measurement interval and the length of the checkpointing interval. The TPC-C specification, however, defines the upper bound on the checkpointing interval to be 30 minutes.[3]

The other TPC-C metric is the price/performance ratio or dollars per tpmC. This metric is computed by dividing the total five-year system cost for the system under test with the reported tpmC.[11]

Performance Evaluation Methodology

In this section, we first describe the configuration of the system under test (SUT) used for the performance evaluation of the TruCluster system under the TPC-C workload. Then we discuss the testing strategy used to enhance the performance of the SUT.

We show the configuration of the client-server SUT in Figure 5. The server SUT consists of a TruCluster configuration with four nodes; each node is an AlphaServer 8400 5/350 system with eight 350-megahertz (MHz) CPUs and 8 gigabytes (GB) of memory. These nodes are connected together by a MEMORY CHANNEL link cable from the MEMORY CHANNEL adapter on the node to a single MEMORY CHANNEL hub. The local storage configuration for each node consists of 6 HSZ40 redundant array of inexpensive disks (RAID) controllers, 31 RZ28 and 141 RZ29 disk drives, connected to the node by SCSI buses to 6 KZPSA disk adapters. Further, each node is connected to FDDI by a DEFPA FDDI adapter. The nodes communicate with the clients over this FDDI.

The client SUT consists of 16 AlphaServer 1000 4/266 systems, each with 512 MB of memory, one RZ28 disk drive, and one DEFPA FDDI adapter.[12] The remote terminal emulators (RTEs) that are used to generate the transactions and measure the various times (i.e., think, response, or keying time) for each transaction are 16 VAXstation 3100 workstations, each with one RZ28 disk drive. From our logical description of the network topology shown in Figure 6, we see that each of the four nodes in the cluster is connected to four client systems, and each RTE is connected to one client system. The four clients associated with each node are connected to a DEChub 900 switch. Each of the four DEChub 900 products contains two concentrators, one DEFHU-MU 14-port unshielded twisted-pair (UTP) concentrator (for FDDI) and one DEFHU-MH concentrator (for the twisted-pair Ethernet). The DEChub 900 switches are connected to an 8-port GIGAswitch system, which is used to route communications between the client and the server.

The software configuration of the server system is the TruCluster software running under the Digital UNIX version 4.0A operating system and the Oracle Parallel Server database manager (Oracle7 version 7.3) installed on each cluster member. The software configuration installed on each client system is the Digital

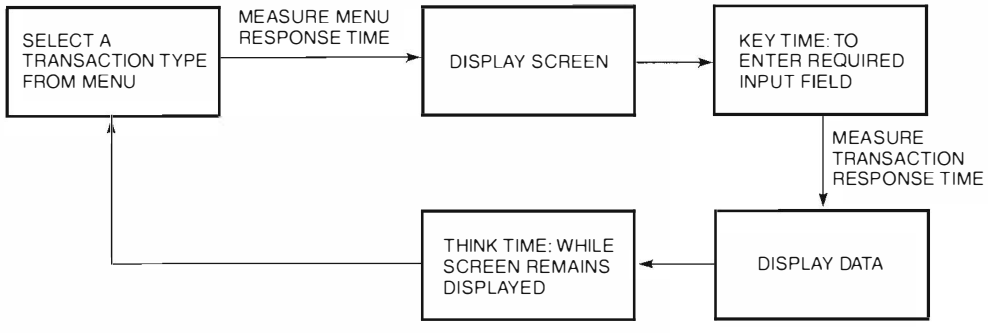


Figure 4
Cycle for Generating a TPC-C Transaction by an Emulated User

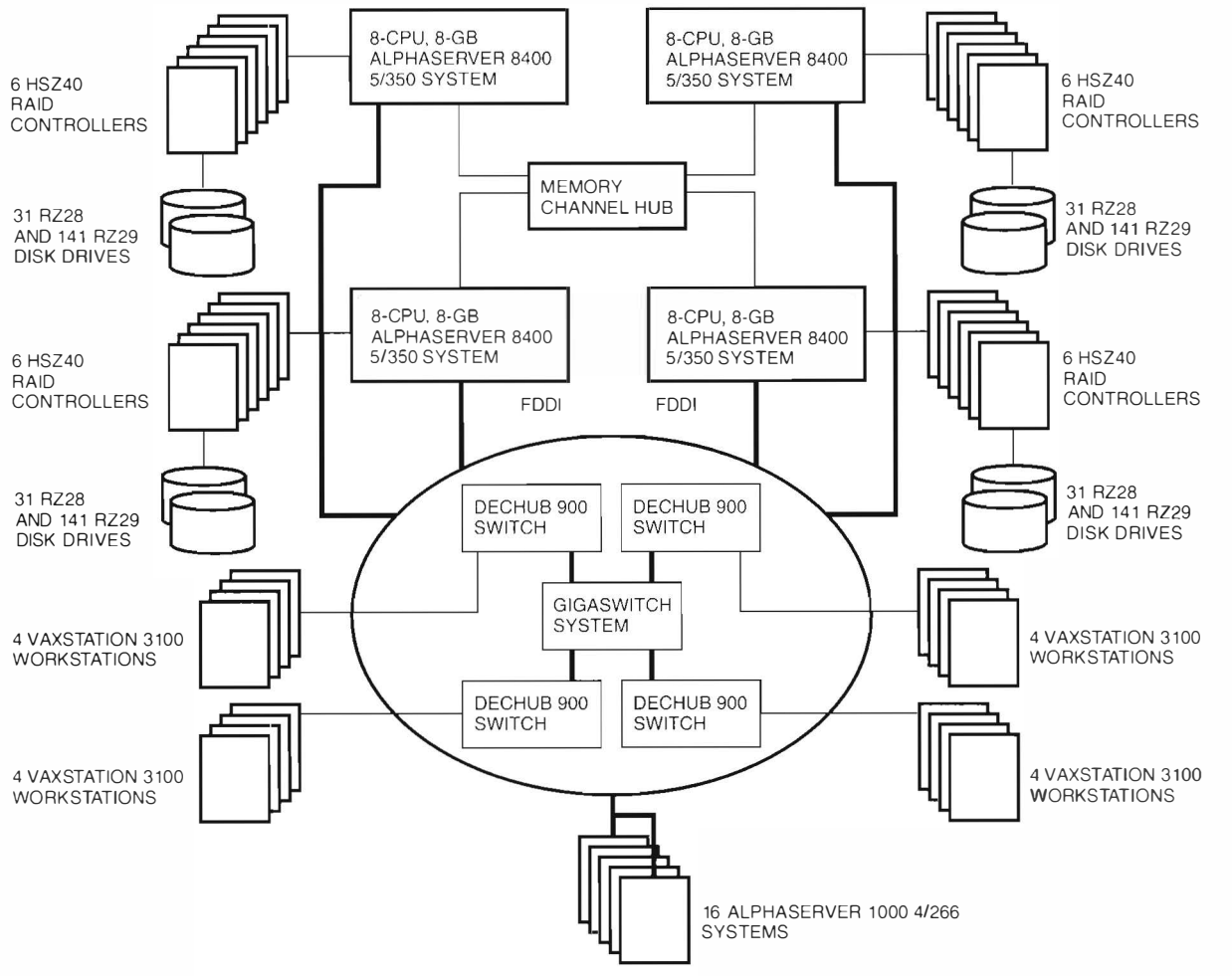


Figure 5
Client-Server System under Test

UNIX version 3.2D operating system and the BEA Tuxedo System/T version 4.2 transaction processing monitor. Further, each RTE runs the OpenVMS operating system and a proprietary emulation package, VAXRTE. In the remainder of this section, we discuss the testing strategy used to generate the transactions on the front end. Then we discuss the tuning done on the back end to achieve the maximum possible tpmC measurements from the SUT.

In conformance with the TPC-C specification, we used a series of RTEs to drive the SUT. The one-to-one correspondence between emulated users on the RTE and the TPC client forms on the client required us to determine the maximum number of users to be generated by the RTE. The main factor we used to determine the number of users was the client's memory size. We assumed that on a client, 32 MB of memory is used for the operating system and 0.25 MB for each TPC client form process. Therefore, with these constraints, each RTE generates 1,620 emulated users. The emulated users then generate transactions randomly based on the predefined transaction mix (as described in Table 1) with a unique seed. This ensures the mix is well defined and a variety of transaction types are running concurrently (to better simulate a real-world environment). We had a local area transport (LAT) connection over Ethernet between each emulated user and a corresponding TPC client form process on the client for faster communication. We show the communication between an RTE, a client, and a server in Figure 7.

We built five order queues on each client corresponding to a transaction type, which allowed us to control the transaction percentage mix. A TPC client form process queues transactions generated by the emulated users to the appropriate order queue using Tuxedo library calls. These transaction requests in each queue are processed in a first in, first out (FIFO) order by the Tuxedo server processes running on the client. We had 44 Tuxedo server processes that were not evenly distributed among the 5 order queues but were distributed so that the number of Tuxedo server processes dedicated to a queue was directly correlated to the percentage of the workload handled by the

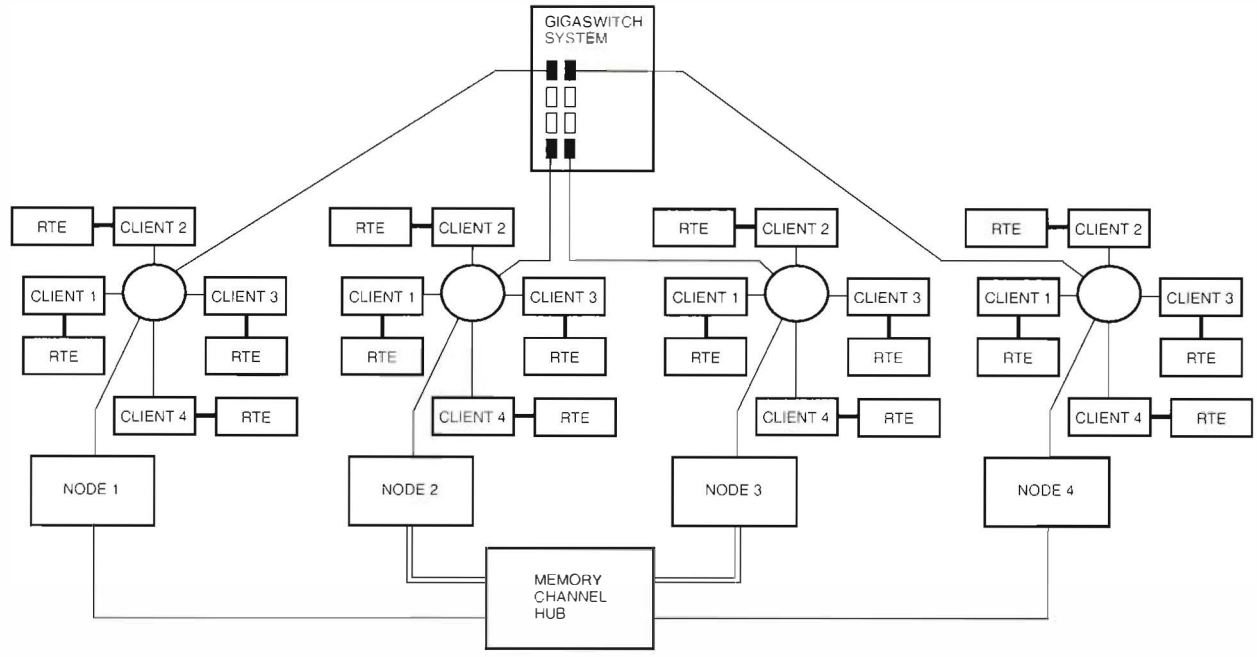


Figure 6
Logical Description of the Network Topology

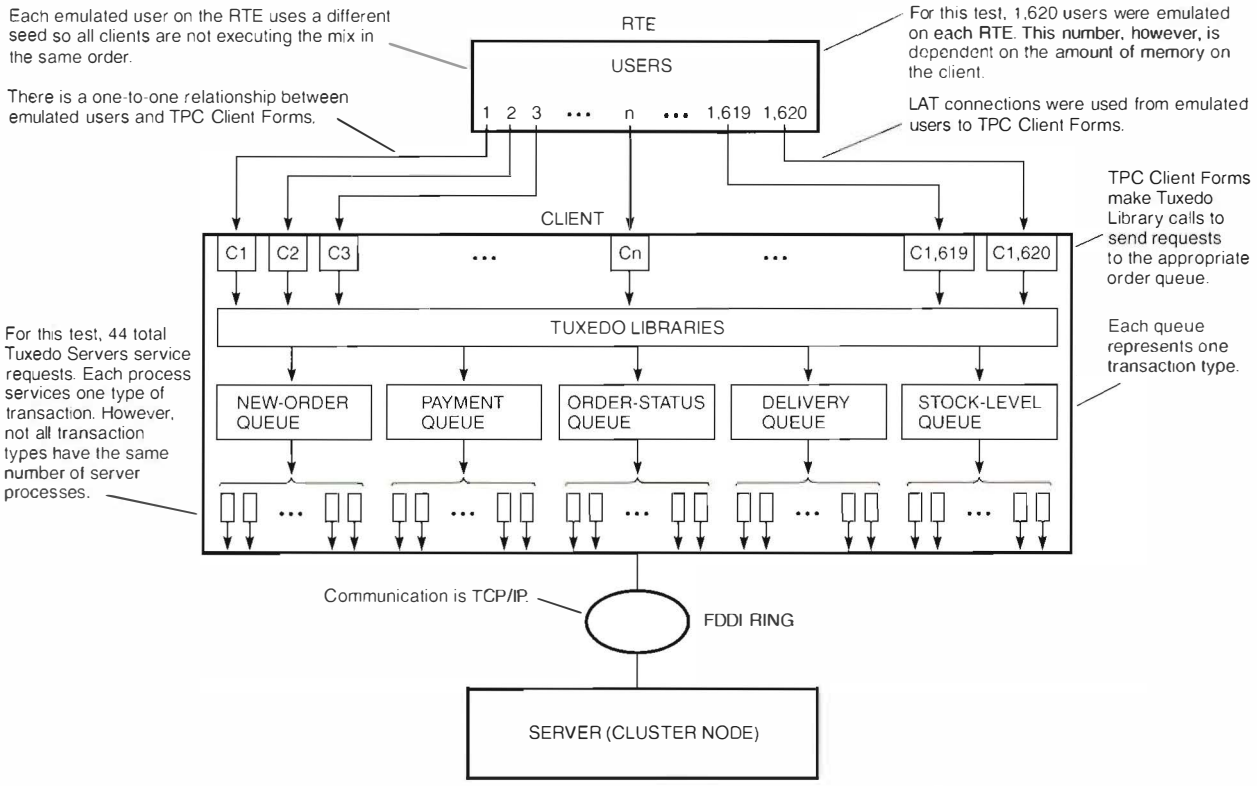


Figure 7
Communication between an RTE, a Client, and a Server

queue. In other words, the greater the percentage of the workload on a queue, the greater the number of Tuxedo server processes dedicated to that queue. The number of Tuxedo server processes per client is computed based on the rule of thumb that each queue should have no more than 300 outstanding requests during checkpointing and 15 at other times. These Tuxedo server processes communicate with the server system (cluster node) using the Transmission Control Protocol/Internet Protocol (TCP/IP) over FDDI to execute related database operations.[13]

The industry-accepted method of tuning the TPC-C back end is to add enough disks and disk controllers on the server to eliminate the potential for an I/O bottleneck, thus forcing the CPU to be saturated. Once the engineers are assured that the performance limitation is CPU saturation, the amount of memory is tuned to improve the database hit ratio. Because all vendors submitting TPC-C results use this style of tuning, the performance limitation for TPC-C is usually the back-end server's CPU power. In fact, tests have shown that if this method of tuning is not followed on the back-end server, the user will not obtain the optimal TPC-C performance results. Instead, the tests reveal a back-end server configuration that has not fully utilized the server's potential by having unbalanced CPU and I/O capabilities. This type of configuration not only reduces the server's throughput capacity but also adversely affects the price/performance of the SUT.

