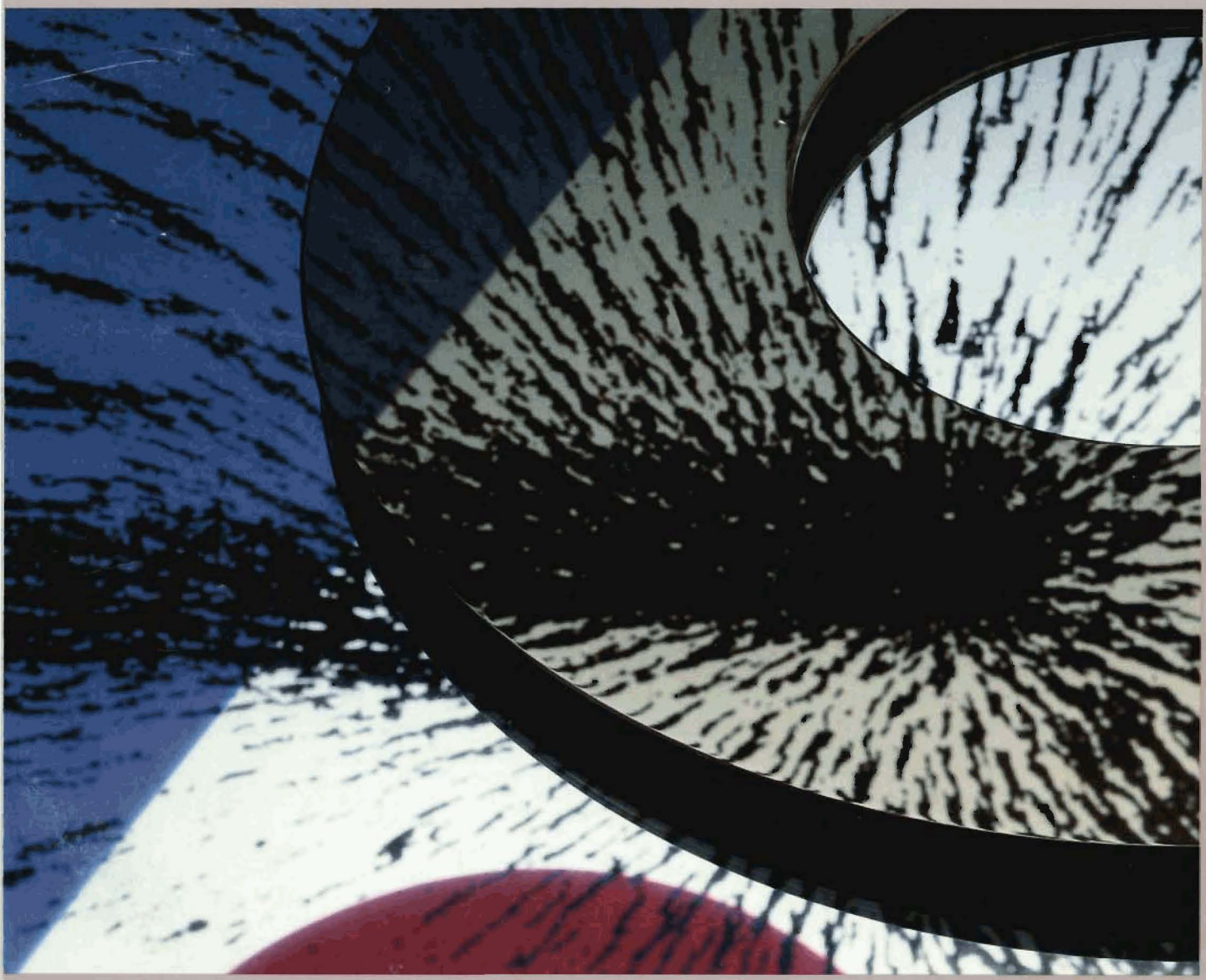
Storage Technology

Digital Technical Journal

Digital Equipment Corporation



Number 8

February 1989

The *Digital Technical Journal* is published by Digital Equipment Corporation, 146 Main Street, Maynard, Massachusetts 01754.

Changes of address should be sent to Digital Equipment Corporation, attention: List Maintenance, 10 Forbes Road, Northboro, MA 01532. Please include the address label with changes marked.

Comments on the content of any paper are welcomed. Write to the editor at Mail Stop MLO1-3/B68 at the published-by address. Comments can also be sent on the ENET to RDVAX::BLAKE or on the ARPANET to BLAKE%RDVAX.DEC@DECWRL.

ISSN 0898-901X

Documentation Number EY-C166E-DP

Book production was done by Digital's Educational Services Media Communications Group in Bedford, MA.

Cover Design

The theme of this issue is storage technology. On our cover, the concept of electromagnetism, basic to this technology, is conveyed by the pattern of iron filings traced on the lines of magnetic force. In the background is the mirror surface of the RA90 magnetic disk, a storage product which represents significant advances in magnetic storage technology.

The cover was designed by Peter Gronsik and Gloria Lee of the Graphic Design Department.

Contents

Editor's Introduction

Jane C. Blake
Editor

The papers in this issue of the *Digital Technical Journal* describe the engineering technologies used in the design, manufacture, and maintenance of Digital's storage and information management products. In his Foreword, Alan Kotok, Corporate Consulting Engineer, sets the order of this issue's papers by discussing them from a systems-level viewpoint. He notes that products at each level of the storage system are planned and designed to be integrated as a whole. This viewpoint is characteristic of the unique approach taken by Digital's engineers in the design of all our products. The systems-level approach provides the framework for our entire design and development process. It ensures that each Digital product will integrate successfully with other units into a complete system.

Both the systems-level view Alan refers to and the content of the papers in this issue are representative of the kinds of information the *Digital Technical Journal* was established to provide. Our goal is to demonstrate Digital's systems-level approach by discussing design projects that result in real products. Begun in 1985, the journal offers explanation of the technological foundations of those products. The journal's first issue, for example, contained papers on the innovations made in the design of the VAX 8600 processor; another issue described the evolution of the Digital Network Architecture and products that incorporate this architecture; in our last issue, topics included the architectural definition process for the VAX 6200 multiprocessor, the complexities of the bus interface, and the CVAX chip set on which the system is based. Journal papers such as these give readers insights into the common ways in which Digital designs and develops its products.

The papers are written by the project engineers who designed and developed these products. These individuals are best qualified to offer the in-depth explanations of the technologies and processes reflecting Digital's approach to designing and integrating systems. Moreover, they can discuss the important decisions made and the difficulties encountered during the development process. These discussions will interest engineers faced with similar issues, professors needing examples of real problems, and students confronting parallel ones in the classroom.

Journal issues focus on current products, often those recently announced. To date, each issue has a single, unifying theme about the new system or technology in which advances have been made. This issue's theme, Storage Technology, includes papers on a new disk drive, the RA90, some newly developed technologies supporting that drive, and enhancements to Rdb and VAX DBMS software products; past issue themes in addition to those noted above include VAXclusters, the VAX 8800 Family, and Software Productivity Tools. The next issue in 1989 will feature Distributed Systems.

Our Advisory Board, chaired by Sam Fuller, vice president of Corporate Research and Architecture, selects the themes for future issues. The board comprises four Digital vice presidents and three senior engineering managers. Once the board has selected future product themes, the editors work with the apposite organizations to bring together papers that relate to these products and their development.

Currently published twice each year, the *Digital Technical Journal* is distributed at no cost to Digital's own engineers and educators in the fields of computer science and engineering, approximately 15,000 readers in all. Copies of individual issues are also sold to interested parties by the Digital Press of Digital Equipment Corporation.

I thank Donna Charette and Peter Van Roekens of the Storage and Information Management Group for their help in preparing this issue.

Jane Blake

Kenneth H. Bates Ken Bates is a consulting software engineer with the I/O Performance Group in Storage Systems Engineering. As a project leader, he models the characteristics of mass storage products to improve their performance. Ken developed both the MSCP disk server in the HSC controller and the specification for disk shadowing. In addition to his 13 years of work at Digital, Ken has served as vice president at Real Time Systems, Inc. and vice president of Technical Services at MLPI Business Systems, Inc.

Robert G. Bean In 1970, Bob Bean joined Digital after receiving his B.S. degree in life sciences from M.I.T. He first wrote software for PDP-8 and PDP-11 systems, then managed the group that developed the RT-11 product. Joining the Storage Systems Engineering Group in Colorado Springs in 1978, Bob helped to define the Digital Storage Architecture, then contributed to the design and implementation of the HSC50 storage controller. He is currently a senior consultant software engineer working on the development of new storage subsystems.

Barbara A. Crane Barbara Crane came to Digital in 1977 as part of the Advanced Development Group which developed thin-film disks at Digital. Now in Colorado, she is responsible for resolving the RA90 head and media technical specifications with the design engineering group, as well as coordinating quality, delivery and volume commitments for the product. She has also conducted failure analysis of RA90 disk drive prototypes. Barbara received an S.B. in materials science and metallurgy from M.I.T. She is a member of the American Society of Metallurgists.

Larry W. Emlich A consultant to the Advanced Service and Delivery Systems Development Group, Larry Emlich is currently working on enhancements to VAXsimPLUS software. He is a coinventor of the technology developed for this product and has applied jointly for its patent. Prior to his work on VAXsimPLUS, Larry led a project that produced the first release of SPEAR, a software package for error analysis and reporting. He is the recipient of Digital's Challenge of Excellence Award, 1988. Larry came to Digital in 1971 from the Univac Corporation (now UNISYS).

Ashok M. Joshi Ashok Joshi is a software engineer interested in object-oriented database systems. The project on which he is now working is the KODA subsystem, which provides record storage for Rdb/VMS and DBMS software. For the Rdb/VMS project, he developed hash indexing and record placement features, and worked on optimizing the lock protocols. Ashok recently came to Digital after receiving a bachelor's degree in electrical engineering from the Indian Institute of Technology, Bombay, and a master's degree in computer science from the University of Wisconsin, Madison.

Reinhard Kretschmer Reinhard Kretschmer earned his diploma in physics from the University Marburg, West Germany, in 1983. Immediately thereafter he joined Digital to work on component engineering for magnetic media. Reinhard is currently the supervisor of Product/Process Reliability Engineering in the European Storage Systems Group. He and his team in Kaufbeuren, West Germany, have performed fundamental phase margin application engineering work, which is one of the key techniques for assurance and advancement of disk drive reliability.

Richard F. Lary Richie Lary, a senior consulting engineer in Storage Systems Engineering, is currently working on the next generation of storage controller products. He helped design the HSC50 and UDA50 controllers, and the RA80 disk drive, as well as the Digital Storage Architecture. Richie was a member of the original VAX architecture team and developed microcode for the 11/60 and 11/780 systems. He has implemented several languages and operating systems on the PDP-8 system. Richie joined Digital after receiving his B.S. degree in mathematics (1969) from the Polytechnic Institute of Brooklyn. He holds 11 patents relating to computer architectures and implementations.

Herman D. Polich Currently a member of the Computer Systems Service Engineering Group in Colorado, Herman Polich is working with a team that is defining the service requirements for the next generation of high-end storage products. Prior to this project, Herman was the technical leader of the VAXsimPLUS development team software and is listed as a coinventor on a patent application pending for the technology used in the VAXsimPLUS product. Herman is a systems maintainability engineer and has been with Digital since 1973.

T.K. Rengarajan T.K. Rengarajan joined Digital in 1987 after receiving an M.S. in computer science from the University of Wisconsin. He also holds an M.S. in computer-aided design by analysis from the University of Kentucky and a B.E. in mechanical engineering from the Regional Engineering College in Tiruchirapalli, India. Rengarajan has worked in the areas of boundary element methods, image databases, secure process connections, and database concurrency control. His current interests lie in the fields of object-oriented CAD database systems and high-performance database system architectures.

Karen E. Rodwell A senior software engineer in the Database Systems Group, Karen Rodwell is responsible for ANSI SQL compatibility support within the Rdb engine. Karen came to Digital in 1984 as a co-op student and then joined the company as a developer working on the Rdb/VMS project. She holds a B.S. in computer science and a B.S. in mathematics with a minor in design (1985) from Rivier College. Currently, Karen is pursuing an M.S. in computer science at Boston University.

Siegbert Sadowski Siegbert Sadowski is a supervisor in the Distributed Design Engineering Group, European Storage Systems. As the leader of the phase distribution analyzer project, he provided the conceptual framework for the test system and guided the design and implementation work. For this work, he was awarded a patent in 1988. Siegbert joined Digital in 1983 after nine years with Dietz Computersysteme, where he developed processors, mass storage controllers, and software for timesharing systems. He holds a Diplom Ingenieur (FH) degree (1974) from Fachhochschule Bochum, West Germany.

Michael D. Sidman A consultant engineer in the Storage Systems Advanced Development Group, Mike Sidman manages a team which he formed to develop new, practical servomechanical systems and strategies for Digital disk products. He currently holds three patents and has several patents pending for his work on continuous-plus-embedded position control systems. As Digital's Graduate Engineering Education Program Scholar, Mike received a Ph.D. (1986) from Stanford University. He has published articles about his work in the area of control systems for disk drives and robotics. Mike is a member of Phi Kappa Phi, Tau Beta Pi, and Eta Kappa Nu.

Alan B. Smith Alan Smith, a consultant engineer, came to Digital in 1983 from Sperry Research Center. Alan holds five patents and has published numerous papers about his work in such areas as bubble-domain and magneto-optic materials. For the RA90 product, he designed electronic lapping guides and is now working on advanced head designs for future disk drive products. Alan received a Ph.D. in applied physics from Harvard University, an M.E.E. from Rensselaer Polytechnic Institute, and a B.S.E.E. from Swarthmore College. He is a member of Sigma Xi, IEEE (senior member), and the American Physical Society.

Peter M. Spiro Peter Spiro is currently the project leader for KODA, the database kernel for both Rdb/VMS and VAX DBMS software. A senior software engineer, Peter has worked on KODA recovery and journaling, on-line backup, access methods, and buffer management. Prior to joining Digital in 1985, Peter was a charcoal-maker in Mali, West Africa, for the Peace Corps and a farmer with the University of Wisconsin Agriculture Department. He holds M.S. degrees in forest sciences and in computer sciences from the University of Wisconsin.

William A. Wright A principal software engineer, Bill Wright has been in the Database Systems Group since joining Digital in 1981. He is presently involved in development efforts for future enhancements to Digital's current database products. Bill has worked on VAX DBMS, Rdb/VMS, and KODA software, and served as the project leader for the development of VAX DBMS version 3.0. His main areas of interest are access methods (B-trees), performance, and product quality. Bill received a B.A. in computer science from Dartmouth College in 1981.

Foreword

Alan Kotok
Corporate Consulting Engineer and Storage Architect

This issue of the *Digital Technical Journal* presents papers on some of the technologies employed in the products developed by Digital's Storage and Information Management Group (SIMG). The name Storage and Information Management was chosen by the group in the summer of 1988, replacing the name Storage Systems, since the charter of the group has expanded to encompass not only storage devices but the total management of our customers' information. Since 1986, SIMG has had responsibility for all database software produced within Digital. Now SIMG is expanding its role to include other aspects of data management as well.

The "Storage" sections of SIMG have responsibility for information storage hardware ranging from primary storage with CPU memories, through secondary storage, such as solid-state storage units and magnetic and optical disk systems, to tertiary storage, such as magnetic tape systems. All secondary and tertiary storage systems developed within SIMG conform to a comprehensive set of architectural specifications known collectively as the Digital Storage Architecture (DSA). The purposes of the DSA are to preserve customer investment in storage devices across generations and to allow multiple groups — both hardware and software — to develop interworking products. To that end, DSA includes specifications for the interfaces between host computer systems and storage controllers, between controllers and storage drives, and the required functions at each level.

The first paper following this Foreword is "The Hierarchical Storage Controller, A Tightly Coupled Multiprocessor as Storage Server" by Richie Lary and Bob Bean. The HSC implements the controller functionality of the DSA in a multihost, multidrive environment. It is the prime storage controller employed in VAXcluster systems. The paper describes the hardware and software structure of the HSC used to achieve high performance in this environment. The second paper, "Performance Aspects of the HSC Controller" by Ken Bates, explores the effects of contention between the multiple processes operating in the HSC on its overall performance.

DSA also prescribes how faults detected in storage elements are to be reported. In the past, these faults were logged for later human analysis to determine what service might be needed or to help diagnose system failure after the fact. A new technique, described by Larry Emlich and Herman Polich in their paper "VAXsimPLUS, A Fault Manager Implementation," allows real-time software analysis of faults and triggers evasive actions to prevent system failures through use of spare components.

The next set of four papers deals with magnetic disk drive technology. The DSA prescribes an interface known as the Standard Disk Interface (SDI) between storage controllers such as the HSC and disk drives. The family of disks that interface in this manner is known as the RA series of drives. The first such drive was the RA80, introduced in 1981. Since then, several generations of drives have been introduced, with increasing capacity and performance, leading to the recently introduced RA90 disk drive. The paper "Disk Drive Technology Improvements in the RA90" by Barbara Crane describes the many areas of technology that were addressed in bringing forth this latest member of the RA series.

There follow three more articles dealing with specific areas of technology used in modern disk drives. The first of this set is on "Control Systems Technology in Digital's Disk Drives." Mike Sidman discusses the problems associated with causing the actuators that move the recording heads to follow tracks on the disk surface at densities of beyond 1500 tracks per inch. The paper describes the servomechanism system used to counter the many sources of positioning errors.

The RA90 drive, as described in Barbara Crane's paper, employs read-write recording heads fabricated using thin-film technology. The paper "Magnetic Domain Observations in Thin-film Heads Using Kerr Microscopy" by Al Smith describes the development of a methodology for observing the magnetic fields generated by these heads. Such observations aid the development and manufacturing process of the thin-film heads.

The third paper in this subgrouping, "Margin Analysis on Magnetic Disk Recording Channels," describes a technique for characterizing the performance parameters of the recording channel. This technique, called Phase Margin Analysis, is implemented by a test system called the Phase Distribution Analyzer. The tester, as described by Reinhard Kretschmer and Siegbert Sadowski, is designed to be used in both design and production environments to enhance both reliability and data integrity of disk drives.

In 1986 the (then) Storage Systems Group was given the responsibility for development of Digital's database systems. Two cornerstones of our database systems are VAX DBMS, a CODASYL database, and Rdb/VMS, a relational database. These are the underlying database management systems which support various database query and transaction processing environments. Both systems have undergone continual enhancement over the past several years. Two papers are included herein which deal with aspects of these systems.

The first article, "High Availability Mechanisms of VAX DBMS Software" by T.K. Rengarajan, Peter Spiro, and Bill Wright, addresses mechanisms employed to allow DBMS to take full advantage of the parallelism inherent in VAXcluster systems, while maintaining service in this distributed processing environment despite the failure of individual cluster members. The paper explains the use of the VMS lock manager to coordinate the activities proceeding on the various members of the cluster. It also describes how "failover" is accomplished when a cluster member ceases to communicate.

The second paper, "A Relational Database Management System for Production Applications," describes how Rdb/VMS software, which was originally developed for ad-hoc query applications, was improved to handle large, production applications. Seven different areas of the system were enhanced to produce the current package, as described by Ashok Joshi and Karen Rodwell.

This selection of nine papers addresses only a few points in the broad field of storage and information management technology. Many of the technologies employed within SIMG were not even mentioned. We hope to touch on these in a future issue of the *Digital Technical Journal*.

Richard F. Lary
Robert G. Bean

The Hierarchical Storage Controller, A Tightly Coupled Multiprocessor as Storage Server

The many hosts in a VAXcluster system simultaneously access a common subsystem of storage devices, which must provide much more comprehensive services than a traditional disk controller. Moreover, the functional sophistication of VAXcluster controllers goes far beyond simple data transfers. This subsystem, the Hierarchical Storage Controller, employs a tightly coupled multiprocessing architecture that permits an I/O control processor to control many data channels without participating in the data flow itself. This paper describes this architecture, emphasizing the mechanisms that govern control and data flow.

The Hierarchical Storage Controller (HSC) product was initially conceived as a high-performance disk controller for a single-host computer. Shortly after the HSC project commenced, however, Digital initiated the VAXcluster program. Since the HSC goals closely matched those needed for a cluster storage server, the HSC project was redirected to produce such a device.

Digital's disk controllers have traditionally performed I/O for a single computer with one to eight disk drives. VAXcluster systems, on the other hand, can have up to 16 nodes on the CI bus, and the aggregate performance of this set of computers is quite large. Therefore, the HSC controller had to be designed to handle significantly more disk drives — the HSC70 model can attach to up to 32 — and deliver more performance than a traditional disk controller.

The performance of disk drives is governed by the laws of mechanics and economics. The energy needed to rotate a disk varies as the inverse third power of the rotation period. The energy needed to move a head assembly varies as the inverse fourth power of the average seek time. Because of these physical limitations, disk performance has improved more slowly than other parameters of computer systems, such as logic speed and memory size.

There is nothing that a disk controller can do to increase performance by increasing the rotational speeds of disks. The apparent performance of a set of disk drives can be improved, however, by adding intelligence to the disk controller to optimize the handling of multiple simultaneous requests to those drives. Some of these optimization techniques merely offset the natural performance decrease that occurs when many devices are accessed simultaneously through a single control point. Others actually improve the performance of individual devices. A number of optimization techniques are used in the HSC design:

- Overlapped Seeks – The controller can seek data on one disk drive while transferring data on another. The first drive will thus seek as if it had a dedicated controller.
- Optimized Seeks – When multiple requests are outstanding to one drive, the controller can modify the order in which they are serviced to minimize the drive's total head movement.
- Multiple Simultaneous Transfers – The HSC design can simultaneously transfer data from several disk drives, making each drive perform as if it had a dedicated controller. The amount of hardware required to perform multiple simultaneous transfers grows linearly with the number of simultaneous transfers allowed. Therefore, the few controllers (like the HSC) that implement this technique allow only a small fixed number of simultaneous transfers.
- Interdrive Rotational Optimization – Since simultaneous transfers are limited, the controller allows the drive that first reaches its desired sector(s) to transfer before other drives that are also "on cylinder." This technique reduces the number of missed revolutions due to missed transfer opportunities.

- Intradrive Rotational Optimization – When several requests want the same cylinder on a drive, the HSC design services them in the order in which they come under the disk head. That reduces the number of missed revolutions on that drive.
- Intrarequest Rotational Optimization – When a single request spans several sectors, the controller can service them in the order in which they come under the disk head. That also reduces the number of missed revolutions on that drive.

Many Digital systems have implemented the first three techniques with a mixture of controller hardware and host software. DECsystem-10 and DECSYSTEM-20 systems implement the fourth technique in software, at the expense of some host overhead. The HSC design is the only Digital product that incorporates the last two techniques.

Performing these optimizations requires considerable processor power; handling the System Communication Architecture (SCA) protocol on the CI bus at high speed requires even more. The hardware and software structures that deliver that power are described in this paper.

Processing Elements in the Hardware

The HSC design is organized internally as a heterogeneous multiprocessor that uses a combination of processor-private and shared-global memories to reduce memory contention. Figure 1 is a block diagram of the HSC hardware.

There are two types of processors executing within an HSC controller. Most of the intelligence resides in the policy microprocessor, called the P.io. The two current HSC models use different hardware for the P.io. The HSC50 device uses a P.ioc incorporating an 11/23 microprocessor that executes 250,000 instructions per second. The HSC70 device uses a P.ioj incorporating an 11/73 microprocessor that executes 500,000 instructions per second. The architecture of the HSC hardware can accommodate up to four P.io processors in one device. Except for special debugging versions in the laboratory, however, each current HSC controller contains a single P.io processor.

Most of the work in an HSC device is done by fast but relatively inflexible RISC machines, called K's within Digital. All K's have at their cores the same processing element (a 16-bit, 16-register datapath built from commodity 2901 chips), and they execute the same basic instruction set. The K's can be divided into three groups, based on their complement of microperipherals and the microcode they execute:

- K.ci, which schedules and performs message transmission and reception on the CI bus
- K.sdi, which schedules and performs read, write, and format operations, seek operations, and status inquiries to a set of four disk drives
- K.sti, which schedules and performs read and write operations, tape movement operations, and status inquiries to a set of four tape formatters

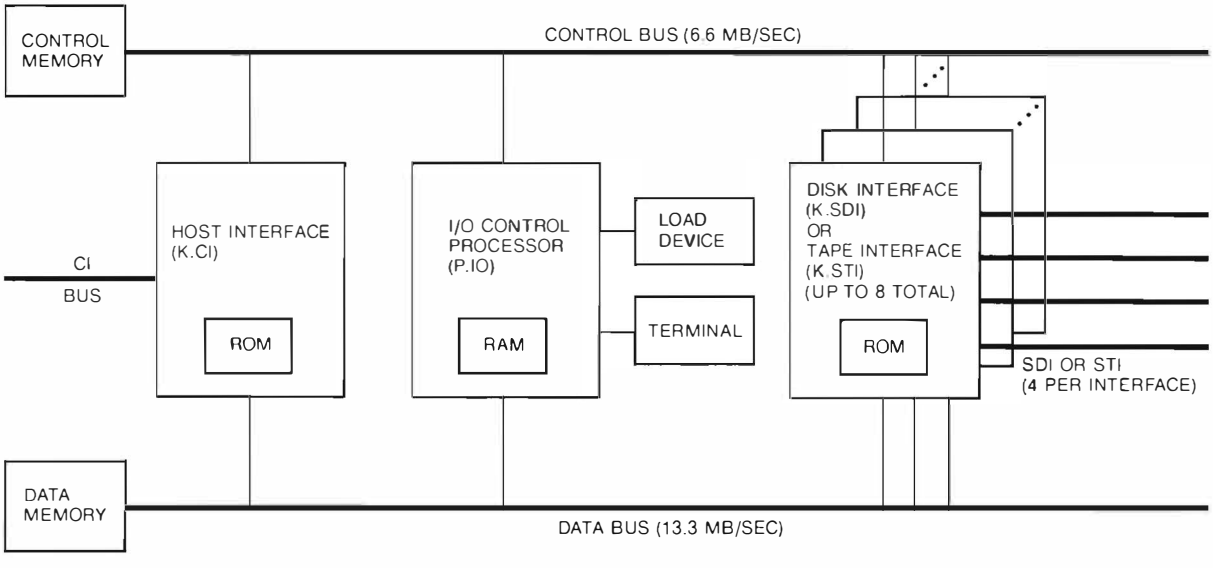


Figure 1 Block Diagram of the HSC Hardware

Each K in an HSC device executes 6.67 million instructions per second (MIPs) and must perform time-critical operations in real time on its associated device. Each K must also simultaneously perform scheduling and bookkeeping functions based on shared-memory communication with other K's and the P.io. To aid in these tasks, each K is multiprogrammed in hardware into two virtual machines: a control processor, and a data processor. These two processors alternate instruction cycles in the K's datapath and appear to be two 3.33 MIP processors sharing the same set of 16 registers.

Memories and Buses

The design of the memory system is crucial to the performance of a shared-memory multiprocessor system. A central memory and its access bus are shared resources. They must be able to handle the combined demands of all the processors in the system. Because the HSC design was intended for a specific application, we reduced the load significantly on the memory system by functionally partitioning it into the following three pieces:

- Each processor uses a private memory bus to access local, private memory in which processor instructions and data are stored. The private memory bus for the P.io is a modified version of the Q-bus system, used in many of Digital's low-end systems. In the HSC implementation, this asynchronous bus can perform a 16-bit (word) memory operation in approximately 650 nanoseconds (ns). The P.io private memory consists of dynamic RAM located on a common memory board called the M.std board. Each K in the HSC design fetches instructions from a private PROM memory and uses fast static RAM for local data storage.

- The data structures used for interprocessor communication and control are located in control memory and accessed by means of the control bus. The control bus is an unpended bus (i.e., operations proceed to completion once they start) with a 300-ns cycle time. This bus performs both byte and word reads and writes to the 128-Kword control memory in one bus cycle. The bus also implements an interlock operation, which consists of reading a memory location and then writing the constant 8000(hex) back into that memory location in two consecutive bus cycles. The P.io and each K's control processor connect to the control bus.

  The control memory is located on the M.std board. This memory was implemented with static RAM on early HSC versions, but to save cost, was reimplemented in fast dynamic RAM when it became available.

- Data moving between the CI interface and the disk drives is stored in data memory and accessed by means of the data bus. The data bus is an unpended bus with a 150-ns cycle time; it performs word reads and writes to the 128-Kword data memory in one bus cycle. No interlock or byte operations are supported. The data memory is implemented with fast static RAM located on the M.std card. The P.io and each K's data processors connect to the data bus.

Interprocessor Communication and Control Flow Mechanisms

At the center of the HSC architecture and design are the mechanisms used by the many processors in the storage subsystem to exchange control information and data. We now describe those mechanisms and provide an example of how they are used to perform a disk read.

At the start, the design team chose some basic strategies to govern the details of the design. These strategies included the following:

- A common set of basic, general-purpose mechanisms should be applied wherever possible. Rather than defining custom interfaces for each element of the subsystem, the design team defined a relatively small number of general mechanisms. These mechanisms could then be adapted as necessary to individual interfaces.

- The P.io should have minimal overhead. The limited processing power of the P.io was viewed as a key limiting factor in subsystem performance. Therefore, the interface design minimizes the number of interrupts processed by the P.io and, most important, removes the P.io from the datapath of error-free operations.

- A common interprocessor and interprocess interface was needed. The K's and the P.io software processes all communicate with common mechanisms. That commonality provides significant flexibility when deciding whether to implement a given function in software or hardware.

- Since the K's are significantly harder to program than the P.io and their programs are kept in unalterable PROM memory, the complexity of these programs had to be minimized to ensure that they would be bug-free when the first HSC controller shipped. On the other hand, the K's are by far the most powerful computational elements in the controller, so they have to perform as much of the work as possible to unburden the P.io.