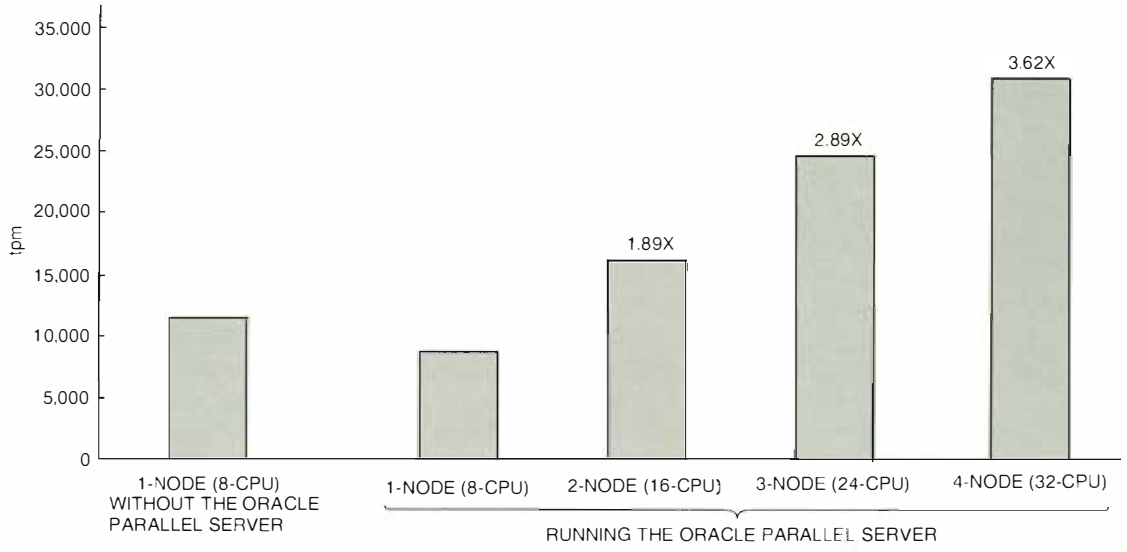
On the back end, we used TruCluster technology features to achieve the maximum possible transactions per minute (tpm).[14] We balanced the I/O across all the RAID controllers and disks of the cluster and distributed the database across all the server nodes. We distributed the database such that each node in the cluster had an almost equal part of the database. The TPC-C benchmark execution requires a single database view across the cluster. We used the DRD and DLM services of the TruCluster software to present a contiguous view of the database across the cluster. If both the database and the indexes could have been completely partitioned, we could have achieved close to linear scaling per node. However, since the Oracle Parallel Server does not have horizontal partitioning of the indexes, we could not completely partition the indexes across the cluster.[15] This resulted in 15 percent to 20 percent of internodal access, which means that 15 percent to 20 percent of the new orders were satisfied by remote warehouses, therefore making our TPC-C results more realistic.

We also tuned the physical memory to trade off memory for database cache and the DLM locks. Heuristically, we observed a 40-percent gain in throughput on a single-node AlphaServer 8400 5/350 server system running TPC-C when the memory size was increased from 2 GB to 8 GB. This is because, with more data being served by memory, the number of processor stalls decreases, and the database-cache hit ratio improves from 88 percent to more than 95 percent.[16] Tuning physical memory beyond 2 GB is called very large memory (VLM). We used the tpm results of the AlphaServer 8400 system to tune the physical memory size and configuration. We show these measured tpm results for the AlphaServer 8400 cluster systems in Figure 8.

To achieve optimal server performance, it is important to tune the amount of memory used by the Oracle System Global Area (SGA) and the DLM. Our testing found that using VLM to increase the size of the SGA to 5.0 GB of physical memory yielded optimal performance in a TruCluster environment. However, it is important to note that on a single-node server that does not run the Oracle Parallel Server, we could assign 6.6 GB of physical memory to the SGA. (One reason that the SGA was smaller in an Oracle Parallel Server environment is that memory needed to be set aside for the DLM.) Consequently, as seen in Figure 8, the tpm on a single-node cluster system running the Oracle Parallel Server (8.4K tpm) is less than a single-node cluster not running the Oracle Parallel Server (11.4K tpm).

In an Oracle Parallel Server environment, we assigned 1 GB of memory to the DLM for the following reasons: The DLM, under the 64-bit Digital UNIX operating system, requires 256 bytes for each lock. In addition, the DLM must be able to hold at least one other location (and sometimes three) for lock callback. As a result, each lock requires between 512 bytes and 1 kilobyte (KB) of physical memory. To tune the system, we added more locks to increase the granularity of the locks and reduce lock contention. We observed that for this configuration, a system of this size supporting the Oracle Parallel Server requires 1 million locks (occupying 1 GB of memory) for the DLM when using 5.0 GB of memory for the SGA. Again heuristically, we observed that if we used less memory for the DLM, the total number of locks per page was reduced. The decrease in locks per page increases contention across nodes and hence reduces the tpm as the number of nodes increases.

With the help of engineers from Digital's MEMORY CHANNEL Group, we were able to use a hardware data analyzer to measure the percentage of the MEMORY CHANNEL interconnect's bandwidth used when running the TPC-C benchmark. By using the data analyzer, we determined that we do not approach saturation of the PCI-based MEMORY CHANNEL hardware during a TPC-C test, even though it is capable of sustaining a peak throughput rate of 100 MB/s. In fact, we observed that the MEMORY CHANNEL bandwidth was not saturated; a TPC-C test required a peak throughput rate of only 15 MB/s to 17 MB/s from the MEMORY CHANNEL. As stated previously, the benchmark specification forces 15 percent of the database accesses



Notes: 1. Each node is an 8-CPU AlphaServer 8400 5/350 cluster system.
2. The number preceding the X indicates a multiple of the tpmC measured on a single node running the Oracle Parallel Server.

Figure 8
TPC-C Results on the AlphaServer 8400 Family

to be remote, resulting in database accesses across the MEMORY CHANNEL. Using the DRD administration service available with the UNIX TruCluster software, we measured the DRD remote read percentages to match the 15-percent remote accesses rate. The DRD remote write performance was only 3 percent to 4 percent during the steady state and rose to 10 percent to 11 percent during a database checkpoint. It is important to note that the TPC-C benchmark performs random 2K I/Os using the Oracle Parallel Server. Small, random I/O transfers are much more difficult to perform than large, sequential transfers. Because the MEMORY CHANNEL interconnect not only has sufficient bandwidth for TPC-C but also provides excellent latency (less than 5 microseconds), we are able to report very good scaling results.

In the section TPC-C Benchmark, we discussed that the time taken for a checkpoint impacts the throughput. Therefore, we focused on improving the checkpointing time to increase the tpmC number. First, we used a dedicated PCI bus on each node for the MEMORY CHANNEL interconnect and thus obtained a 5-percent improvement in performance during checkpointing. Next we implemented the highly optimized "fastcopy" routine in DRD, which packs data on the PCI when transmitting through the MEMORY CHANNEL interconnect.

Performance Measurement Results

In this section, we present our results for the TruCluster configuration running the TPC-C workload and compare them with results from competitive vendors. We conducted the test on a database with 2,592 warehouses and 25,920 emulated users. The database was equally divided, which means each node contained 648 warehouses and served 6,480 emulated users. We show the initial cardinality of the database tables in Table 2. The cardinality of the history, orders, new-order, and order-line tables increased as the test progressed and generated new orders. We conducted the experimental runs for a minimum of 160 minutes.[10] The measurement on the SUT began approximately 3 minutes after the simulated users had begun executing transactions. The measurement period of 120 minutes, however, started after the SUT attained a steady state in approximately 30 minutes. In agreement with the TPC-C specification, we performed 4 checkpoints at 30-minute intervals during the measurement period.

On the SUT, we measured a maximum throughput of 30,390.65 tpmC, which unveiled a new record high in the competitive market for database applications and UNIX servers. We repeated the experiment once

Table 2
Initial Cardinality of the Database Tables

Warehouse	2,592
District	25,920
Customer	77,760,000
History	77,760,000
Order	77,760,000
New order	23,328,000
Order lines	777,547,308
Stock	259,200,000
Item	100,000

more to ensure the reproducibility of the maximum measured tpmC. Digital Equipment Corporation and Oracle Corporation also present a price/performance ratio of $305 per tpmC.

In Table 3, we present the total occurrences of each transaction type and the percentage transaction mix used to generate the transactions in each test run. We compare the percentage transaction mix in Table 1 and Table 3 and observe that our measurements are in agreement with the TPC-C specification. We present the 90th percentile response time measured for each transaction type in Table 4. The 90th percentile response time we measured is well below the TPC-C specification requirement (compare Table 1 and Table 4). In Table 5, we present the minimum, average, and maximum keying and think times. Again, we comply with the TPC-C specification (compare Table 1 and Table 5).

Now we compare the maximum throughput achieved on the AlphaServer 8400 5/350 four-node TruCluster configuration with results from Tandem Computers and from Hewlett-Packard Company (HP).[17] The Tandem nonstop Himalaya K10000-112 112-node cluster reported 20,918.03 tpmC at $1,532 per tpmC. Observe that Digital's measured tpmC are 45 percent higher than Tandem's, and Digital's price/performance is 20 percent of Tandem's cost. In Figure 9, we compare Digital's performance with HP's. The HP 9000 EPS30 C/S 48-CPU four-node cluster system using the Oracle Parallel Server Oracle7 version 7.3 reported 17,826.50 tpmC at $396.[18] Again, observe that the tpmC we measured on Digital's TruCluster configuration are 59 percent higher than HP's at 77 percent of the cost.

Table 3
Measured Total Occurrences of Each Transaction Type and Percentage Transaction Mix

Transaction Type	Total Occurrences	Percentage in Mix
New order	3,645,228	44.47
Payment	3,540,119	43.19
Order status	336,255	4.10
Delivery	337,423	4.12
Stock level	337,730	4.12

Table 4
Measured 90th Percentile Response Time

Transaction Type	90th Percentile Response Time
New order	3.4
Payment	3.2
Order status	0.9
Delivery (interactive)	0.4
Delivery (deferred)	5.0
Stock level	1.7

Table 5
Measured Keying/Think Times

Transaction Type	Minimum (Seconds)	Average (Seconds)	Maximum (Seconds)
New order	18.0/0.00	18.1/12.2	18.8/188.1
Payment	3.0/0.00	3.1/12.1	3.7/201.4
Order status	2.0/0.00	2.1/10.1	2.7/125.6
Delivery	2.0/0.00	2.1/5.2	2.7/74.9
Stock level	2.0/0.00	2.1/5.2	2.7/62.7

Conclusion and Future Work

In this paper, we discussed the performance evaluation of Digital's TruCluster multicomputer system, specifically the AlphaServer 8400 5/350 32-CPU, four-node cluster system, under the TPC-C workload. For completeness, we gave an overview of TruCluster clustering technology and the TPC-C benchmark. We discussed tuning strategies that took advantage of TruCluster technology features like the MEMORY CHANNEL interconnect, the DRD, and the DLM. We tuned memory to use VLM for the database cache and made memory allocation trade-offs for DLM locks to reduce processor stalls and improve cache hit ratios.

One common concern is performance scalability of cluster systems, that is, incremental performance growth with the size of the cluster. In Figure 8, we showed the measured performance of an SMP server, both with and without the Oracle Parallel Server, and cluster configurations with two, three, and four SMP servers. We do not see linear scaling because the Oracle Parallel Server imposes a significant amount of overhead on each cluster node. This overhead equates to approximately a 25-percent reduction in tpmC on a per node basis. However, it is important to note that due to the time constraints of obtaining audited results for the product announcement, the testing team was unable to fully tune the server and saturate the server CPUs. In future testing, additional performance tuning is planned to further optimize server performance.

The performance testing of the TruCluster multicomputer system was time-consuming and expensive. Thus, answering "what if" questions regarding sizing and tuning of varying cluster configurations under different workloads using measurements is an expensive (with respect to money and time) task. To address this problem, we are developing an analytical performance cluster model for capacity planning and tuning.[19] The model will predict the performance of cluster configurations (ranging from two to eight members) with varying workloads and system parameters (for

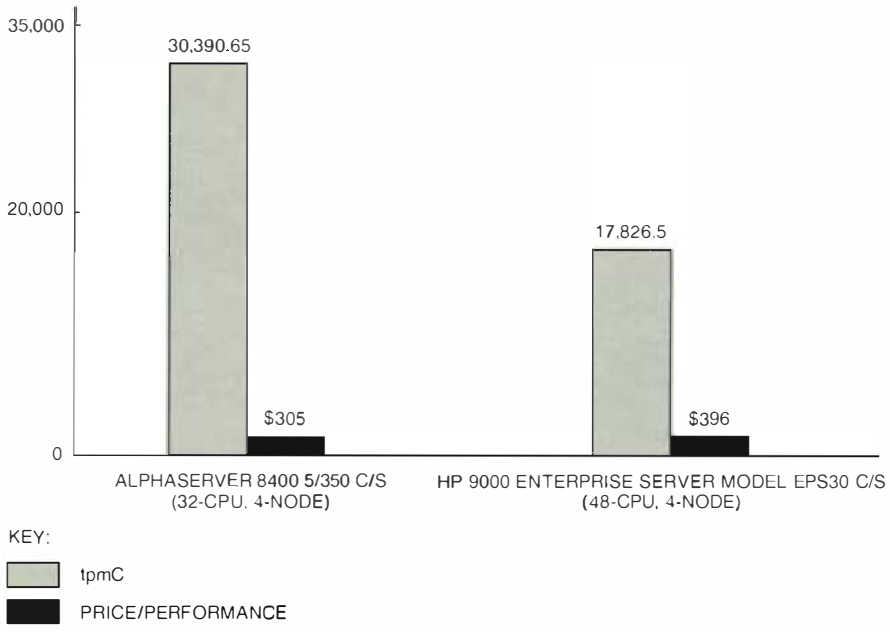


Figure 9
Comparison of TPC-C Results

example, memory size, storage size, and CPU power). We will implement this model in Visual C++ to develop a capacity planning tool.

Acknowledgments

Many people within several groups and disciplines in both Digital and Oracle contributed to the success of this performance project. We would like to thank the following individuals from Digital: Lee Allison, Roger Deschenes, Tareef Kawaf, Maria Lopez, Joe McFadden, Bhagyam Moses, Ruth Morgenstein, Sanjay Narahari, Dave Stanley, and Greg Tarsa of the CSD Performance Group; Brian Stevens and Tim Burke of the Digital UNIX Engineering Group; Jim Woodward, Digital UNIX Performance Team member; Sean Reilly, Doug Williams, and Zarka Cvetanovic of the AVS Performance Group; and Don Harbert and Pauline Nist, the test sponsors. Lastly, we would like to thank Jef Kennedy, Peter Lai, Karl Haas, and Vipin Gokhale of the Oracle Performance Group.

References and Notes

1. Throughout this paper, we use the term cluster interchangeably with TruCluster.

2. W. Kohler, A. Shah, and F. Raab, "Overview of TPC Benchmark C: The Order Entry Benchmark" (Transaction Processing Performance Council, Technical Report, December 1991).

3. Transaction Processing Performance Council, *TPC Benchmark C Standard Specification, Revision 3.0*, February 1995.

4. S. Leutenegger and D. Dias, "A Modeling Study of the TPC-C Benchmark," *ACM SIGMOD Record*, vol. 22, no. 2 (June 1993): 22–31.

5. R. Gillett, "MEMORY CHANNEL Network for PCI," *IEEE Micro*, vol. 16, no. 1 (February 1996): 12–19.

6. *TruCluster for Digital UNIX Version 1.0* (Maynard, Mass.: Digital Equipment Corporation, Software Product Description 63.92, October 1995).

7. W. Cardoza, F. Glover, and W. Snaman Jr., "Design of the TruCluster Multicomputer System for the Digital UNIX Environment," *Digital Technical Journal*, vol. 8, no.1 (1996): 5–17.

8. *TruCluster Software, Hardware Configuration* (Maynard, Mass.: Digital Equipment Corporation, Order No. AA-QL8LA-TE, December 1995).

9. *TruCluster Software Installation and Configuration* (Maynard, Mass.: Digital Equipment Corporation, Order No. AA-QL8MA-TE, September 1995).

10. Checkpointing is a process to make the copy of the database records/pages on the durable media current; systems do not write the modified records/pages of the database at the time of the modification but at some deferred time.

11. This cost includes the hardware system cost, the software license charge, and the maintenance charges for a five-year period.

12. The AlphaServer 1000 4/266 system can be configured with as much as 1 GB of memory. Due to a supply shortage of denser error correction code (ECC) memory, the clients in the SCT could be configured with a maximum memory of 512 MB.

13. Digital Equipment Corporation and Oracle Corporation, "Digital AlphaServer 8400 5/350 32-CPU 4-Node Cluster Using Oracle7, Tuxedo, and Digital UNIX," TPC Benchmark C Full Disclosure Report filed with the Transaction Processing Performance Council, April 1996. Also available from the TPC Web page.

14. Note that these results were not audited; per TPC-C specification, we refer to them as tpm instead of tpmC.

15. Horizontal partitioning of the indexes allows the user to have each node in the cluster store indexes that are mapped only to tables that are local.

16. T. Kawaf, D. Shakshober, and D. Stanley, "Performance Analysis Using Very Large Memory on the 64-bit AlphaServer System," *Digital Technical Journal*, vol. 8, no. 3 (1996, this issue): 58–65.

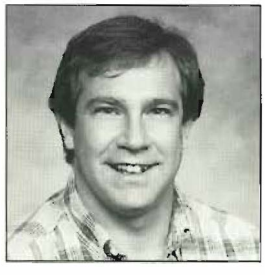
17. These results were withdrawn by Tandem on April 12, 1996, and hence are not included in Figure 9.

18. Hewlett-Packard Company, General Systems Division, and Oracle Corporation, "HP 9000 Enterprise Parallel Server Model EPS30 (4-Node) Using HP-UX 10.20 and Oracle7," TPC Benchmark C Full Disclosure Report filed with the Transaction Processing Performance Council, May 1996. Also available from the TPC Web page.

Biographies

Judith A. Piantedosi
A principal software engineer in the CSD Performance Group, Judy Piantedosi evaluates I/O performance on Digital UNIX systems, specializing in characterizing NFS file servers. Judy is the project leader of the TruCluster capacity planning modeling effort and Digital's technical representative to the Standard Performance Evaluation Corporation (SPEC) System File Server (SFS) Subcommittee. Judy joined Digital in 1987 to help solve customer hardware/software problems when using System V. Before joining Digital, Judy was employed at Mitre Corporation. She was the lead software designer on the Joint STARS Radar Evaluation Activity, a radar simulation built to provide proof of concept to the U.S. Air Force for the Joint STARS project. She was responsible for implementing several radar models into the simulation. Judy holds a B.A. from Boston College (1984).

Archana S. Sathaye
Archana Sathaye is currently a consultant to Digital in its CSD Performance Group. From 1987 to 1994, she was an employee of Digital and worked on several reliability, availability, and performability modeling projects for OpenVMS Cluster systems and other high-end CPU products. She resigned from Digital and accepted a position as adjunct assistant professor in the Department of Electrical and Computer Engineering at the University of Pittsburgh. Archana holds a Ph.D. in electrical and computer engineering from Carnegie Mellon University (1993); an M.S. from Virginia Polytechnic and State University (1986), a B.Sc (1981) and an M.Sc (1983) from the University of Bombay, India, all in mathematics. She is an affiliate member of ACM SIGMETRICS and has authored or coauthored several papers on reliability, availability, and performability modeling and control synthesis.



D. John Shakshober
John Shakshober is the technical director of the CSD Performance Group. The Computer Systems Division Performance Group evaluates Digital's systems against industry-standard benchmarks such as those of the Transaction Processing Performance Council (TPC) and the Standard Performance Evaluation Corporation (SPEC). In this function, John has been responsible for integrating Digital's state-of-the-art software technologies with Digital's Alpha-based products since their introduction in 1992. Prior to joining the CSD Performance Group, John modeled the performance of the 21064 and 21164 Alpha 64-bit VLSI microprocessors and was a member of the VAX 6000 Hardware Group. He joined Digital in 1984 after receiving a B.S. in computer engineering from the Rochester Institute of Technology. John also received an M.S. in electrical engineering from Cornell University in 1988.

Performance Analysis Using Very Large Memory on the 64-bit AlphaServer System

Tareef S. Kawaf
D. John Shakshober
David C. Stanley

Optimization techniques have been used to deploy very large memory (VLM) database technology on Digital's AlphaServer 8400 multiprocessor system. VLM improves the use of hardware and software caches, main memory, the I/O subsystems, and the Alpha 21164 microprocessor itself, which in turn causes fewer processor stalls and provides faster locking. Digital's 64-bit AlphaServer 8400 system running database software from a leading vendor has achieved the highest TPC-C results to date, an increased throughput due to increased database cache size, and an improved scaling with symmetric multiprocessing systems.

Digital's AlphaServer 8400 enterprise-class server combines a 2-gigabyte-per-second (GB/s) multiprocessor bus with the latest Alpha 21164 64-bit microprocessor.[1] Between October and December 1995, an AlphaServer 8400 multiprocessor system running the 64-bit Digital UNIX operating system achieved unprecedented results on the Transaction Processing Performance Council's TPC-C benchmark, surpassing all other single-node results by a factor of nearly 2. As of September 1996, only one other computer vendor has come within 20 percent of the AlphaServer 8400 system's TPC-C results.

A memory size of 2 GB or more, known as very large memory (VLM), was essential to achieving these results. Most 32-bit UNIX systems can use 31 bits for virtual address space, leaving 1 bit to differentiate between system and user space, which creates difficulties when attempting to address more than 2 GB of memory (whether virtual or physical).

In contrast, Digital's Alpha microprocessors and the Digital UNIX operating system have implemented a 64-bit virtual address space that is four billion times larger than 32-bit systems. Today's Alpha chips are capable of addressing 43 bits of physical memory. The AlphaServer 8400 system supports as many as 8 physical modules, each of which can contain 2 CPUs or as much as 2 GB of memory.[2] Using these limits, database applications tend to achieve peak performance using 8 to 10 CPUs and as much as 8 GB of memory.

The examples in this paper are drawn primarily from the optimization of a state-of-the-art database application on AlphaServer systems; similar technical considerations apply to any database running in an Alpha environment. As of September 1996, three of the foremost database companies have extended their products to exploit Digital's 64-bit Alpha environment, namely Oracle Corporation, Sybase, Inc., and Informix Software, Inc.

The sections that follow describe the TPC-C workload and discuss two database optimizations that are useful regardless of memory size: locking intrinsics and OM instruction-cache packing. (OM is a post–link time optimizer available on the Digital UNIX operating system.)[3] VLM experimental data is then presented in the section VLM Results.

TPC-C Benchmark

The TPC-C benchmark was designed to mimic complex on-line transaction processing (OLTP) as specified by the Transaction Processing Performance Council.[4] The TPC-C workload depicts the activity of a generic wholesale supplier company. The company consists of a number of distributed sales districts and associated warehouses. Each warehouse has 10 districts. Each district services 3,000 customer requests. Each warehouse maintains a stock of 100,000 items sold by the company. The database is scaled according to throughput (that is, higher transaction rates use larger databases). Customers call the company to place new orders or request the status of an existing order.

Method

The benchmark consists of five complex transactions that access nine different tables.[5] The five transactions are weighted as follows:

1. Forty-three percent—A new-order transaction places an order (an average of 10 lines) from a warehouse through a single database transaction and updates the corresponding stock level for each item. In 99 percent of the new-order transactions, the supplying warehouse is the local warehouse and only 1 percent of the accesses are to a remote warehouse.
2. Forty-three percent—A payment transaction processes a payment for a customer, updates the customer's balance, and reflects the payment in the district and warehouse sales statistics. The customer resident warehouse is the home warehouse 85 percent of the time and is the remote warehouse 15 percent of the time.
3. Four percent—An order-status transaction returns the status of a customer order. The customer order is selected 60 percent of the time by the last name and 40 percent of the time by an identification number.
4. Four percent—A delivery transaction processes orders corresponding to 10 pending orders for each district with 10 items per order. The corresponding entry in the new-order table is also deleted. The delivery transaction is intended to be executed in deferred mode through a queuing mechanism. There is no terminal response for completion.
5. Four percent—A stock-level transaction examines the quantity of stock for the items ordered by each of the last 20 orders in a district and determines the items that have a stock level below a specified threshold. This is a read-only transaction.

The TPC-C specification requires a response time that is less than or equal to 5 seconds for the 90th percentile of all but the delivery transaction, which must complete within 20 seconds.

In addition, the TPC-C specification requires that a complete checkpoint of the database be done. A checkpoint flushes all transactions committed to the database from the database cache (memory) to nonvolatile storage in less than 30 minutes. This important requirement is one of the more difficult parts to tune for systems with VLM.[6]

Results

Table 1 gives the highest single-node TPC-C results published by the Transaction Processing Performance Council as of September 1, 1996.[4]

For a complete TPC-C run, a remote terminal emulator must be used to simulate users making transactions. For performance optimization purposes, however, it is convenient to use a back-end-only version of the benchmark in which the clients reside on the server. The transactions per minute (tpm) derived in this environment are called back-end tpm in Table 2 and cannot be compared to the results of audited runs (such as those given in Table 1). However, when a performance improvement is made to the back-end-only environment, performance improvements are clearly seen in the full environment.

Tuning for the system is iterative. For each data point collected, clients were added to try to saturate the server; then the amount of memory was varied for the database cache. A trade-off between database memory, system throughput, and checkpoint performance required us to tune each data point individually. The system was configured with a sufficient number of disk drives and I/O controllers to ensure that it was 100-percent CPU saturated and never I/O limited. The experiments reported in this paper use database sizes of approximately 100 GB, spread over 172 RZ29 spindles and 7 KZPSA adapter/HSZ40 controller pairs, with each HSZ40 controller using 5 small computer systems interface (SCSI) buses.

Tuning Specific to Alpha

UNIX databases on Digital's Alpha systems were first ported in 1992. For database companies to fully use the power of Alpha's 64-bit address space, each database vendor had to expand the scope of its normal 32-bit architecture to make use of 64-bit pointers. Thus, each database could then address more than 2 GB of physical memory without awkward code segments or other manipulations to the operating system to extend physical address space.

By 1994, most vendors of large databases were offering 64-bit versions of their databases for Digital's Alpha environment. As a group chartered to measure database performance on Alpha systems, Digital's Computer Systems Division (CSD) Performance Group worked with each database vendor and with the Digital System Performance Expertise Center to improve performance.

Table 1
TPC-C Results

System	Throughput	Price/ Performance	Number of CPUs	Date
AlphaServer 8400 5/350, Oracle Rdb7 V7.0, OpenVMS V7.0	14,227 tpmC	$269/tpmC	10	May 1996
AlphaServer 8400 5/350, Sybase SQL Server 11.0, Digital UNIX, iTi Tuxedo	14,176 tpmC	$198/tpmC	10	May 1996
AlphaServer 8400 5/350, Informix V7.21, Digital UNIX, iTi Tuxedo	13,646.17 tpmC	$277/tpmC	10	March 1996
Sun Ultra Enterprise 5000, Sybase SQL Server V 11.0.2	11,465.93 tpmC	$191/tpmC	12	April 1996
AlphaServer 8400 5/350, Oracle7, Digital UNIX, iTi Tuxedo	11,456.13 tpmC	$286/tpmC	8	December 1995
AlphaServer 8400 5/300, Sybase SQL Server 11.0, Digital UNIX, iTi Tuxedo	11,014.10 tpmC	$222/tpmC	10	December 1995
AlphaServer 8400 5/300, Oracle7, Digital UNIX, iTi Tuxedo	9,414.06 tpmC	$316/tpmC	8	October 1995
SGI CHALLENGE XL Server, INFORMIX-OnLine V7.1, IRIX, IMC Tuxedo	6,313.78 tpmC	$479/tpmC	16	November 1995
HP 9000 Corporate Business Server, Sybase SQL Server 11, HP-UX, IMC Tuxedo	5,621.00 tpmC	$380/tpmC	12	May 1995
HP 9000 Corporate Business Server, Oracle7, HP-UX, IMC Tuxedo	5,369.68 tpmC	$535/tpmC	12	May 1995
Sun SPARCcenter 2000E Oracle7, Solaris, Tuxedo	5,124.21 tpmC	$323/tpmC	16	April 1996
Sun SPARCcenter 2000E, INFORMIX-OnLine 7.1, Solaris, Tuxedo	3,534.20 tpmC	$495/tpmC	20	July 1995
IBM RS/6000 PowerPC R30, DB2 for AIX, AIX, IMC Tuxedo	3,119.16 tpmC	$355/tpmC	8	June 1995
IBM RS/6000 PowerPC J30, DB2 for AIX, AIX, IMC Tuxedo	3119.16 tpmC	$349/tpmC	8	June 1995

Table 2
Amount of Memory versus Back-end tpm, Database-cache Miss Rate, and Instructions per Transaction

Database Memory (GB)	Back-end (Normalized tpm)	Relative Database-cache Miss (Percentage)	Relative Instructions per Transaction
1	1.0	1.0	1.0
2	1.3	0.73	0.75
3	1.5	0.58	0.63
4	1.6	0.50	0.57
5	1.7	0.42	0.50
6	1.8	0.40	0.45

Two optimizations generally realized 20 percent gains on Alpha systems.[7] These were

1. Optimization of spinlock primitives supported now by DEC C compiler intrinsics
2. OM profile-based link optimization, which performs instruction-cache packing during the final link of the database

In addition, the Digital UNIX operating system version 3.2 and higher versions have optimized I/O code paths and support advanced processor affinity and other scheduling algorithms that have been optimized for enterprise-class commercial performance. With these optimizations, database performance on Digital's Alpha systems has been significantly improved.