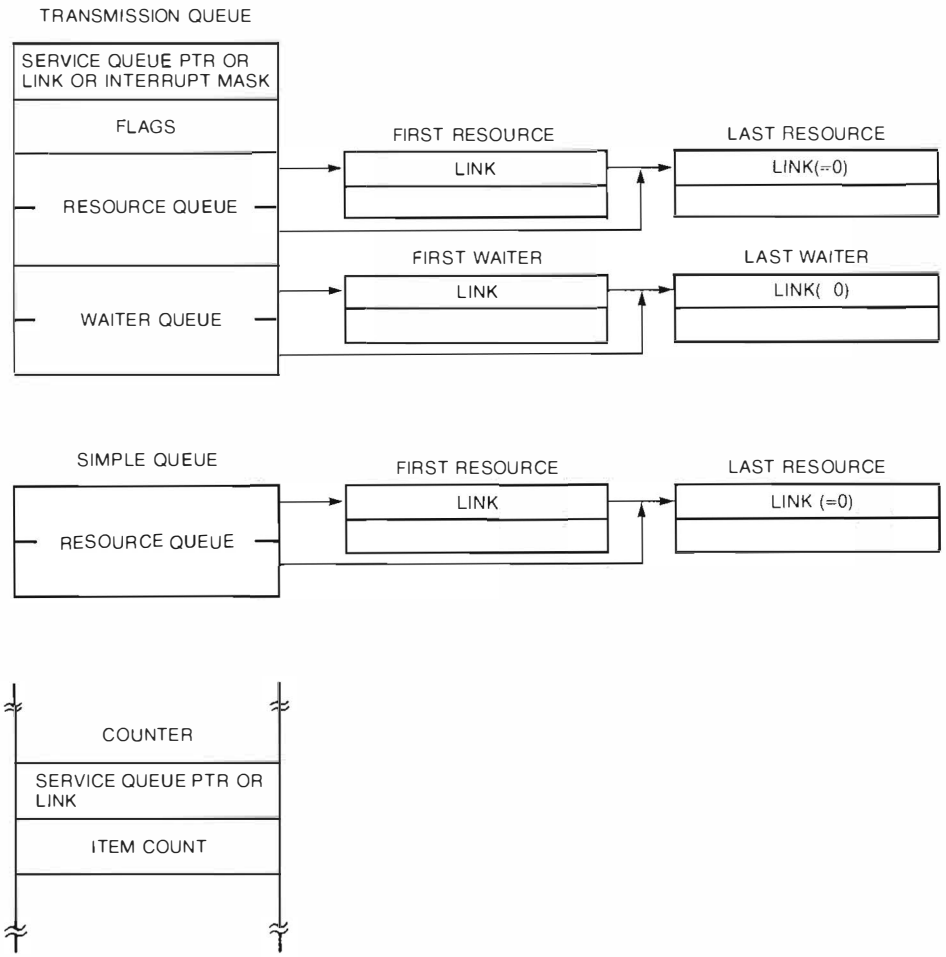


Figure 2 Transmission Queues

Mutual Exclusion

As mentioned earlier, the control bus provides an interlock operation consisting of a read of the target control-memory address followed immediately by a substitution of the constant 8000(hex) into that location. Each data structure described below contains a single location that is accessed via the interlock operation as part of accessing the data structure. The data structures are all organized so that the value 8000(hex) cannot be a valid value in that lockable location. Therefore, an interlock operation to that location that returns the value 8000(hex) implies that the data structure is currently locked. A simple write of a value other than 8000(hex) to that location will unlock the data structure. This mechanism permits each processor to hold locks on elements within its own interface without depriving other processors of lock access to their private data structures. Lock contention is limited in most cases either to pairs of processors contending for a mutually shared data structure or to the few structures, called common resource pools, shared by all processors.

Data Structures and Primitive Operations

The interprocessor data structures consist of transmission queues, simple queues, counters, and a specialized queue called the interrupt queue. The flow of information between these data structures is governed by other data structures called routes. These data structures and the operations performed on them are described below.

Transmission Queues Transmission queues pair resources with processes waiting for those resources. As shown in Figure 2, transmission queues have two physical queue heads, one each for resources and waiters, and linkage information used by the interrupt queue mechanism, described later. The "resource" associated with each queue can be any

physical or informational entity of interest to more than one process or processor in the system. Transmission queues are used for K operation-completion queues, free-buffer lists, free-control-memory lists, and P.io message queues.

The name "transmission queue" has its roots in a research project that was a precursor to the HSC project; unfortunately, the name implies a list of messages awaiting transmission, which is not the function of these queues. Transmission queues primarily serve a process scheduling function.

Simple Queues Simple queues are subsets of transmission queues used for queuing resources for which a P.io software process can never wait. Unlike transmission queues, in which unsatisfied receivers can block waiting for an item, simple queues are usually polled by unsatisfied receivers. As Figure 2 shows, a simple queue is nothing more than a queue head for the resource queue.

Counters To meet the goal of separating the P.io from the control data flow, a mechanism that will account for completed operations is required. For example, a single Mass Storage Control Protocol (MSCP) command to read many sectors is decomposed into several independent transfers, which will complete in an indeterminate sequence due to the effects of optimizations and error-recovery strategies within the controller. This mechanism will have to signal when the last of these transfers completes so that the response indicating an MSCP completion can be sent to the host. The data structure used for this purpose, called the counter in Figure 2, consists of a count field and linkage information.

Interrupt Queue The interrupt queue is a specialized form of the transmission queue. In this case, however, the linkage information is replaced by information used to generate an interrupt to the P.io. The resources on the resource queue for the interrupt queue are transmission queues that require scheduling action. This mechanism is explained later in the discussion of the send algorithm.

Routes Routes are arrays of elements (called route vectors) that control the individual steps (called stations) of a data-transfer operation. MSCP commands are decomposed into individual transfer-work descriptors that describe a manageable fraction of the overall transfer. The K's and the software processes operate upon these transfer descriptors, as described later. The descriptors move from work queue to work queue, traveling through the subsystem. Each station of the route guides a processing element in determining which operation to perform on the transfer, and what to do with the completed result.

For example, a simple disk-read operation is a two-station route: the first station is the K.sdi to source the sector data into the HSC data buffers; the second station is the K.ci to transmit data to the host. In contrast, a disk write-compare operation is a four-station route:

1. The first station, the K.ci, fetches data from host memory.
2. The second station, the K.sdi, writes data to the disk.
3. The third station, the K.sdi again, reads data from the disk.
4. The fourth station, the K.ci again, fetches the original data again from host memory and compares it with the data read from the disk.

As shown in Figure 3, route vectors contain four items of information:

1. An opcode to dictate the operation to be performed at this station
2. A destination indicator to determine where the completed operation should be sent if it succeeds

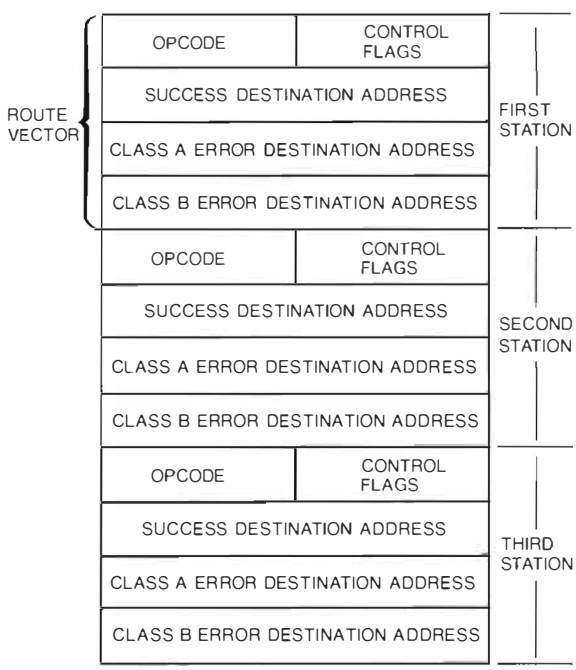


Figure 3 Route Vectors

```
Lock resource queue head and obtain value.
IF resource queue head NOT 0 and NOT 1:
     /*Resource available on queue.*/
     Remove item from head of resource list.
     Write link from removed item into resource queue
     head to unlock.
ELSE
     /*No resource available on queue.*/
     IF target queue is a transmission queue AND
     receiver is a software process:
          /*Indicate software process waiting for resource.*/
          Link process control block on waiter queue.
          Write "1" into resource queue head to unlock.
     ELSE
          /*Receive failed. No item available.*/
          Write "0" into resource queue head to unlock.
```

Figure 4 Algorithm for Receive Primitive

3. Additional destination indicators to define where the completed operation should be sent if it fails
4. Control flags to govern resource allocation and route termination

Receive and Send Standardized receive and send primitives act on both the transmission queues and simple queues.

By convention, a resource queue head in a simple queue or a transmission queue will be zero if the queue is empty and there are no waiters in the queue. The queue head will be one if the queue is empty and one or more software processes are waiting for a resource. And it will be greater than one if resources exist on the queue. By definition, queue heads for simple queues cannot contain a value of one since there can be no waiters on simple queues.

Figure 4 shows the algorithm for the receive primitive. If a software process executes this primitive and finds no resource, the process control block will be queued on the waiter queue and the value of the queue head changed to one.

The send algorithm, shown in Figure 5, is much more interesting than the receive algorithm. The same algorithm can be used to send to both simple queues and transmission queues without knowing the type of destination queue beforehand. The algorithm is driven by the queue head contents to the correct final result. After the lock is obtained, the resource queue is checked for a non-empty condition. If the resource queue is not empty, the new resource will be added to the queue, and the operation is finished. If the resource queue is empty, the queue head value will be checked to determine if any P.io software processes are waiting on the queue. If the queue head value is zero (no processes waiting), the new resource will be added to the queue, and again, the operation is completed.

If the queue head is one, however, (indicating that a software process is waiting for the queue), the queue is by definition a transmission queue. In this case, the sender uses the linkage information to turn the transmission queue head itself into a resource that is sent to the "service queue" indicated in the linkage fields of the transmission queue. This algorithm is recursive and continues until a flag in the service queue indicates that the target queue is the interrupt queue. Once that happens, the linkage information in the interrupt queue head will cause an interrupt to the P.io.

The interrupt queue thus becomes a queue of queues as shown in Figure 6. The resources in the resource queue of the interrupt queue are transmission queue heads, each in turn describing sets of resources and processes waiting for those resources.

Upon taking an interrupt from the interrupt queue, the P.io processes each transmission queue linked on the interrupt queue. That action gives resources to each waiting process (until one or the other list has been exhausted) and schedules any unblocked processes for execution.

This algorithm dictates that only send operations providing a resource to a waiting process will result in entries on the interrupt queue. Moreover, only the transition of the interrupt queue from empty to non empty will result in a hardware interrupt. Thus the P.io is interrupted only when a send has occurred that is of interest to the P.io's process scheduler.

```
Clear link in item to be sent.
Lock resource queue head and obtain value.
IF resource queue head NOT 0 and NOT 1:
     /*Resource queue not empty. Add new item
       to resource list.*/
     Link new item at end of resource list.
     Write original resource queue head back to unlock.
ELSE
     /*Resource queue empty. Add new item
       to resource list.*/
     Link new item at end of resource list.
     IF resource queue head originally 0:
          /*No software processes waiting on queue.
            operation finished.*/
          Write new item address in resource queue head to unlock.
     ELSE
          /*Software process(es) waiting on queue, or
            target is interrupt queue. Determine which.*/
          IF flag indicates NOT "interrupt queue":
               /*Send transmission queue to its service
                 queue.*/
               IF flag indicates transmission queue not already
               linked on service queue:
                    Flag transmission queue as on service queue.
                    Write new item address in resource queue head to unlock.
                    SEND transmission queue to service queue.
               ELSE
                    /*Target queue is "interrupt queue".
                      Generate interrupt if appropriate.*/
                    IF flag indicates interrupt not already generated:
                         Flag interrupt as generated.
                         Use interrupt mask from queue head to
                         generate hardware interrupt.
```

Figure 5 Send Algorithm

Furthermore, multiple transmission queues can be linked on the interrupt queue, and all can be serviced at once. That capability allows multiple process-scheduling events from a single hardware interrupt. As the system gets busier, interrupt processing tends to consolidate due to this batching effect, and executive overhead decreases as a percentage of total cycles consumed. As the system gets busier, it actually becomes more efficient at processing scheduling events.

We noted earlier that this send algorithm is recursive. A resource can be sent to a transmission queue, which in turn can be sent to its service queue, which in turn can be sent to its service queue, and so forth, until the interrupt queue is reached. In practice, however, the current HSC implementations use only three levels (a queue of queues of queues of resources). The reason for that limitation is to curtail computation in certain critical sections of K microcode.

Downcount Figure 7 shows the algorithm for downcount operations on counters. When a downcount operation turns the value of the counter to zero, the counter will be sent to its service queue. Usually embedded inside other data structures, a counter represents a count of events that must occur before the operation in which the counter is embedded can commence. As each event occurs, the value of the counter is reduced with a downcount operation. When the counter value becomes zero, the counter (and therefore implicitly the structure in which it is embedded) are sent to the counter's service queue. This queue is usually a work queue for the processing of the larger structure.

For example, when an MSCP transfer command is processed, a data structure is generated describing the completion message to be sent to the host. This data structure includes a counter containing the number of component operations that must be finished before the completion message can be sent.

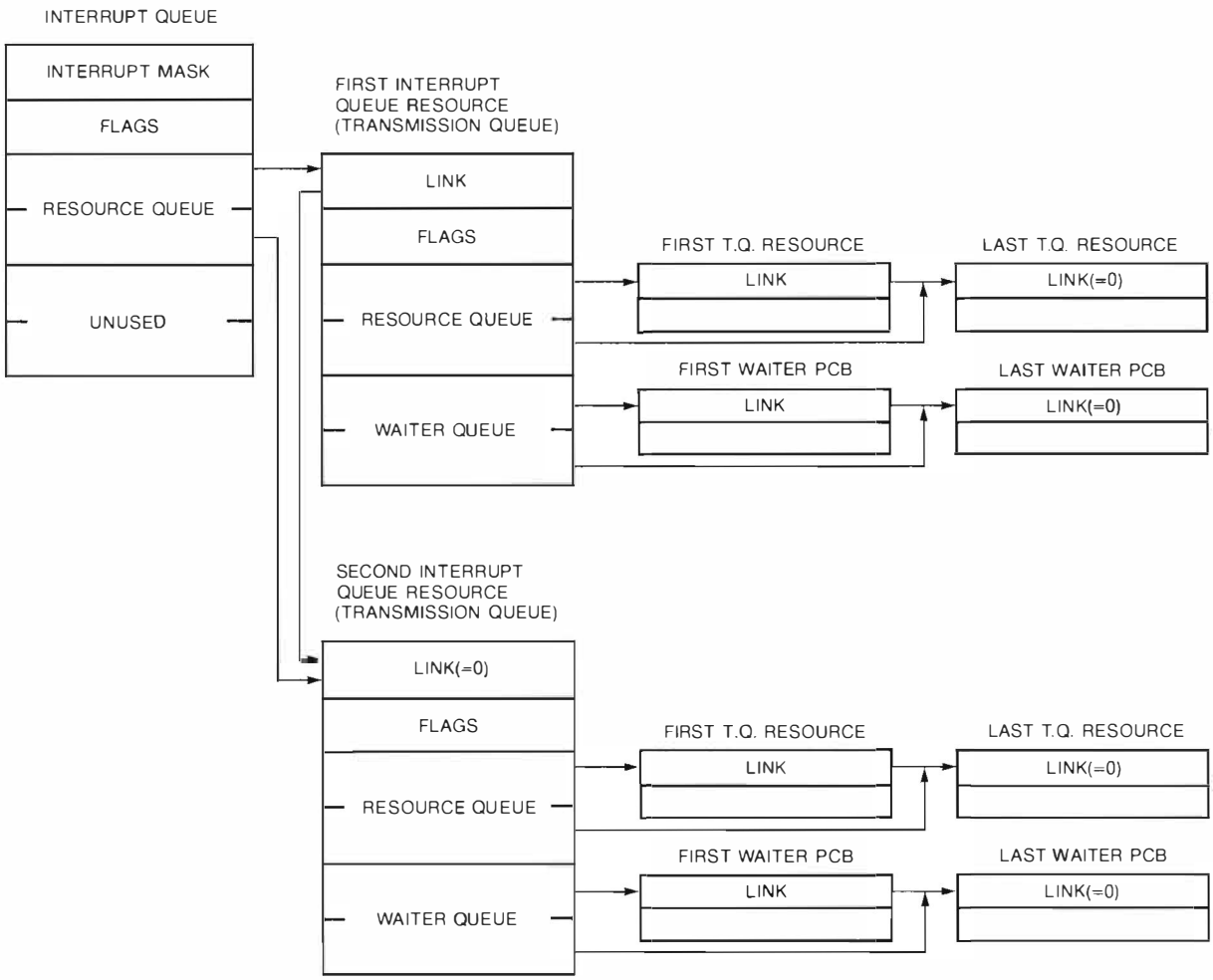


Figure 6 Interrupt Queue

```
Lock count word and obtain value.
Decrement value.
Write decremented value into count word to unlock.
IF decremented value = 0:
     /*Count transitioned to 0.*/
     SEND counter to service queue.
```

Figure 7 Algorithm for Downcount Operations

The service queue for this counter is the K.ci work queue for message transmission. As each component operation completes, a downcount operation is performed. When the counter value reaches zero, the counter (and the completion-message structure in which it is embedded) is sent to the K.ci message-transmission queue, and the completion message is transmitted to the host.

Work Queues and Work Descriptors

Each K has an interface to the rest of the system through its work queues and through the operations performed on standardized work descriptors. This section describes some of the unique aspects of each K interface.

K Control Areas The activities of each K are controlled through a master data structure in control memory called the K control area. These control areas contain the queue heads for the K work queues, resource queue heads for resources devoted to the K, and constants used as parameters for K operations. In all cases the queues in the K control areas are simple queues.

The K.ci control area contains work queues for message and data transmissions, and a pointer to the SCA open-connection database.

The K.sti control area contains work queues for data transfer and formatter communication operations, as well as information necessary to detect and report formatter state changes.

The K.sdi control area is similar to that of the K.sti. It contains work queues for data transfer and drive communication operations also, as well as drive state information. Unlike the transfer descriptors in the K.ci and the K.sti, however, those in the K.sdi are not queued directly on its work queues. Rather, transfer work is provided to the K.sdi by means of a special data structure, the disk rotational access table (DRAT).

As shown in Figure 8, DRATs consist of two types of elements: arrays of simple queue heads, one queue for each unique sector position in a single disk rotation; and a counter containing the total number of transfers linked on the various queue heads. For example, an RA81 disk drive has 51 sectors per track; therefore, a DRAT for this drive contains 51 transfer queue heads.

A transfer descriptor is queued to the transfer queue corresponding to the physical sector position at which the transfer is to start. Figure 8 shows two transfer descriptors, each describing the transfer of four contiguous sectors, and illustrates how they are linked in the DRAT. The K.sdi interrogates these transfer queues as the disk rotates. Each time a new sector pulse arrives from the disk, the K.sdi changes the queue polled for transfer work to the next transfer queue head in the array. Each index pulse from the disk resets K.sdi to the start of the array. In this fashion, the scanning mechanism for K.sdi transfer work is coupled in lockstep with the disk rotation. That coupling provides the basis for the rotational optimization algorithms used in the subsystem.

A unique DRAT is created for each set of transfers on a given track on a given disk. These transfers can be either individual pieces of a single, larger MSCP transfer or independent smaller MSCP transfers that fortuitously reference sectors on the same track. These smaller transfers can therefore be serviced during the same rotation. DRATs, interspersed with disk-head motion commands, are queued to the K.sdi to cause the transfer operations to occur. The typical work sequence of the K.sdi is as follows:

- Initiate head-motion operation
- Perform all transfers described in the next DRAT
- Initiate head-motion operation
- Perform all transfers in the next DRAT, and so forth

DRATs also provide a synchronization mechanism for fetching host data during a write operation to a disk. During a write, data must be fetched from the host into data buffers inside the subsystem before the actual disk transfers can occur. Because of limited buffering capacity, data cannot be prefetched for writes until reasonably close to the time when it will actually be needed. Figure 8 also shows retrieval queues, which are queues of transfer descriptors for data that must originate elsewhere before the disk portion of the transfer can occur.

As the first step of DRAT processing after initiating the head-motion operation, the K.sdi sends all work on the retrieval queue to the K that will provide the data (typically K.ci). The route vectors for these transfers are arranged so that, when the data-fetch operation completes, the transfer descriptor will be sent to the DRAT transfer queue for the sector at which the disk portion of the operation is to take place. In this fashion, data-fetch latencies for write operations overlap with disk-head positioning and rotation time. The DRAT transfer queues start out empty when the head motion is initiated, but they fill with work as the K's which source the data complete their portions of the data-transfer operations and forward the transfer descriptors to the DRAT transfer queues.

The DRAT contains a counter describing all the transfers that must take place before this DRAT can be completed. As each transfer completes, the K.sdi

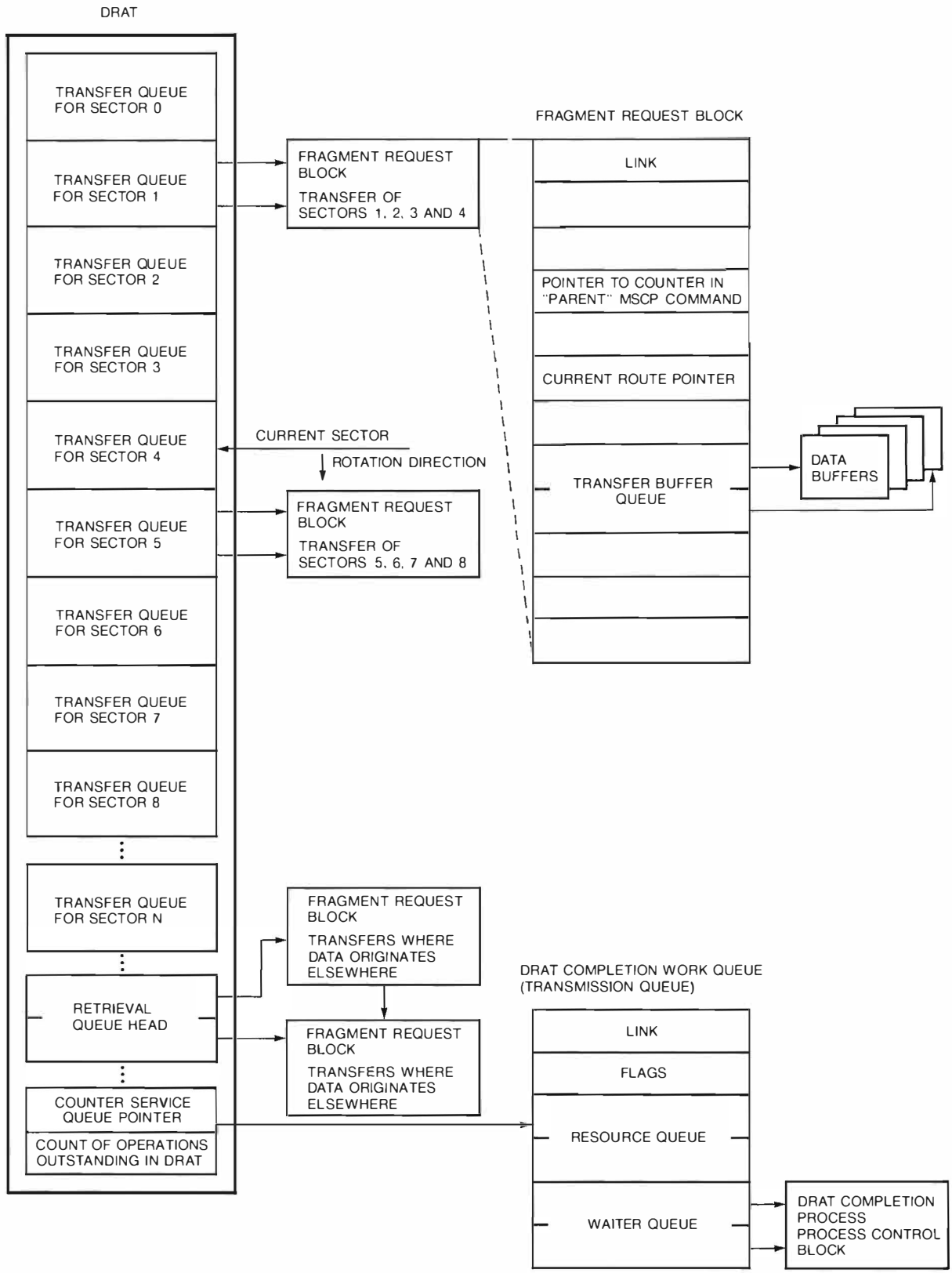


Figure 8 Disk Rotational Access Table

performs a downcount operation on the DRAT counter. When that counter becomes zero, the DRAT will be sent to a completion queue to be processed by the P.io software; the K.sdi then begins the head motion for the next track. Thus a non-zero DRAT counter indicates that additional transfer work must still be performed on the current track; the K.sdi cannot move the disk heads until all that work has been accomplished.

Fragment Request Blocks Thus far, we have referred to transfer descriptors in a generic sense. In actuality, all transfer operations within the subsystem are described by a standardized data structure called the fragment request block (FRB), shown in Figure 8. FRBs follow routes from work queue to work queue and cause the data-transfer operations to occur.

In addition to various constants needed at different steps of the transfer, FRBs contain three fields that are central to the subsystem data-flow algorithm. The first field is a route pointer, which points to the current route station in this FRB's route vector. The route pointer is advanced each time the FRB is forwarded to the next station on its route. The second field is a buffer queue head that links the data buffers used to hold the data during the operation. The third field is a pointer to a counter on which a downcount operation is performed when the FRB completes its route. As explained later, this counter usually resides in an MSCP control structure. The counter is the means by which the subsystem recognizes when all the individual components of MSCP transfers have completed.

Read Operation Control and Data Flow

The mechanisms and data structures described heretofore control data transfers more by using data structures than by using explicit algorithmic decisions by the various processors. In this section, we will follow a typical disk-read operation through the HSC design, demonstrating how the various data structures and operations interrelate. For simplicity, we assume that the sectors being read are contained on a single track, and that the HSC controller is idle when the read command arrives.

MSCP Command Arrival

The first event for our read operation is the arrival of the MSCP command message in a K.ci reception buffer. The sequence of events is shown in Figure 9. The K.ci first detects this arrival, inspects the specific CI protocol fields enveloping the MSCP packet, and determines if the message is valid over an open connection. The K.ci then adjusts the connection database as necessary to reflect any SCA credits passed with the command. Finally, the K.sci copies the message to a control-memory data structure called a host message block (HMB).

The HMB is sent to the reception queue for the connection over which the HMB was received. The reception queue is a transmission queue on which the P.io MSCP server is blocked waiting for work. The send algorithm sends this transmission queue to the interrupt queue and triggers an interrupt. The P.io process scheduler takes this interrupt and activates the MSCP server.

MSCP Server Processing

The MSCP server first parses the command and chooses it for transfer, since the disk is idle. The sequence of events is shown in Figure 10. (If the disk were busy, the request would be placed on the optimization queues.) Next, the server converts the HMB holding the MSCP command into an HMB containing the MSCP completion message that will be sent if the operation completes successfully. The server also initializes a counter in this HMB to a value of one. The server then designates the work queue for the K.ci message transmission as the one to which the counter (and thus the HMB) will be sent upon becoming zero. Finally, the server allocates a DRAT, builds a seek command for the drive, and begins to generate the individual FRBs for the transfer.

Each FRB except the last one describes four sectors of the transfer; the last FRB can describe up to eight sectors. This distinction is made because, if the transfer is small, the overhead of generating FRBs is large compared to the transfer time. Therefore, little of the potential rotation optimization will result, and it will be more advantageous to describe the entire operation with a single FRB. Note that as a transfer that does not cross a track boundary grows larger, however, the probability increases that rotational optimization can return a significant gain. In that case the FRB generation overhead is justified.

As each FRB is generated, it is initialized with the parameters needed to govern both the disk and the host sides of the transfers. In addition, the FRB's route pointer is initialized to point to the route vector for the first station on the disk-read route. If more than one FRB is generated from the seek command, the operation counter in the HMB will be incremented for each additional FRB. Each completed FRB is then queued to the DRAT transfer queue corresponding to the sector position at which the transfer starts. Finally, a counter within the DRAT is incremented to reflect the newly added

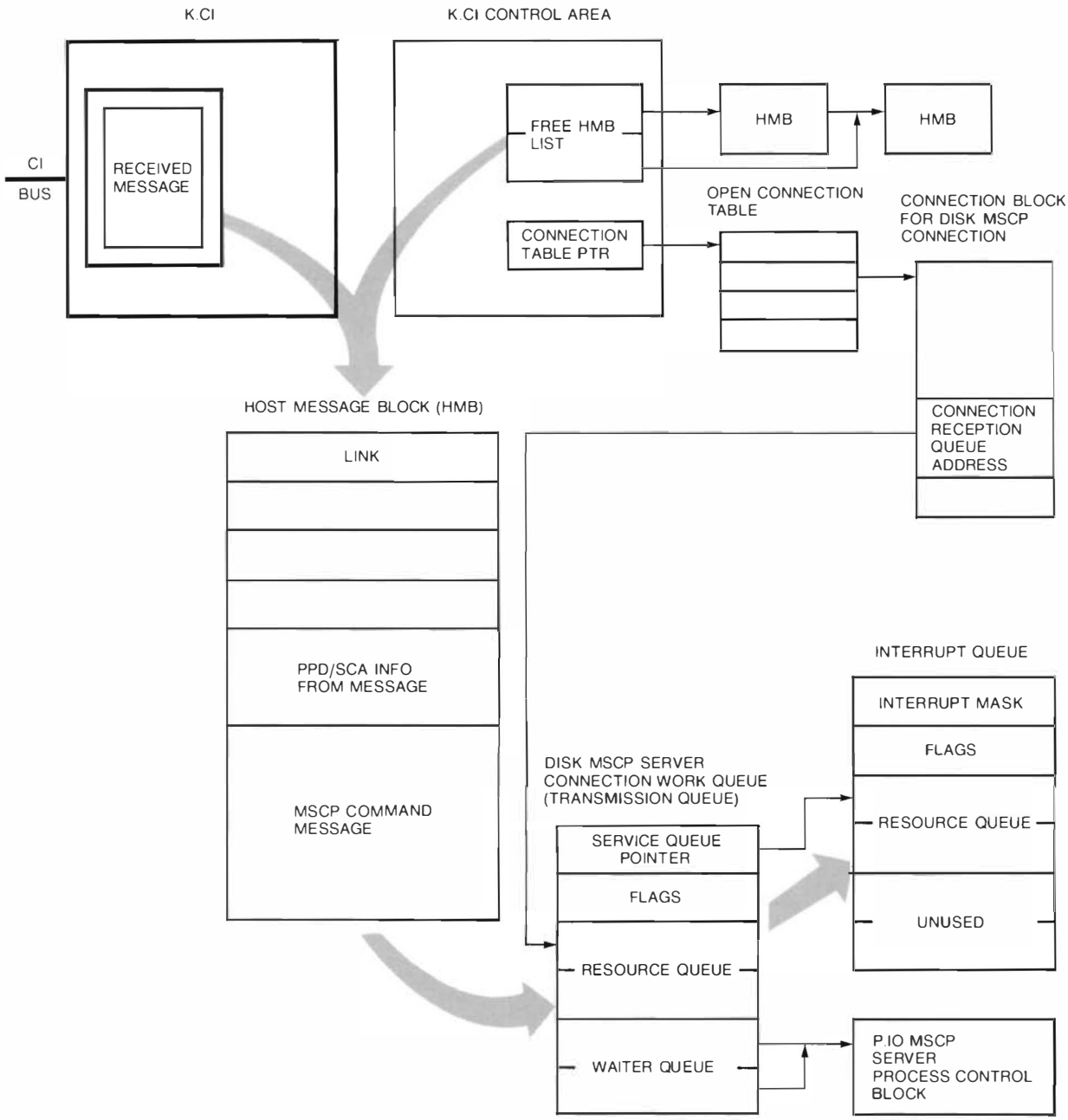


Figure 9 MSCP Command Arrival

operation. This DRAT counter contains the number of FRBs on the track, and its service queue is the work queue for the MSCP server to recognize DRAT completion. The MSCP server performs all these operations without blocking; thus no context switches are needed. When all these data structures have been generated, the seek command and the DRAT will be sent to the K.sdi's work queues. The MSCP server then returns to wait for more input commands.