Lock Optimization

Locks are used on multiprocessor systems to synchronize atomic access to shared data. A lock is either unowned (clear) or owned (set). A key design decision leading to good multiprocessor performance and scaling is partitioning the shared data and associated locks. The discussion of how to partition data and associated locks to minimize contention and the number of locks required is beyond the scope of this paper.

The implementation of locks requires an atomic test-and-set operation. On a particular system, the implementation of the lock is dependent on the primitive test-and-set capabilities provided by the hardware.

Locks are used to synchronize atomic access to shared data. A shared data element that requires atomic access is associated with a lock that must be acquired and held while the data is modified. On multiprocessing systems, locks are used to synchronize atomic access to shared data. A sequence of code that accesses shared data protected by a lock is called a critical section. A critical section begins with the acquisition of a lock and ends with the release of that lock. Although it is possible to have nested critical sections where multiple locks are acquired and released, the discussion in this section is limited to a critical section with a single lock.

To provide atomic access to shared data, the critical section running on a given processor locks the data by acquiring the lock associated with the shared data. In the simplest case, if a second processor tries to acquire access to shared data that is already locked, the second processor loops and continually retries the access (spins) until the processor owning the lock releases it. In a complex case, if a second processor tries to acquire access to shared data that is already locked, the second processor loops a few times and then, if the lock is still owned by another processor, puts itself into a wait state until the processor owning the lock releases it.

The Alpha Architecture Reference Manual specifies that "...the order of reads and writes done in an Alpha implementation may differ from that specified by the programmer."[8] Therefore, process coordination requires a special test-and-set operation that is implemented through the load-locked/store-conditional instruction sequence. To provide good performance and scaling on multiprocessor Alpha systems, it is important to optimize the test-and-set operation to minimize latency. The test-and-set operation can be optimized by the following methods:

- Use an in-lined load-locked/store-conditional sequence through an embedded assembler or compiler intrinsics.
- Preload a lock using a simple load operation prior to a load-locked operation.
- If a lock is held, spin on a simple load instruction rather than a load-locked instruction sequence.

The basic hardware building block used to implement the acquisition of a lock is the test-and-set operation. On many microprocessors, an atomic test-and-set operation is provided as a single instruction. On an Alpha microprocessor, the test-and-set operation needs to be built out of load-locked (LDx_L) and store-conditional (STx_C) instructions. The LDx_L ... STx_C instructions allow the Alpha microprocessor to provide a multiprocessor-safe method to implement the test-and-set operation with minimal restrictions on read and write ordering. The load-locked operation sets a locked flag on the cache block containing the data item. The store-conditional operation ensures that no other processor has modified the cache block before it stores the data. If no other processor has modified the cache block, the store-conditional operation is successful and the data is written to memory. If another processor has modified the cache block, the store-conditional operation fails, and the data is not written to memory. Optimizing the test-and-set sequence on Alpha systems is a complex task that provides significant performance gains.

Figure 1 shows code sequences that Digital's CSD Performance Group has given to database vendors to improve locking intrinsics in the Alpha environment. These code sequences can be used to implement spinlocks in the DEC C compiler on the Digital UNIX operating system.

Using OM Feedback

As previously mentioned, OM is a post–link time optimizer available on the Digital UNIX operating system. It performs optimizations such as compression of addressing instructions and dead code elimination through the use of feedback. The performance improvement provided by OM on Alpha 21164 systems is dramatic for the following two reasons.[5]

- The 21164 microprocessor has an 8-kilobyte (KB) direct-mapped instruction cache, which makes code placement extremely important. In a direct-mapped cache, the code layout and linking order maps one for one to its placement in cache. Thus a poorly chosen instruction stream layout or simply unlucky code placement within libraries can alter performance by 10 to 20 percent. Routines are frequently page aligned, which can increase the likelihood of cache collisions.
- The high clock rate of the Alpha 21164 microprocessor (300 to 500 megahertz [MHz]) requires a cache hierarchy to attempt to keep the CPU pipelines filled. The penalty of a first-level cache miss is 5 to 9 cycles, which means that an

```
//TEST_AND_SET implements the Alpha version of a test and set operation using
//the load-locked .. store-conditional instructions.  The purpose of this
//function is to check the value pointed to by spinlock_address and, if the
//value is 0, set it to 1 and return success (1) in R0.  If either the spinlock
//value is already 1 or the store-conditional failed, the value of the spinlock
//remains unchanged and a failure status (0,2, or 3) is returned in R0.
//
//The status returned in R0 is one of the following:
//  0 - failure (spinlock was clear; still clear, store-conditional failed)
//  1 - success (spinlock was clear; now set)
//  2 - failure (spinlock was set; still set, store-conditional failed)
//  3 - failure (spinlock was set; still set)
//
#define TEST_AND_SET (spinlock_address) asm( "ldl_l     $0,($16);  "    \
                                             "or        $0,1,$1;   "    \
                                             "stl_c     $1,($16);  "    \
                                             "sll       $0,1,$0;   "    \
                                             "or        $0,$1,$0   ",   \
                                             (spinlock_address));

// BASIC_SPINLOCK_ACQUIRE implements the simple case of acquiring a spinlock. If
// the spinlock is already owned or the store-conditional fails, this function
// spins until the spinlock is acquired. This function doesn't return until the
// spinlock is acquired.
//
#define BASIC_SPINLOCK_ACQUIRE(spinlock_address)                              \
{       long status = 0;                                                      \
                                                                              \
        while (1)                                                             \
        {                                                                     \
                if (*(spinlock_address) == 0)                                 \
                {                                                             \
                        status = TEST_AND_SET (spinlock_address);             \
                        if (status == 1)                                      \
                       {                                                      \
                                MB;                                           \
                              break;                                          \
                       }                                                      \
                }                                                             \
        }                                                                     \
}
```

Figure 1
Code Sequences for Locking Intrinsics

instruction-cache miss rate of 10 to 12 percent can effectively stall the CPU 70 to 80 percent of the time. Conversely, decreasing the miss rate by 2 percent can increase throughput by 10 percent.

OM performs profile-based optimization. A program is first partitioned into basic blocks (that is, regions containing only one entrance and one exit), and instrumentation code is added to count the number of times each block is executed. The instrumented version of the program is run to create a feedback file that contains a profile of basic block counts. OM then uses the feedback to rearrange the blocks in an optimal way for the first-level caches on the Alpha chip. The details of the procedure for using OM may be found in the manpage for cc on the Digital UNIX operating system but can be summarized as follows:

- Build executable with -non_shared -om options, producing prog.
- Use pixie to produce prog.pixie (the instrumented executable) and prog.Addrs (addresses).
- Run prog.pixie to produce prog.Counts, which records the basic block counts.
- Now build prog again with -non_shared -om -WL, om_ircorg_feedback.

VLM Results

Figure 2 shows the increase in throughput realized when using VLM. Note that throughput nearly doubles as the amount of memory allocated to the database cache is varied from 1 GB to 6 GB. Of course, the overall system requires additional memory beyond the database cache to run UNIX itself and other

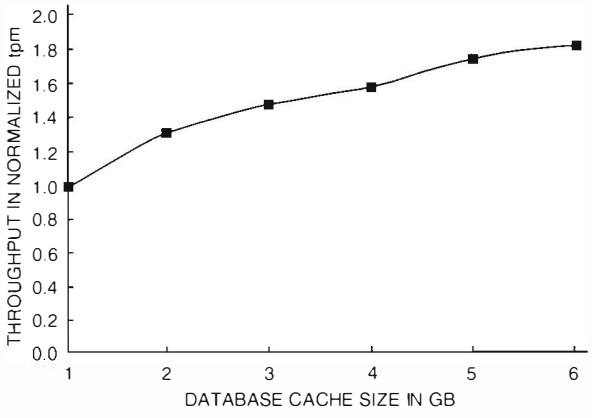


Figure 2
Database Cache Size versus Throughput

processes. For example, an 8-GB system allows 6.6 GB to be used for the database cache.

Performance Analysis

Why does the use of VLM improve performance by a factor of nearly 2? Using statistics within the database, we measured the database-cache hit ratio as memory was added. Figure 3 shows the direct correlation between more memory and decreased database-cache misses: as memory is added, the database-cache miss rate declines from 12 percent to 5 percent. This raises two more questions: (1) Why does the database-cache miss rate remain at 5 percent? and (2) Why does a small change in database-cache miss rates improve the throughput so greatly?

The answer to the first question is that with a database size of more than 100 GB, it is not possible to cache the entire database. The cache improves the transactions that are read-intensive, but it does not entirely eliminate I/O contention.

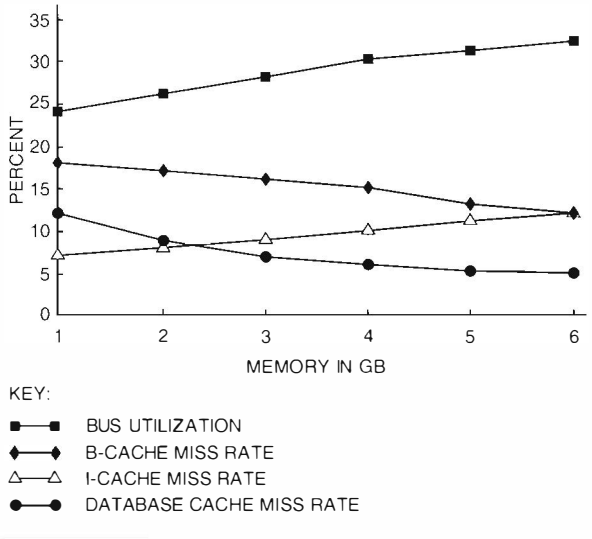


Figure 3
Cache Miss Rates and Bus Utilization

To answer the second question, we need to look at the AlphaServer 8400 system's hardware counters that measure instruction-cache (I-cache) miss rate, board-cache (B-cache) miss rate, and the bandwidth used on the multiprocessor bus. With an increase in throughput and memory size, the VLM system is spanning a larger data space, and the bus utilization increases from 24 percent to 32 percent. Intuitively, one might think this would result in less optimal instruction-and data-stream locality, thus increasing both miss rates. As shown in Figure 3, this proved true for instruction stream misses (I-cache miss rate) but not true for the data stream, as represented by the B-cache miss rate. The instruction stream rarely results in B-cache misses, so B-cache misses can be attributed primarily to the data stream.

Performance analysis requires careful examination of the throughput of the system under test. The apparent paradox just related can be resolved if we normalize the statistics to the throughput achieved. Figure 4 shows that the instruction-cache misses per transaction declined slightly as the memory size was increased from 1 GB to 6 GB—and as transaction throughput doubled. Furthermore, the B-cache works substantially better with more memory: misses declined by 2X on a per-transaction basis. Why is this so?

Analysis of the system monitor data for each run indicates that bringing the data into memory helped reduce the I/O per second by 30 percent. If the transaction is forced to wait for I/O operations, it is done asynchronously, and the database causes some other thread to begin executing. Without VLM, 12 percent of transactions miss the database cache and thus stall for I/O activity. With VLM, only 5 percent of the transactions miss the database cache, and the time to perform each transaction is greatly reduced. Thus each thread or process has a shorter transaction latency. The shorter latency contributes to a 15-percent reduction in system context switch rates. We attribute the measured improvement in hardware miss rates per transaction when using VLM to the improvement in context switching.

The performance counters on the Alpha microprocessor were used to collect the number of instructions issued and the number of cycles.[9] In Table 2, the relative instructions per transaction results are the ratios of instructions issued per second divided by the number of new-order transactions. (In TPC-C, each transaction has a different code path and instruction count; therefore the instructions per transaction amount is not the total number of new-order transactions.) The relative difference between instructions per transaction for 1 GB of database memory versus 6 GB of database memory is the measured effect of eliminating 30 percent of the I/O operations, satisfying more transactions from main memory, reducing context switches, and reducing lock contention.

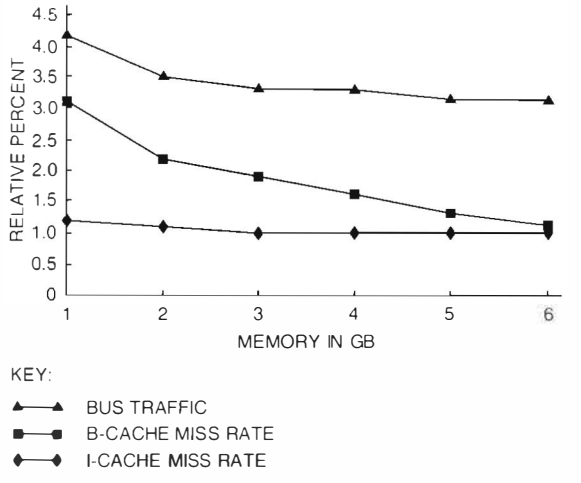


Figure 4
Normalized Cache Miss Rates and Bus Traffic

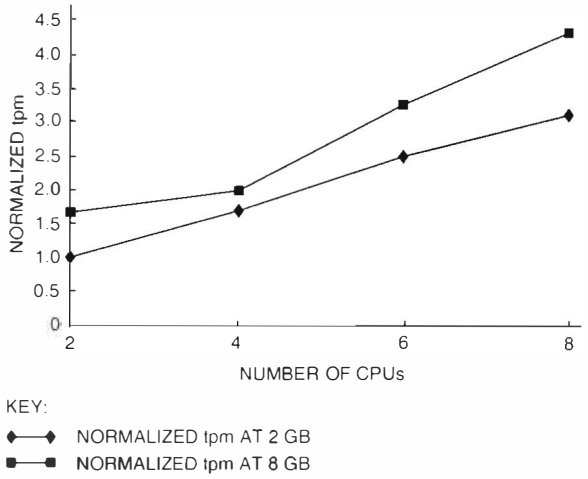


Figure 5
CPU Scaling versus Memory

Improved CPU Scaling—More Efficient Locking

A final benefit of using VLM is improved symmetric multiprocessing (SMP) scaling. Because the TPC-C workload has several transactions with high read content, having the data available in memory, rather than on disk, allows an SMP system to perform more efficiently. More requests can be serviced that are closer in cycles to the CPU. Data found in memory is less than a microsecond away, whereas data found on disk is on the order of milliseconds away.

We have shown how this situation improves the overall system throughput. In addition, it improves SMP scaling. Figure 5 shows the relative scaling between 2 CPUs and 8 CPUs with only 2 GB of system memory (1.5 GB of database cache) compared to the same configurations having 8 GB of system memory (6.6 GB of database cache).

We used the performance counters on the Alpha 21164 microprocessor to monitor the number of cycles spent on the memory barrier instruction.[9] Memory barriers are required for implementing mutual exclusion in the Alpha processor. They are used by all locking primitives in the database and the operating system. With VLM at 8 GB of memory, we measured a 20-percent decline in time spent in the memory barrier instruction. Larger memory implied less contention for critical disk and I/O channel resources and thus less time in the memory barrier instruction.

Conclusions

Open system database vendors are expanding into mainframe markets as open systems acquire greater processing power, larger I/O subsystems, and the ability to deliver higher throughput at reasonable response times. To this end, Digital's AlphaServer 8400 5/350 system using VLM database technology has demonstrated substantial gains in commercial performance when compared to systems without the capability to use VLM. The use of up to 8 GB of memory helps increase system throughput by a factor of 2, even for databases that span 50 GB to 100 GB in size.

The Digital AlphaServer 8400 5/350 system combined with the Digital UNIX operating system to address greater than 2 GB of memory has made possible improved TPC-C results from several vendors. In this paper, we have shown how VLM

- Increased the throughput by a factor of nearly 2
- Increased the database-cache hit ratios from 88 percent to 95 percent

By using monitor tools designed for the Alpha platform, we have measured the effect of VLM in issuing fewer instructions per transaction on the Alpha 21164 microprocessor. When transactions are satisfied by data that is already in memory, the CPU has fewer hardware cache misses, fewer memory barrier processor stalls, faster locking, and better SMP scaling.

Future Digital AlphaServer systems that will be capable of using more physical memory will be able to further exploit VLM database technology. The results of industry-standard benchmarks such as TPC-C, which force problem sizes to grow with increased throughput, will continue to demonstrate the realistic value of state-of-the-art computer architectures.

Acknowledgments

Many people from a variety of groups throughout Digital helped tune and deliver the TPC-C results. In particular, we would like to thank Lee Allison, Roger Deschenes, Joe McFadden, Bhagyam Moses, and Cheryl O'Neill (CSD Performance Group); Jim Woodward (Digital UNIX Group); Sean Reilly, Simon Steely, Doug Williams, and Zarka Cvetanovic (Server

Engineering Group); Mark Davis and Rich Grove (Compilers Group); Peter Yakutis (I/O Performance Group); and Don Harbert and Pauline Nist (project sponsors).

References and Notes

1. D. Fenwick, D. Foley, W. Gist, S. VanDoren, and D. Wissell, "The AlphaServer 8000 Series: High-end Server Platform Development," *Digital Technical Journal*, vol. 7, no. 1 (1995): 43–65.

2. At the time this paper was written, 2 GB was the largest size module. Digital has announced that a 4-GB option will be available in January 1997.

3. L. Wilson, C. Neth, and M. Rickabaugh, "Delivering Binary Object Modification Tools for Program Analysis and Optimization," *Digital Technical Journal*, vol. 8, no. 1 (1996): 18–31.

4. Transaction Processing Performance Council, *TPC Benchmark C Standard Specification, Revision 3.0*, February 1995.

5. W. Kohler, A. Shah, and R. Raab, *Overview of TPC Benchmark: The Order Entry Benchmark*, Technical Report (Transaction Processing Performance Council, December 1991).

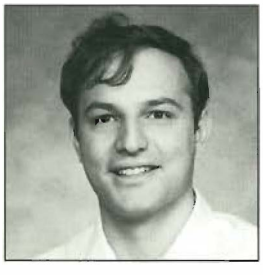
6. More information about the TPC-C benchmark may be obtained from the TPC World Wide Web site, http://www.tpc.org.

7. J. Shakshober and B. Waters, "Improving Database Performance on Digital Alpha 21064 with OM and Spinlock Optimizations" (CSD Performance Group, Digital Equipment Corporation, July 1995).

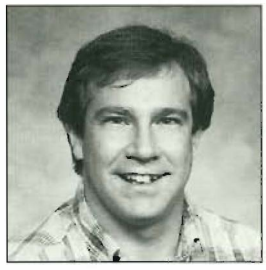
8. R. Sites, ed., *Alpha Architecture Reference Manual* (Burlington, Mass.: Digital Press, 1992).

9. B. Wibecan, *Guide to IPROBE* (Digital Equipment Corporation, December 1994).

Biographies



Tareef S. Kawaf
Tareef Kawaf received a B.S. in computer science (magna cum laude) from the University of Massachusetts at Amherst. He is a member of Phi Beta Kappa. Tareef joined Digital in 1994 to work on performance enhancements and tuning of high-end systems and is a senior software engineer in the CSD Performance Group. He worked on attaining the world record-setting TPC-C results on the AlphaServer 8400 5/300 and 5/350 systems and the four-node AlphaServer 8400 5/350 cluster system running a state-of-the-art database application. Tareef has received two excellence awards from Digital for his work in TPC-C performance measurement on the AlphaServer 8000 series.



D. John Shakshober
John Shakshober is the technical director of the CSD Performance Group. The Computer Systems Division Performance Group evaluates Digital's systems against industry-standard benchmarks such as those of the Transaction Processing Performance Council (TPC) and the Standard Performance Evaluation Corporation (SPEC). In this function, John has been responsible for integrating Digital's state-of-the-art software technologies with Digital's Alpha-based products since their introduction in 1992. Prior to joining the CSD Performance Group, John modeled the performance of the 21064 and 21164 Alpha 64-bit VLSI microprocessors and was a member of the VAX 6000 Hardware Group. He joined Digital in 1984 after receiving a B.S. in computer engineering from the Rochester Institute of Technology. John also received an M.S. in electrical engineering from Cornell University in 1988.

David C. Stanley
Dave Stanley joined Digital in 1984. He is a principal software engineer in the CSD Performance Group and was the project leader for the TruCluster system that achieved a world-record result for the TPC-C benchmark. Dave has also led several TPC-C audits on the AlphaServer 8000 series running a state-of-the-art database application. He is a secondary representative at the TPC General Council and a member of the TPC-C Maintenance Subcommittee. Prior to these responsibilities, he was a microprocessor application engineer at Digital Semiconductor, where he ran competitive benchmarks on the MicroVAX II processor chip versus the Motorola 68020. Dave received a B.S.E.E. from the State University of New York at Buffalo (1981).

Building Collaboration Software for the Internet

Dah Ming Chiu
David M. Griffin

Collaboration software for the Internet's World Wide Web involves the development of shared information systems for network computing. The AltaVista Forum version 2.0 software from Digital contains extensions to World Wide Web technology that facilitate collaboration on the Internet. The extensions consist of a toolkit and a set of collaboration applications. The toolkit components include a built-in database with an indexing and search capability. Generic applications include discussion, document sharing, and calendar applications and administrative functions for managing users, teams, and access control.

The Internet and the World Wide Web (WWW) have changed the scope of network computing. As the Internet user population has grown, so has the demand for better ways to collaborate on the Internet. Some examples include the ability to share and discuss issues of common interest, coauthor documents, and track project status. Although today's WWW is ideal for publishing information, it requires considerable customized programming to support collaboration. The AltaVista Forum version 2.0 product is both a set of collaborative applications and a toolkit (platform) that facilitates easy, efficient, and rapid development of collaborative applications for the Internet for both UNIX and Windows NT systems.

In this paper, we describe our experiences in building collaboration software for the Internet. We begin with a brief discussion of WWW technology and groupware applications. Then we present our design philosophy and the framework of the software and discuss the applications supplied by AltaVista Forum. Following that, we discuss the various experiences gained in developing software for the new Internet paradigm. We conclude the paper by discussing our plans for future development efforts.

World Wide Web Technology

Today's Internet was originally a government-funded computer network that facilitated collaboration among academic researchers. Information exchange was conducted by means of electronic mail (e-mail) and file transfer. Over time, bulletin-board style discussions were supported by the Network News Transfer Protocol (NNTP), which propagated textual discussion threads to a large number of NNTP servers for viewing. With the development of the WWW technology, collaborating over the Internet has become even easier.

The WWW technology consists of the following elements:

- Universal resource locator (URL), a convention for information naming and linking
- Hypertext markup language (HTML), a text-based language for information rendering

- Hypertext Transfer Protocol (HTTP), a simple client-server protocol to transport information associated with a URL
- Web browser, a program that renders HTML documents, provides URL caching, and supports a directory for URLs
- Web server, a server that responds to requests for information from the Web browsers

Information Access

WWW technology has transformed the way users access information through computer networks. Access to information on the Internet was primarily text-based; with the WWW, users are able to access information in multimedia format. The combination of functionality (information linking, graphical interface, and caching), extensibility (for dealing with new protocols and new information types), ease-of-use, and low cost appealed to a wide range of users in homes, offices, and corporations. In addition, the Mosaic-style of "point-and-click" graphical Internet browser has become the most widely accepted user interface for network computing.

The most popular use of the WWW today is for publishing information, and the process is comparable to the way a newspaper publishes or a television station broadcasts information. The roles of the information provider and the information consumer are clearly defined. The information provider gathers and organizes the pertinent information, converts it to the HTML scripting format, and makes it available on a Web server. The information consumer, after obtaining initial access to the Web server (as one might tune into the correct television station), can then browse and search for various types of information available on that server. The linking capability of URL and HTML allows the references or links to additional information on various servers to be easily published along with the original information.

In contrast, multiple information providers work in collaboration to generate the content of shared information. For the purposes of this paper, we will assume that there is only one type of user—information collaborators.

Collaboration and Groupware

The WWW is useful for many types of collaboration. For example, a project team may need to keep track of project status and individual progress; people with a common interest (e.g., film enthusiasts) may want to share and discuss their views on that topic; a customer support group may need a system to provide on-line answers to real-world customer problems; or several authors may wish to work on a document together.

Today, several computer applications facilitate such collaboration. Collectively, these applications are known as groupware. Lotus Notes is a popular groupware application. Typically, groupware applications support the following capabilities:

- Management of a set of users and groups
- Storage of shared information in a database (sometimes with replication capability)
- Viewing information stored in the databases by means of a graphical interface
- Protection of the collaboration environment when necessary through authentication and access control

Groupware systems are built to run in homogeneous client environments, such as the Microsoft Windows environment. They rely on specific client-server technology, which is often proprietary, to support remote operations.

The popularity and rapid growth of the Internet and the WWW have created an open, universal, and easy-to-program infrastructure that can readily serve several groupware functions. Engineers at Digital's Internet Software Business Group recognized the potential of using the WWW as the underlying infrastructure for groupware solutions and at the same time saw that the groupware applications available today have features that the WWW lacks. Our goal was to add groupware features to the WWW to facilitate collaboration.

We started exploring the idea of using the Internet and the WWW for groupware applications in the summer of 1994. By the end of that year, we had built a prototype that supported the simplified discussion (bulletin-board) features of an internal product known as DEC Notes.[1] This prototype generated considerable interest among active DEC Notes users who were seeking a similar solution built around an Internet infrastructure. Based on their feedback, the prototype was redesigned and became a product.[2]

By September 1995, we had built several collaborative applications to run over the WWW. In a workshop organized by the World Wide Web Consortium and the Massachusetts Institute of Technology, we participated in discussions on how to extend the WWW technology to support collaboration. All the workshop participants presented their ideas to the WWW Consortium for review.[3]

Design

In this section, we summarize our design philosophy and discuss the framework and applications developed for the AltaVista Forum product. For our design, we adopted an object-oriented approach, which meant that we would have to modularize the various components for reuse and modification.

Design Philosophy

Our fundamental design philosophy required using the Internet and its infrastructure as building blocks for our collaboration software. After years of experimenting and collaborating to develop an open process, the Internet developers realized that the Internet had reached a state of critical mass. In the case of networks and connectivity, reaching critical mass is a tremendous impetus for agreeing on a common standard. As more and more users access the Internet, the need for software development for the Internet also increases. In addition, the very nature of the Internet demands an open standardization process to ensure the long-term viability of a product.

Our philosophy also included the reuse of existing open software as building blocks whenever possible. In addition to our choice of building upon the Internet and the WWW technology, we selected the Tool Command Language (Tcl) as the primary language for developing most of our application and user interface functions.[4] We also took advantage of the database library in the Berkeley UNIX distribution for built-in database support.[5]

Another objective was to make sure our software would be easy to port to all the relevant operating system platforms. This principle guided our selection of components and helped us isolate a small set of platform-dependent functions into a special library for porting the software.

As stated earlier, we tried to take an object-oriented approach whenever possible. The advantages of our approach became increasingly apparent as more people became involved with the software development. The object-oriented approach made component reuse feasible.

Framework

Our framework organizes the AltaVista Forum software into two layers: toolkit and applications. The tools required to build the applications overlap each other. We have used them to build generic applications, including a discussion application that supports users discussing a set of related topics, much like newsgroups do; a calendar application that supports users' abilities to schedule events on a specific date and at a particular time; and a newspaper application that provides a personalized news filtering service. We envision that, over time, the framework we have developed will support a number of diverse applications. Figure 1 shows the AltaVista Forum toolkit and application layers.

The toolkit is a combination of both C and Tcl code that creates the following interface components:

- Built-in database. The application uses a built-in database to store its object instances. The database is a very simple relational model with an object hierarchy relationship facility available to those applications that need it. The library also provides inversions on certain attributes to support fast retrieval and sorting based on attribute values.
- Built-in indexing and search. An indexing and search function complements the database by providing a high-speed query facility. For less-structured objects, it is often easier to index them and look them up using a search tool.
- Graphical user interface support. The use of a graphical user interface insulates applications from having to deal with HTML directly and cope with its changes over time. Abstract definitions of user interface objects also tend to simplify and clarify the code and create a more uniform appearance on the screen.
- Access control. All applications require some form of access control to regulate who can access, create, modify, and delete various objects.
- Internationalization. An internationalization facility gathers strings that appear in the user interface into message catalogs for later translation to different languages.
- Platform-specific support. A special library isolates those operating system—dependent functions that vary from platform to platform. Certain file system accesses and date/time library accesses are examples of this component.