The Disk Portion of the Transfer

Upon receiving the first DRAT, the K.sdi issues a seek command to the drive, then waits for the heads to settle and the drive to indicate its readiness to transfer. This sequence of events is shown in Figure 11. When the drive is ready, the K.sdi polls the various transfer queues, as described earlier. In turn, it encounters the first FRB to come under the heads. The K.sdi first uses the FRB's route pointer to find its

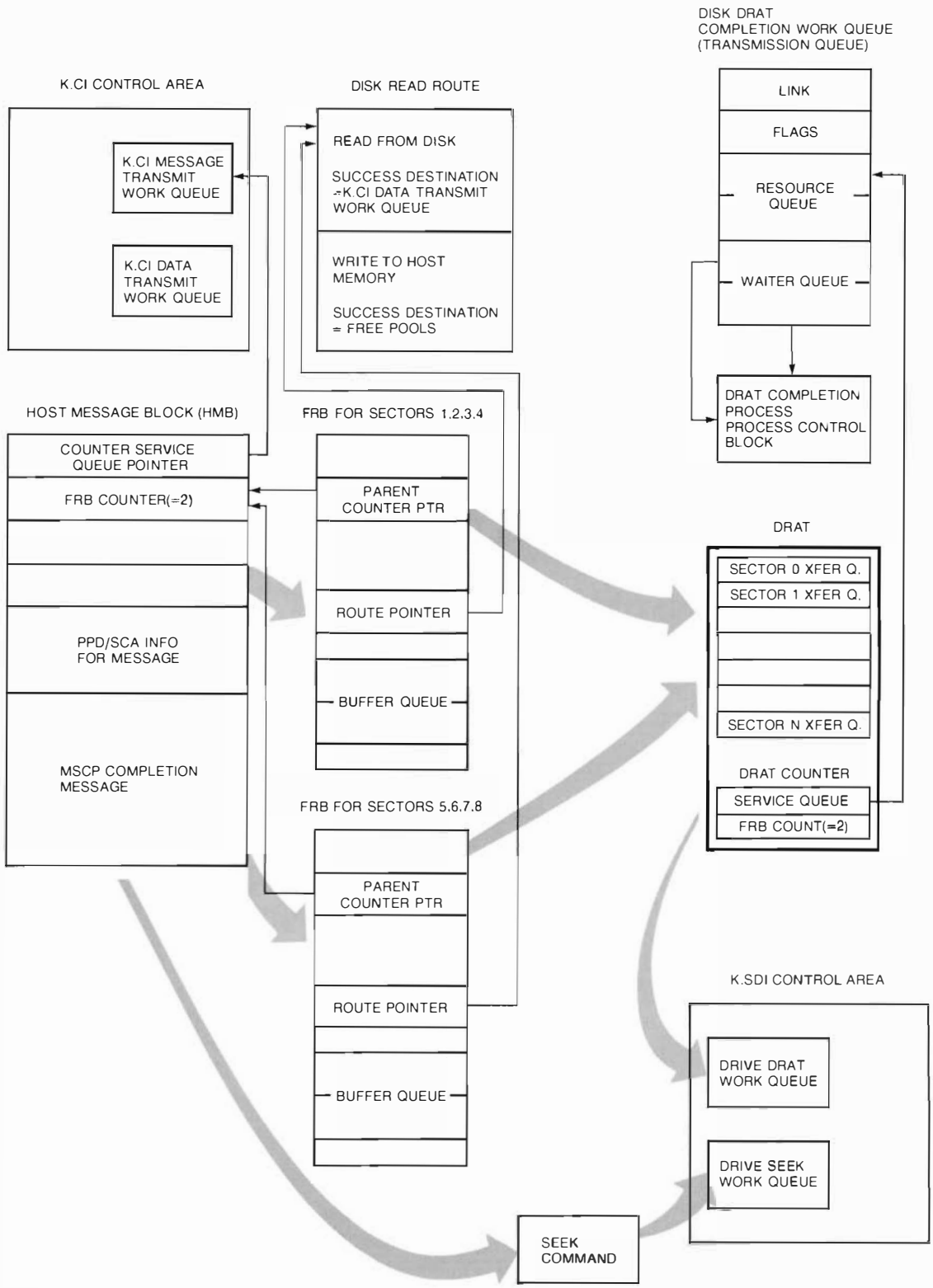


Figure 10 MSCP Server Processing

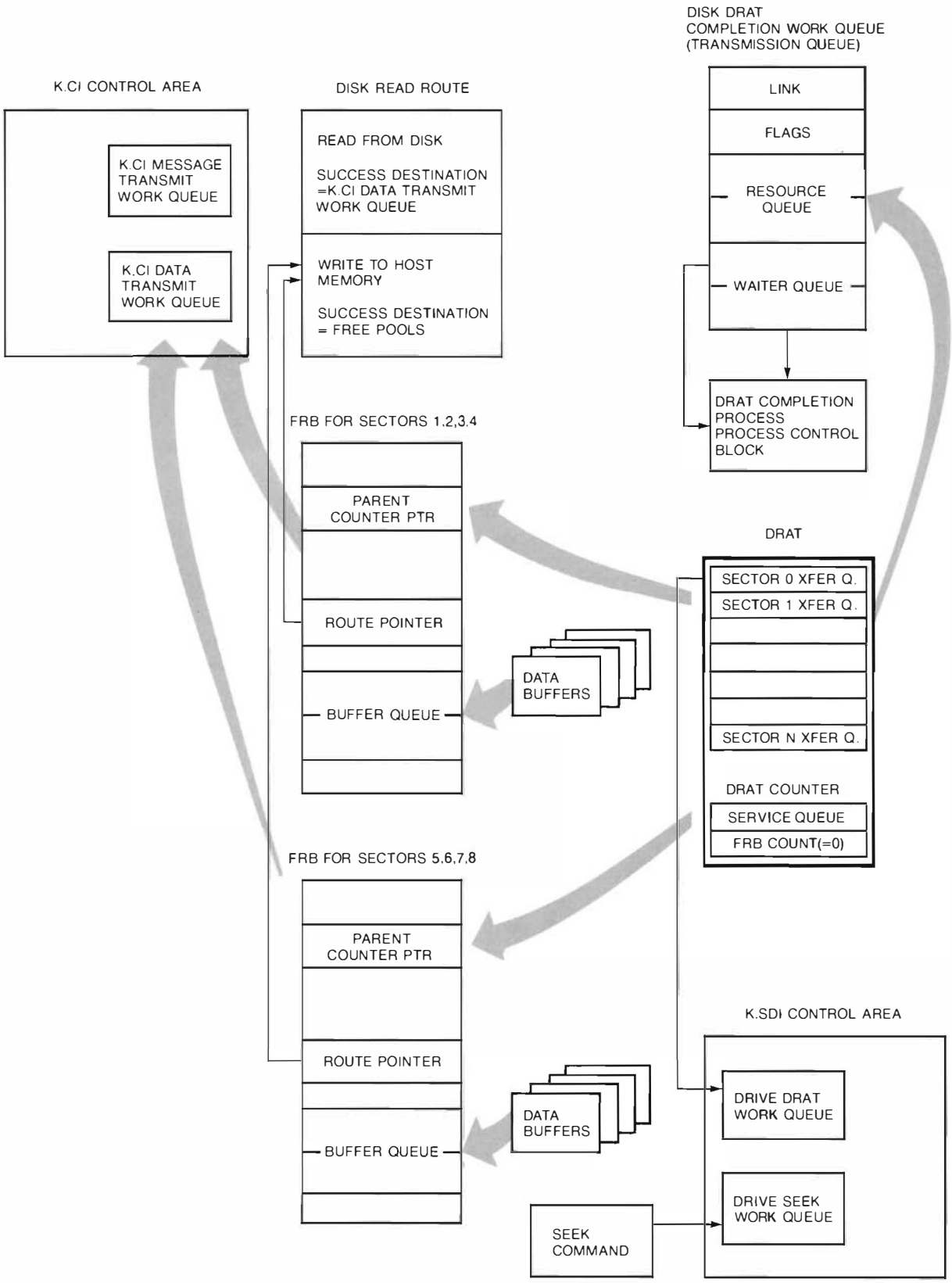


Figure 11 Disk Portion of Transfer

route vector and then determines that the operation is a read, meaning that buffers must be allocated. The K.sdi allocates the necessary data buffers and then reads the sectors. If no errors occur, the K.sdi advances the route pointer in the FRB, removes it from the transfer queue, and sends it to the success destination in the route. In this case, this destination is the data-transmission work queue for the K.ci. Finally, the K.sdi performs a downcount operation on the DRAT counter to reflect the completion of processing for an FRB.

This sequence repeats for each FRB in the DRAT until the last FRB completes. At that point, the DRAT counter will be zero, causing the DRAT to be sent to the counter's service queue. The K.sdi then waits for a new DRAT to appear in its work queues.

The service queue for the DRAT counter is a transmission queue that contains the completion process for the MSCP server. The send algorithm causes an interrupt-queue interrupt and the activation of the process. This completion process first checks the seek-optimization queue for this drive and, since in this case it finds nothing else to do, puts the DRAT in the DRAT free list.

The Host Portion of the Transfer

Upon receiving the first FRB from the K.sdi, the K.ci examines the route vector by means of the FRB route pointer. The K.ci then determines that the operation is a transfer of data to host memory. In turn, the necessary CI packets are generated from the data in the buffers attached to the FRB and transmitted to the host. As Figure 12 shows, after the last buffer has been transmitted and successfully acknowledged, the K.ci will advance the FRB route pointer and forward the FRB to the next station on its route. Because the K.ci is the last station for a disk read, the next-station destination is the free FRB queue. The send command then returns the FRB to the free list.

Flags in the K.ci route vector also instruct the K.ci to return the buffers attached to the FRB to the free-buffer list prior to routing the FRB. These flags also cause a downcount operation on the HMB counter. Thus the routing operation for this last station in the route involves automatic resource deallocation and completion accounting. Having completed and routed the FRB, the K.ci then returns to its work queues and repeats the cycle for the next FRB that arrives.

This process continues until the last FRB in the transfer has been processed by the K.ci. Here, the downcount operation on the HMB counter will decrease it to zero, and the K.ci will send the counter (and thus the HMB) to its own message-transmission work queue. After routing the FRB, the K.ci will find the HMB on its message-transmission work queue and transmit the MSCP completion message to the host.

Errors

The aforementioned routing and downcount operations occur as outlined only if no errors are encountered. If an error occurs, however, the transferring K routes the FRB to its error destination, avoiding the downcount operation. In the K.sdi case, this action prevents the DRAT from completing (which ensures that the disk heads will not move) until the error-recovery routines have tried to recover all the DRAT's FRBs. In the K.ci case, this process keeps the HMB from being sent to the message-transmission queue until all FRBs have been recovered. If the data can be recovered without having to resubmit the FRB to a K (e.g., purely mathematical corrections, such as ECC errors), the error-recovery routine will perform the routing and downcount operations.

Benefits

The control scheme for a read operation, described above, has many elements that seem quite complicated when viewed individually. Moreover, many of them do not return significant benefits for serial operations on a single, isolated command. The real power of this parallel design comes into play when large request rates are realized. In a busy system, the P.io executes two basic loops: processing new MSCP commands into suboperations on seek queues, and processing completed DRATs into DRATs that describe the next transfer in the sequence. At most, only one interrupt per MSCP command and one interrupt per track visited by a command are necessary. Sometimes, fewer interrupts are required.

In parallel with the DRAT-stuffing activity of the P.io, each K.sdi processes transfers that are synchronized with the head movement and rotation activity of its drives. The P.io maintains a queue of two DRATs per drive to the K.sdi. Upon completing the transfer on a track, the K.sdi can start the next seek on that drive immediately. Upon finishing an FRB transfer, each K.sdi will forward the data to the K.ci, which transmits data in parallel over the CI bus. As each error-free transfer completes, the K.ci automatically accounts for that completion. When all completions have finished, the MSCP completion message will be automatically transmitted.

Summary and Lessons for the Future

The HSC architecture and implementations have met or exceeded most of our original expectations. The architecture currently supports two hardware

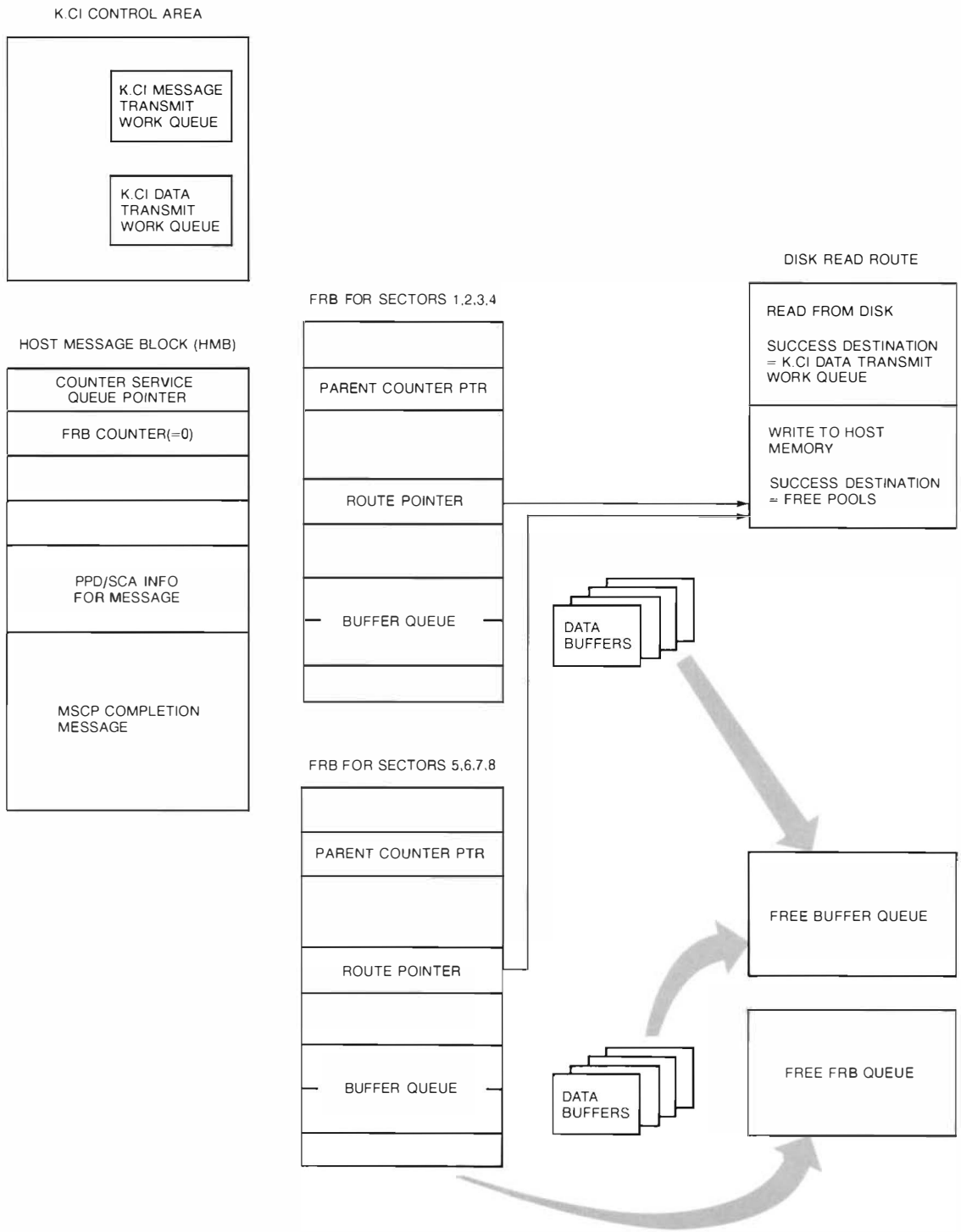


Figure 12 Host Portion of Transfer

implementations (the HSC50 and HSC70 devices), and four major software revisions have been made since the architecture was originally conceived in 1978 (based on advanced development work begun in 1976). All these changes were achieved with no substantive alterations to the basic machine design or the K microcode.

With a single policy microprocessor, the HSC70 design can process over 1000 requests per second and move approximately four megabytes of data per second on behalf of VAXcluster hosts. At the same time, the design can perform significant storage management functions, such as disk shadowing and device error recovery.

Based on almost a decade of hindsight, some of our initial design decisions were more restrictive than we realized.

- When we created the HSC design, we chose the 11/23 as the policy microprocessor for the HSC50 device. Although the 11/23 was reliable, inexpensive, and fast enough for our needs, its 16-bit address space significantly complicated our hardware and software design.
- In the software implementation, we placed much emphasis on maximizing performance by maximizing parallelism wherever possible. This emphasis extended to error recovery, the management of the CI communication state, and other infrequently executed algorithms. Although clearly warranted for the critical path, the use of highly parallel algorithms for the less frequently executed procedures added significantly to their complexity with no commensurate performance return. In retrospect, limiting infrequently executed code paths to straightforward, serial algorithms would have saved a lot of implementation and debugging time.
- With the exception of some of the device utility programs, the HSC P.io software is implemented entirely in machine language. At the time, performance considerations warranted such a decision for the critical-path routines. It is now clear, however, that the same infrequently executed code sections that would benefit from algorithmic simplification would also benefit from implementation in a suitable high-level language.

Kenneth H. Bates

Performance Aspects of the HSC Controller

The HSC controller tries to maximize its request and data rates while minimizing its request response time. Delays, which reduce performance, are caused by resource contention in the processor, the buses, and the I/O processes, as well as in the functions of the server, the processor, and the disk drives. Algorithms to minimize these delays and to take advantage of any idle time are incorporated in the design. However, the gains from one algorithm may come at the expense of another. Therefore, many techniques are used to solve this problem, including dynamic bandwidth adjustment, fragmentation, request merging, and seek reordering.

One feature that sets an HSC intelligent controller apart from its more traditional counterparts is its ability to enhance the performance of the I/O subsystem. Although the controller may not modify the functions of the I/O stream, it can improve the overall performance dramatically by influencing a number of areas that are amenable to optimization. Before investigating the means whereby these optimizations may be implemented, however, it is worthwhile to explore the areas contributing to the time taken by an I/O request.

Performance Equations

Although I/O operations are extremely complex, it is possible to ignore effects such as parallel processing in order to obtain a simplistic view of the division of time spent in the processing of an I/O request. Two equations may be constructed to represent the overall time taken by an I/O operation. The first equation treats the total time required for an I/O as being composed of several distinct time components:

$$t_{io} = t_{host} + t_{cont} + t_{ctrl} + t_{acc} + t_{xfr}$$

in which

t_{io} = The total time required for the I/O operation. This time span begins when the user program issues the I/O directive (e.g., read, write, QIO) and ends when that directive ends. This time is also referred to as the response time of an I/O request.

t_{host} = The time spent by the host to perform I/O initiation and completion processing. This component includes the processing time required for the user-level statement (read, write, etc.), as well as the time spent by the various drivers invoked during the life of the I/O operation. In the simplistic model portrayed here, this component also includes any time spent in the adapter.

t_{cont} = The time that I/O processing is suspended due to contention for various required resources. Note that this delay is not necessarily confined to any one component but may exist at one or more points in the I/O path.

t_{ctrl} = The time spent by the controller during the processing of the I/O request.

t_{acc} = The time required by the drive heads to move from their current position to the position required to transfer the requested data. This time includes both the physical movement of the heads and the time required for the media to rotate into the correct position.

t_{xfr} = The time required to transfer the specified amount of data. This component includes both the drive transfer time and the transit time of the various buses in the I/O path.

Since these components are additive, a reduction in any term on the right side of the equation will result in a comparable reduction of the total I/O time. Of these terms, only t_{host} and those portions of t_{xfr} and t_{cont} that happen outside the controller are considered fixed and invariant. Therefore, they cannot be optimized or reduced by the controller. This equation is of interest when it is desired to allocate the time spent by an I/O request to the discrete components along the path of the request.

The second I/O equation is recursive, treating an I/O operation as being composed of discrete operations at multiple levels, with each operation contributing an amount to the overall response time:

$$r_i = t_{si} + t_{wi} + r_{i-1}$$

in which

r_i = The response time of the I/O operation at level i.

t_{si} = The time required for servicing at level i (i.e., the time that level i performs useful work on the I/O operation).

t_{wi} = The time that the I/O operation waits at level i. During this period, no useful work is being done on the operation, although it is still at level i. This time reflects the delay contributed to the response by level i.

r_{i-1} = The response time at the next level below i. This time reflects the period during which level i passes work to a lower level for further processing. Note that r_{i-1} represents the response time of the next lower level; therefore, it implicitly includes the response times of all lower levels.

In other words, the response time of any component in the system is the sum of the service time and the delay contributed by that component, plus the response time of all lower levels in the system.

Although this statement may seem obvious, the view of what is meant by useful work can vary from one component to another. For example, the controller would certainly view the movement of the disk arm (a seek) at level i as a delay; the disk drive (at level $i-1$), however, would clearly see a seek as useful work.

Performance Metrics

The performance of a controller is generally measured using three metrics: the request rate, the request response time, and the data rate. The request rate is the rate at which the controller services I/O requests and is usually expressed in requests per second. Being highly dependent on the values for t_{acc} and t_{xfr} caused by variations of the input workload, this rate must be defined together with a description of the characteristics of the I/O stream used to obtain the rate. The request rate is usually specified as the point at which the controller saturates and cannot process any additional requests.

The second metric, the request response time, refers to the total amount of time required for an I/O operation, or t_{io}. This response time is also specified as a function of a given I/O workload, characterizing the response times associated with a particular I/O stream. Note that the request response time is not simply the reciprocal of the request rate, as is often thought, but may vary from one request to another. This metric should therefore be stated as a distribution function, either graphically or as a mean with a standard deviation.

The third performance metric is the data rate, which refers to the volume of data processed by the controller per unit of time. As might be expected, this metric is inversely proportional to the t_{acc} and t_{xfr} components of the equation. The data rate is usually specified at the point at which the controller is incapable of sustaining an increase in the data rate, even though the request-processing rate of the controller may not be saturated.

The twin performance goals of a high-performance controller are to minimize the request response time and to maximize both the request and data rates. To meet these goals, not only must all service times and delays associated with an I/O request be minimized, but the waiting times and delays associated with lower levels must be utilized to perform any possible additional work through the mechanism of parallel processing. Unfortunately, the algorithms invoked to increase the request rate often increase the response time of some requests, while the algorithms which minimize the request response time may decrease the request and data rates. Therefore, these goals are often at odds with each other. How the HSC intelligent controller balances these conflicting goals is the subject of this paper.

Contention

Contention arises whenever more than one entity requires simultaneous access to a single resource. Although this problem may be overcome by adding more resources, there may be certain physical or economical restrictions on the number of these resources available. Since contention may delay the completion of an I/O request, the HSC controller faces the task of how to resolve situations in which these resource contentions exist.

Processor Contention

One of the most important resources within the HSC controller is the inherent intelligence represented by the I/O processor. This entity first receives, decodes, and validates the Mass Storage Controller Protocol (MSCP) command from the host. The processor then performs a series of actions:

1. Transforms the host command into a format suitable for transmission to the drive over the standard disk interconnect (SDI) bus

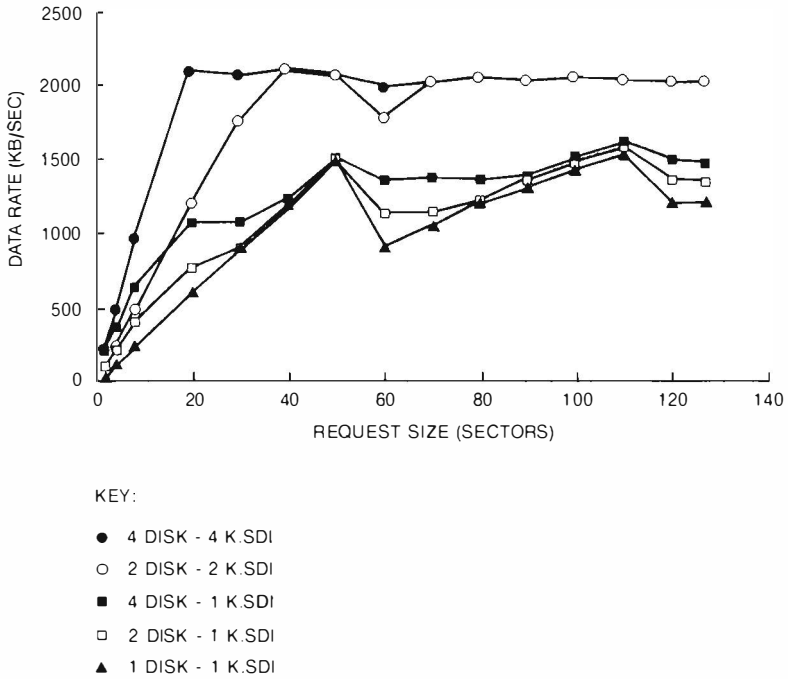


Figure 1 Effect of Processor Contention on Data Rate

2. Issues the command to the drive at the most propitious moment
3. Monitors the progress of the command while the drive executes it
4. Transfers data between the host and the drive
5. Verifies the successful completion of the command and attempts to correct any errors that were detected
6. Notifies the host when the command has completed

Not only must the processor accomplish all these actions quickly, but it must do so for many commands issued in rapid succession by different hosts and addressed to multiple drives. This process leads to contention.

To eliminate a large amount of this contention, the HSC controller divides the work among multiple processors. This division is based on the fact that all work in the controller can be divided into two major categories: policy determination, and policy implementation. Since policy determination includes algorithms and decisions that may change from one implementation to the next, a RAM-based processor called the P.io is entrusted with this task.[1] On the other hand, policy implementation, such as the SDI protocol, is relatively invariant and may safely be encoded in a ROM-based machine called the K.sdi. The relative speed difference between some functions also requires this division of labor. Incoming requests from the host arrive with a separation time of many milliseconds, whereas the timings on the SDI and CI buses are measured in fractions of microseconds. This difference is reflected in the power and speed of the processor chosen as the ROM-based I/O engine for the K.sdi, the 2901 bit-slice microprocessor.

Even with the speed advantage of these microprocessors, however, the relative speed of the SDI and CI buses would cause severe contention problems if only one microprocessor were dedicated to policy implementation. For this reason (as well as allowing expansion capability), a single microprocessor called the K.ci has been dedicated to CI-bus processing, while a single K.sdi has been dedicated to servicing a maximum of four disks, with additional K.sdi microprocessors being utilized for additional disks. With this arrangement, the software in each individual microprocessor can be optimized for one generic type of operation. If MSCP requests are directed to disks on multiple K.sdi processors, relatively fast disk operations can proceed in parallel with the processing of other requests by the slower P.io.

Processor contention can be demonstrated by the difference between multiple disks on one K.sdi and each disk on separate K.sdi's. Figure 1 was obtained

by running several tests with varying numbers of disks and K.sdi processors. Each test consisted of the same I/O work pattern: a read request was issued at the first sector on the disk, the number of sectors read was varied from 1 to 127, and three requests were always outstanding to the HSC controller to ensure that transfers were always taking place.

As shown, the difference in data rates between the two configurations becomes quite significant both as the number of disks connected to a single K.sdi increases and as the amount of data requested per transfer increases. Note that the data rate sustainable by a single disk at a request size of 20 sectors does not double when two disks are connected to the same K.sdi. Instead, the data rate increases only from over 500 kilobytes (KB) per second to around 750KB per second. When a second disk is connected to a separate K.sdi, however, the data rate for both disks is almost exactly double that for one disk, attaining well over 1100KB per second.

These differences are attributable to the contention caused by the requirement for simultaneous transfers from the different disks to the single K.sdi. Since one K.sdi cannot transfer data on two separate disks at the same instant in time, the data rate can increase only when the second (and subsequent) disks transfer data during the t_{acc} of the first disk (since the K.sdi will perform overlapped seeks, as explained later). In this context, the K.sdi may be viewed as a source of contention.

One of the tasks faced by controller designers is how to handle this source of contention. Within each K.sdi, contention arises whenever a disk is ready to transfer data while the K.sdi is already busy transferring data on another disk. Although the K.sdi cannot transfer data from two disks simultaneously, it can minimize the effects of this contention. This minimization is done by code that senses the current rotational position of each drive and initiates the transfer as soon as the data passes under the heads. This action is taken irrespective of the arrival order of the requests for data on the different disks, thus reducing the potential waiting time.

The K.sdi also utilizes the concept of a current drive for cases in which two or more drives are ready to transfer at the same time. Here, the K.sdi will first choose the current drive to initiate the transfer, then increment the current drive to implement a round-robin scheduling scheme among the competing drives. Once initiated, the transfer will continue uninterrupted as long as data remains to be transferred, which may be up to one full group on the disk.

The uppermost curve in Figure 1 illustrates a second form of contention: bus bandwidth limitations. The anticipated data rate for four disks, each transferring data on a separate K.sdi, is in excess of 4000KB per second with a 40-sector request size. The bandwidth of the CI780 adapter used in this test is slightly over 2000KB per second, however, so the actual data rate will be the lower of the two values. Since this decrease is caused by contention outside itself, the HSC controller can do nothing to obviate it.

Bus Contention

Other sources of contention are the various buses within the HSC controller itself. The data bus, the main route for data passing through the controller, can sustain a rate of 13.3 megabytes (MB) per second. This rate has traditionally been viewed as only 50 percent effective however, since most data must pass over the bus twice. On disk reads, for example, data must pass over the bus from the disk to the buffer memory in the HSC controller, and again when the same data is read from the buffer memory and transferred to the host over the CI bus. For this reason, the effective transfer rate of the data bus is generally thought to be limited to 6.6MB per second.

In fact, however, this figure is considerably higher than 6.6MB per second since the data transfer to and from the multiple hosts occurs by means of the K.ci, which has a maximum data rate of approximately 4.3MB per second. Subtracting this value, there is 9MB per second of bandwidth available for use.

Thus six RA82 disk drives, each with a sustained transfer rate of 1.4MB per second, will approach saturation on the data bus. Severe problems would result if a seventh drive began transferring data at high speed, thus exceeding the limitation of 9MB per second. The HSC controller deals with this problem in a manner that not only addresses this issue but also compensates for the varying transfer rates of different SDI devices. The controller utilizes a mechanism called the data-bus bandwidth semaphore, which functions as a throttle on the data transfer rate.

This semaphore is initialized to a value representing the maximum transfer rate the data bus can sustain. When a drive comes on line to the HSC controller, the disk-server software will interrogate the drive over the SDI bus. Among other items, the software determines the relative transfer rate of the drive, expressed in increments of 100Kbits per second. When a transfer to that drive is requested, the K.sdi will examine the drive's speed parameter to determine if it is less than the value remaining in the bandwidth semaphore. If so, the drive's transfer-speed parameter will be subtracted from the

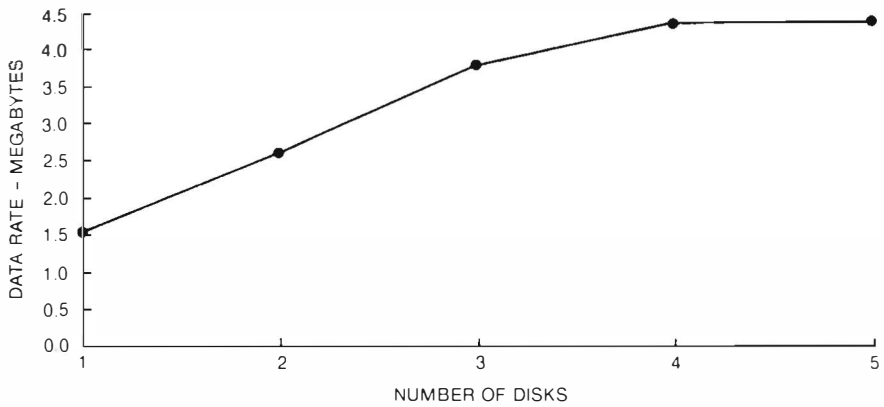


Figure 2 Effect of Bandwidth Semaphore on Data Rate

semaphore value and the transfer initiated. Upon completion of the transfer, the value of the transfer-speed parameter is added back to the value of the semaphore.

As multiple drives begin to transfer at the same time, this additive effect eventually causes the transfer-speed parameter to exceed the value remaining in the semaphore. When this condition occurs, the K.sdi will delay the next requested transfer until a disk completion increments the bandwidth semaphore to a value large enough to allow the new transfer request to proceed. In this manner, disks with differing transfer rates may be intermixed on the HSC controller without bus contention, and overly restrictive I/● initiation policies are not needed.

Figure 2 demonstrates the effect of this semaphore when a full disk read was initiated to a varying number of disks. These reads were initiated by the MSCP ACCESS command, which reads the data but does not return it over the CI bus, thereby eliminating the CI bandwidth restrictions discussed earlier. Each ACCESS command was directed to an RA82 drive on a single K.sdi.

As shown, the data rate for one drive is about 1.5MB per second and increases in a very linear manner as the second and third drives are added. The data-bus bandwidth semaphore is initialized to a value of 661, or 66.1Mbits per second. (This value is considerably less than half the maximum data-bus bandwidth of 9MB, or 72Mbits per second, for reasons explained later.) Since the transfer-speed parameter of the RA82 drive is 192 (peak rate of 19.2Mbits per second), the addition of a fourth drive exceeds the semaphore value by 107. Thus the transfer is postponed slightly, causing the nonlinearity in the curve. As the fifth drive is added, the increase in the data rate becomes miniscule, with the semaphore allowing additional transfers only when one drive is occupied performing a seek operation.

Since the K.ci is the lowest priority requester on the data bus, it is not included in the data bus bandwidth calculations. In essence, it will always transmit and receive on the CI bus, using whatever bus cycles remain after any bus use by the K.sdi's.

Process Contention

The HSC software is a highly complex real-time multitasking operating system, consisting of more than 50 separate processes. Therefore, some form of process contention is bound to arise. An example of this contention is the process that is responsible for correcting errors detected by the 170-bit error correcting code (ECC). This software is totally compute bound and can take nearly 100 milliseconds of continuous compute time in severe cases. Since this compute time prohibits other work from being done on the system, the ECC process has a very low priority relative to other tasks, allowing it to work in background mode. In this manner, high-priority requests for work from the K.sdi and the K.ci will be serviced with minimum waiting; the time for the already lengthy ECC process will be increased by only a small percentage.

This contention is quite important when considering the division of work for a typical I/O request. As mentioned earlier, the P.io merely provides the policy determination of which request to transfer next; the K.sdi and the K.ci do the actual work. This work must be directed from the P.io, however, and no new I/● requests will be issued if the processor is blocked awaiting the completion of ECC processing. Now, t_{ctrl} typically contributes only a fraction of the t_{io} time, and t_{wc} and t_{xfr} are idle times to the P.io. Therefore, it is worthwhile to interrupt the

compute-bound ECC process for the small time required to initiate new work. After sending a new I/O request to the drive for action, the P.io will return control to the ECC process, which will continue its compute-bound activity during the drive's t_{acc} and t_{xfr} time.

Resource Contention

As might be expected, some of the resources subject to contention are the buffers utilized by the CI780 port in the VAX CPU to receive the data sent by the HSC controller as a result of a disk read operation. Surprisingly, the K.ci is not only aware of this possible contention in the VAX host, but has been designed to reduce the amount of contention for these external buffers whenever possible.

The contention arises because the CI780 port maintains two buffers to receive the incoming data. The K.ci can send data at a rate in excess of the port's capability to empty and recycle the buffers. Therefore, if more than two buffers are sent in rapid succession, the CI780 port will be forced to reject (NAK) the incoming data, thus necessitating a retransmission by the K.ci.

To reduce the CI bandwidth wasted on retransmissions, the K.ci has been designed with the reception capability of the host-buffer in mind. The K.ci sends data to the host one buffer at a time. After sending a buffer, the K.ci checks to see if more data is waiting to be transmitted to another host. If so, the K.ci will send to the next host, proceeding in a round-robin fashion among all hosts. In this manner, a sequence of buffers to be sent to one host will be sent not in one burst but gradually, over a period of time. Since the K.ci effectively introduces a pause between the transmission of buffers to a host, the receiving CI780 port can empty one buffer before receiving another, thus eliminating the need for the K.ci to transmit the same buffer more than once.

Although this scheme will be effective only when there is data to be sent to more than one host, it is precisely at this time that a retransmission would have the greatest impact on performance. Without this scheme, the second host would have to wait for one (or more) retransmissions to the first host, effectively doubling the potential delay the second host would see. If data exists to be sent to only one host, then although a retransmission will undoubtedly be required (if more than two buffers are sent), other hosts will not be affected by this delay.

In a similar fashion, the contention for data buffers within the HSC controller also exists for the K.ci when the host issues a disk write command. In this instance, the K.ci must supply buffers to receive the data sent by the host to be written to disk. On one hand, the K.ci must not always allocate all the required buffers since this allocation could potentially remove buffers from service for a lengthy period of time, thus preventing their use elsewhere. On the other hand, a last-minute allocation of buffers is a risky business since if they are not available when the data comes in, the K.ci must NAK the data from the host and request retransmission, which causes delays. The K.ci addresses this problem with both an internal hardware solution and a software solution, while allowing full control of the policy to reside in the P.io.

Within the K.ci, there are two hardware buffers dedicated to the reception of data received from the CI bus. Once received into these buffers, data must be copied into a data buffer that the K.ci has allocated before more data arrives over the CI bus, or else the data will be lost. To control the number of data buffers to preallocate, the P.io will inform the K.ci of the maximum number of buffers to preallocate. The K.ci will maintain two internal variables, one keeping the maximum number of buffers to allocate less the number currently allocated (called SP.MMA), the other keeping track of the number of buffers needed less the number currently allocated (called SP.NMA).

Upon determining that data is required from the host, a K.ci will add the required number of buffers to SP.NMA and then check both variables. If both are greater than zero (implying that buffers are needed and the allocation limit has not been reached), the K.ci will allocate a data buffer, decrement both variables, and repeat the check. If either variable is less than or equal to zero (implying either sufficient buffers have been allocated or the K.ci is at the allocation limit), then the K.ci will transmit a request to the host for the data. SP.MMA will be incremented after the data is received from the host and transferred from the reception buffer into the data buffer.

By following this algorithm, the K.ci will allocate some portion of the required buffers prior to requesting data from the host but will limit this allocation in order not to severely deplete the pool of free buffers. Since the values for SP.MMA and SP.NMA are determined from information obtained from the P.io, this policy is fully controlled by the P.io.

If data arrives from the host and there is no data buffer available (possibly as a result of this algorithm), the K.ci will not retain the data in the CI reception buffer because the K.ci would then have to send a NAK, thus wasting CI bandwidth. Instead, the K.ci receives the data and simply discards it, requesting it again at a later time. By doing that, the reception buffer will be free to receive more data, thereby minimizing the impact caused by the K.ci's

asking for more data than it could momentarily receive. The number of retransmissions of the requested data will be the same or less than the NAK case, thereby conserving valuable CI bandwidth.

Latency

As mentioned earlier, the t_{wi} component of response time represents the waiting time, or request latency, associated with a particular request at level i. Latency is also included in r_{i-1}, which represents the response time for the work done at I/O levels below i. From the viewpoint of the user who has issued the high-level I/O request, however, the total time between the beginning and ending of the request is considered to be request latency, even though lower levels are performing useful work. These latencies may be subdivided into several distinct areas within the HSC controller.

MSCP Processing Overhead

The first task undertaken by the MSCP server is the verification and accounting associated with all incoming MSCP requests. Although certainly required, the various checks and validations consume a significant amount of time, which contributes to the latency of the request. Therefore, the only possible reductions in request latency come from using highly optimized software coding and faster processors, like the ones in the HSC70 controller.

The software in the HSC controller is not only optimized (e.g., the register autoincrement mode when traversing sequentially through structures) but also takes advantage of certain presumed I/O characteristics of the VMS system. For example, it has been observed that significantly more VMS I/O requests are reads than writes.[2] Because of this characteristic, the default code paths within the HSC software are set to a read case, allowing the majority of the I/O handling to proceed slightly faster. Although a small amount of code must clearly exist to handle the other cases, this code is invoked less frequently. Small optimizations such as this one contribute to the latency reduction.

Following its verification, the request must be decomposed into fragments suitable for sending to the K.sdi for the actual work. In addition to the processing delays, contention may also be encountered here because additional resources may be required to contain these fragments.

The final portion of overhead contributed by the MSCP processor is, ironically enough, the delays inherent in the code designed to reduce request latency. This code, which includes features such as fragmentation and seek ordering policy, will contribute a small delay by its very presence. On a system with a small load of I/O requests, this overhead may contribute a noticeable amount to the request latency since the optimizations within the HSC controller do not begin to take effect until the load increases. The HSC code attempts to minimize the impact of these intrusions by not invoking optimization algorithms when the workload is small. Some small amount of code must remain, however, contributing a small but measurable amount to the latency of the request.

K.sdi Processing

Although the K.sdi processors are extremely fast, some delays due to internal processing are inevitable. Each K.sdi consists of two 2901 microprocessors, one to process data transfers, the other to process SDI control messages. These processors share a common 150-nanosecond (ns) time-sliced clock, resulting in an effective cycle time of one 150-ns instruction every 300 ns for each microprocessor. The data received from the drive is not buffered within the K.sdi. Therefore, that data must be removed from the internal holding register and placed on the data bus before the next data word arrives, or else a data-late condition will occur. (The data-late condition requires an error-recovery procedure, necessitating a reread of the failing sector at the expense of another complete revolution of the disk.)

To prevent the data-late condition from occurring, the bandwidth semaphore previously mentioned is utilized as a latency semaphore. That action effectively moves the semaphore from the frequency domain into the time domain, although it retains its representation of the peak transfer rate in increments of 100Kbits per second. To derive the actual value chosen for this semaphore, let us consider two hypothetical drives, each chosen for a transfer rate that corresponds to an integral number of K.sdi clock cycles. If the first drive can transfer data at a peak rate of 21.33Mbits per second and the second at 26.67Mbits per second, the first will transfer one complete 16-bit word every 750 ns, and the second will transfer the same word in 600 ns.

Since the current limit of the SDI bus is 22.5Mbits per second, the fastest drive that can be supported lies somewhere between the limits set by these two drives. Between four and five K.sdi clock cycles are therefore required to transfer a complete word. To eliminate any possibility of a data-late condition, the shorter of these two transfer times must be used, requiring every fourth clock cycle to be available on the data bus. Now, the data-bus interface in the K.sdi cannot begin a new data-bus request in the same cycle in which the previous request was

granted. Therefore, the number of available cycles must be reduced by one to account for this possibility, which changes the requirement for the fastest drive to every third clock cycle being available for data to be moved to and from the data bus. Since drives that can transfer at least 21.33Mbits per second will require every third bus cycle, the bus will be 100 percent utilized (from a latency standpoint) when three drives are transferring. The rates of three of these drives total 64Mbits per second, which translates into a bandwidth semaphore value of 640. In essence, if drives faster than 21.33Mbits per second are utilized, the bandwidth semaphore must be less than 640.

Following this same line of reasoning, drives capable of transferring between 17.78 and 21.32Mbits per second require 900 ns for every 16-bit word, or every fourth bus cycle when all previously mentioned effects have been considered. Using four drives at 17.78Mbits per second yields a peak rate of 71.1Mbits per second, or a bandwidth semaphore of 711 or less if drives faster than 17.78Mbits per second are allowed.

The current value of 661 (66.1Mbits per second) will support drives with peak transfer rates greater than 17.78Mbits per second and less than 21.33Mbits per second, with a slight safety margin.

Disk Overhead

Although somewhat paradoxical, the actual transfer of data by the disk drive is viewed as latency by higher I/O levels. Therefore, data transfer must be considered as a source of delay. Once the request has been issued to the drive, several sources of latency become evident.

The first source is the seek time, or the time required to position the disk head to the required area on the disk. Although the movement itself is unavoidable, the P.io attempts to ameliorate this delay by reordering multiple requests to minimize the sum of their delays. The K.sdi will also issue the SDI command to initiate the seek and then look for work to perform on another drive while waiting for the seek to complete. In this manner, one K.sdi can have up to four seek requests outstanding simultaneously on four different drives.

Once the disk head reaches the desired cylinder, a second source of latency is introduced while the disk surface rotates to bring the sector containing the desired data under the head. With a rotational speed of 3,600 RPM, this motion can take a significant amount of time, potentially exceeding the time required to move the heads across several cylinders.

The final source of latency introduced by the disk is the actual transfer of data from the disk to the controller. A finite amount of time is required for this action, which could become significant for large transfers.

Latency Optimizations

Since the sources of latency were known, the HSC controller was designed to reduce the effects of each. As mentioned earlier, including algorithms to reduce the latency and increase the data rate may also introduce an additional delay into the chain. Therefore, the HSC design bypasses these optimizations if they contribute to the delay rather than reduce it.

Dynamic Bandwidth Adjustment

Within the HSC controller, the K.sdi processors are prioritized according to their requester numbers — the lower the number, the lower the priority. The priority arbitration will be invoked only when more than one K.sdi attempts to access the bus simultaneously; control of the bus is awarded to the K.sdi with the highest priority. If a situation arises in which a number of low-speed drives are on high-priority requesters, these drives may be selected, thus delaying the transfer initiation on the high-speed drives.

To prevent that situation, the P.io implements a dynamic bandwidth-adjustment algorithm. This algorithm, invoked whenever a new disk drive becomes available to or disappears from the HSC controller, loads the SDI transfer-speed parameter for each drive. Obtained from the drive and made known to K.sdi, this parameter is compared with the bandwidth semaphore when a transfer is to be initiated, as explained earlier. Upon executing, this algorithm scans every disk on every K.sdi on the HSC device, sequencing from the K.sdi with the lowest priority to that with the highest. Within each K.sdi, the algorithm compares the speed parameter for the drive in question with the highest speed seen to date on any previous K.sdi. The algorithm then assigns the higher of these two values to the drive.

This algorithm promotes slower drives on a higher priority requester when a faster drive has been seen on a lower priority requester. By doing that, the relatively slow drive on the higher priority K.sdi has its SDI speed rating increased to that of the fastest drive seen on any previous K. Thus, although it may still initiate the transfer in preference to the higher speed drive, the slower speed drive will decrement the bandwidth semaphore as if it were capable of transferring data at the same rate as the fast drive. When multiple drives begin transferring or they compete for the bandwidth semaphore to

initiate a transfer, this algorithm increases the weight given to relatively slow drives on a higher priority K.sdi. The lower priority drives then have a greater chance to transfer.

Routing

As noted earlier, the P.ioc performs policy determination, and the individual K.sdi and K.ci processors actually implement the policy. In keeping with this philosophy, the concept of routing is an integral part of the HSC design. With this concept, the individual fragments containing the transfer information also contain pointers to a memory structure known as a route. This structure consists of several items, among them the operation code and the location of the next station on the route. Each K processor is aware of these routing structures and can cause a data fragment to flow from one station on the route to the next without intervention by the P.io.

An example of this routing occurs when an MSCP read request is received by the HSC controller. The P.io will set up a fragment describing the transfer, including a pointer to the route associated with a read request, and send this fragment to the K.sdi for processing. The K.sdi will note that a data buffer is needed, allocate that data buffer, read the data on the disk pointed to by the fragment into that buffer, and then forward the fragment to the next station on the route. For a read command, the next station will be the K.ci; the opcode of that station will instruct the K.ci to send the data contained in the attached buffer to the host. Once the data has been sent, the station code will instruct the K.ci to return the data buffer to the pool of free data buffers and send an end message to the host to complete the transfer.

The important aspect of this scenario is that once the initial MSCP verification and fragment set-up have been completed by the P.io, all work will be handled by the individual K processors without P.io intervention. The P.io is now free to process other requests since the set-up by the P.io occupies only a small fraction of the total time required for the transfer. This feature allows the HSC design to attain high data and request rates.

Rotational Position Sensing

The work request that the P.io sends to the K.sdi for processing is contained in a disk rotational access table (DRAT). A DRAT is restricted to a single group on the disk surface and may contain multiple discontiguous transfer fragments. To reduce the time spent waiting for disk rotation, the K.sdi does not look in a DRAT for a fragment and then wait for the disk surface to rotate to that position. Instead, the K.sdi monitors the current position of the disk surface relative to the disk head, then looks into the DRAT to see if any corresponding work requests exist. If the DRAT contains more than one transfer fragment, this technique will allow the transfer to begin as soon as any data is accessible.

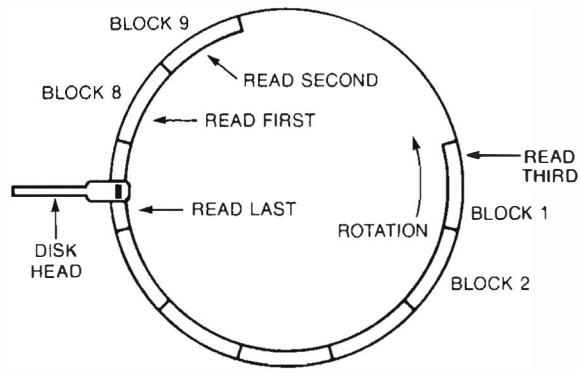


Figure 3 Request Fragmentation

Fragmentation

More than one fragment must be contained in a DRAT to fully realize the benefits of fragmentation. Although the HSC controller clearly cannot manufacture I/O requests to fill a void, it can decompose one contiguous host request spanning multiple sectors into several distinct fragments. Since this decomposition takes a certain amount of time (contributing to t_{ctrl}), the HSC controller will not decompose a request less than eight sectors long. If the request exceeds eight sectors, the P.io will create a fragment four sectors long, reduce the original request by four sectors, and then check this remainder. If the remainder is greater than eight sectors, this decomposition will continue until the remaining transfer fragment has been reduced to eight or fewer sectors. At this point, all fragments will be placed in a DRAT and issued to the K.sdi for transfer. Since the K.sdi can begin the transfer whenever the disk head passes over a transfer fragment, transfers could begin earlier than would otherwise be possible. Figure 3 shows the request fragmentation.

As an example of this decomposition, consider a transfer request for 57 blocks. On an RA82 disk (having 58 sectors and 57 logical blocks per track), the rotational position of the head when the request is received is probabilistic. That is, the odds are only 1 in 58 that the head is positioned at the beginning of the transfer. With fragmentation, however, the original request will be subdivided into 14 individual fragments: 13 with four sectors, and one with five. With this arrangement, there are now 14 possible places for the transfer to begin, increasing the

odds to 14 in 58. Of greater importance is the actual latency reduction obtained by this method.

In the nonfragmented case, the average time to the beginning of the one and only fragment is one-half the rotation time, or 8.3 milliseconds. With fragmentation, the average time will be slightly more than one-half the smallest fragment (due to the presence of the five-sector fragment), or 0.585 milliseconds. In both cases the transfer time will be the time to send 57 sectors, or about 16.3 milliseconds. The total time to complete the request will therefore be 24.6 milliseconds without fragmentation, but only 16.9 milliseconds with fragmentation, a reduction of over 30 percent.

Request Merging

The benefits of fragmentation increase in direct proportion to the number of fragments contained in a DRAT. The DRAT structure contains no host-specific information but is concerned only with transferring data from disk to buffers within the HSC controller. The MSCP server can combine into one DRAT the transfer requests from different hosts for the same group on the disk. Increasing the number of fragments will allow the previously mentioned optimizations to come into play. As the individual fragment requests complete and as the buffers are sent to the K.ci by the K.sdi, the K.ci will obtain the host information associated with each specific fragment and send the data to the correct host.

Of course, there is no guarantee that a specific application running on a host will send multiple I/O requests to the same disk area simultaneously. There is a high probability, however, that more than one outstanding request to the same area on an individual disk will exist.[2] A typical VAXcluster system contains many hosts, each of which runs numerous applications and system programs. Therefore, the probability is high that requests can be merged into one DRAT.

Work Queue Depth

As explained earlier, the K.sdi begins work only when a DRAT appears on its input work queue and places the DRAT back on an input queue of the MSCP server only when processing ends. The issue of when to queue a DRAT to the work queue of the K.sdi for processing constitutes yet another optimization opportunity for the HSC controller. At first glance, it appears best to construct a DRAT, fill it with fragments, and send it to the K.sdi whenever work requests arrive from the host. Further examination reveals that this procedure is not always optimum.

When the first host request arrives for an idle disk, the MSCP server within the P.io will construct the DRAT and send it to the K.sdi for action. The K.sdi will then notify the server of transfer completion by returning the DRAT. The time for the actual transfer consists of t_{acc} and t_{xfr}. Although that time will vary depending on the specific transfer parameters, it will typically be many milliseconds. If a second host request arrives for the same disk while the K.sdi is still processing the first DRAT, a second DRAT will be constructed and sent to the K.sdi. If a third request arrives while there are still two DRATs outstanding, however, it will be deferred by placing it in an internal list within the HSC controller. By deferring host requests when more than one DRAT is outstanding, the MSCP server may have the opportunity to merge multiple host requests into one DRAT. That action increases the benefits obtained through fragmentation.

When work has completed on the first DRAT, no delays will be introduced due to the K.sdi waiting for work since a second DRAT is immediately available for the K.sdi to process. The K.sdi will send the first DRAT to the P.io, signaling process completion. The MSCP server in the P.io will now construct a third DRAT from the requests placed in the deferred list and send this DRAT to the work queue of the K.sdi for action. Since the K.sdi is currently busy processing the second DRAT, the third DRAT will arrive in time to prevent the K.sdi from waiting for work, causing the cycle to repeat. In this case, however, there is a possibility that more than one host request has arrived for the same disk area. If these requests were placed in the same DRAT, the previously mentioned latency reductions will occur. If only one host request has arrived, no penalty will be incurred, although no benefits will be obtained by multiple host requests either.

Seek Ordering

Since the t_{acc} portion of a transfer usually exceeds the other time components by a considerable amount, the HSC design contains two algorithms that minimize the time spent moving the disk heads. The algorithms reorder the input request stream into a sequence that reduces the distance the heads must travel over the disk surface. Because more than one request must be outstanding (from one or more hosts) for this reordering to occur, the internal list of deferred requests is used to determine which one to service next.

As each DRAT is constructed and sent to the K.sdi for processing, the P.io will update an internal variable associated with each disk that indicates the current head position. (This variable shows the head position upon completion of the last DRAT, which does not necessarily correspond to the current head position.) When new host requests arrive and deferral is necessary, these requests are placed

in the internal holding list, which is maintained in ascending cylinder sequence.

When a DRAT completes, the first algorithm in the P.io will search the internal holding list for the request with the next higher cylinder number than that indicated by the current head-position variable. That search causes the disk head to move from low to high cylinder numbers. If the deferred list has no requests with higher numbers, the direction will be set to scan downwards and the next lower cylinder number will be used. Once the list has been exhausted in the downwards direction, the direction will be set to up and the process repeated.

This algorithm sequence is analogous to an elevator moving from the lowest floor where a button has been pushed to the highest floor where a button has been pushed, then back to the lowest floor, stopping at each floor where a button has been pushed. Each disk cylinder is guaranteed to be serviced by proceeding in this fashion. Moreover, requests continually arriving for a specific cylinder will not lock out other, more distant cylinders, since the scan immediately moves on to the next cylinder once one has been serviced.

As the number of outstanding requests mounts, this algorithm may add latency to some requests. Returning to the elevator analogy, consider a request for service made on the second floor of a ten-story building when the elevator is at the fifth floor and rising. It is reasonable to expect the elevator to stop at the sixth through tenth floors on the way up. If it also stops for requests on the ninth through third floors on the way down, however, the second-floor request will wait an extremely long time. Translating this elevator analogy to the actual disk case, the latency of an individual request (second floor) has been sacrificed to ensure a high request rate (ninth through third floors). Although clearly a goal of any high-performance controller, an increased request rate is of questionable value if latency is traded off to attain it.

For this reason, the HSC design invokes a second algorithm when the number of outstanding requests exceeds seven. In that case, the upward scan proceeds as before but will go directly to the lowest cylinder when there are no more higher cylinders. Using our analogy, the elevator would proceed directly to the lowest floor once the highest floor had been reached. Although this algorithm reduces the overall request rate somewhat, it reduces the maximum latency figures even more. Thus a reasonable balance is achieved between request rate and latency.

More specifically, a request arriving while the first algorithm is in effect may have to wait for $2c$ cylinders of time, in which c is the total number of

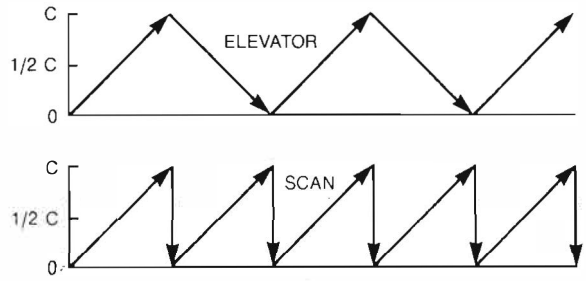


Figure 4 Seek Reordering Algorithms

cylinders. With the second algorithm in effect, however, the maximum waiting time decreases to c, a potential reduction of 50 percent in individual request latency. This second algorithm is known as the scan algorithm because it effectively scans the disk surface from low to high cylinders, then drops back to low again and repeats the scan.

These two algorithms are illustrated in Figure 4. The elevator algorithm smoothly transitions from low to high cylinder numbers, then smoothly transitions from high back to low. The scan algorithm, on the other hand, smoothly transitions from low to high, then abruptly moves to the lowest cylinder and begins its upwards scan again.

The increases in the throughput rate for requests are depicted in Figure 5. This curve was generated by a host program that made single-block requests distributed randomly over the entire disk surface. The vertical axis represents the number of requests per second being processed, which is limited by the mechanical time to physically move the disk heads. The horizontal axis represents the number of requests that the host has outstanding at any one time.

As shown, an increase in request rate occurs when the number of requests outstanding from the host increases from one to two. This improvement is due to the fact that the MSCP server will place the second request on the K.sdi work queue, allowing the K.sdi to begin work immediately on the second request after the first request completes. No improvement is seen for three requests, but one is seen when four requests are outstanding. When three requests are outstanding, one request (the first DRAT) is being serviced by the K.sdi, one (the second DRAT) is waiting for service, and only one is maintained in the internal holding list. When a fourth request arrives, it is placed in the internal holding area with the third request, and the P.io may choose which of these two internally held requests to issue next, allowing optimization to occur.

The request rate drops slightly when between nine and ten requests are outstanding. At nine, one request is being serviced by the K.sdi, one is in the

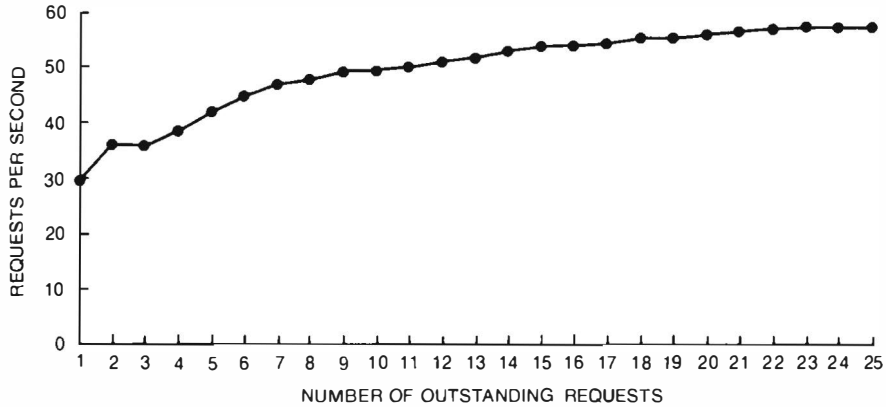


Figure 5 Effect of Seek Reordering on Request Rate

K.sdi work queue, and seven remain in the internal holding list. When ten requests are outstanding, the number of requests in the internal holding list exceeds seven. In that case, the MSCP server will switch from the first algorithm to the second (from the elevator algorithm to the scan algorithm). As shown, a slight drop in the overall request rate occurs and the remainder of the curve shows a slower rise since request latency is now given preference over request throughput.

Shadowing

Although primarily designed for high data reliability and availability, shadowing also offers an additional means to enhance I/O performance. Very little can be done to reduce latency for write requests to a shadow set since the data must be written to all members. The HSC design does attempt to minimize this delay by writing the data in parallel to all members rather than writing it to each member sequentially. Here, the time required for writing to a shadow set is only slightly longer than writing to an individual volume. Moreover, that time is far less than the amount required if the individual members of a shadow set were written serially.