Armed with all the components in the toolkit, an AltaVista Forum application consists of a set of functions, each responding to a different user request. The organization of an application is modular. A function can call various objects that are defined separately as part of the application, including the following:

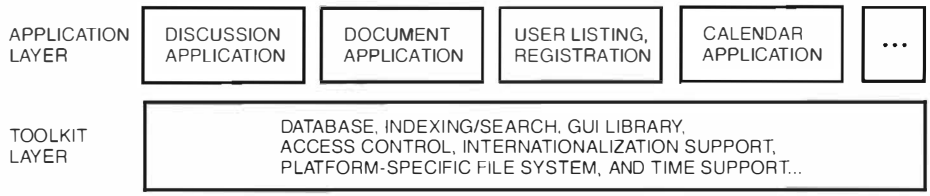


Figure 1
AltaVista Forum Toolkit and Application Layers

- Graphical objects such as definitions of buttons, toolbars, various objects that are part of a form (e.g., select boxes, radio buttons, check boxes, text boxes), and icons.
- Database entries, the definitions of their attributes, and default values.
- User interface aggregate objects such as forms, views, dialogs, and error messages.
- Default access control policies, including default groups, access rights, and their mappings, to control who can access individual forums and what actions they can take within them.

This approach encapsulates the details in low-level modules, making the software more readable and maintainable. It also makes it easy for different functions to reuse the objects.

To further facilitate code sharing, the framework also allows applications to inherit a set of functions and objects that have been grouped together as a pseudoapplication. For example, the access control management functions can be grouped into a pseudoapplication and certain button and toolbar definitions can be grouped into another pseudoapplication. All applications that need access control and the common graphical objects that lend a consistent "look-and-feel" can inherit those functions and objects from pseudoapplications.

The AltaVista Forum product works in conjunction with the Web browser and the Web server. The Web browser submits requests to the Web server whenever the user opens a link. If the link points to a file, then the Web server sends the file to the browser, which is the normal interaction. The link can also point to programs on the server; in this case, the Web server invokes the program and then the program responds to the user.

When the link points to the AltaVista Forum, the Web server invokes the AltaVista Forum dispatcher program through the common gateway interface (CGI). Based on the information passed along with the user request, the dispatcher invokes a specific application, which, in turn, calls various tools in the toolkit to respond to the user's request. Figure 2 illustrates the interaction of the AltaVista Forum software with the Web browser and server.

Parameters are passed to the dispatcher from segments of the URL. The dispatcher parses the URL into the pieces that provide the overall control of the program: (1) the forum name, (2) the access control area name, (3) the message name, and (4) the message arguments.

Each forum is an instance of an application object. For example, many discussion forums are available on various topics. Each discussion forum has its own name at the time of creation; however, the same discussion application can be used to manage all the forums.

An access control area contains a set of forums and a common user/group database. An administrator group helps administer the user/group database and establish overall access control policies for the environment. A user registers only once with an access control area. Based on the access control area location, the hypertext server not only knows where to find the user's credentials for authentication purposes but also knows how to authenticate the user and pass the authenticated user identity to the AltaVista Forum environment. Given the user identity and the access control location, AltaVista Forum software can also look up the user profile, check access control, and perform other user-specific functions.

The message name and message arguments then select particular actions to perform within the application.

Generic Applications

The AltaVista Forum product supplies a set of generic applications that make the software immediately usable. The applications are described in this section.

User and Group Management and Lookup This application provides an interface for user registration (either by the user or by an administrator). Users can supply and modify their business card information such as phone numbers and e-mail addresses. Users can also set certain preference parameters that help the AltaVista Forum software tailor its responses (e.g., native language and preferred display formats). In addition, groups can be created and modified as a set of users. This application also provides the interface for listing and searching for user and group information for all forums. As discussed earlier, the AltaVista

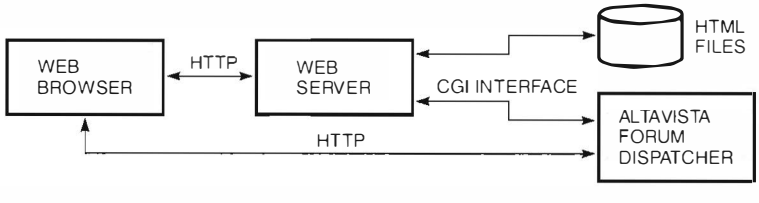


Figure 2
Interaction of AltaVista Forum, Web Browser, and Web Server

Forum product can support all the users that a Web server can handle since only one repository of users and groups is necessary.

Community, Team, and Personal Vistas A vista is another term for home page, which is a place for the user to log in to the WWW. Once in the community vista, the user sees a set of public forums and links to perform various tasks, e.g., register oneself, look up teams or join a team, perform AltaVista Forum administrative tasks (if an administrator), and so on. For this reason, the community vista is also called the summit. In much the same way, a team vista keeps track of all the forums and links for a group of users, and a personal vista performs this function for a single user. Both team and personal vistas can own forums that are not visible to the public community vista.

Discussion Much like a bulletin-board discussion group or Digital's DEC Notes software, this application permits users to share ideas on a set of related topics. Users create topics and replies that form a hierarchical tree (also known as threaded topics), providing a way for users to navigate through existing discussions. Other methods of reading the existing discussion are also provided. These include chronologically navigating through items not read; listing unread items only and selectively reading them; and searching for topics and replies containing certain words that were entered during a particular time period by a certain author. Users can also create multiple discussion forums to discuss different topics; this is true for the following applications as well.

Document Sharing The document sharing application enables users to organize documents of the same type into hierarchically organized folders. In addition, it keeps track of versions of the documents, attachments, and comments. As with the discussion application, users can browse through and search for specific documents using a variety of methods.

Newspaper The newspaper application lets users select a specific source of information and then define filters to present only those items of potential interest. A good example of an information source on the Internet is one of the real-time news feeds. Using the newspaper application, it is also possible to read and monitor other information sources, e.g., e-mail sent to a distribution list or information appearing on a set of WWW sites.

Calendar The calendar application permits users to enter a set of scheduled events (or a to-do list) and present the events as a calendar (sometimes called a diary). The application supports requests to add items to the calendar, thus allowing the calendar to be used as a scheduling tool. Although a calendar forum can be set up for each person, it is equally useful to have a team calendar, a community calendar, or even a calendar for a specific type of event.

Experiences

In this section, we summarize some of our experiences and discuss the lessons we learned along the way. As a result of our decision to rely on the Web browser as the universal user interface, we had to resolve some unique user interface issues. Because we chose to use Tcl for developing higher-level objects, we had to cope with using an interpretive language. We designed the database and indexing and search interfaces based on extensibility and portability goals. Finally, in the design of access control, we had to carefully weigh the pros and cons of simplicity and flexibility.

Coping with the User Interface Defined in HTML

Very early in the design phase, we decided to make AltaVista Forum client-independent, with the exception of dependence on the Web browser. This decision was based on the fact that the Web browser was already freely available on most of the platforms. We expected the browser to become a ubiquitous network front end, allowing us to focus on building groupware functions on the server. This meant that we were faced with the task of designing the user interface using HTML.[6]

Since HTML was evolving, our first step was to define graphical objects in more abstract constructs supported by our toolkit. Each construct encapsulates the specifics into a representation of a graphical artifact in HTML in the toolkit. Thus as HTML evolves, or as the page design changes, only one area needs to be updated. For example, a select box object on a form may be defined as follows:

```
forum selectbox s.language \
      -mapto language \
      -label "Select a language:" \
      -labelbreak

s.language add_option English 1 selected
s.language add_option French 2
```

In this example, a select box is defined to begin with a label and some spacing and then to contain two options: English and French, with English as the default. The values 1 and 2 are internal representations of the selected values. Also, the "map-to" switch specifies that this object must correspond to the language attribute in the database, a feature that was included to simplify database update.

Note that although a label is specified, no specification is provided to represent that label in a particular font or typeface. Neither is the actual spacing for label break specified. These decisions are made in the forum

select box part of the toolkit procedure, which translates this object into HTML.

Most of the early Web browsers were single-window based. This limitation was especially problematic for us because most of our applications provide some organization to the information content. A much more natural way of browsing for our environment would include at least two windows: one showing the context and the other showing the content of a specific item. For this reason, we introduced multiple navigational methods. For example, the discussion application

- Allows hierarchical navigation (previous, next, up)
- Allows navigation in chronological order (next unseen, what's new)
- Provides a category view that lists topics according to their category
- Supports content-based search or an index-like function

Newer versions of Web browsers support frames, which have multiple window-browsing capabilities (although the standards in this area are still a bit vague). We are updating our applications to take advantage of these new features.

Usability studies guided our decisions as we were designing forms and dialog boxes. It is likely that many potential users of our product are familiar with Windows-style user interface objects. Because the early Web browsers (e.g., Mosaic) were UNIX-based, little attention was given to providing a human–computer interface that resembled the more widely used Windows interface. However, our usability studies indicated that many personal computer (PC) users had difficulty using Web browsers out-of-the-box. For example, a user might expect a dialog box to have certain standard buttons, such as OK, cancel, and clear. Ideally, the user would know what to do with these buttons without any training. To make our software easy to learn, we tried to follow the same user interface style that was already familiar to most users. Since we were limited by HTML and browser design, this was not a simple task. Thus we were often forced to produce rough facsimiles of the more well-known interface artifacts.

In summary, we found usability studies to be extremely valuable when designing end-user applications. For this reason, it is important to allocate enough time in the product design cycle to collect user feedback before beginning product development.

The Pros and Cons of Using an Interpretive Language

As mentioned earlier, we selected Tcl as the language for building the AltaVista Forum toolkit. Tcl is a highly portable, extensible, and freely available language that was originally designed to be embedded in a larger framework.[4] However, it is also an interpretive language, which supported our goal of rapid and iterative development of collaborative applications for the WWW.

We extended standard Tcl to provide a set of commands and objects that formed the AltaVista Forum toolkit: database, HTML generation, access control, internationalization, user profile management, and platform-specific support. Many of these extensions supported an object-based environment (i.e., the environment supported standard Tcl objects and our simple inheritance mechanism). The use of these extensions made it easier to develop applications than it would have been with Tcl (or any other language), alone. As a result, these extensions form the basis for future development tools.

From the beginning, we knew that the choice of an interpretive language was going to involve trade-offs. In fact, performance, which was our most critical trade-off, continues to be a concern for the engineering team. Although the performance of an interpreted language is lower than that of a compiled language, fast processors have made the use of an interpreter worthwhile because of the reduced expense of developing applications. The use of Tcl in the AltaVista Forum software certainly takes advantage of this. Although the applications and part of the toolkit are written in Tcl, many critical parts are implemented in a compiled language (such as C) to stay within performance requirements. The engineering team is continually searching for ways to improve performance while accommodating requests for new features and tracking the rapidly evolving WWW environment.

The second trade-off was the absence of a sophisticated debugging and profiling environment. Partly due to the limitations of Tcl and partly due to the stateless nature of WWW transactions, some of the more sophisticated development tools that programmers expect to see are not readily available. Despite these shortcomings, rapid development is still possible; however, we expect even larger gains as we correct these problems in the future.

Interfacing to the Database

Several factors (primarily portability and cost) influenced our decision to build a hybrid database rather than the more customary relational database. The database in the AltaVista Forum toolkit consists of a B-tree indexed file (from the Berkeley ndbm package) for storage of basic attributes about documents, which is backed by the file system for the nonstructured data. This design, combined with the search engine (described in the next section), is quite effective for the types of applications we initially developed with the AltaVista Forum toolkit.

In effect, the database is organized as a collection of documents (or entries) that have unique identifiers (document IDs), hierarchical document numbers, and

a set of attributes that is similar to a relational database table. The toolkit provides each entry with a set of built-in attributes (such as title, creation and modification dates, and author). The applications can then deliver additional attributes.

The toolkit provides the means to retrieve, modify, and iterate through the collection of entries in a straightforward manner. Because the attributes are part of the application description and are not stored in a separate database, the toolkit can use its knowledge of the attributes to simplify certain common operations. For example, because transferring data from HTML forms to the database and back is a basic operation in collaborative applications, the toolkit can link fields on forms to database attributes, making it possible to store them with a single command. To support a dynamic development environment, the toolkit also upgrades databases in real time as new attributes are added or deleted. This permits the application developer to concentrate on the task at hand rather than worry about database management tasks.

Although the primary organization mechanism is a flat table indexed by document identifiers, the database integrates a hierarchical relationship between entries when necessary. Because hierarchies are common in collaborative applications (e.g., folders/documents and topics/replies), it was important to reflect this in a natural way in the database.

In addition to attributes, the database offers properties. Compared to attributes, which are stored for each entry in the database, properties are stored within each forum. Application designers can use these properties in any way they desire: they are simple key-value relationships. The AltaVista Forum software uses properties to implement a variety of features, from access control policies to the background color of the screen display.

User properties are an extension of standard forum properties. They act like forum properties except that they are tied to the user who is executing the transaction. User properties keep database locking to a minimum because, in collaborative applications, a user will typically execute only one transaction at a time.

Indexing and Search: The Way of the Future?

One key design decision was to include an indexing and search engine as a basic component of the product. Although the database is often the central piece of a groupware product, an indexing and search engine often plays a similar role for a WWW site. This development is completely consistent with the philosophy of the WWW—information is linked as needed, not necessarily following any structure. Database use is more suitable for information objects that have some uniformity in their definitions.

The basic function of the indexing engine is to map a set of words to a document containing those words. (The term document is used in a generic sense. It can be any logical entity associated with a set or words.) The indexing information must be stored in such a way that subsequent searches based on individual words (and phrases) are efficient and speedy. The indexing engine in the AltaVista Forum toolkit is basically the same indexing engine available on the AltaVista Web site.[7] Designed and implemented at Digital's System Research Center, it is highly scalable and efficient.

The built-in database functions as a repository for entries with a predefined set of attributes. It provides fast retrieval when the entries are identified using either an entry ID or a hierarchical ID, and it provides simple creating, updating, and sorting functions associated with retrieval. The indexing and search engine complements the AltaVista Forum database: it provides a content-based search method and functions at higher speed. Since the search engine is extremely fast and scalable, we also use it to index some of the attribute values in the database. This allows us to use the search engine for certain compute-intensive searches that otherwise would be performed by the database.

Based on our experience, we expect the capabilities of the indexing and search engine to continue to expand. As the popularity of the WWW technology continues to grow, the volume of published information will also increase. Only a small amount of this information can be effectively captured in databases. The indexing and search engine is an invaluable tool for mining useful information out of the vast amount of data stored in these databases.

The Dilemma of Access Control

Designing access control is very challenging because users and administrators have different requirements. On the one hand, administrators want a high degree of flexibility in controlling access. Their issues include the following:

- What type of information is subject to access control?
- Should access control be defined for every possible access/action type?
- Should there be arbitrary flexibility in defining groups (including nesting)?

On the other hand, users have stated that they do not like products in which access control operations are complex, especially in the case of a product that is supposed to help people collaborate. In a majority of scenarios, they argue that very little access control is needed.

For this reason, we tried to strike a balance between administrators' needs and users' preferences. Although we recognize the importance of access control, we did not give it precedence over product usability. Since usability was our priority, and the time available to

work on it was limited, we divided our efforts between making access control flexible and choosing default options that would promote collaboration.

We defined access control for the whole database (forum), rather than for individual entries and attributes of entries. However, some entry-level access control is necessary. For example, it is preferable to let only the owner (or the creator) of an entry modify and delete that entry. As a result, we allowed the group definition to include entry-specific logical users, rather than provide a general mechanism for entry-level access control. Therefore, a group may contain a member who is the owner of the current entry. During access control checking, the current entry's owner is looked up and matched against the currently logged-in user.

Instead of letting the administrator define access control for each possible incoming access/action, our framework allows the application definition to group accesses together into logical access rights. For example, for the discussion application, we defined the following access rights:

- Read—Includes all read URLs (different views, whether for a single entry or a list of entries)
- Contribute—Includes adding a topic or reply
- Modify—Includes any form of modification or deletion
- Moderate—Includes such functions as creating keywords, polling options, controlling number of levels of replies, and setting certain entries as hidden
- Administrate—Change access control or other kinds of resource consumption policies

By defining these access rights, the administrator only needs to establish who can do these five operations, rather than define numerous other kinds of operations. It is still possible to change and add to this group of access rights by making simple modifications to the application definition.

Our basic strategy for making access control easy to manage is to set up default policies of access control that apply to as many situations as possible, within reason. The default policy is added to the application definition. If the administrator is satisfied with the default policies, then the access control can be used as supplied. For the discussion application, the default policy is the following:

- Read—All users, including anonymous
- Contribute—All users, excluding anonymous
- Modify—Owner (creator) of entry and moderators
- Moderate—Owner of the forum
- Administrate—Owner of the forum

To simplify implementation, we chose not to allow nesting of groups. Our design allows for adding it in the future as long as it makes management of access control policies easier.

Future Directions

To date, we have received encouraging feedback from users. Of the ways that we can continue to improve the AltaVista Forum product, we feel the following deserve the highest priority.

First, we need to provide better ways to help users deal with information overflow. Although we have built ways to filter and search information into our application, further simplification is necessary. We are working on smart agents that bring the relevant information to the user's fingertips.

Second, a number of the functions that we provide can be more easily performed on the client machine. The Java language is the best candidate for providing these functions since it enables us to handle a wide variety of client platforms. Initially, we are looking into using Java to improve certain user interface problems, such as opening additional windows on the client machine to notify users of new information.

Third, synchronous collaboration using video, audio, and whiteboard will soon become feasible and cost effective. It is important for us to help bring users together through both synchronous and asynchronous methods of collaboration. For example, users should be able to use the calendar application to schedule a meeting over the Internet, and Windows should be available to the user automatically.

Fourth, as the AltaVista Forum software matures, we hope to add to its performance and increase its scalability. As its environment evolves, we are looking into ways to bypass the CGI interface and use a compiled language for more of the toolkit implementation. We also hope to add support for large commercial databases.

Finally, we will continue to add innovative applications to our product. We recently built a prototype of a customer-support application that keeps track of problem reporting. We are looking into other applications such as project management, group review, and survey and decision-support systems.

Acknowledgments

We wish to thank the AltaVista Forum development and management teams for their contributions to the product. In particular, we wish to thank Peter Hurley for his leadership in starting the effort; Ralph DeMent, Bob Travis, David Marques, and Rick Frankosky, who have worked with us throughout the lifetime of the product and with whom we have developed a special camaraderie; and Dan Kalikow, who was the first adopter and has cheered us on ever since.

References and Notes

1. DEC Notes is a discussion application running primarily on VAX systems connected on a DECnet network. Still a very popular tool within Digital, it is used for collaborating on many topics, ranging from product development, customer support, and marketing to various personal interest topics.
2. The product was originally called Workgroup Web Forum. It was subsequently merged into a larger family of products and the product name became AltaVista Forum.
3. For more information on the World Wide Web Consortium/MIT Laboratory for Computer Science Workshop on the World Wide Web and Collaboration held September 11–12, 1995, see http://www.w3.org/pub/WWW/Collaboration.
4. J. Ousterhout, *Tcl and the Tk Toolkit* (Reading, Mass.: Addison-Wesley Publishing Company, 1994).
5. *NDBM(3). 4.3 BSD Unix Programming Manual Reference Guide* (University of California, Berkeley, 1986).
6. During this time, Java was still on the drawing board, or at least not generally supported by Web browsers. We did expect to use Java to enhance our user interface over time.
7. For access to Digital's indexing and search engine, visit the AltaVista Web site at http://altavista.software.digital.com.

Biographies

Dah Ming Chiu
Dah Ming Chiu was a consulting engineer in Digital's Internet Software Business Unit and a technical leader in developing the AltaVista Forum groupware product for the Internet. Before that project, he worked for the Networks Architecture Group on congestion and flow control, network monitoring, name service, and the X.500 standard. Previous to that, he worked on performance modeling and analysis network protocols and graphical workstation design. Dah Ming is currently an architect in the Internet Solutions Group of Sun Microsystems, Inc. He received a Ph.D. in applied mathematics (1980) from Harvard University and a B.Sc. in electrical engineering (1975) from the Imperial College, London University. He holds three patents in the areas of congestion control and network monitoring and is a coauthor of *Network Monitoring Explained.*

David M. Griffin
Dave Griffin joined Digital in 1981. He is a principal software engineer in the AltaVista Collaboration Engineering Group, where he leads the AltaVista Forum Toolkit team for version 3.0. Dave also led the toolkit team for version 2.0 and was the primary designer and implementer of the (Workgroup Web Forum) version 1.0 toolkit and the author of the document-sharing application for version 1.0. Prior to this work, Dave led the DECdns server project (part of the DECnet/OSI program) and designed and implemented the hierarchical cells and cell-renaming facilities in the DCE Cell Directory Service. He has been involved in the development of a number of distributed information systems for Digital and other companies. He holds two patents in distributed systems technology.

Further Readings

The *Digital Technical Journal* is a refereed, quarterly publication of papers that explore the foundations of Digital's products and technologies. *Journal* content is selected by the Journal Advisory Board, and papers are written by Digital's engineers and engineering partners. Engineers who would like to contribute a paper to the *Journal* should contact the managing editor, Jane Blake, at Jane.Blake@ljo.dec.com.

Topics covered in previous issues of the *Digital Technical Journal* are as follows:

Spiralog Log-structured File System/OpenVMS for 64-bit Addressable Virtual Memory/ High-performance Message Passing for Clusters/ Speech Recognition Software
Vol. 8, No. 2, 1996, EY-N6992-18

Digital UNIX Clusters/Object Modification Tools/eXcursion for Windows Operating Systems/ Network Directory Services
Vol. 8, No. 1, 1996, EY-U025E-TJ

Audio and Video Technologies/UNIX Available Servers/Real-time Debugging Tools
Vol. 7, No. 4, 1995, EY-U002E-TJ

High Performance Fortran in Parallel Environments/Sequoia 2000 Research
Vol. 7, No. 3, 1995, EY-T838E-TJ
(Available only on the Internet)

Graphical Software Development/Systems Engineering
Vol. 7, No. 2, 1995, EY-U001E-TJ

Database Integration/Alpha Servers & Workstations/ Alpha 21164 CPU
Vol. 7, No. 1, 1995, EY-T135E-TJ
(Available only on the Internet)

RAID Array Controllers/Workflow Models/PC LAN and System Management Tools
Vol. 6, No. 4, Fall 1994, EY-T118E-TJ

AlphaServer Multiprocessing Systems/DEC OSF/1 Symmetric Multiprocessing/Scientific Computing Optimization for Alpha
Vol. 6, No. 3, Summer 1994, EY-S799E-TJ

Alpha AXP Partners—Cray, Raytheon, Kubota/ DECchip 21071/21072 PCI Chip Sets/DLT2000 Tape Drive
Vol. 6, No. 2, Spring 1994, EY-F947E-TJ

High-performance Networking/OpenVMS AXP System Software/Alpha AXP PC Hardware
Vol. 6, No. 1, Winter 1994, EY-Q011E-TJ

Software Process and Quality
Vol. 5, No. 4, Fall 1993, EY-P920E-DP

Product Internationalization
Vol. 5, No. 3, Summer 1993, EY-P986E-DP

Multimedia/Application Control
Vol. 5, No. 2, Spring 1993, EY-P963E-DP

DECnet Open Networking
Vol. 5, No. 1, Winter 1993, EY-M770E-DP

Alpha AXP Architecture and Systems
Vol. 4, No. 4, Special Issue 1992, EY-J886E-DP

NVAX-microprocessor VAX Systems
Vol. 4, No. 3, Summer 1992, EY-J884E-DP

Semiconductor Technologies
Vol. 4, No. 2, Spring 1992, EY-L521E-DP

PATHWORKS: PC Integration Software
Vol. 4, No. 1, Winter 1992, EY-J825E-DP

Image Processing, Video Terminals, and Printer Technologies
Vol. 3, No. 4, Fall 1991, EY-H889E-DP

Availability in VAXcluster Systems/Network Performance and Adapters
Vol. 3, No. 3, Summer 1991, EY-H890E-DP

Fiber Distributed Data Interface
Vol. 3, No. 2, Spring 1991, EY-H876E-DP

Transaction Processing, Databases, and Fault-tolerant Systems
Vol. 3, No. 1, Winter 1991, EY-F588E-DP

VAX 9000 Series
Vol. 2, No. 4, Fall 1990, EY-E762E-DP

DECwindows Program
Vol. 2, No. 3, Summer 1990, EY-E756E-DP

VAX 6000 Model 400 System
Vol. 2, No. 2, Spring 1990, EY-C197E-DP

Compound Document Architecture
Vol. 2, No. 1, Winter 1990, EY-C196E-DP

Technical Publications by Digital Authors

R. Abugov and X. Dietrich, "A Yield Based Replacement for Capability Indexes," *Advanced Semiconductor Manufacturing Conference and Workshop* (November 1995).

P. Bhat, "A Case for a Knowledge-based Performance Tuning Advisor," *CMG95: Proceedings of the 21st Annual Conference for the Resource Management and Performance Evaluation of Enterprise Computing Systems* (December 1995).

C. Boutin, "From Manager to Individual Contributor–Would You Rather Be a Worker Bee?" *Proceedings of the 43rd Conference of the Society for Technical Communication* (May 1996).

W. Bowhill, "A 300MHz CMOS RISC Microprocessor," *IEEE Journal of Solid State Circuits* (November 1995).

C. Brench, "Modeled and Measured Results from Two Standard EMI Problems," *IEEE Transactions on Electromagnetic Compatibility* (August 1995).

A. Charny, "Scalability Issues for Explicit Rate Allocation in ATM Networks," *IEEE Infocom '96: Proceedings of the 15th Annual Conference of the IEEE Computer and Communications Societies* (March 1996).

A. Charny, "Timescale Analysis for Explicit Rate Allocation in ATM Networks," *IEEE Infocom '96: Proceedings of the 15th Annual Conference of the IEEE Computer and Communications Societies* (March 1996).

J. Clement, "Pulsed-current Duty Cycle Dependence of Electromigration-induced Stress Generation in Aluminum Conductors," *IEEE Electron Device Letters* (May 1996).

T. Collins, "POLYCENTER License System: Enabling Electronic License Distribution," chapter 10 in *Integrated Network Management V: Proceedings of the Fourth International Symposium on Integrated Network Management* (London: Chapman & Hall, ISBN 0-41271-570-8, 1995).

T. Collins, "The Wolf as a Metaphor for Software Agent," chapter five in *Bots and Internet Beasties* (Indianapolis, Ind.: Sams.net Publishing, ISBN 1-57521-016-9, 1996).

A. Conn, "Time Affordances: The Time Factor in Diagnostic Usability Heuristics," *Human Factors in Computing. CHI '95 Proceedings* (May 1995).

Z. Cvetanovic, "Performance Characterization of the Alpha 21164 Microprocessor Using TP and SPEC Workloads," *Proceedings of the IEEE Second International Symposium on High-performance Computer Architecture* (February 1996).

M. Desai, R. Cvijetic, and J. Jensen, "Sizing of Clock Distribution Networks for High Performance CPU Chips," *Proceedings of the 33rd Design Automation Conference* (June 1996).

M. Desai and Y. Yen, "A Systematic Technique for Verifying Critical Path Decays in a 300MHz Alpha CPU Design Using Circuit Simulation," *Proceedings of the 33rd Design Automation Conference* (June 1996).

M. Elbert and R. Howe, "Manufacturing Process Study and Certification," *IEEE 34th Annual Spring Reliability Symposium* (April 1996).

L. Elliott, R. Shuman, J. Rose, and T. Spooner, "The Electromigration and Failure Behaviour in Layered Tungsten Via Structures," *Materials Research Society Symposium Proceedings* (April 1995).

J. Emer et al., "Predictive Sequential Associative Cache," *Proceedings of the Second International Symposium on High-performance Computer Architecture* (February 1996)

J. Emer, R. Stamm et al., "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," *Proceedings of the IEEE ACM 23rd International Symposium on Computer Architecture* (May 1996).

A. Flanders and M. Raven, "Using Contextual Inquiry to Learn about your Audiences," *The Journal of Computer Documentation* (February 1996).

K. Geblert and D. Scipione, "In Situ Monitoring of Product Wafers," *Solid State Technology* (March 1996).

J. Grodstein, E. Lehman, H. Harkness, and Y. Watanabe, "Logic Decomposition During Technology Mapping," *IEEE/ACM International Conference on Computer-aided Design* (November 1995).

P. Gronowski, "A 433MHz 64b Quad-issue RISC Microprocessor," *Digest of Papers, IEEE International Solid State Circuits Conference* (February 1996).

P. Gronowski, "Dynamic Logic and Latches Part II—Practical Implementation Methods and Circuit Examples Used on the Alpha 21164," *VLSI Circuits Symposium* (June 1996).

E. Hanson and H. Woodward, "Process Control Methodology for PSG and PETEOS Films in a Highly Interactive Multiprocess CVD System," *Advanced Semiconductor Manufacturing Conference and Workshop* (November 1995).