Since a shadow set replicates data across multiple spindles, the real potential for performance improvement comes when read requests are directed to the shadow set. Many different techniques may be utilized to access data in a parallel fashion; however, the t_{acc} component tends to dominate the disk-latency portion of the transfer. Using the same reasoning that advocates seek reordering, we can conclude that any mechanism that reduces disk access time will have an impact on disk performance.

The shadowing process in an HSC controller takes advantage of the data replication by sending an incoming read request to what is thought to be the least busy drive. Several decision-making steps are required that function as follows:

1. If any drive is currently idle, send the request to that drive.
2. If there is only one drive with only one request outstanding to the K.sdi work queue, send the request to that drive.
3. If one drive has a shorter deferred-request list than the other drive(s), send the request to that drive.
4. If one drive is closer to the target cylinder, send the request to that drive.
5. If none of the above conditions exist, make a random choice of which drive to send the request to.

Following these steps, incoming requests will tend to be sent to the drive with the shortest amount of work pending. Since t_{acc} usually dominates the time required for each work request, distributing the work across multiple drives will reduce the latency of the individual requests by a large amount.

Summary

The design of a high-performance controller involves many complex issues, a large number of which impact the various performance metrics. The designers must address these issues when the system design is initiated since the largest performance improvements are seen with architectural and algorithmic approaches. It is extremely difficult to change either approach once the product design has solidified.

The HSC controller utilizes both approaches to improve performance, as well as the more traditional approach of simply reducing the amount of code that exists in the critical path. In addition to designing the HSC product to achieve high request and data rates, the reduction of request latency has always been a major portion of the design. The result of this design, begun over a decade ago, remains the highest performing I/O engine in Digital's current product offering, and probably in the industry.

References

1. R. Lary and R. Bean, "The Hierarchical Storage Controller, A Tightly Coupled Microprocessor as Storage Server," *Digital Technical Journal* (February 1989, this issue): 8–24.

2. K. Bates, "I/O Workload Characterization," Digital Internal Publication TR-8741 (June 1987).

Larry W. Emlich
Herman D. Polich

VAXsimPLUS, A Fault Manager Implementation

The VAXsimPLUS service tool is a fault manager developed for Digital Storage Architecture (DSA) fixed media disk drives. The VAXsimPLUS tool uses four functions — detect, diagnose, recover, and report — to manage system faults effectively. As part of its recovery capability, VAXsimPLUS provides Autocopy — the dynamic disk substitution of a drive in the DSA disk system. The VAXsimPLUS fault manager was built from four proven entities: the VAXsim monitoring tool, the SPEAR error log analysis tool, volume shadowing, and the VAX/VMS mail utility. The implementation of the VAXsimPLUS fault manager has resulted in the increased availability of a major system component.

The VAXsimPLUS service tool was developed to monitor and collect fault and event information on a Digital VAXcluster system, a VAX system, or a DECnet network (the detect function). Through an on-board knowledge-based system, the VAXsimPLUS tool uses fault and event information to predict a system or device fault (the diagnose function) and to initiate repair operations (the recover function). After the diagnosis of the fault, VAXsimPLUS informs designated individuals of the actions to be taken (the report function).

For the Digital Storage Architecture (DSA) disk system, VAXsimPLUS software provides not only monitoring and analysis, but dynamic substitution of a disk drive (Autocopy). The predictive capability of VAXsimPLUS allows time for Autocopy to take place before data availability is lost or downtime occurs. The Autocopy process takes place on-line and is transparent to the system manager of the suspect drive.

For the initial release, the knowledge-based system would be designed for use with only DSA fixed-media disk drives. We would thus first address the requirements for a major system component and a popular and widely used device. Once the tool is established, VAXsimPLUS designers will broaden the product so that it can be used with other system components.

Fault Management

The VAXsimPLUS tool acts as an automatic fault manager. It ensures that the fault does not impact the manager of the system and that the fault is efficiently removed. A prerequisite of fault management is fault tolerance. For if a system cannot tolerate a fault, then downtime occurs immediately and there is nothing to manage. Digital's DSA disk subsystems provide a level of fault tolerance and reporting that supports the implementation of fault management.

The efficient removal of a fault requires diagnosis to find the failing replaceable (or adjustable) component. Then, an impact evaluation is necessary to determine when to make the repair. Finally, a notification mechanism is needed to control how and when an external repair agent becomes involved.

This procedure comprises the following four functions:

1. Detect. A fault is detected by noticing a difference between expected and actual behavior. When dealing with noise-prone media, this means detecting an above-normal error rate.
2. Diagnose. The symptoms and related information are examined to locate the fault.
3. Recover. Based on the diagnosis, the system is recovered from the fault. Recovery may be automatic or manual. It may be temporary (a workaround) or permanent (repair).
4. Report. When manual recovery is required, all the necessary information is included in an English-language message and the appropriate people are notified.

When we started the project, we had access to the following software entities:

1. VAXsim Service Tool. This product monitors a VAXcluster system. It classifies error events and applies thresholds against each class. If a thresh-

old is exceeded, a notice is sent by electronic mail to the system manager.

The VAXsim tool also presents system managers with a pictorial display of their systems. Summary fault information is included to assist the system manager in making decisions concerning system integrity and storage management.

2. SPEAR Service Tool. This product analyzes error logs. It extends beyond the simple VAXsim classification to correlate multiple events and identify failing components.

3. Controller-based Shadowing. This feature of a hierarchical storage controller (HSC) enables the system manager to keep multiple copies of a disk structure. A disk write operation, for example, would be sent to each drive in the appropriate shadow set. A disk read would receive data from one of the drives in the set. If a drive fails, there is no data loss or loss of availability.

4. VAX/VMS Mail Utility. The mail facility of the VMS operating system allows the system manager to send electronic mail to any person on the network.

After some preliminary investigation, we decided to build our fault manager from these four entities, rather than start from scratch. We could use the VAXsim tool for our detect function, components of SPEAR for the diagnose function, shadowing for recover, and mail for report. The use of these products would greatly reduce the amount of work necessary to produce an automatic fault manager and would make the tool available much sooner.

We were not able to use the SPEAR product as it existed, but did utilize some of the generalized code. Prior to the development of the VAXsimPLUS product, SPEAR was incapable of analyzing error correction code (ECC) and standard disk interconnect (SDI) events. Thus, we wrote the analysis code so that it could be incorporated into both the SPEAR and VAXsimPLUS tools.

In the balance of this paper, we describe how we implemented the detect, diagnose, recover, and report functions in the first release of VAXsimPLUS software.

Detect Function

The VAXsimPLUS monitor accumulates new error and event data and records that information into the VAXsimPLUS monitor database. In a cluster environment, this database is shared by all host nodes.

The VAXsimPLUS monitor accumulates this information by attaching itself to the VAX/VMS ERRFMT process using a "mailbox" interface. A mailbox is a software data structure that is treated as a record-oriented device for general interprocess communication. (Communication using a mailbox is similar to other forms of device-independent I/O. Senders write to a mailbox; receivers read from that mailbox.) The monitor processes the error log data as it passes through ERRFMT. It immediately updates its database with information that includes the device type, device name, error code, and the time when the error occurred. The monitor also examines the contents of the ERROR FLAGS field of the error packet to determine if the error was recovered or not. That information is used to classify the error.

Classification of Errors

The VAXsimPLUS monitor classifies each error according to the device and error type with similar error types grouped together. The four error classes, media, soft, hard, and informational, are described as follows.

- The media error class is used to hold head-disk assembly (HDA) related errors. Examples of errors in this class are ECC errors, positioner errors, and drive-detected servo errors.
- The soft error class holds non-HDA-related errors that are recoverable through retries. Examples of errors in this class include SDI Command Timeout, Lost Receiver Ready, and drive-detected communication errors.
- The hard error class holds the same errors as the soft error class except that recovery was indicated as unsuccessful by the ERROR FLAGS field of the error log packet.
- The informational error class holds informational events. An example of an event in this error class is MSCP COMMANDABORTED.

In VAXsim software, only ECC errors are included in the media error class. In VAXsimPLUS software, we group all HDA-related errors into the media error class. Some HDA-related errors can also result from a logic failure, and we can therefore classify them either as soft or hard. For example, an RA81 Write-and-Off-Track servo error could be caused by either bad embedded servo data on the media or a servo module failure to which the servo data is fed. By grouping all HDA-related errors into the same error class, we allow all media errors to be counted together. Thus media failures can be predicted earlier because the fault manager is triggered sooner than if different media errors were counted separately.

Error Rate Thresholding

Each time an error is added to the VAXsimPLUS monitor database, the error rate is recalculated to determine if the fault manager process should be

activated. To describe how this works, we must first define the following terminology.

The *historical average error rate* is the average number of errors for a device over the previous 25 days. This average is calculated separately for each error class. Abnormally high bursts of errors caused by either a transient error occurrence or an earlier failure, which was subsequently repaired, are filtered from this number through a process called clipping.

The *current evaluation period* is the count of errors during the previous 24 hours.

The *margin* is a number assigned to each device type per error class to allow for normal fluctuations in the error count. The historical error rate and the margin are added to determine the threshold.

Triggering the Fault Manager

The fault manager is initiated when an error class enters into one of the two error states: warning or alarm. As shown below, by adjusting the margin value, the VAXsimPLUS monitor is more sensitive to errors in the hard error class, whereas it is less sensitive to errors in the information error class.

The following conditions cause an error class to enter a warning or alarm error state.

- Warning error state, media and soft error classes – The count of errors in the current evaluation period equals the historical average error rate plus the margin.
- Warning error state, hard error class – The count of errors in the current evaluation period equals the historical average error rate plus one half the margin.
- Warning error state, informational error class – The count of errors for the current evaluation period equals the historical average error rate plus twice the margin.
- Alarm error state, hard error class – The count of errors for the current evaluation period equals the historical average error rate plus the margin.

When a device is in an error state, triggering of the fault manager can reoccur if the number of errors exceeds twice the error count at the previous trigger. For example, if the number of errors in the soft error class for a device is 8 when the warning error state is entered, triggering will next occur when the count reaches 17 or greater.

The default margin used in VAXsim software for all devices was 15. For VAXsimPLUS, we lowered this margin for the DSA fixed media disk drives to allow faster triggering of the fault manager. Some of the thresholds used by the fault manager for recognizing a failure mode are lower than 15; therefore, having a lower threshold was also necessary to ensure that triggering occurs before the errors become frequent enough to warrant initiating recovery and repair actions.

Diagnose Function

Collect

The diagnose function needs access to all available symptoms for effective fault isolation. At the beginning of the project, we were not certain of the best way to obtain this information.

The VMS operating system logs all errors that happen anywhere in the system. Logging involves sending each event message to a system error log file and also to a mailbox, which is opened by the VAXsimPLUS monitor process. (Mailbox is explained earlier in this paper in the section Detect Function.)

In a VAXcluster system, every cluster node has its own VAXsimPLUS monitor mailbox and its own separate error log. This method works well for internal errors; but for shared controllers (HSC50 and HSC70), it means that error reports for a shared device are kept in multiple files, complicating the event correlation piece of fault isolation.

Errors that happen during a system-manager-invoked operation are reported to the node which initiated the operation. Other errors — those detected by self-test — are broadcast to all cluster nodes. In the former case, only one error log contains the symptoms. In the latter case, all error logs receive a copy of the report. VAXsimPLUS software is then faced with the problem of trying to merge this data without keeping duplicate information.

Originally, VAXsim kept all essential data in a single cluster-wide database. This data was sufficient for a gross "alert" (tell someone that something may be wrong) but not enough for a complete diagnosis.

For the new diagnose function, we needed a common database; but we also required the additional information that VAXsim had discarded. Furthermore, we needed to be able to recognize and eliminate duplicate records from HSC controllers.

One alternative was to make the VAXsimPLUS monitor process create a merged, complete error log. This process takes more time at high priority and uses more of the system's disk space to hold the additional (and redundant) file.

Another alternative was to make the VAXsimPLUS monitor process retain more information in its database so that a complete diagnosis could be performed. This approach seemed optimal, but it demanded major changes to the code and might have weakened system performance.

We chose to make the diagnose function use system error log files for input. We could then schedule the process to run as a low-priority, background job. Neither the VAXsimPLUS monitor nor the diagnose code would be drastically changed. The only problem was that we needed to merge the error logs to get all the information about shared disk drives.

To isolate this task, we wrote a separate program to allow the system manager to specify a list of input files and a single output file. The program opened all the selected input files, merged them chronologically, and wrote nonduplicate records to the selected output file, which then became the input file to the major diagnose process. To keep irrelevant data out of the diagnosis, this process limited collection to the most recent seven days or the time span since the last repair, whichever was more current. We chose seven because we needed something greater than one for multiple event correlation, but we also did not want obsolete or irrelevant data. The thought was that if we were unable to diagnose a problem with seven days worth of data, then more data would be of no help.

Since duplicates could only be caused by an HSC controller that sent messages for errors detected outside the context of a command, we could identify potential duplicates through the absence of a command reference number. Now that a potential duplicate was identified, we needed a way to determine whether or not we already knew about the event.

We decided on the following procedure. The first record identified as a potential duplicate would mark the logging node as the "master" for that disk drive, based on our theory that this node would continue to receive these records. Any potential duplicate that came from another node would be ignored. Note that comparing all the record fields is time-consuming and not necessarily accurate.

We needed one modification to the strategy. The selected master for a disk drive might become unavailable for a period of time and would therefore not receive error messages from the controller. To offset this, we added a timer. Instead of simply rejecting any record from another node, the time elapsed between the new record and the previous record from that device was determined. If the time was more than five minutes, the new reporting node became the master for the disk drive in question.

Many tests have confirmed the validity of our strategy. The first node to report an error is usually the fastest or least busy and continues to be the first node to report until it becomes unavailable. Since the error report is broadcast to all nodes, there should be no great difference in the time of the report. Even if the node times are badly skewed, the risk of losing information is low. The difference between counting 30 errors rather than 29 errors does not usually change a diagnosis.

We used a similar strategy to prevent the VAXsimPLUS monitor from counting duplicates in its database.

We changed the monitor code to use the VAXcluster lock manager in the following way. Whenever a potential duplicate appeared, the monitor process would attempt to get the "token" for the drive. If this token was available, the monitor logged the event in the cluster-wide database and kept the token, becoming the master for that drive. If the token was unavailable, then the monitor would know that another monitor held the token and would simply ignore the event.

If the master for a drive became unavailable, it would release the token. The next time any monitor process asked for the token, it would be granted; and the associated drive would receive a new master logger.

Of course, a cluster-wide error logging system would solve all these duplication problems; but the cost of such a solution is often prohibitive. It is best to keep high-priority jobs as simple as possible and leave any complexity to background tasks.

Analyze

We examined the rules that had been developed for Digital's MASSBUS drives and decided to revise them for use with the RA-series drives. Since these rules were written for removable media drives, we made extensive changes to make them effective with the DSA RA-series drives. The RA drives analyzed in VAXsimPLUS V1.0 are primarily HDA-based and require a troubleshooting approach different from the one for removable media drives. RA drives are connected by a serial interface versus a parallel interface for MASSBUS drives. They also contain a great many more internally detected error codes and diagnostics than their MASSBUS predecessors.

Rules After listing all possible failure modes, we began writing the rules. The items we considered most important were the following:

- Which errors can be symptoms of each of the failure modes?
- In what pattern must these errors occur to indicate a failure mode, i.e., should the errors be random or occur all on the same head?
- Can multiple errors be indicated as a result of one failure? For example, a drive-detected Write-and-Off-Track error might also result in a controller-detected Lost-Read/Write-Ready error.

- What is the minimum number of errors needed to have confidence that the correct failure mode is being indicated?

We designed the tests to isolate a failure to the bad field replaceable unit (FRU). We further designed the tests to indicate what is wrong with that FRU by assigning a theory number to each failure. The Digital Customer Support Centers use the theory number to recommend the proper troubleshooting and repair actions. The repair centers also use the theory number to assist them in determining the failing components for intermittent failures.

The development team recognized that sometimes devices fail too quickly for analysis to complete recovery actions before the device fails solid or data loss occurs. We also realized that for some errors the partitioning is not adequate to provide isolation to a single FRU. However, we felt these problems could be overcome a majority of the time. Field test showed our goals in both cases were met or exceeded.

We developed tests to check for the following failure modes.

- SDI Path Failure – This test checks first for hardware- and then firmware-detected errors over the SDI communications path from the controller to the drive.
- Drive-Detected Error Test – This test analyzes all non-media-related drive-detected errors.
- Head Matrix Failures – This test checks for a bad head matrix chip within the preamp module. Each head matrix chip is associated with either three or four read/write heads.
- Bad Surface – This test checks for most errors occurring on heads associated with one media surface.
- Bad Heads – This test searches for head failures.
- Scratches – This test inspects for radial and circumferential scratches on both the data and servo surfaces.
- Servo Logic/Head Failure – This test searches for random servo-related failures.
- Bad Read Path – This test checks for failures in the read path external to the HDA.
- Forced Error – In this test the error pattern is such that a failure mode cannot be detected, but at least one disk data block was revectored to a replacement block with the forced error flag. The forced error flag indicates that data loss may have occurred.

Order of Failure Modes The analysis control process runs through the tests until a test fails, and then analysis ceases. The order in which the analysis tests are run is important since multiple tests can fail on the same error pattern. For example, the head matrix test checks for errors on heads associated with each head matrix circuit. If this test fails, the failing head matrix circuit is indicated. One of the later tests checks for bad heads. If the bad head test was run prior to the head matrix test, multiple heads might be identified as bad and the incorrect FRU replaced. Therefore the tests are run in a sequence that prevents the reporting of erroneous device failures.

Thresholds Each failure mode uses a threshold to determine when to notify a system manager, Field Service, and Autocopy that a device is beginning to fail. This threshold is not necessarily the same as the number of errors needed to have confidence in the accuracy of the failure mode. Our two main concerns in selecting these thresholds were as follows:

1. If the threshold is too high, notification may not occur early enough, resulting in lost data and downtime.
2. If the threshold is too low, a transient error situation might result in unnecessary service calls.

Failure modes that can result in data loss were given two thresholds for notification: a higher threshold was assigned when all errors are recoverable, and a lower threshold was specified when at least one of the errors results in possible data loss. In this way we prevented notification from occurring too soon for transient error situations that may be corrected by automatic error recovery mechanisms such as Bad Block Replacement.

Recover Function

VAXsimPLUS software does not do error recovery. However, the software can often provide a temporary repair (work-around) by automatically inserting a spare disk drive when diagnosis determines that another drive may be failing.

This recovery method uses the VMS/HSC shadowing mechanism, the original intent of which was to allow the system manager to maintain the same data on multiple drives. If one drive failed, the data could be found on another drive in the set.

All drives assigned to the same data set are grouped together into a "shadow set." Instead of referring to a drive name, the operating system refers to a shadow set name.

The system manager may mount a new drive into an existing shadow set at any time. When this is

done, all data from the shadow set is copied onto the new drive.

When data is written to the shadow set, it is sent to all drives in the set. When data is read from the shadow set, one drive is chosen (the one with heads closest to the target block).

If an error is detected during the read, the good data can be retrieved from another drive in the shadow set. If the block in error is "replaced," this good data is moved to the replacement block as well.

Shadowing requires at least twice the storage space to ensure redundancy. The VAXsimPLUS recovery mechanism (Autocopy) offers a low-cost alternative by allocating one spare drive for many system drives. Shadowing is started only when the diagnose function of VAXsimPLUS determines that the drive is about to fail.

When we considered the management of spares, our first approach was to create a pool of spare drives that would stay idle until needed. We would make them members of a special shadow set so they could not be used for anything else. When we needed a spare, we would simply remove it from the spare pool by dismounting it from the special shadow set.

We found two problems with this approach:

1. If the spare drive remains idle for a long period of time, it may develop problems of its own. We would need to periodically test the drives.

2. The spare drives would be idle and nonproductive unless and until another drive failed.

These problems led us to another solution: instead of keeping spares in a separate pool, we would add them to "default" shadow sets. This solution allows the spares to be used for host I/O while waiting for a failure.

Following is the system set-up for VAXsimPLUS recovery:

1. All candidates for recovery are mounted as single-drive shadow sets. In other words, the drive is in a shadow set all by itself. There will be no protection until a spare drive is mounted into the same shadow set.

2. Spare drives are mounted into the shadow set of one of the candidates for recovery.

After set-up, each disk drive is in one of the following categories:

1. Unprotected. The drive is not in a shadow set at all. Some volumes contain read-only data and programs that can be easily restored from backup tapes or other media.

2. Potentially protected. The drive is in a single-member shadow set. If VAXsimPLUS diagnosis detects an impending failure, it attempts to mount a spare drive into the shadow set. If the failure is sudden or there are no available spare drives, this drive would be unprotected. Optionally, a drive can be designated potentially protected if the data on it is frequently backed up or seldom modified.

3. Potentially unprotected. The drive is one member of a two-member shadow set, but the other member is also a spare drive which may be moved to a failing drive at any time (unless this drive fails first). A drive that is read often for performance reasons or one that merits extra protection from sudden catastrophic failure can be placed in this category.

4. Fully protected. The drive is one member of a permanent multiple-member shadow set. This set-up is recommended for maximum protection of data but at the cost of using redundant drives.

5. Spare. This drive is always a member of a two-member shadow set. When there are no failing drives, the spare is attached to a potentially unprotected drive. If the VAXsimPLUS recovery process chooses this drive to support a failing drive, it is dismounted and remounted into the failing drive's shadow set if the failing drive is in the potentially protected category. The spare remains in this shadow set until the failing drive is repaired. It then returns to its default position as a member of a potentially unprotected shadow set.

The location of spare drives must be known to facilitate system reloads. The VAXsimPLUS database keeps track of spares and candidates for recovery, and a new command has replaced the VMS MOUNT command for spare drives. The ASSIGN DRIVES option of the VAXsimPLUS command is executed within the system's startup file so that spares are mounted into the proper shadow sets on a system reboot.

When the diagnose function of VAXsimPLUS determines that a drive is beginning to have problems, it tells the recover function to locate a spare (if the failure mode calls for this kind of recovery).

The recover function looks in its database for a spare drive of the same type that is not supporting another faulty drive. The spare drive must be physically attached to the same HSC controller as the failing drive because the shadowing feature is provided by the controller.

If a free spare is found, it is taken out of the default shadow set and mounted into the failing

drive's shadow set. This action invokes an automatic copy operation of the data from the failing drive to the spare. The spare then continues to shadow the failing disk.

Now we were faced with the question of what to do with the failing drive after the spare has become a member of the shadow set. Should it be dismounted and left down until the fault is removed? We decided to allow the system manager to resolve this problem. The report function (explained in the next section) would tell the system manager about the fault and about the use of the spare. If the system manager so desires, he or she may dismount the failing drive with another VAXsimPLUS command. This special dismount marks the drive in the database so that it does not remount during a system startup.

We could have provided an automatic dismount of the failing drive, but we could not be sure this would always be the best solution.

In a cluster, nodes may be coming up and going down all the time. All nodes must therefore know where each spare should be mounted and which drives have been dismounted because of failures (so they are not remounted). Each node must also know which drives are candidates for recovery.

All of this information is kept in a database which is accessible from all nodes in the cluster. The VAXcluster lock manager prevents nodes from accessing the database before an operation is complete. Assigning a spare, for example, may take several commands. Another node, which may be restarting, must not make initial assignments until this spare assignment operation is complete. For this reason, every node must acquire the single recover resource before it assigns candidates or spares.

When a failing drive has been repaired, the system manager mounts it and observes its behavior for a period of time before releasing the spare. If the repair seems sound, the system manager manually removes the spare with another VAXsimPLUS command, allowing it to return to its default position.

Whenever a spare is placed into the shadow set of a failing drive or back to its default drive, it must receive a copy of the other drive's data. This operation may take several minutes. While it is copying data, the spare cannot be made to do anything else. If another drive begins to fail while the spare is returning to its default position, the recovery must wait (if this is the only available spare) until the copy is finished.

This recovery mechanism is not without drawbacks. It simply fills in the gap between complete redundancy and manual backup procedures. The mechanism cannot be used for all failures. It will not help when the failure does not give sufficient warning. And it demands that drives are fixed quickly enough to ensure that a spare is always available.

We believe, however, that the mechanism is especially useful in unattended environments where the criticality of data permits something less than total redundancy.

Report Function

After the diagnose function has determined that a drive is failing and the recover function has attempted to use a spare (when applicable), the report function notifies the system manager. The system manager can then schedule a permanent repair.

It is important to avoid "over-reporting." Once a drive begins to fail, it may trigger the monitor threshold several times. While automatic rediagnosis is recommended occasionally, it is not always necessary to repeat notification. In fact, if the diagnosis does not change, there will be no further reports until 24 hours after the initial report. The reasoning here is that Field Service is probably on the way to fix the problem. If the thresholds are still being exceeded after a full day, a reminder may need to be sent.

If a spare drive has not been assigned to the failing drive, there are other cases when a secondary notification may be sent.

If a new diagnosis shows that a previously recoverable fault is now causing data loss, another report is sent, regardless of when the initial report was sent. The worsening condition warrants a reminder.

Also, a new notification is sent if a new diagnosis results from a change of information from the failure mode. For example, an initial diagnosis might indicate a bad head. As the disk continues to be used, it becomes apparent that the bad head was really head-to-disk interference and the HDA has become contaminated resulting in a different failure syndrome.

These secondary notifications are not sent if the recover function has assigned a spare to the failing drive. If the drive is enjoying temporary redundancy, the chances are better of surviving until the Field Service engineer arrives to make a permanent repair. Reminders should only be necessary when there is no redundancy.

All notices are sent by the mail facility of VMS to one of three mailing lists:

1. Customer list. These notices alert the system manager to the fault and state whether or not automatic recovery was started. They also include information to use when calling Field Service.

2. Field Service list. These notices give Field Service personnel detailed information about the fault.

3. Monitor list. These notices concern devices that are not fully supported by VAXsimPLUS software.

When calling Field Service, the system manager must report the event code that is part of the mailed message. Field Service personnel can then feed this code into an information base and retrieve a complete repair plan for the fault. This plan includes the suspect parts and any other information the repair agent may need.

The repair agent may also read the VAXsimPLUS mail message sent to the Field Service mailing list to obtain the facts about the event code.

We established a monitor list to send alerts for devices that were not yet fully supported by VAXsimPLUS. In other words, we needed to carry on VAXsim support without the "PLUS" for all devices.

A message is sent to the monitor list when a nondisk device reports a large number of errors. In these cases, there is little or no diagnosis. The message is just a warning that some unexpected activity is taking place. When VAXsimPLUS software is able to diagnose everything on the system, this mailing list may be eliminated.

Members of a mailing list may be anywhere in the network. For this reason, VAXsimPLUS makes several attempts to send the mail (in case the addressee's node is temporarily unavailable).

The report function is the last stage of automatic fault management before a person is involved. Once the fault has been placed in human hands, VAXsimPLUS must wait for the service engineer to make the repair, log it, and release the spare (if one was in use).

If the repair attempt fails to solve the problem, VAXsimPLUS software may not give the same diagnosis because no information prior to the last repair can be factored into subsequent diagnoses.

Previous data (prior to the repair attempt) is not used in analysis because its inclusion would mix data from two unrelated faults in the analysis. The problem with this approach occurs when a repair attempt is unsuccessful and data that is related to the fault is rejected. However, we chose this approach because we were faced with the following choices:

1. Disregard repairs. This proposal could result in bad multiple-event correlation if a new fault were inserted or developed after a good repair.

2. Factor the repair into subsequent diagnoses. This solution is ideal, but we were unable to develop such a mechanism in a reasonable time-frame.

3. Assume the repair was successful and count nothing prior to it in any diagnosis. This method is easy and reduces the chance of bad diagnosis from multiple faults.

In a future release, we hope to move to option 2. Until then, we risk seeing incomplete diagnoses when an incorrect repair is made.

Summary

The VAXsimPLUS software tool was developed as a system fault manager to provide predictive capabilities for all system elements and an increase in system availability. Such an approach requires cooperation from the device level, through the subsystem level and the system level in fault coverage, error detection, diagnosis, and recovery. These layers then feed into the fault management system to allow fault isolation and repair. The outcome of this approach is the ability to recover from faults at the lowest level possible.

Acknowledgments

The people who contributed their expertise to the success of the VAXsimPLUS software are too numerous to list here. However, the authors would like to thank the following individuals for their development work on the VAXsimPLUS product: Debbie Hoeppner, Carl Grundstrom, Roger Sulzbach, Gary Lengyel, Robert Winant, Jim Nicholson, and Bruce Kelsey. We would like to particularly acknowledge the pioneering work done by Bruce Moore on the monitoring and display features. Additional thanks to Keith Norman and Dick Stevenson for their support in the writing of this paper.

Barbara A. Crane

Disk Drive Technology Improvements in the RA90

The RA90 product represents significant advances in the technology used in Digital's disk drives. To ensure high storage capacity, engineers balanced the interdependent factors of size and number of disks and heads, rotational speed, and bit and areal data densities. As a result, RA90 formatted capacity is 1,216 megabytes. Improvements in drive speed were made by reducing seek time to an average 18.5 milliseconds. To increase reliability, engineers established performance criteria for key subassemblies. They then designed components to meet these standards by reducing the effects of heat, contamination, and wear. The RA90 disk drive is also designed to be easily installed and to operate in international environments. An appendix to this paper presents some fundamental concepts of magnetic recording technology.

An industry-leadership product, the RA90 disk drive stores 1.216 gigabytes (GB) in its formatted space and occupies one-half of a 10½-inch by 19-inch rack slot. Access times average about 18 milliseconds (ms), and reliability of the drive is among the highest in the industry.

Requirements for a Storage Device

As computer systems become ever more widely used and applications more sophisticated and interrelated, the need for data storage devices increases. The ideal storage device would be inexpensive, fast, space-efficient, easy to use and install, gracefully integrated into the host system environment, and reliable relative to the current state-of-the-art. Of the three types of technology available today for storing data, none satisfies all of these goals to the same degree.

Semiconductor memory technology is our fastest storage and retrieval technology, but also the most expensive. Because of its speed, semiconductor memory is central to the operation of the CPU; we therefore think of it as "primary" storage. Tape and optical technologies are very cost-effective for large volumes of information, but relatively slow; they are usually considered "tertiary" storage.

Magnetic disk products are intermediate in speed and cost, and are available in relatively large capacity units; this technology constitutes "secondary" storage. Magnetic recording devices have been used for information storage in computer systems since the 1940s and are anticipated to continue in use for many years. The basic operating principles of these devices have remained constant over the years, although with tremendous improvements in implementation.

The RA90 team set out to make significant improvements in Digital's disk drives relative to the criteria for secondary storage products:

- High speed
- Large capacity
- Reliability and data integrity
- Simple system interface and integration
- Competitive cost

To accomplish these improvements, we went beyond the traditional product development method of design first, purchase parts second, and finally manufacture. We integrated the design process, supplier capabilities and expertise, and manufacturing requirements and processes to achieve a higher performance, better quality product. The engineering and manufacturing teams worked closely together to design the product both for performance and for manufacturability.