C.-L. Huang, J. Faricelli, N. Khalil, and R. Rios, "An Accurate Gate Length Extraction Method for Sub-quarter Micron MOSFETs," *IEEE Transactions on Electron Devices* (June 1996).

H. Jakiela, "Performance Visualization of a Distributed System: A Case Study," *Computer* (November 1995).

R. Kelsey, "Bad Fixes, Change Specifications and Linguistic Constraints on Problem Diagnosis," *Software Engineering Notes* (March 1996).

J. Kern, "The Chicken is Involved, But the Pig is Committed—Building Commitment Through Cascading Teams," *Quality Progress* (October 1995).

N. Khalil, J. Faricelli, and J. Huang, "Two-dimensional Dopant Profiling of Submicron MOSFETs Using Nonlinear Least Squares Inverse Modeling," *Journal of Vacuum Science and Technology* (January/February 1996).

J. Kitchin, "Design for Reliability in the Alpha 21164 Microprocessor," *Proceedings of the IEEE 34th Annual Spring Reliability Symposium* (April 1996).

B. Mirman, "Choice of Models and Failure Indicators for Thermally Loaded Solder Joints," *Proceedings of the Technical Program NEPCON EAST '96* (June 1996).

W. Nagorski, W. McGee, E. Piccioli, and L. Bair, "Automatic Test Chip Documentation Synthesis," *Proceedings of the 1996 IEEE International Conference on Microelectronic Test Structures* (March 1996).

L. Noack and M. Kantrowitz, "I'm Done Simulating; Now What? Verification Coverage Analysis and Correctness Checking of the DECchip 21164 Alpha Microprocessor," *Proceedings of the 33rd Design Automation Conference* (June 1996).

O. Ramahi, "Adaptive Absorbing Boundary Conditions in Finite-difference Time Domain Applications for EMC Simulations," *IEEE Transactions on Electromagnetic Compatibility* (November 1995).

O. Ramahi et al., "Dynamic Analysis of V Transmission Lines," *Conference Proceedings of the 12th Annual Review of Progress in Applied Computational Electromagnetics* (March 1996).

S. Rege, "A Distributed System Client/Server Architecture for Interactive Multimedia Applications," *COMP96: Digest of Papers 41st IEEE Computer Society International Conference* (February 1996).

R. Rios, N. Arora, C.-L. Huang, N. Khalil, J. Faricelli, and L. Gruber, "A Physical Compact MOSFET Model, Including Quantum Mechanical Effects, for Statistical Circuit Design Applications," *IEEE/Electron Devices Technical Digest* (December 1995).

K. Roselle, "Estimating Crosstalk in Multiconductor Transmission Lines," *IEEE Transactions on Components, Packaging, and Manufacturing Technology Part B: Advanced Packaging* (May 1996).

N. Rubin, "Efficient Instruction Scheduling Using Finite State Automata," *Proceedings of the 28th Annual International Symposium on Microarchitecture* (November 1995).

C. Smith and T. Vallone, "Mentoring: Providing Professional and Organizational Benefits," *Proceedings of the 43rd Conference of the Society for Technical Communication* (May 1996).

N. Sullivan, S. Dass, G. Pollard, W. Jones, and T. Lindsay, "A Comparison of State-of-the-Art DUV Lenses," *Proceedings of the Society of Photo-Optical Instrumentation Engineers* (February 1995).

H. Teegan, "Distributed Performance Monitor Using SNMP V2," *IEEE/IFIP Network Operations and Management Symposium* (April 1994).

M. Tsuk, "FD-TD Analysis of Electromagnetic Radiation from Modules-on-Backplane Configurations," *IEEE Transactions on Electromagnetic Compatibility* (August 1995).

A. Villani and H. Nguyen, "Correlation of the Mechanical to the Thermal Strength of Ceramic Packages," *American Ceramic Society Transactions: Hybrid Microelectronic Materials* (November 1994).

W. Zahavi, "Modeling the Performance Budget—A Case Study," *CMG95: Proceedings of the 21st Annual Conference for the Resource Management and Performance Evaluation of Enterprise Computing Systems* (December 1995).

Recent Digital U.S. Patents

The following patents were recently issued to Digital Equipment Corporation. Titles and names supplied to us by the U.S. Patent and Trademark Office are reproduced as they appear on the original published patent.

D353,800	M. S. Lewis, L. A. Treseder, R. M. Tusler, and G. Suzda	Modular Enclosure for Electronic Equipment
5,371,807	N. Kannan and M. S.	Register Method and Apparatus for Text Classification
5,371,822	F. Horwitz and E. Thomson	Method of Packaging and Assembling Opto-electronic Integrated Circuits
5,371,868	G. P. Konig, H. S. Yang, and W. Hawe	Method and Apparatus for Deriving Addresses for Stored Address Information for Use in Identifying Devices during Communication
5,371,870	P. M. Goodwin, D. Smelser, and D. A. Tatosian	Stream Buffer Memory Having a Multiple-entry Address History Buffer for Detecting Sequential Reads to Initiate Prefetching
5,371,874	M. Gagliardo, J. Lynch, K. Chinnaswamy, and J. Tessari	Write-read/Write-pass Memory Subsystem Cycle
5,371,889	J. Klein	Journalling Optimization System and Method for Distributed Computations
5,372,262	J. M. Benson and J. E. Fritscher	Frame Assembly for Rack-mountable Equipment
5,373,421	C. Detsikas and T. Spellman	Fiber Optic Transceiver Mounting Bracket
5,375,068	R. S. Palmer and L. G. Palmer	Video Teleconferencing for Networked Workstations
5,375,199	J. R. Harrow and F. P. Messinger	System Monitoring Method and Device, Including a Graphical User Interface to View and Manipulate System Information
5,377,190	H. Yang, K. K. Ramakrishnan, B. Spinney, and K. R. Jain	Frame Removal Mechanism Using Frame Count for Token Ring Networks
5,377,327	K. R. Jain, K. K. Ramakrishnan, and D.-M. Chiu	Congestion Avoidance Scheme for Computer Networks
5,377,354	N. Scannell, A. Redmond, P. Bares, A. Clark, S. Dawson, and S. Himbaut	Method and System for Sorting and Prioritizing Electronic Mail Messages
5,378,945	H. Partovi, S. Butler, and L. Tran	Voltage Level Converting Buffer Circuit
5,379,419	J. S. Heffernan, P. L. Savage, S. J. Pittman, and R. V. Sunkara	Methods and Apparatus for Accessing Non-relational Data Files Using Relational Queries
5,381,052	R. Kolte	Peak Detector Circuit and Application in a Fiber Optic Receiver
5,381,146	R. Kolte	Voltage-tracking Circuit and Application in a Track-and-hold Amplifier

5,382,831	B. Lee, E. Atakov, and J. Clement	Integrated Circuit Metal Film Interconnect Having Enhanced Resistance to Electromigration
5,383,096	M. C. Benson and L. M. Mazzone	I/O Expansion Box
5,384,779	M. Patrick and J. A. Daly	State Machines for Configuration of a Communications Network
5,385,289	C. Bloch, P. McKinley, and R. Ranganathan	Embedded Features for Registration Measurement in Electronics Manufacturing
5,385,630	A. Philipossian, H. Soleimani, and B. Doyle	Process for Increasing Sacrificial Oxide Etch Rate to Reduce Field Oxide Loss
5,386,514	V. Boaen, R. Lary, B. Rubinson, D. Thiel, C. Van Ingen, W. Watson, R. Willard, and E. A. Gardner	Queue Apparatus and Mechanics for a Communications Interface Architecture
5,386,523	N. A. Crook, M. J. Seaman, and D. L. A. Brash	Addressing Scheme for Accessing a Portion of a Large Memory Space
5,386,524	V. Boaen, R. Lary, B. Rubinson, D. Thiel, C. Van Ingen, W. Watson, and R. Willard	System for Accessing Information in a Data Processing System
5,387,495	J. C. K. Lee, M. Castro, F. Tung, C. Lee, and A. Ahmad	Sequential Multilayer Process for Using Fluorinated Hydrocarbons as a Dielectric
5,387,530	A. Philipossian and B. Doyle	Threshold Optimization for SOI Transistors through Use of Negative Charge in the Gate Oxide
5,388,099	N. Poole	Backplane Wiring for Hub in Packet Data Communications System
5,388,222	L. A. P. Chisvin, J. F. Rantala, J. K. Grooms, and D. W. Hartwell	Memory Subsystem Input Queue
5,388,224	B. Maskas	Processor Identification Mechanism for a Multiprocessor System
5,388,247	P. Goodwin, K. Thaller, and B. Maskas	History Buffer Control to Reduce Unnecessary Allocations in a Memory Stream Buffer
5,388,263	R. K. Peterson, J. R. Ellis, and C. G. Nylander	Procedure State Descriptor System for Digital Data Processors
5,389,757	E. Soulière	Elastomeric Key Switch Actuator
5,390,173	B. Spinney, R. Simcoe, G. Varchese, and R. Thomas	Packet Format in Hub for Packet Data Communications System
5,390,286	K. K. Ramakrishnan	Reticular Discrimination Network for Specifying Real-time Conditions
5,390,299	S. L. Rege, K. K. Ramakrishnan, and D. A. Gagne	System for Using Three Different Methods to Report Buffer Memory Occupancy Information Regarding Fullness-related and/or Packet Discard-related Information
5,390,302	J. Johnson, M. Howell, and C. Whitaker	Transaction Control
5,390,318	K. K. Ramakrishnan and P. Biswas	Cache Arrangement for File System in Digital Data Processing System
5,390,327	C. Lubbers and D. Thiel	Method for On-line Reorganization of the Data on a RAID-4 or RAID-5 Array in the Absence of One Disk and the On-line Restoration of a Replacement Disk
5,392,219	S. Birch, G. Gavrel, and Z. Memon	Determination of Interconnect Stress Test Current
5,394,143	J. Murray and G. Antoshenkov	Run-length Compression of Index Keys
5,394,347	R. Kita, S. Tremblay, and T. Lynch	Method and Apparatus for Generating Tests for Structures Expressed as Extended Finite State Machines
5,394,401	M. Patrick and J. A. Daly	Arrangement for a Token Ring Communications Network
5,394,529	J. Brown, J. Meyer, and S. Persels	Branch Prediction Unit for High-performance Processor

Call for Papers
Network Products and Technologies

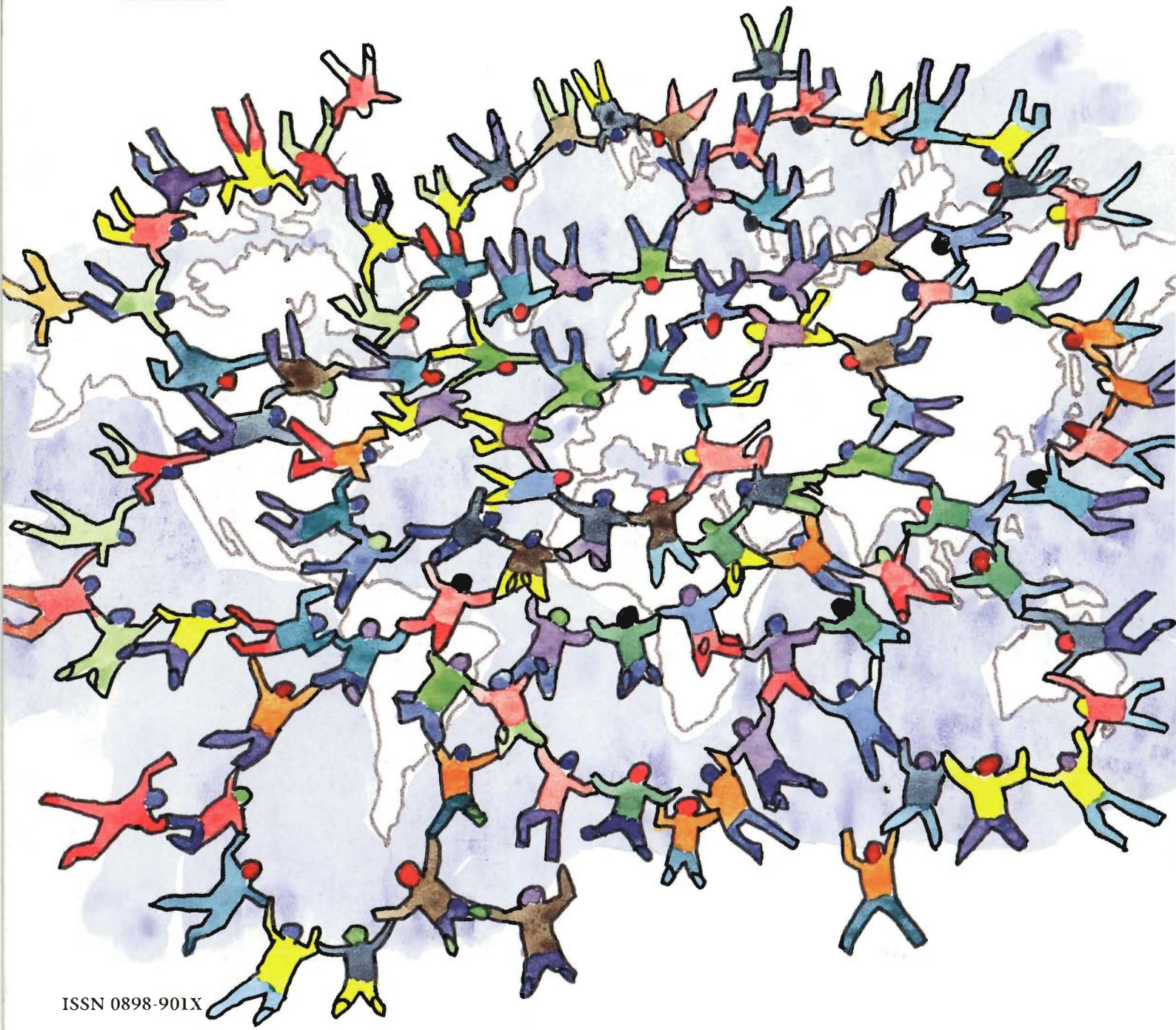
The *Digital Technical Journal* seeks technical papers in all areas of networking technology for an issue to be published in the fall of 1997. Digital's engineers and industry partners interested in participating in the special issue should send topics and brief abstracts (100 words) by February 10, 1997, to

Jane Blake, Managing Editor
Digital Technical Journal
Digital Equipment Corporation
50 Nagog Park, AKO2-3/B3
Acton, MA 01720-9843
Email: jane.blake@ljo.dec.com
508-486-2544

Notice of the topics accepted will be sent to all authors by February 28, 1997. The manuscript-submission date for accepted topics is May 30, 1997.

For information on topics published in the *Journal*, the audience, writing guidelines, and the peer-review process, see http://www.digital.com/info/dtj/dtj-guide.htm or contact the Managing Editor at the address above.

digital™



ISSN 0898-901X

Printed in U.S.A. EC-N7285-18/96 12 14 21.5