We frame the following discussions around four of the criteria for secondary storage listed above — capacity, speed, reliability, and interface and integration. In each section, we describe aspects of the development process and the design decisions made

by the RA90 team to meet the stated criteria. An appendix at the end of this paper presents an overview of the basic considerations in the technology and design of a magnetic disk drive.

Capacity

For any storage product, it is desirable to store as much information as possible in the smallest space. This section looks at the interdependent factors that determine effective space utilization and storage capacity. In general, the key product decisions are made relative to the size and number of the disks and heads, and the bit and data density.

Among the several factors that influence the selection of disk size and number is the space allotted for the drive itself. Digital's disk drives are built to fit into a standard rack slot 10½-inches high by 19-inches wide by 30-inches deep. One or more disk drives should use this space as efficiently as possible. If 14-inch diameter disks were selected, then a full slot would be used, as with the RA81 and RA82 drives. If an 8-inch or 9-inch diameter disk were selected, one-half of a slot would be used. If a 5¼-inch disk were selected, up to eight standard-format drives would fit into a full slot. Cost also has a bearing on this decision. As the number of drives increases, so does cost per MB, because positioner hardware and electrical systems must be duplicated for each drive.

Rotational speed and bit density also ultimately influence the choice of disk size. Together these factors affect transfer rate and consequently the disk capacity. The rotational speed of disks determines the rotational latency component of access time. Many drives today rotate in the range of 3600 rpm, resulting in an 8.33 ms average rotational latency (time required for one-half revolution). The transfer rate is determined by the velocity at which the disk moves past the head and by bit density along a track. For a given rotational speed and bit density, large diameter disks have a higher transfer rate than small diameter disks. The second factor, higher bit density, also results in faster transfer rates. Faster transfer rates are desirable, up to the limit for standard interconnects to the controller and CPU, such as Digital's Standard Disk Interconnect, or SDI.

To provide larger capacity and more cost-effective storage, we increased bit density. Moreover, we wanted to keep the rotational speed high to minimize rotational latency time. However, the combination of high rotational speed and increased bit density can increase the transfer rate beyond interconnect capabilities. By changing to smaller diameter disks, we ensured the drive transfer rate remained within the capabilities of existing interconnects and retained a relatively low rotational latency time of 8.33 ms. Later in this section we discuss the details of our decisions on areal density.

The RA90 drive was conceived as a half-rack-size disk drive in which two RA90 drives fit side by side in a standard rack. This decision determined the physical parameters. Within the confines of the half-rack space, we needed to house the head-disk assembly (HDA), electronic control module, power supply, cooling unit, and operator control panel.

We built cardboard prototypes to evaluate different configurations. Working within the half-rack size constraints, we considered disk size variations of approximately 8 or 9 inches (200 or 230 millimeters [mm]) outside diameter and 2½ or 4 inches (63.5 or 100 mm) inner diameter.[1] Disk sizes were evaluated to assess the available physical space; the size would allow for clamping at the inner edge and for surface irregularities and chamfer at the outer edge. The ratio of inner to outer recording radii helped to determine the variation of pulse width, amplitude level, and overwrite level which the head, disk, and electronics must tolerate. Arm and positioner configurations were sketched and modeled to assess seek distance and power requirements.

Based on the information these investigations provided, we chose disks with approximately 9.055 inches (230 mm) outer diameter and approximately 3.937 inches (100 mm) inner diameter, and arms mounted to a rotary positioner. Of the 2.5 inches of radial distance available, we use approximately 1.5-inches distance for recording data; the remaining 1.0 inch is used for clamping, landing zones, and flexure and wire routing. Inside the HDA we could fit seven disks and eight head-arms. See Figure 1.

As shown in Figure 2, the outer head-arms have only one recording head; the middle head-arms have two heads, which record and read on adjacent disks.

The HDA is mounted to a carrier. The HDA and its carrier are oriented so that disks are vertical in operation. The modules are mounted vertically, on a carrier, next to the HDA; a copper-plated divider between the HDA and modules minimizes extraneous signal transmission. The blower, power supply, and operator control panel are then mounted on the exterior of the chassis, as shown in Figure 3.

Given the size and number of components that can be shoehorned into a cabinet, the other variable affecting capacity is areal data density on the disks. Areal data density is the product of circumferential bit density and radial track density. We originally targeted the RA90 drive to provide at least 900MB of formatted capacity. However, because of anticipated competition from other companies, we raised

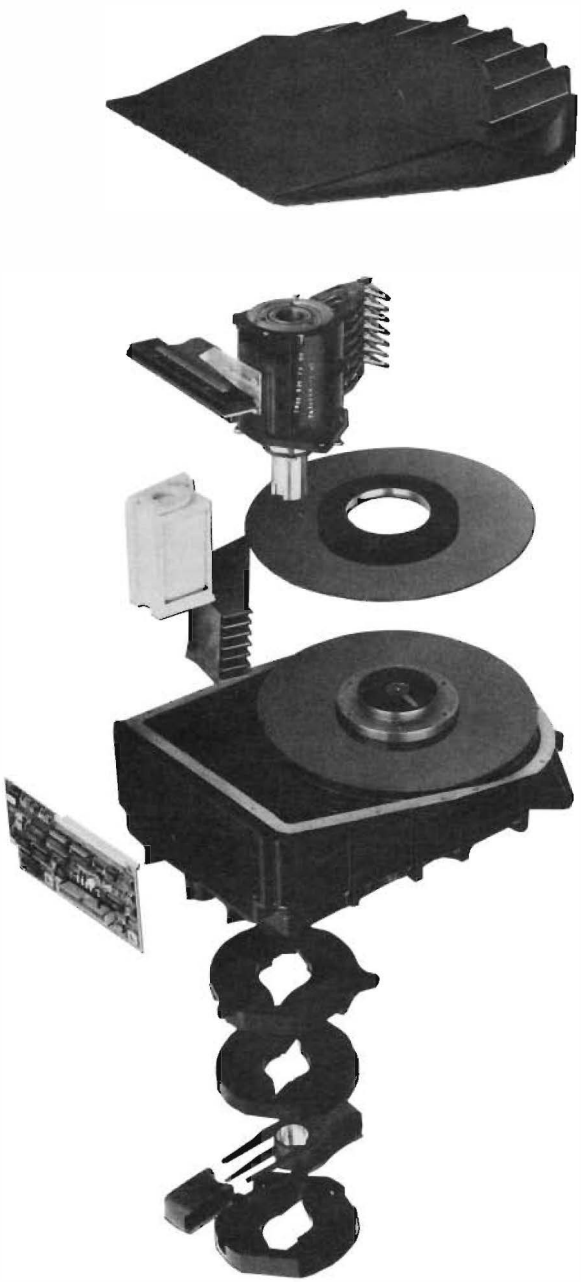


Figure 1 Exploded View of RA90 HDA Showing Orientation of Key Components

the capacity to 1,200MB, or 1.2GB. Because the physical rack size was unchanged, this change meant an increase in areal density from approximately 30 megabits (Mb) per square inch to about 40Mb per square inch.

Our advanced development group, using various combinations of bit and track density, had demonstrated areal densities up to and beyond 40Mb per square inch. We had to choose specific track and bit densities that would result in the desired net areal density.

Radial track density is limited by several factors:

- Repetitive and nonrepetitive runout of the spindle bearing
- Mode-frequency structural responses of the mechanical subsystem
- Signal level required by the read/write channel
- Control of the recording head tolerances

Based on demonstrations and prototypes, evaluation of signal amplitudes of prototype thin-film recording heads and disks, and assessment of position error, we believed we could achieve 1,750 tracks per inch in the drive.

Circumferential bit density is limited by

- Signal level and signal-to-noise ratio
- Precision of pulse location within the bit-cell timing window and intersymbol interference
- Transfer rate capabilities of the drive-controller-CPU interconnect

The signal level is determined primarily by the head design, disk and head materials, and flying height. The read/write channel electronics manipulate the signal pulses to improve bit-cell precision. Based on these constraints, we selected a data density of 22,839 bits per inch, for an overall recording density of approximately 40Mb per square inch.

Given this density, the number of disks, and Digital's standard disk format, the RA90 provides 1,216MB of formatted capacity. (Digital's disk format assures very high data integrity by using some of the raw capacity for multiple copies of headers, error-correction coding, etc.) In addition, Digital Storage Architecture (DSA) requires spare and diagnostic capacity. Unformatted capacity, for reference only, is 1,607MB. Achievement of these densities was dependent on thin-film head and disk technologies developed by teams in Massachusetts, Colorado, and Arizona.

Thin-film heads are manufactured by processes similar to those used to manufacture semiconductor chips; recording elements are masked and deposited on a thick wafer, or puck. The puck is then diced into individual units, each of which becomes a recording head. Thin-film head structures provide accuracy of track dimensions necessary for 1,750 tracks per inch, gap dimensions required for clear pulses at almost 23,000 bits per inch, and efficient coil and pole structures for signal amplitude. (See Figure 4.)

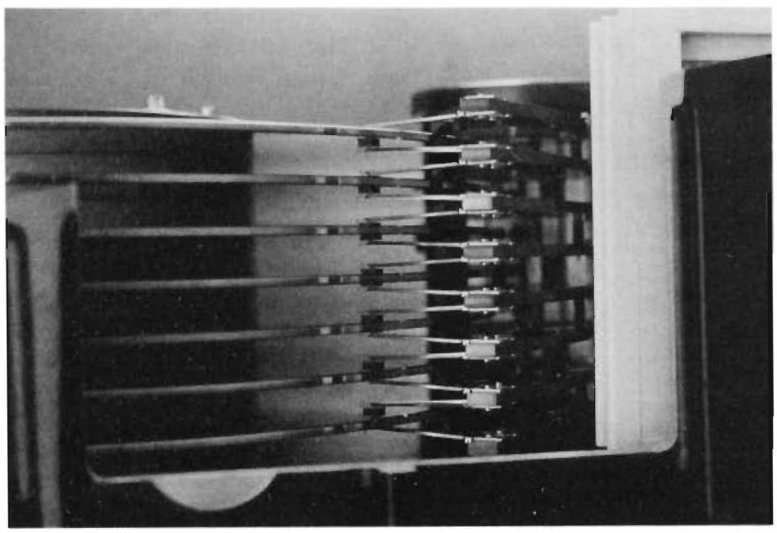


Figure 2 HDA Showing One Head on Each Outer Arm, Two Heads on Inner Arms Facing Opposing Disks

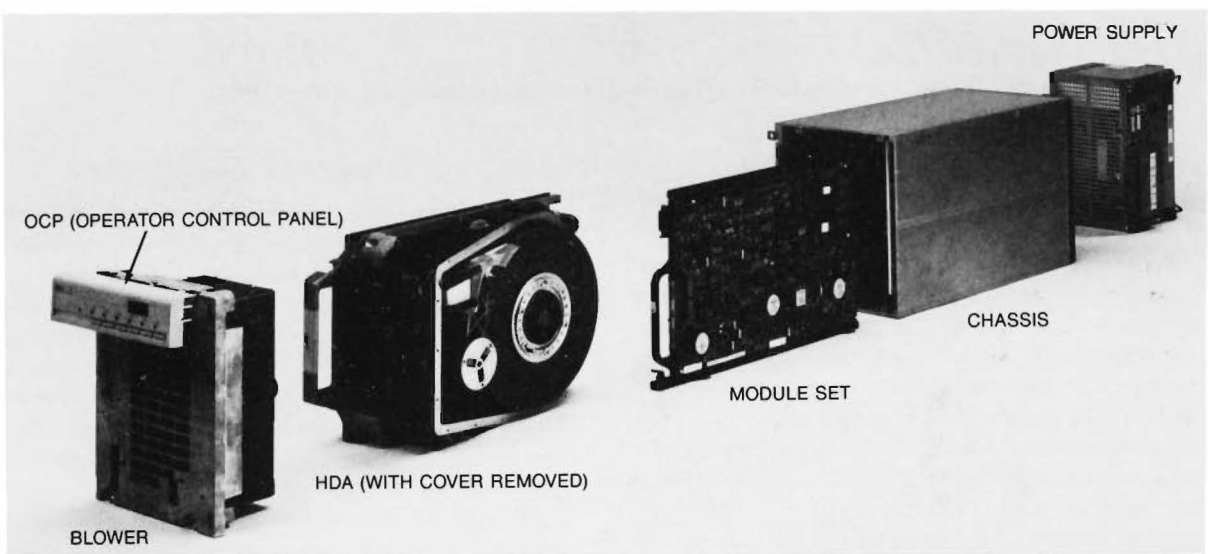


Figure 3 Expanded View of RA90 Drive, Showing Major Components

Thin-film disks are manufactured by plating or sputtering sublayers, a recording layer, and a protective layer on an aluminum substrate. These processes produce a wear-resistant disk which can be written and read back with good signal-to-noise ratios at this density. The disks have relatively few defects that require revectoring during initial format at the factory.

Speed

Based on our model of data request and response times in systems environments, we knew that a key parameter to improving system performance was access time of disk drives. Access time is composed of seek/settling time and rotational latency time. Rotational latency time, 8.33 ms, was determined by transfer rate, disk diameter, bit density and rotational

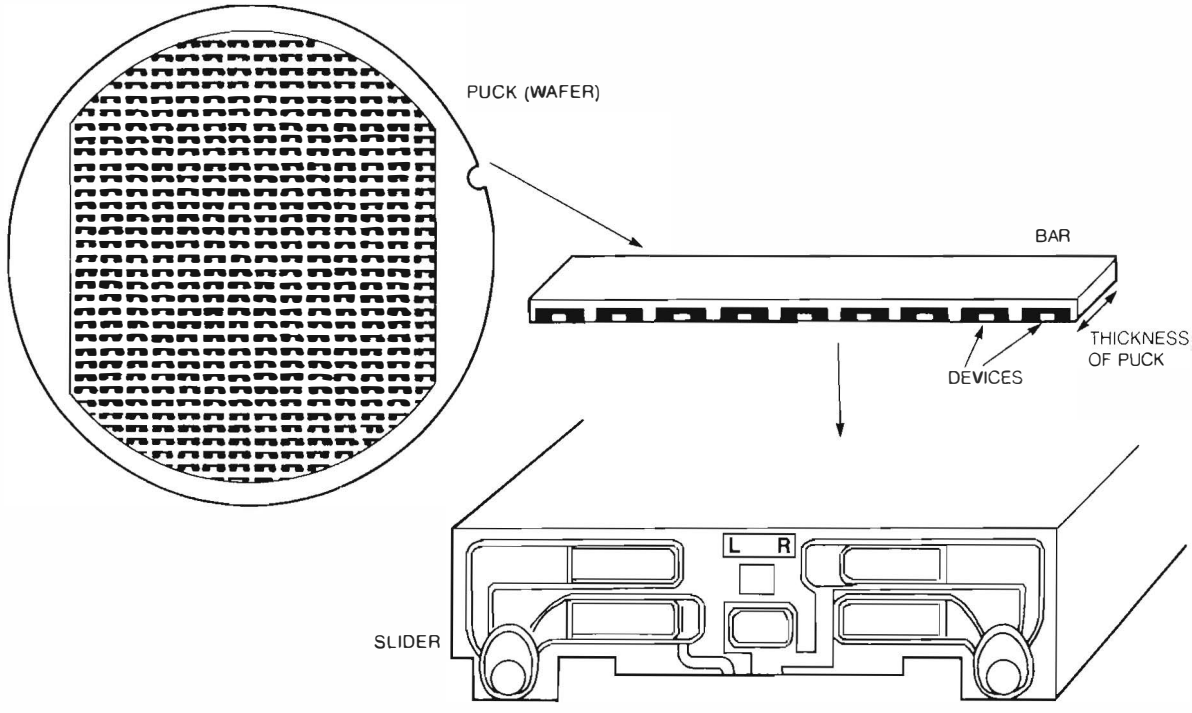


Figure 4 Summary of Thin-film Head Manufacturing Process Steps

velocity requirements, as described earlier in the section Capacity. Our target for average seek/settling time was about 18 ms.

Seek/settling time is the time required to move head-arm assemblies across disks and lock in the servo system to verify the on-track positioning. Average seek time is defined as the average of the times required for all possible track-to-track seek combinations. The average time is approximately equal to the time required for a seek across one-third of the total surface. Settling time is the time required to fine-tune and verify the head position prior to reading or writing. The actuator motor drives the positioner assembly at high acceleration and deceleration rates up to 22 gravitational units to achieve desired seek times.

To reduce seek time without impairing reliability, we must evaluate the design choices: increased power from actuators, reduced mass in the positioner, increased or decreased disk diameter, and multiple heads per arm or multiple positioners.

Increased power from actuators can generate higher acceleration rates, but also can generate more heat and structural vibrations. Reduced mass in the positioner and head-arm assemblies lowers the power requirements dramatically: power is proportional to mass raised to the fourth power.

$$\text{Power} = K \times \text{Mass} \times \text{Mass} \times \text{Mass} \times \text{Mass}$$

However, low mass positioners and head-arms may be less stiff and have lower frequency resonances which interfere with servo system settling.

Disk diameter affects seek time, because the diameter determines the maximum distance over which heads must travel and the arm length required to cover the disk. The smaller the diameter of the disk, the less distance the head must travel; accordingly, the arm length and inertial mass of head-arms can be reduced.

Multiple heads per arm or multiple positioners covering different zones on the disk are other approaches to this problem. However, both approaches significantly increase the cost of components required and the complexity of assembly without corresponding increases in capacity. In addition, multiple heads per arm increase the mass and cost of the arm. For these reasons, the optimal choice for the RA90 drive was one positioner per drive, one head per surface.

As noted above, the servo system is the second part of the total seek/settling time. The RA90 uses a combination of a dedicated servo surface and embedded servo information on each data surface to achieve accurate positioning at 1,750 tracks per inch. (This measurement is equivalent to 571 microinches, or 14.27 microns, from track centerline to centerline.)

A dedicated servo surface provides continuous feedback of the cylinder number to the servo head, allowing rapid seeking. Embedded servo data provides detail on exact track position relative to data heads, which may vary as a result of temperature, physical shifts, or vibration. For each recording head, embedded servo data determines offsets and positioner bias force over time for optimal performance.[2]

In summary, the RA90 achieves a 18.5 ms average seek time and 4.0 ms single-track seek time by using an efficient positioner structure, high-energy neodymium-iron-boron (Nd-Fe-B) actuator magnets, and the dedicated plus embedded servo strategy described here. Coupled with an average transfer rate of 2.8MB per second and optimizations provided by controllers in the DSA, the RA90 delivers both high throughput and responsiveness in a system environment.

Reliability

To achieve the reliability goals for the RA90 product, we relied on our experience with previous products, reliability modeling tools such as PREDIC (a Digital program based on MIL-Std-217-E), and test data for proposed components. With these, we portioned reliability "budgets" for each key subassembly, or field-replaceable unit (FRU). Continuing to lower level assemblies, we estimated a required reliability for every critical part or assembly.

By working through each assembly level, we developed a reliability budget for each component. Some components, such as gate arrays, heads, and disks, were required to perform for mean times of 10 million hours or more between failures (MTBFs). Once requirements were identified, we could work on controlling the parameters that influence reliability:

- Reduced early life failures
- Heat and wear resistance
- Cleanliness
- Interconnects
- Incoming quality

In addition to these parameters, we also briefly describe in this section the heads and media reliability test we performed prior to production.

Early Life Failures

Electronic components tend to fail very early in their useful life, or to fail at a very low rate thereafter. To remove this early failure mode from our product in the field, we implemented several forms of testing. Most electrical components in the RA90 disk drive are subjected to power, resistive load, or thermal cycling by the vendor that produces the parts. At the end of our manufacturing process, we run an "extended test" in which drives are operated in a system-like environment for many hours. The number of hours (currently 96) is set to ensure that drives function reliably and is based on current ongoing reliability test (ORT) data.

A limited number of drives are cycled through ORT testing. The ORT test cycle is a much more extensive test cycle than extended test, usually lasting longer than one month. This testing verifies the reliability of the product produced by our process and makes apparent any problems that might occur in the field.

Heat and Wear Resistance

Before placing a drive in service, it is critical to reduce any factors that could shorten the inherent design life of the drive's components. Designers must, therefore, anticipate the effects of heat, contamination, mechanical wear, and environmental degradation. The heart of the HDA, heads and disks, underwent particularly stringent design evaluation and testing to ensure adequate life.

Localized heating shortens the life of electrical and mechanical components. Heat provides activation energy for ion migration, oxidation reactions, and breakdown of lubrication layers. To address this problem, we took several approaches. First, we thermally stressed the components to reduce early life failures that might result from manufacturing process problems. Second, we did extended reliability testing to study thermal effects. Third, we revised the design to reduce thermal stress.

We selected and are continuing to change to lower power dissipation components on our modules: bipolar to field effect transistors (FET), bipolar and emitter coupled logic (ECL) to complementary metal-oxide-semiconductor (CMOS) logic. We have also improved module layout to spread out heat-generating power components, thus minimizing hot spots.

The actuator motor is composed of a high-power Nd-Fe-B magnet structure and is located outside the HDA. (See Figure 5.) By locating the motor outside, we had to deal with possible torsion of the positioner shaft during seek and settling on track. On the other hand, the location allows cooling air from the blower to circulate more directly on the actuator and prevents dissipation of heat during positioning inside the HDA. Furthermore, the design removes the possibility of data erasure caused by small particles of Nd-Fe-B dislodging during assembly or handling (e.g., kitting).

The interior of the HDA is about 25 degrees Fahrenheit warmer than the ambient environment

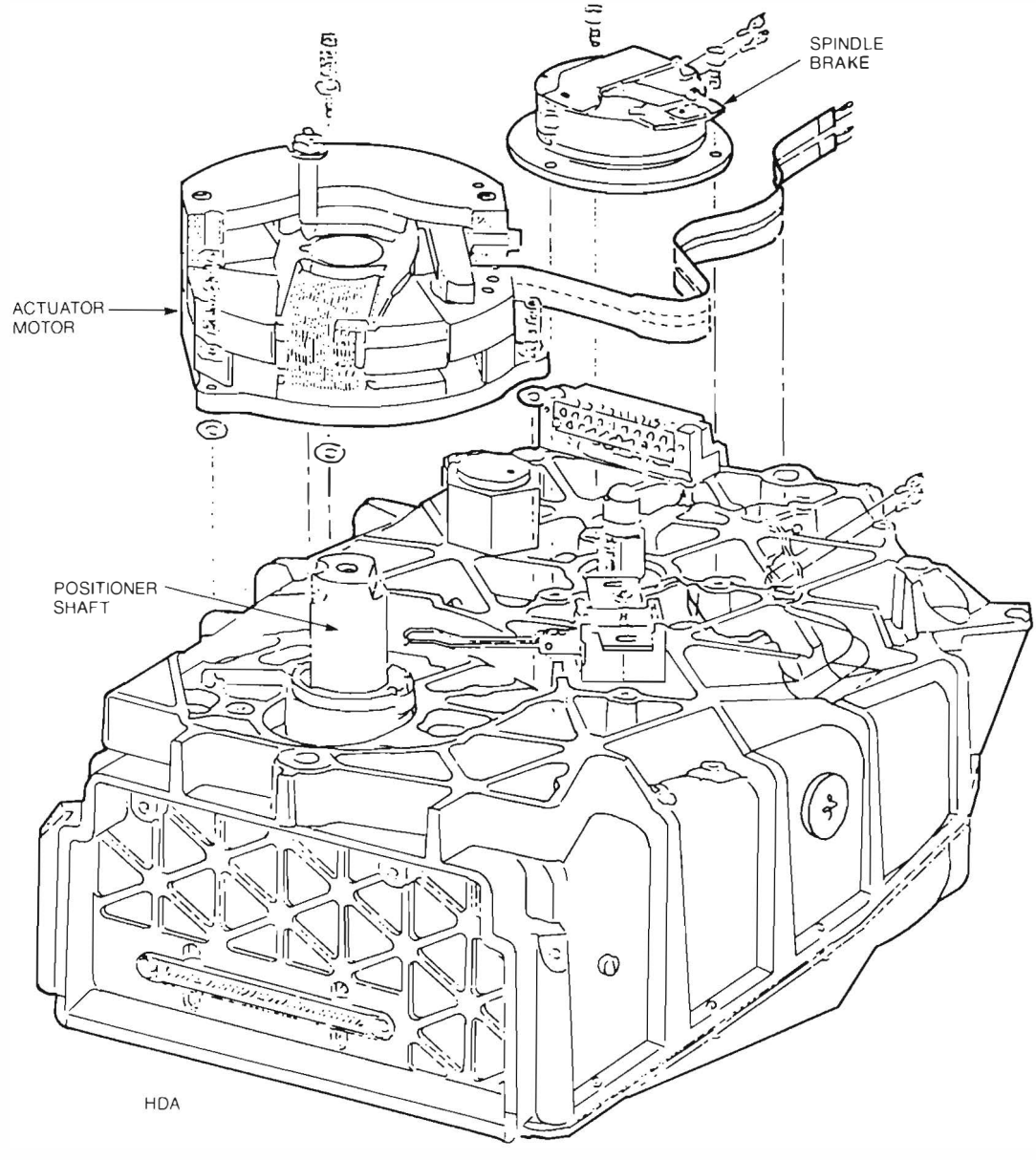


Figure 5 HDA Exterior with Actuator Motor and Spindle Brake Shown

in which it operates. This temperature difference is primarily due to frictional heating of the air circulated by spinning disks and to the spindle motor which rotates the disks. The spindle motor is located inside the hub on which the disks are mounted. When the spindle motors start and stop, wear can occur, both at the head-disk interface and in the motor bearings.

To attentuate disk wear, we use a licensed disk lubricant that improves wear characteristics during takeoff and landing of heads in the landing zones. Disks were tested for frictional characteristics in service with the recording heads, and for loss and migration of lubricant at temperatures above operating range specifications and at speeds of 1.5 to 2 times normal rotational velocities. Results were evaluated both by comparison with minimum and maximum acceptable lubricant thickness, and by comparison with existing products known to function acceptably in the field.

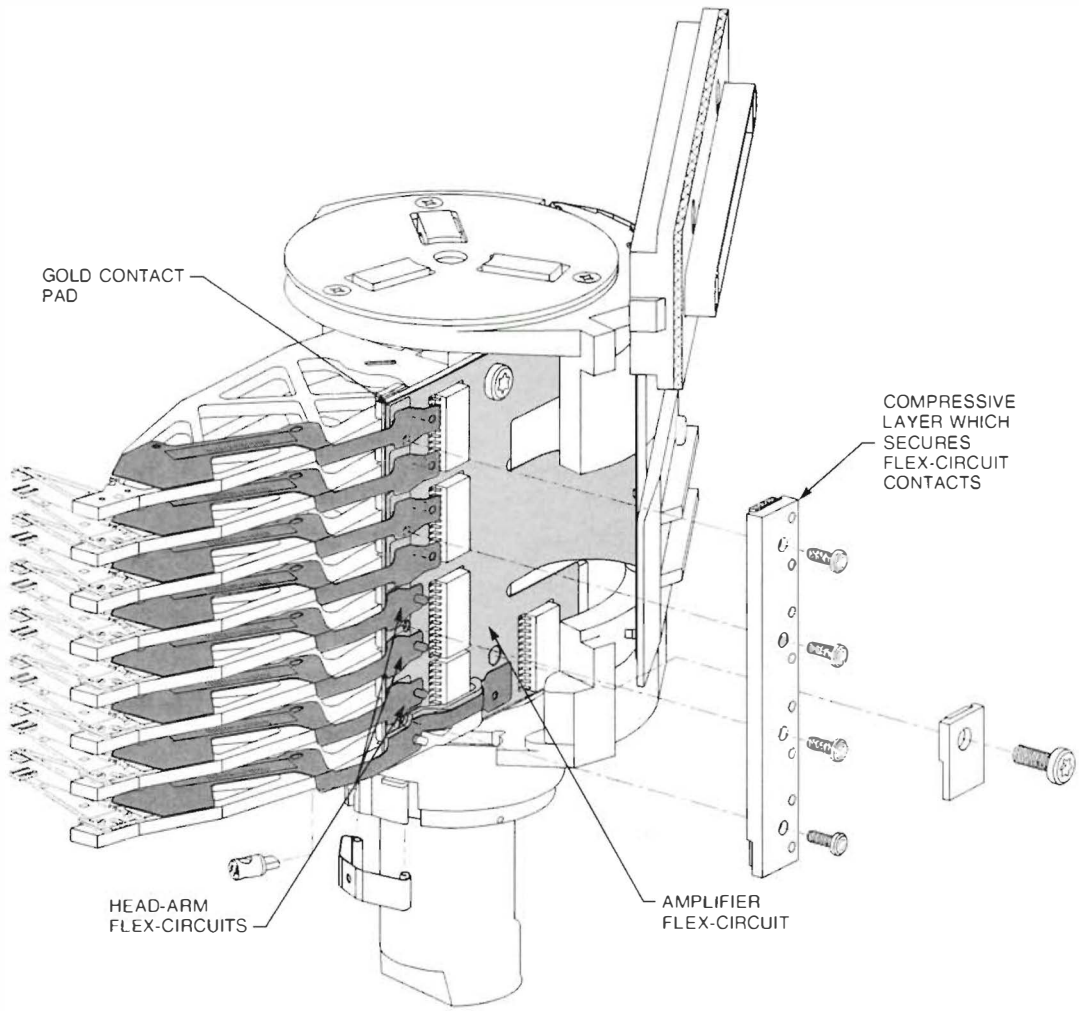


Figure 6 Flex-circuit Gold Pad Interconnect

Cleanliness

RA90 recording heads fly approximately 10 micro-inches (0.25 microns) away from the disks. To ensure that particles of smoke, dirt, dust, oil, and so forth will not be scraped and squeezed between the head and disk, disk drives must be ultraclean. The average office environment typically contains 500,000 to 1,000,000 particles per cubic foot of sizes that could interfere with head flying height behavior. The HDA environment is controlled during manufacture to ensure that it is many orders of magnitude cleaner than the ambient office or computer room environment.

Achieving this cleanliness means providing a clean environment: designing parts so that they can be readily cleaned and fastened together with a minimum of mechanical work; working with vendors to receive parts as clean as possible; and developing a manufacturing process that builds assemblies without excessive particulate generation, thus maintaining cleanliness.

We considered several techniques to electrically interconnect head-arms to the amplifier flex-circuit. Soldering has low contact resistance and can be very reliable; but it generates contaminants due to fluxes and is inconvenient to rework. Pin and socket connectors have had reliability problems in some applications.

To solve this problem, our flex-circuit vendor developed a method that has high reliability, very low contact resistance, is clean, and permits easy rework. This electrical interconnect, shown in Figure 6, uses gold-plated pads on the flex-circuits. Location holes are punched in both head-arm and amplifier flex-circuits. A strip of plastic posts is mounted on the positioner assembly. The amplifier and head-arm flex-circuits are located by pressing

the location holes over the posts. A compressive, gas-tight layer and screws fasten the assembly and allow disassembly for rework. This connection method has additional advantages of low mass and compactness.

Nongalling material combinations such as silicon bronze fasteners in aluminum castings were selected to minimize contamination during the assembly process. The spindle-brake/ground-brush assembly is located outside the HDA to prevent particles generated by grounding from contaminating the HDA. The location also allows more effective heat dissipation during braking.

In the disk stack assembly, we originally designed the disk-hub tolerances at a minimal spacing. We wanted to minimize weight imbalance created by disks off-center relative to the hub, and we hoped to remove balancing as a process step. However, misalignments and small imperfections in circularity caused disks and hubs to scratch or gall during assembly, thus generating particles. We opened up the tolerance and corrected the imbalance by adding balance weights to finished assemblies.

Finally, we considered the environmental resistance of critical HDA components. The HDA is not hermetically sealed. We install a breather filter on the HDA enclosure to permit the HDA to equilibrate with air pressure differences. This attention to air pressure is critical because assemblies are built in Colorado Springs at an ambient air pressure of about 24 inches of mercury, and the assembly may operate at locations ranging from sea level to an altitude of 8,000 feet.

The breather filter may also allow undesirable environmental agents to enter the HDA. So we also place a chemical filter in line with the breather filter to trap many common corrosive agents, such as Cl_2, H_2S, SO_2, and NO_2. Earlier in the assembly process, balance weights for the bottom plane of the spindle are inserted through a tapped hole in the baseplate. The breather/chemical filter is threaded into this hole, closing the HDA opening.

Interconnects

We set out to improve drive reliability by minimizing the number of physical interconnects between electrical components and assemblies, and by controlling their type. Gate arrays of up to 6,000 gates per chip increase reliability, reduce footprint, and minimize assembly and rework. Testability was improved by minimization of cable interconnects between subassemblies.

HDA-to-power-supply and HDA-to-module electrical connectors are self-guided blind interconnects. The HDA in its carrier and module set are slid along "ways," and guides position the electrical connectors to the power supply. The interconnects are gold-plated to provide low contact resistance and are designed for multiple insertions.

Incoming Quality

We worked with our suppliers to receive extremely high quality components, as we believe that such components function more reliably, and because any testing we might do to sort good from bad is likely to inflict damage. For example, if recording heads are already at 98 or 99 percent quality, testing typically causes rejection of several percent good parts, may miss some bad parts, and inflicts several percent handling damage. The result is increased cost without improved quality; quality cannot be tested into the product. Components are tested at the source sites as an integral part of their manufacturing process. We do not re-test heads or other components at the Colorado site prior to use in the drive.

Preproduction Testing for Head and Media Reliability

Thin-film heads and disks used in the RA90 drive underwent extensive reliability testing prior to the start of production. They continue to be tested in a manufacturing audit mode. Since the design of both the heads and the disks were new technologies for Digital, we needed to convince ourselves as well as our customers that reliability was proven.

Recording heads and disks were tested in HDA and test-bed configurations. A key test mode was start/stop testing, where disks are started and stopped with heads taking off and landing. In this test, disks are spun up from zero to 3600 rpm at a slow acceleration rate (similar to power brownout conditions) and spun down under similar deceleration rates. Heads contact disks for relatively long periods under these conditions. The number of cycles was 3 to 20 times the maximum number expected in various service environments.

Following start/stop testing, disk and head electrical signals were evaluated to assess two possibilities: (1) the decrease in signal integrity due to media wear or head structure deterioration, and (2) increases in defects. Visual examinations were made to evaluate wear. For these tests, we used interference microscopy, which is helpful in identifying microtextures and imperfections on mirror-like surfaces. The results indicated excellent durability of the surfaces at the interface and undiminished data integrity.

Data Integrity

We have outlined some of the hardware designs that ensure data integrity, such as placing the actuator motor with its magnet structure outside the HDA to reduce risk of magnetic contaminants. To further ensure data integrity, we rely on DSA and VAXsimPLUS software. DSA provides powerful error correction for data — up to 8 bursts of 10 bits each in one block, or sector — and replaces blocks that appear to be deteriorating based on corrections performed. VAXsimPLUS software provides assessment of the overall data integrity of a drive based on multievent correlations of errors. This assessment permits the repair of a failing unit before application interruption or data loss.[3]

System Interface and Integration

A variety of factors affect system interface and integration of the disk drive. These include

- Ease of installation and repair of the drive
- Compatibility with existing hardware and software systems
- On-board diagnostics
- Differing power requirements
- Appropriate acoustic levels

Ease of Installation and Repair

The RA90 drive meets all the communication protocol and physical interconnect standards of DSA. An RA90 drive may simply be "plugged in" to an existing system, turned on, and used. New cables and new controllers are not required.

The blower is mounted on the front of the cabinet with quarter-turn fasteners. The power supply is similarly mounted on the rear of each drive chassis. The operator control panel snaps onto the front of the blower unit. Two flexible circuits, one ribbon cable, four self-guiding blind connectors, and one snap-in connector provide all power and signal interconnects between these FRUs.

On-board Diagnostics

The RA90 drive has on-line diagnostics which evaluate its status and report to the controller at startup and periodically during operation. Many problems can be resolved by the drive itself or by the drive and controller working together. Status is also evaluated by VAXsimPLUS software, which assesses data integrity issues and prevents application interruption for the system.

International Requirements for Power and Acoustics

Our design requirements called for the RA90 disk drive to be functional in an international environment as well as in the United States. Power, labeling, and acoustic requirements are areas that are often critical in integrating a product into system environments worldwide.

Power The RA90 drive has a universal power supply designed to function at 43 to 63 Hertz, and 86 to 132 or 174 to 264 volts. The "country kit" supplied with each unit provides the appropriate cables for a given outlet and the snap-in operator control panel (OCP) with labels in the appropriate language.

Acoustics Many countries have limits on the acoustic noise produced by electronic equipment, particularly when used in an office environment. The RA90 drive must meet acoustic environment standards for each and every country in which it is marketed.

The RA90 drive makes its biggest contribution to noise when it moves cooling air to HDA, modules, and power supply. Ambient operating temperatures for disk drives range from 10 to 40 degrees C. Cooling required to keep components at an acceptable temperature is greater at higher ambient temperatures within this range. A blower that provides adequate cooling at the upper end of the operating environment has excess capacity in more typical computer room environments and is noisier than desired.

The RA90 drive has a variable-speed blower for cooling and includes a sensor that adjusts blower speed as the ambient temperature varies. When needed, large volumes of air are pumped over the HDA, modules, and power supply. When thermal environments are more favorable, blower speed is reduced; and acoustic noise generation is lowered from 6.4 bels to 5.8 bels.

Damping material installed in the chassis provides additional reduction of acoustic noise. The variable-speed blower and damping material result in a disk drive that is comparable to the ambient noise level of a typical business office or conversational speech.

Summary

Innovative design and close cooperation between engineering, manufacturing, field, and marketing teams allowed the RA90 team to develop a disk drive with a capacity of 1.216GB, a seek time of 18.5 ms, competitive reliability, and an easy-to-use system interface. Moreover, we met internal design cost goals.

Tremendous efforts in technology advancements and implementation were made by key vendors and Digital groups in Shrewsbury, MA; Tempe, AZ; Forge Road, Colorado Springs, CO; Marlboro, MA; Kaufbeuren, West Germany; and San German, Puerto Rico. In addition these groups maintained open communications to ensure product information flowed smoothly between them. We learned as we went, keeping previous experience in mind. For many of us, the RA90 disk drive is significant in its performance specifications and in the teamwork leading to steady progress on this project.

Acknowledgments

The number of people on the RA90 team is quite large and the names too numerous to list here. Nonetheless, I would especially like to thank those who supplied information and photographs for this article, and those whose careful reviews improved the content, accuracy, and style of this article: Bill Brown, Xuan Bui, Tom Fava, Don Jones, Ahmad Kassak, Mark Lewis, Keith Norman, Gary Rauch, John Read, Mike Riggle, Rob Stubblefield, Barbara Wilson, Pat Witt, Sam Yun. I would also like to thank my manager, Chris Wehrli, for support and encouragement throughout the RA90 project, including the writing of this article.

Appendix: Magnetic Recording Technology

Recording Process

Storage and retrieval of data, or recording and readback, is done by heads and disks. The basic operation is identical in theory to audio and video tape recorders, and flexible media drives. However, the specific implementation details, such as size and shape of the components and materials, differ somewhat.

In disk drives, data is organized in circular tracks around the disk surfaces, which rotate at high speed. As shown in Figure A, the recording heads are mounted to an arm/positioner assembly, which can move heads across disks. The heads stop over any track to read the circumferential data path.

The recording transducer is mounted at the rear edge of the slider, a structure which controls the aerodynamic behavior of the recording head. Figure B shows these structures. A transducer has two poles, with a small gap in between. Each pole is a soft magnetic material which is easily magnetized but retains little or no magnetization on its own.

A coil is wrapped around one pole of the transducer. Applying a changing electrical current into the coil induces a magnetic field in the head transducer material. The magnetic field travels around the easily magnetized path of the poles. Since magnetic fields do not require a "conductor," the field jumps the gap as well. Near the gap, the field also "leaks out" into the surrounding space.

The disk is coated with a hard magnetic material. The material requires a strong magnetic field to become magnetized in a particular direction and remains magnetized after such a field is applied. This coating is deposited on top of various substrate preparation layers, which smooth the surface and ensure that the magnetic layer will deposit uniformly. The magnetic layer is then overcoated with a very thin layer designed to assure that the disk will resist any possible environmental effects.

If the leakage field created by the head is strong enough inside the disk's magnetic layer, then the disk's magnetic domains will be oriented parallel to the applied field. This write process is illustrated in Figure B. Data is encoded by controlling the timing of polarity reversal of the applied electrical current relative to a fixed clock rate. As the electrical current applied to the coil alternates in polarity, the induced field in the head also alternates direction, thus writing bits of data.

Readback Process

The readback process operates in a complementary fashion to the recording process. (See Figure C.) After recording, the disk surface has data bits written as separate magnetized areas. Each of the magnetized areas, or bits, behaves like a small permanent magnet, with a magnetic field emanating from it. The head senses this field, which diminishes as the spacing between the gap and data bits increases.

As the disk moves past the head, the field emanating from each of these magnetized areas induces a field in the recording head poles. This alternating magnetic field, traveling the easily magnetized path around the poles, induces an alternating electrical current in the coil, which in turn creates a time-varying readback voltage. The voltage changes are compared to a fixed clock rate and decoded into the 1s and 0s of the data.

Flying Height

One surface of the recording head is ground to a precision shape, contour, and size; it will become the air bearing surface. The disk rotates rapidly beneath the recording head. This rotation of the disk creates an air layer which pushes the recording head, with its aerodynamic slider, away from the disk. The recording head is mounted on a suspension, or flexure, which provides a counterbalancing force by pushing the head toward the disk. The balance of the lifting force created by the disk rotation and the

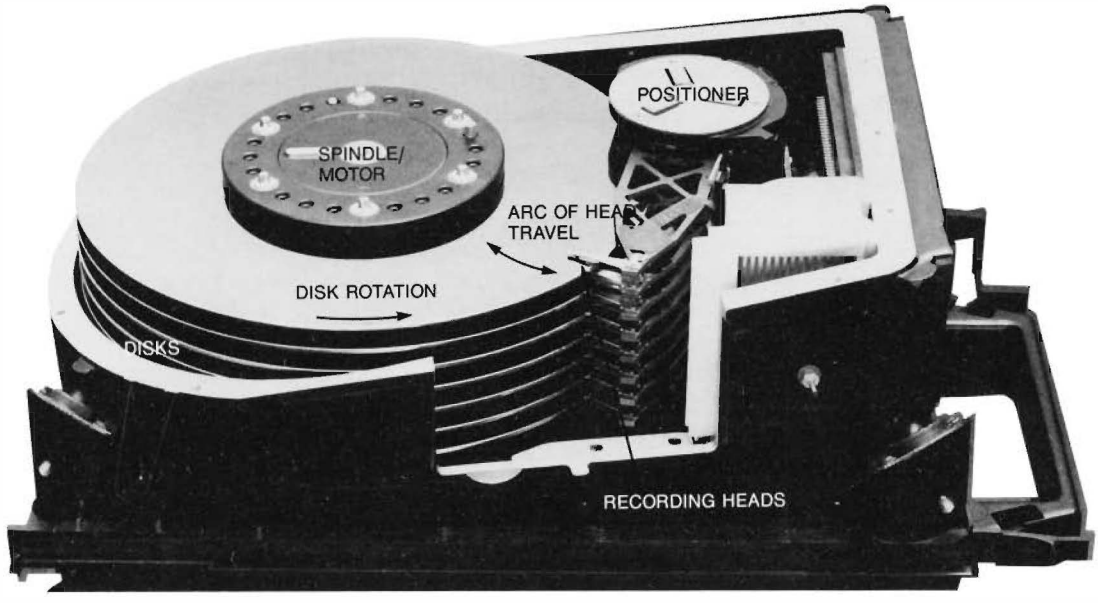


Figure A Example Disk Drive

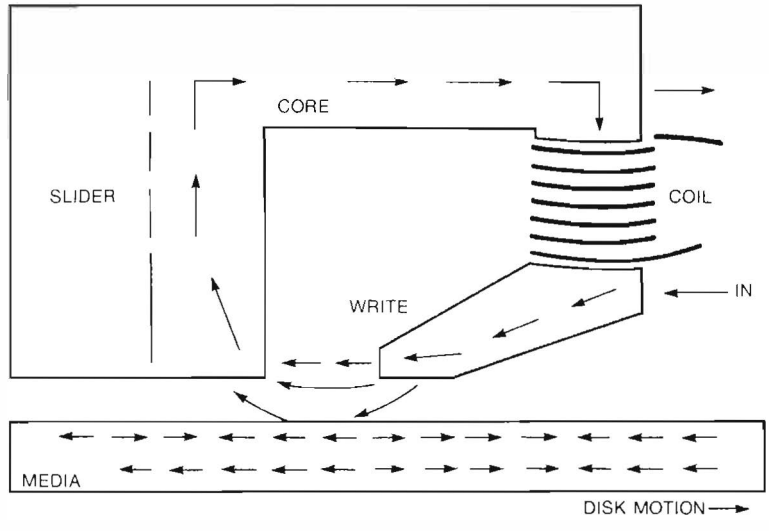


Figure B Side View of Recording Head Showing Magnetic Write Process

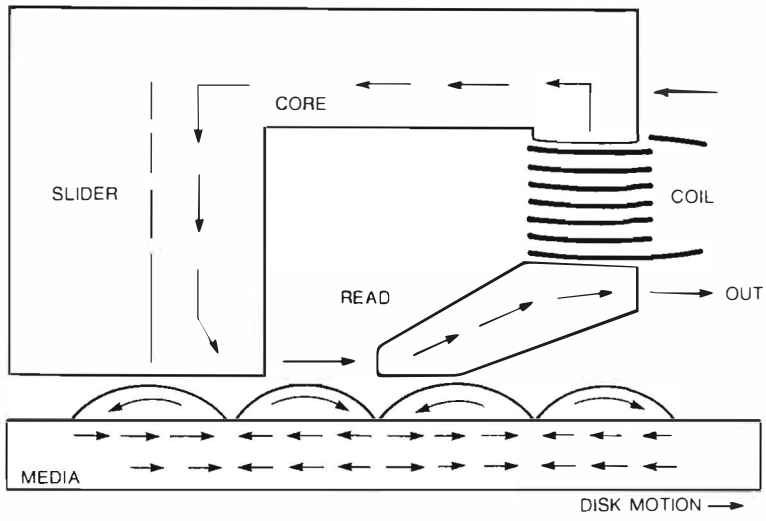


Figure C The Magnetic Read Process

suspension force determine the spacing between head and disk, the flying height of the head.

The flying height of the recording head is not constant across the disk surface. Two factors control the flying height: air velocity and yaw angle. As the recording head moves from inner to outer diameters, air velocity due to disk rotation increases, and tends to push the head further away. As the disk drive is stopped, the air layer diminishes, and the heads land in the disk landing zone. When the drive is started, the air layer builds up and the heads take off.

The RA90 disk drive uses a rotary positioner by which the heads-arms are pivoted around a fixed point to move the recording heads across the disk. The head has a variable yaw angle as it is positioned from inner to outer diameters. In general, the larger the yaw angle (positive or negative), the lower the head flies. The combination of air velocity and yaw angle result in a flying height which is at its lowest at both inner and outer diameters, and peaks near the middle diameter.

The flying height must be controlled so that it is always sufficiently low to read and write data accurately but large enough to prevent unintended touchdowns, or damage to the media. The RA90 recording head flies 10 to 13 millionths of an inch (0.25 to 0.325 microns) from the disk, depending on the radius. Working with these variables in flying height is one of the challenges in developing a reliable mechanics set.

Thin-film Disks

The recording disk used in the RA90 drive is also a thin-film structure, but there are no mask and alignment structures; the entire surface is used. The disk substrate is aluminum, which has been polished to a smooth, flat finish. Various sublayers deposited on the disk provide a surface that has more chemical and physical uniformity than the aluminum substrate alloy. Moreover, this surface is appropriate for the magnetic recording layer.

The magnetic recording layer is exceedingly thin, about 3 millionths of an inch (0.075 microns) and must be uniform, with few if any flaws over the entire disk surface. A recorded data bit in the RA90 disk drive is about 0.000044 inch by 0.000400 inch (approximately 1.1 microns by 10 microns); so even "small" defects in this mirrorlike surface could cause significant data dropouts.

The magnetic layer is then coated with a thin layer to provide mechanical protection in the landing zone during stop and start of the disk drive.

Mechanical Integrity

For the heads and disks to function reliably, we need to control anything that might interfere with the flying height or positioning relative to the track. The flying height can be affected by waviness of the disk surface, by contaminants which may interfere or collect on the head or disk, or by vibration. The positioning can be affected by temperature variations and by vibration or movement of the recording head relative to the disk.

Disk Waviness and Flying Height

The waviness of a disk is a measure of the surface contour over which the head must fly while maintaining its spacing relative to the disk. The contour is usually measured as runout and acceleration. Runout is the amount of vertical displacement of the

disk surface during a revolution. Acceleration is the second derivative of vertical displacement with respect to time.

For a given head-disk combination, we must accommodate factors such as

- Mass of the head
- Spring force of the suspension
- Nominal flying height
- Variation in nominal flying height that can be tolerated during a read or write operation
- Relative positions of heads and disks given stacking tolerances
- Surface characteristics that can be achieved on the disk

Modeling of the air bearing and flying height response to disk surface waviness, and correlation to actual flying height measurements, determined our maximum total indicated runout (TIR) of 0.002 inch and maximum acceleration of 1800 inches per second. The outer recording radius is 4.084 inches, or a circumference of 25.66 inches. The flying height is approximately 10 microinches.

For comparison, one can scale up the runout, circumference, and flying height dimensions to those of an airplane flying from San Francisco to Denver, over the Sierra Nevada and Rocky Mountains. The runout of this path is approximately 4,000 feet, or 2.6 miles (about 8.4 times 10^7 times 0.002 inch). The circumference of the disk scales up to 34,000 miles, or almost 17 round trips. The flying altitude of this theoretical airplane would be only 70 feet above the ground at all times.

The acceleration is a measure of the sharpness or abruptness of the peaks and valleys of the surface. Another way to think of this is to imagine a bump over which a car must travel. The bump may be only a few inches high; but if it is abrupt, it will cause the automobile to bounce up and down, possibly even scraping the road surface. Traveling at 10 miles per hour over a very smooth, rounded speed bump is about equivalent to the maximum acceleration permitted in the RA90 disk drive, that is, 1800 inches per second per second.

The design of the head's suspension, or flexure, is intended to produce a head which can fly consistently over the disk surface undulations, without excessive bouncing up and down.

Track Positioning and Mechanical Stability

In discussing disk waviness earlier and the head's response, we addressed one axis of head motion. In addition to the spacing between the head and disk, we must also maintain the head's relationship to the data track on the disk along the track circumference and across its width.

Variations of the head position with respect to the track circumference will lead to variations in the timing of the data bits read and written. Normally this variation is small and slow relative to the data rate and so can be compensated for; but the variation must be factored into the design tolerances on bit density and the data window in the electronics set.

Variations of head position with respect to track width can be very serious. Spacing between actual written track location and attempted read location is offset. Offset is caused by vibration sources, temperature changes, and electronics error.

There are many sources of vibration that can affect the disk drive: inside the HDA, outside the HDA but part of the drive, and the outside environment.

Inside the drive, bearings in the spindle which supports the disks and bearings in the positioner assembly may have small variations in ball size and finish which lead to vibration. The rapid acceleration and deceleration of the head-arm assemblies as they are positioned across the disk surface cause vibrations or resonances which are sufficient to affect positioning.

The air flow inside the HDA is turbulent and affects the flight of the read-write heads. To minimize this turbulence, there is a baffle next to the disks in the region where head-arms contact the disks. This baffle has been designed to effectively extend the disk surface by moving the turbulent region out beyond the recording surface. The baffle has also been designed to function as a support structure for the clean air filter, which slides between the baffle and the rear of the HDA.

The air cooling system or blower of the disk drive can cause vibration of the HDA assembly. Even though the HDA is a large, heavily ribbed casting, the blower caused sufficient vibration that we found it necessary to attenuate the blower vibration by design changes and to modify the damping characteristics of the blower mounts.

Finally, the outside environment may contribute vibration which leads to offsets; for example, forklifts running on concrete floors cause vibration. All of these vibration sources must be tolerated by the complete system.

In addition to vibration, temperature differences can cause offsets. If a disk is at one temperature when data is written, and then warms only two degrees Fahrenheit, the readback signal will be sufficiently offset to create problems unless some compensation is employed.

Servo Systems

Servo systems control the track over which the heads are positioned and compensate for all sources of misposition: repetitive and nonrepetitive runout, temperature variation, acceleration, deceleration, and vibration.[2]

References

1. In this paper, I have followed conventions currently in use in the disk drive industry. Both English and metric units are used, depending on the quantity being measured and the convenience of a particular set of units. Equivalent quantities are given in parentheses. Numbers given are exact, unless described as approximate in the text.
2. M. Sidman, "Control Systems Technology in Digital's Disk Drives," *Digital Technical Journal* (February 1989, this issue): 61–73.
3. L. Emlich and H. Polich, "VAXsimPLUS, A Fault Manager Implementation," *Digital Technical Journal* (February 1989, this issue): 38–45.

Michael D. Sidman

Control Systems Technology in Digital's Disk Drives

Advanced technologies developed by Storage Systems Engineering have resulted in higher track densities and improved performance in Digital's disk products. The adaptive runout correction system improves tracking accuracy. By anticipating runout, the system reduces the effect of this disturbance in a closed-loop servo system. This is the first known use of digital signal processing in a disk drive servo system. Augmented embedded servo technology provides sampled-data head position-error information from the data heads to the servomechanism and improves positioning accuracies. Digital's disk drives also use an automatic bias force correction system which employs digital signal processing. Finally, digital signal processors are used to control the head-positioning system. The digital control this technology provides is repeatable and versatile. Modern control software tools are employed in disk servo control design and analysis.

The use of sampled-data, digital, and adaptive control techniques in disk drive head-positioning servo systems has enabled increased track densities and improved access time performance in Digital's disk products. These servo methods correct for inherent limitations in drive mechanics such as disk runout, thermal arm shift, and actuator bias forces at track densities well beyond 1500 tracks per inch (TPI).

This paper highlights several technical developments used extensively in Digital's rigid disk products. These technologies include embedded servo systems, adaptive runout correction, automatic bias force correction, and digital signal processors for real-time control. Also discussed is rapid servomechanical system design using modern control tools.

Embedded Servo Systems

Embedded servo technology has in part enabled higher track densities in Digital disk products. This technology is featured on all current Digital-designed and manufactured 14-inch, 9-inch, and 5¼-inch rigid disk products. In this method, position information is embedded on each data surface at sector boundaries on every track. The information gives the head-positioning servomechanism feedback about the relative position of the selected data head to the data track it is following.

This method provides sampled-data position-error information from the actual point on the actuating structure that needs to be accurately positioned. Further, it is more accurate at the instant of sampling than other sensing locations, such as a dedicated servo head or a carriage-mounted tachometer. These alternate sensors are often prejudiced at dc and very low frequencies by nonuniform thermal growth and compliance in the structure.

Figure 1 shows the placement of burst-encoded servo data fields. These fields precede customer data fields on a typical disk data surface used in Digital disk products. Data track centerlines are shown as dotted circular lines, $t-2$, $t-1$, t, and $t+1$. A bipolar position-error estimate is derived by processing the relative detected amplitudes of the two bursts, A and B. These bursts are composed of many

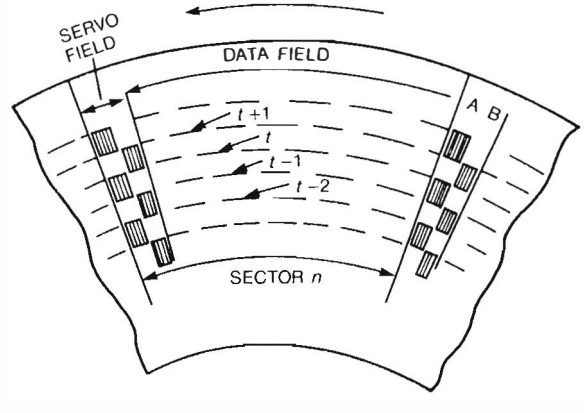


Figure 1 Embedded Burst Servo Encoding on Data Disk Platter

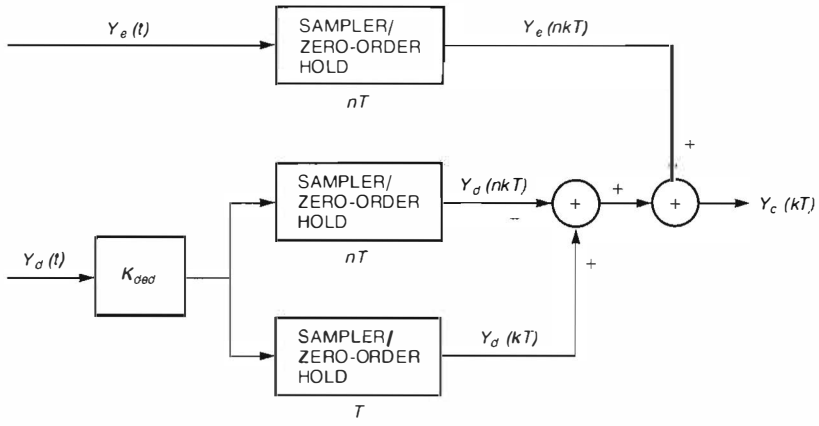


Figure 2 Model of Digital Composite Position-error Estimation

equally spaced transitions. When the data head sits directly over a data track centerline, it equally straddles the two bursts; the result is a zero value for the position-error signal. As the data head moves radially in one direction, the amplitude of one burst linearly increases while the amplitude of the other linearly decreases. Constant-frequency bursts are a useful encoding scheme in part because signal energy is concentrated in a narrow band of frequency. After bandpass filtering and detection, this servo encoding results in a very high signal-to-noise (S/N) estimate of the relative position of the data head to the data track.

The sample rate, and therefore the quality of this position reference, is governed in part by the sector rate. Sector rate is the product of the number of sectors and the rotational rate of the disk. The RA90 disk drive, for example, has a sector rate of 4.2 kilohertz (KHz). Because the sampling is mechanical, embedded servo systems are among the few sampled-data control systems in which *a priori* anti-aliasing filtering is not possible. Therefore, care must be taken to prevent the excitation of actuator modes whose frequencies exceed the Nyquist rate of the system, or half the sector rate.

Augmented Embedded Servo Systems

Sampling at low rates, i.e., less than ten times desired servo bandwidth, introduces substantial phase-loss to the servo loop. The net effect is a limitation in a design's dynamic closed-loop stiffness, runout tracking gains, and vibration disturbance rejection. Consequently, designers are motivated to use augmented embedded servo systems which restore phase and improve servo performance.

An augmented embedded servo system typically utilizes an auxiliary velocity or position sensor on the carriage. If a dedicated servo surface is available, high-frequency components of position-error are sensed nearly continuously between a dedicated surface track and servo head mounted to the carriage. These high-passed components may be blended with the sampled-data embedded position-error signal from the selected data head. Blending restores the phase lost by low-frequency sampling and thus can improve servo tracking accuracy and settling. The resulting composite signal provides a form of interpolation in time between embedded samples. If the crossover frequency between the embedded and high-passed dedicated position-error signals is significantly greater than overall servo loop bandwidth, disturbances appearing only in the auxiliary sensor will have minimal effect on the servo system, as desired.

If a dedicated servo surface is not available, a carriage velocity transducer may be used as the auxiliary sensor. A carriage tachometer used for this purpose is usually implemented by sensing back-EMF on a carriage-mounted coil cutting a field generated by a magnet mounted to the disk drive baseplate. A composite position-error signal may be produced by first passing the velocity estimate signal through a resettable integrator; the integrator is reset to zero at sector boundaries. A composite position-error signal results when the integrated velocity signal is added to the zero-order-held embedded position-error signal.

Interestingly, the dynamic response of embedded servo systems of comparable bandwidth may be quite dissimilar depending on the auxiliary sensor and specific blending method selected.

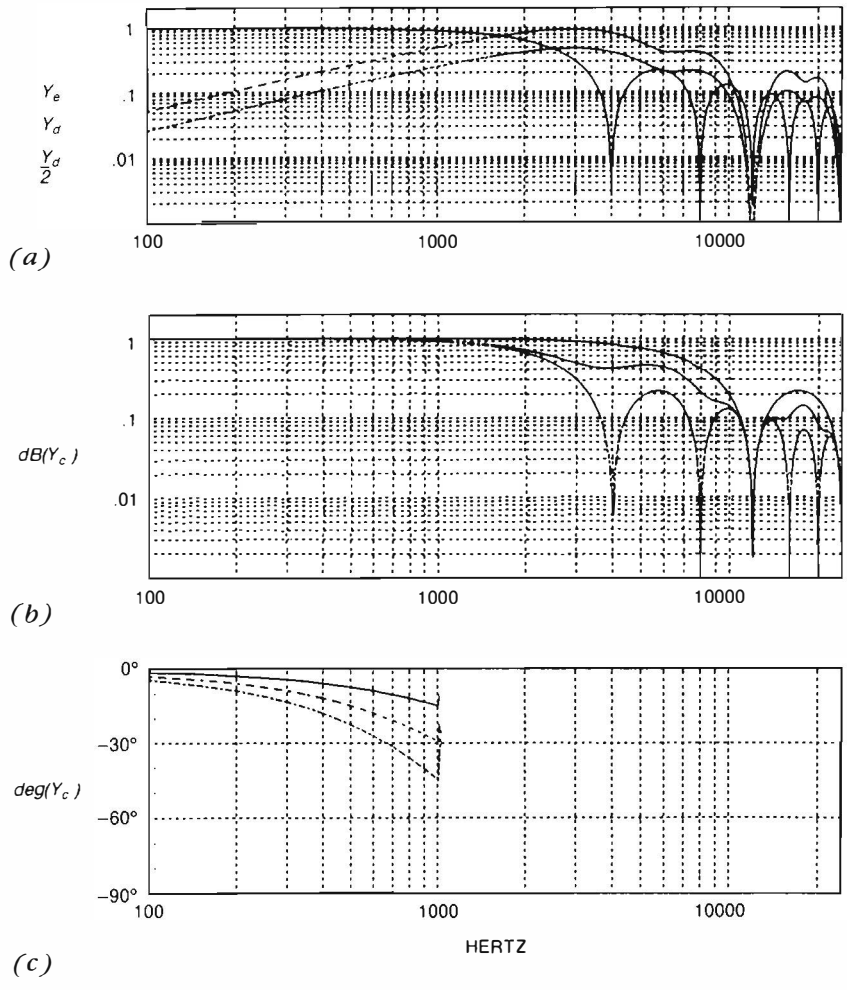


Figure 3 Frequency Response of Digital Composite Position-error Estimators

Digital Composite Position-error Estimation

Embedded and continuous dedicated position-error information may be blended in discrete-time. Figure 2 shows how continuous, dedicated position-error, $Y_d(t)$, may be thought of as being sampled both at the embedded sector rate, $1/nT$, and at an integral multiple, n, of that rate, $1/T$.[1]

The resulting sampled-data composite digital position-error estimate, $Y_c(kT)$, is determined at each sector sample-instant solely by the embedded servo position-error estimate, $Y_e(nkT)$. At intermediate sample instants, the composite digital position-error estimate is determined by embedded position-error plus an estimate of how much the dedicated-servo position-error signal has changed since the last embedded sample. In this way, dc and low-frequency errors between the dedicated and embedded position-error sensors are ignored, and sampling phase-loss is largely restored. In practice, the ratio of the dedicated to embedded sample rates is three or greater. Phase improves as this ratio increases.

Phase may be *partially* restored by using a fraction, K_{ded}, of the available dedicated position-error signal. This partial restoration may be necessary in systems where the mid-frequency estimates of position-error from the dedicated servo head do not correlate well with the embedded position-error estimates. Partial restoration allows the designer to trade off phase improvement against disturbance introduction.

Figure 3a shows the transfer functions from dedicated and embedded position-error (assumed equal) to composite position-error for full and 50 percent digital blending. The dedicated position-error component is effectively high-passed by this process and

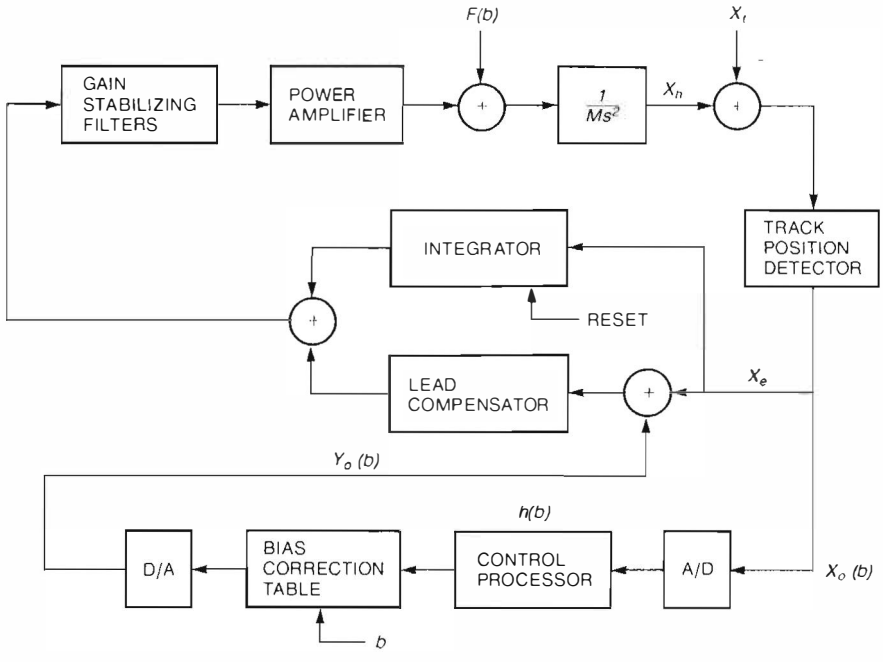


Figure 4 Automatic Bias Force Correction System Block Diagram

sampled in this example at three times (12 KHz) the rate of the embedded position-error contribution. The gain magnitude and phase of the composite blended signal are shown in Figures 3b and 3c. As K_{dcd} increases from 0 to 0.5 to 1.0, some of the high-frequency sampling nulls in the magnitude response are eliminated. Consequently, phase-loss due to sampling decreases.

Automatic Bias Force Correction

Digital's disk drive products utilize an automatic bias force correction system that nullifies the effect of exogenous carriage forces on the closed-loop track-following servo system.[2] This correction system greatly improves track capture and shortens settling times. The system uses digital signal processing to build a smoothed table of optimal correcting levels as a function of coarse actuator position.

Bias forces are usually principally due to windage, cable forces, gravity, and magnetic motor biases in head-positioning actuators. The net force or torque (in the case of rotary actuators) presents an unwanted but repeatable dc disturbance to the position control system. The result is a small position-error offset that is a nonlinear function of coarse actuator or head position. Electronic circuit offsets from sources other than the embedded position sensor may also be grouped with the mechanical sources of error. Stochastic forces, however, principally originating from actuator bearing friction and stiction add to the closed-loop error in disk drives, but generally not predictably.

In the past, proportional-integral-derivative (PID) compensators alone coped with the problems created by bias forces on the head-positioning actuator. But the time taken by the compensator integrator or lag-filter to substantially reject the dc disturbance increases settling time. In recent disk drive designs, access-time performance has become more important and track densities have increased, resulting in the need for both improved positioning accuracies and track capture. Integral control *alone*, therefore, has become a less attractive or even infeasible solution.

The automatic bias force correction system solves the bulk of the problem. The correction system gives the servo system the information it needs to nullify the repeatable disturbance component before it is encountered. The servo system can nullify the disturbance as the servo switches from seek to track-follow mode some distance from destination track. Thus, correction occurs throughout and following the entire head-settle mode. The compensator integrator is then only responsible for rejecting the effect of unpredictable, stochastic errors.

Figure 4 shows a simplified block diagram of the automatic bias force correction system connected to an analog position control system. Position-error information, X_e, the scaled difference between head

position, X_b, and track position, X_t, is digitized by an analog-to-digital (A/D) converter and conditioned by the servo control processor.

A lookup table of optimal correcting levels, $Y_o(b)$, is generated as a function of coarse head position or radial band number, b. After the calibration interval completes and when the heads are instructed to seek to a given track, the corresponding correction level, $Y_\bullet(b)$, for the band in which that track resides is accessed from the table and introduced to the servo system.

The optimal correcting signal level exactly matches the level of position-error signal offset, $X_o(b)$, due to repeatable biases in the system. Knowledge of servo bandwidth, actuator inertia, or gain parameters of the power amplifier, motor, compensator, or track-position detector for a given drive design is not required to make offset correction accurate.

Signal Processing Technique

To determine the offset correction at a given track, the repeatable dc component of the corresponding position-error must be determined. This may be done in part by averaging the measured closed-loop position-error samples while using a proportional-plus-derivative (PD) or lead compensator over one or more integral revolutions of the disk. Averaging reduces the effect of runout on the offset estimate. Since offset is a function of gross head position, the disk is divided into a number of bands, each consisting of the same number of tracks. Position-error offset is measured at tracks in the center of each band and stored for subsequent processing.

An added complication is that the offset may be affected by the direction from which the track is accessed. Figure 5a shows the measurement of position-error offset while using a lead or PD compensator over the range of actuator travel; 22 bands denoted by band number, b, are used in this example. Notice that there is a difference between the offset curve for tracks approached in the forward direction (represented by circles) and the curve for tracks approached from the reverse direction (represented by triangles). This difference is generally attributed to the side force required by the balls in the actuator bearing to squeeze the grease out from under themselves after arriving at a new track. Noisy data results from a number of sources, including actuator friction and limited A/D converter resolution.

Let the forward and reverse measured position-error values be given by $X_{\bullet_{fwd}}(b)$ and $X_{o_{rev}}(b)$, where $1 \leq b \leq B$. B is the number of bands on the disk.

It is important to substantially eliminate the effects of stochastic disturbances on the bias estimate. Otherwise these effects may be exaggerated when correction is applied to the control system. Elimination of these effects is the role of subsequent digital signal processing.

The first step is to compute the average of the forward and reverse sets of data, $X_{o_{avg}}(b)$ as shown in Figure 5b. The next step is to compute an average forward offset value, $X_{off_{fwd}}$, and reverse offset value, $X_{off_{rev}}$, to be added to the bias correction when a track is first accessed from the given direction.

$$X_{\bullet_{avg}}(b) = \tfrac{1}{2}\left[X_{\bullet_{fwd}}(b) + X_{o_{rev}}(b)\right]$$

$$X_{off_{fwd}} = -X_{off_{rev}} = \frac{1}{2B}\sum_{1}^{B}\left[X_{\bullet_{fwd}}(b) - X_{\bullet_{rev}}(b)\right]$$

The averaged bias table, $X_{\bullet_{avg}}(b)$, is still quite noisy and requires digital filtering for smoothing. A three-point symmetrical finite-impulse-response (FIR) filter with unit pulse response, $h(b)$, shown in Figure 6, is convolved with sequence $X_{\bullet_{avg}}$ one or more times to produce the smoothed correction table, $X_{filt}(b)$, shown in Figure 5c. The discrete linear convolution filtering operation is given by:[3]

For $2 < b < B - 1$, and $Dim(h) = 3$:

$$X_{filt}(b) = \sum_{i=-1}^{1} h(i)\,X_{\bullet_{avg}}(b - i)$$

When filtered bias correction, $X_{filt}(b)$, and average forward or reverse offset value $X_{off_{fwd}}$ or $X_{off_{rev}}$ are added to the servo, the position-error is reduced to the amount shown in Figure 5d. This represents the amount prior to the use of a lag or integrating filter. For this example, roughly over 85 percent of the bias error is eliminated before integrating control is initiated. Settling is dramatically improved.

Adaptive Runout Correction System

This section describes the adaptive runout correction system (ARCS). ARCS adaptively nullifies the effect of repetitive disk runout in a disk drive head-positioning servo system. Digital signal processing in the form of *circular* convolution compensates for closed-loop servo dynamics and ensures rapid, accurate convergence to an optimal feedforward correcting signal. Convergence on a signal occurs even in the presence of significant high-frequency disturbances, measurement noise, and nonrepetitive runout. Correction of individually selected runout frequency components is permitted and imposes no additional on-line computation; computation is extremely minimal in any case. Results in disk servo systems typically indicate better than 95 percent correction of repetitive runout errors after just two iterations.

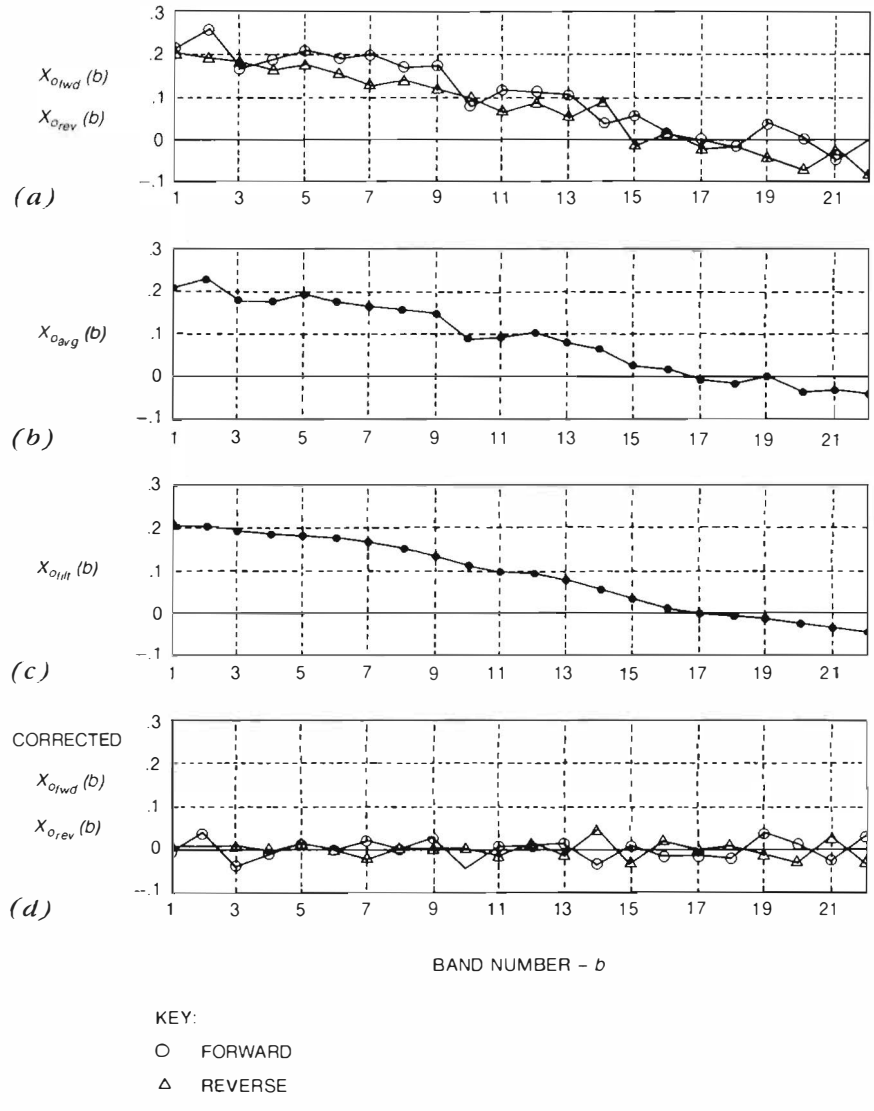


Figure 5 Automatic Bias Force Correction System Signal Processing

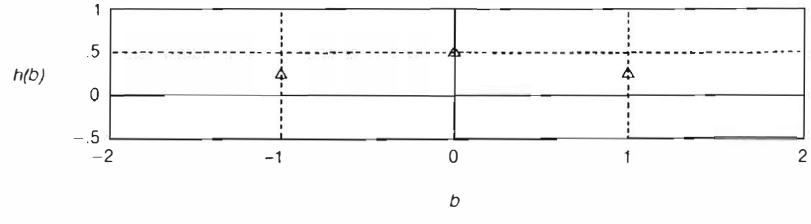


Figure 6 Unit Pulse Response of Bias Force Smoothing Filter

Adaptive Disturbance-rejection Systems

Adaptive disturbance-rejection techniques such as ARCS inject a feedforward signal that anticipates a disturbance. The intent is to reduce the effect of the disturbance at some point in the control loop. This technique may be particularly helpful if closed-loop system gain is not adequate to deal with the disturbance. The additive correcting signal used does not modify the loop dynamics. Therefore, this adaptive technique will not destabilize a system given that adaptation itself is slow relative to loop bandwidth.

It may be more appropriate to classify such a system as an adaptive signal processing system augmenting a control system than as an adaptive control system that modifies plant compensation or control law dynamics to achieve its goals. Adaptation occurs in the process of estimating and iterating to an optimal correcting signal that nullifies the effect of the disturbance at a specific point in the closed-loop system. Prerequisites for employing such a system include a parametric model of the disturbances and a crude model of the inverse closed-loop frequency response of the transfer function from the signal injection point to the point of desired nullification. This model need only be accurate over the spectrum of desired correction.

Runout and Disk Servo System Operation

The goal of a tracking embedded disk servo system is to minimize the radial displacement of a selected data head relative to the data track it is following. This position-error information is measured directly on a sampled-data basis in the sector boundary regions on each data track on each disk where servo position fields are written. The radial position of the data track sector may be considered to be a dynamic reference command for head position and is directly affected by mechanical runout. The position error that results from this command or disturbance is attenuated by the action of the servo system. However, runout rejection is practically limited by control loop bandwidth.

Repetitive runout may be due to mechanical nonconcentricity of disks or to the embedded data tracks in which position information is servo written onto the disk. Runout disturbance may be due to other sources, such as deformation or tilting of disks away from the axial direction. Repetitive runout disturbances may be different for each data head in the drive or may be similar for data heads accessing opposite sides of the same disk in the stack. Stochastic or nonrepetitive runout usually originates principally from ball bearings that support the spindle. This component of runout may be much smaller than or commensurate with the magnitude of repetitive runout dependent on drive and spindle design. Air-bearing spindles, for example, can reduce nonrepetitive runout to negligible values.

Repetitive disk drive runout disturbances are periodic and present themselves as sinusoidal components synchronized to disk rotation. The frequency components of repetitive runout errors typically include fundamental frequency runout and several higher harmonics. For a disk spinning at 3600 rpm, this translates to runout frequencies of 60, 120, and 180 hertz (Hz). Measured position-error signals contain some high-frequency components resulting from local media anomalies that affect the quality of the signal. Thus, there are differences in closed-loop tracking on adjacent tracks even with the same head. However, low-frequency errors, usually up to 180 Hz, generally are similar over the range of radial displacement for a given head.

Servo bandwidths are typically about 500 Hz for many commercial products. Such bandwidths offer limited runout attenuation of about 10 to 20 at 60 Hz, decreasing with frequency.

Role of Adaptive Runout Correction System

An adaptive runout correction system can be used to nullify the repetitive residual tracking error due to runout. Since repetitive runout is predictable or very slowly time-varying for a given data head, measurements of runout disturbances can be infrequently scheduled by the servo control system. These measurements can then be used to give the control system information about the disturbances it is *about* to experience. Stochastic, nonrepetitive disturbances, which by definition are not predictable, are best ignored in such a system by employing an averaging technique.

The net result is improved tracking accuracy and response to such predictable disturbances with otherwise unattainable runout rejection.

Adaptive Runout Correction System Operation

Figure 7 illustrates the components of an adaptive runout correction system in an embedded or sectorized servo disk drive.[2,3]

Relative radial displacement information between a selected data head and its respective data track is provided by a sampled-data position-error signal. This is accomplished by demodulating the readback signal from the head while the head is passing over servo data fields at sector boundaries. The position-error signal corrects the position of the head and is

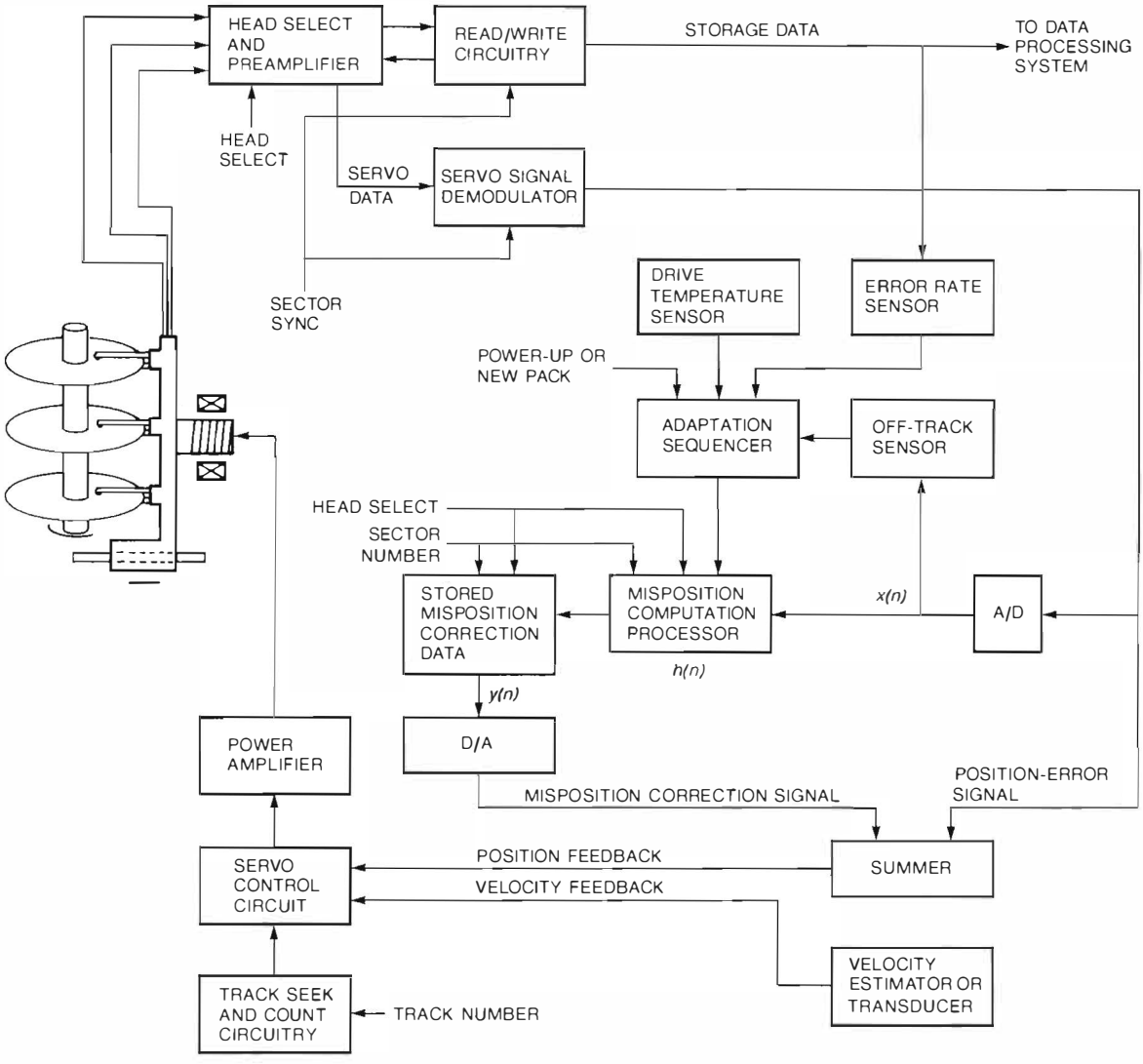


Figure 7 Adaptive Runout Correction System in a Disk Drive Servo

fed back through a summer to the servo compensator or control circuit. This circuit provides commands to the power amplifier responsible for driving current into the carriage actuating motor.

The position-error signal, $x(n)$, where n is an index corresponding to sector number, is also fed into a runout or misposition computation processor with unit pulse response, $h(n)$. This digital signal processor performs digital circular convolution discussed below.

The runout or misposition correction signal, $y(n)$, acts to nullify the effect of runout for the selected data head and is generated by the processor. The signal is stored for immediate lookup in a small random-access memory (RAM) table that contains the stored runout or misposition correction data. This signal is ideally summed back into the servo loop through the summer just before the servo compensator.

The adaptation sequencer provides a trigger to the runout processor and servo system to enable runout adaptation for the disk drive as necessary.

Signal Processing Technique

The basic algorithm involves initially two steps: (1) measuring an integral number of revolutions of sampled position-error signal, $x(n)$, and (2) averag-

ing values obtained across similar sectors on different revolutions to extract the repetitive runout component only. After these steps, high-frequency components in the measurements will still exist corresponding to local media defects or track anomalies.

These high-frequency components may be eliminated completely by performing digital circular convolution with a cosine-based sequence. This approach leaves only the low-frequency sinusoidal components of the position-error signal. For example, the first Fourier or fundamental frequency component of the position-error signal can be extracted by convolving it with a cosine waveform of frequency corresponding to the rotational frequency, 60 Hz.

To nullify the fundamental frequency component at the point of measurement, the processed position-error signal is injected back into the servo system. However, there may be phase lag and attenuation between the point of correction signal injection and the point in the control loop where nullification is desired — the position-error signal. Generally, this phase lag and attenuation are reasonably well known, given that servo bandwidth and mechanical resonances in the structure are much higher in frequency than the repetitive runout components. Therefore, it is reasonable to expect that a corresponding amount of phase lead and gain could be applied to the first Fourier component of position-error signal to produce a suitable correction signal. The process of individual Fourier component extraction, phase and gain adjustment, and rejection of other Fourier components can be easily accomplished simultaneously. The position-error sequence can be circularly convolved with a phase-shifted and scaled cosine wave sequence of appropriate frequency.

If several frequency components of runout need to be corrected, the unit pulse response of the filter would be the *superposition* of several corresponding cosines. Each cosine is at the desired frequency of correction and is individually phase-shifted and scaled appropriately to compensate for the lag and attenuation of the closed-loop control system at each corresponding frequency. Thus, one filter operation can perform all of the above in one step, independent of the number of frequencies to be corrected.

However, the quality of nullification or amount of correction is limited at this point by the accuracy of the *a priori* estimate of closed-loop control system gain and phase. This limitation is the motivation for subsequent iteration or adaptation.

By injecting the most recently computed correcting signal and simultaneously measuring the remaining repetitive runout components in the resulting position-error signal, a second correcting signal can be computed using the above method. When summed with the original, this second signal will improve the runout correction still further. It is desirable to be able to iterate indefinitely using this technique. The system can thus achieve the desired amount of correction or keep up with very slowly varying repetitive runout without having to start fresh at each calibration or adaptation interval.

To ensure convergence to an optimal correcting signal after an arbitrary number of iterations, it is important to process the measured position-error signal in such a way as to reject high-frequency components. If these components were allowed to be introduced back into the servo system, they would tend to build up and cause divergence after several iterations. This is another problem that this signal processing method overcomes.

Digital Filtering Formulae

Assume that measured, sampled-data position-error signal, $x(n)$, for a given tracking data head and data track is repetitive and is given by:

$$x(n) = DC + \sum_{i=1}^{N/2} A_i \sin\left[\frac{2\pi i(n-1)}{N} + \phi_i\right]$$

where

$x(n) = x(n + kN)$, n and k are integers and $1 \leq n \leq N$.

N = number of samples/revolution or data sectors/revolution.

Let the unit pulse response, $h(n)$, of a periodic digital filter operating on $x(n)$ be given by:

$$h(n) = \frac{A_{dc}}{N} + \frac{2A_{fund}}{N}\cos\left[\frac{2\pi i(n-1)}{N} - \phi_{fund}\right] + \frac{2A_{2nd}}{N}\cos\left[\frac{4\pi i(n-1)}{N} - \phi_{2nd}\right] + \frac{2A_{3rd}}{N}\cos\left[\frac{6\pi i(n-1)}{N} - \phi_{3rd}\right]$$

where

A_{dc} is the desired gain of the filter at DC.

A_{fund}, ϕ_{fund} are the desired gain and phase lead of the filter at the fundamental frequency of rotation.

A_{2nd}, ϕ_{2nd} are desired gain and phase lead of the filter at twice the fundamental rotational frequency.

A_{3rd}, ϕ_{3rd} are desired gain and phase lead of the filter at thrice the fundamental rotational frequency.

The above parameters anticipate closed-loop control system and optional anti-aliasing filter gain and phase lag. They then *individually* correct for gain and lag at each frequency where forced sinusoidal correction is applied to nullify disturbance errors.

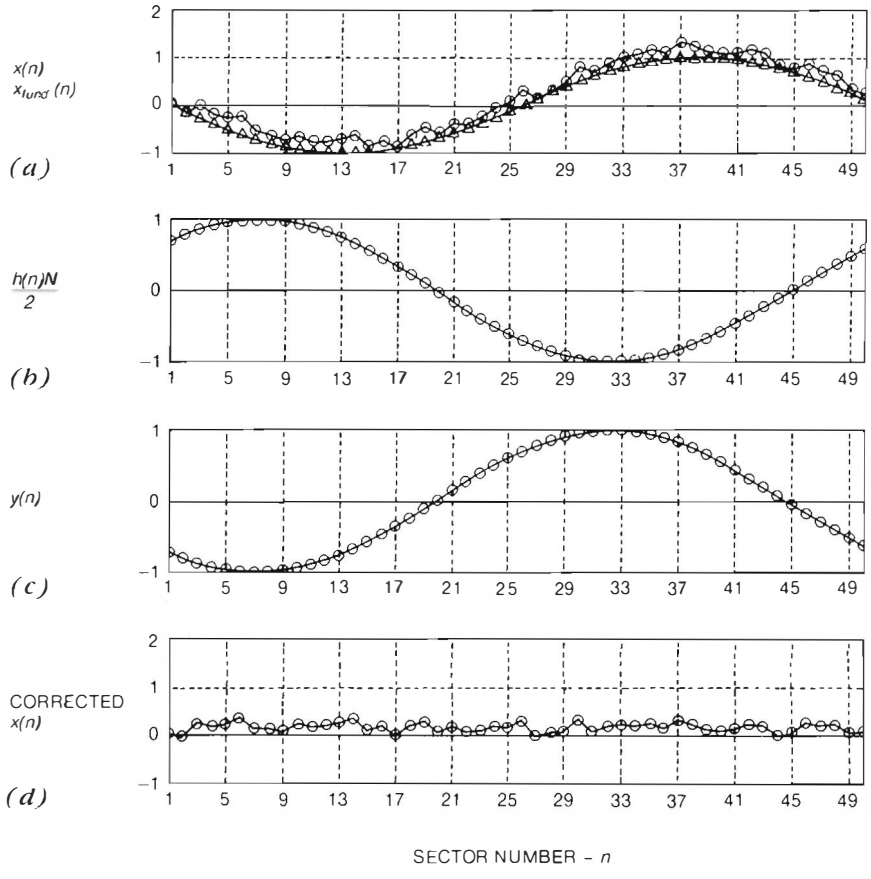


Figure 8 Digital Signal Processing Example

The desired correction signal, $y(n)$, is determined by digital circular convolution:[5]

$$y(n) = \sum_{i=1}^{N} h(i)\, x\left[(i + n - 2)\, MOD\ N + 1\right]$$

$$y(n) = A_{dc}DC + A_{fund}A_1 \sin\left[\frac{2\pi(n-1)}{N} + (\phi_1 + \phi_{fund})\right] +$$

$$A_{2nd}A_2 \sin\left[\frac{4\pi(n-1)}{N} + (\phi_2 + \phi_{2nd})\right] +$$

$$A_{3rd}A_3 \sin\left[\frac{6\pi(n-1)}{N} + (\phi_3 + \phi_{3rd})\right]$$

The resulting periodic correction signal, $y(n)$, lacks frequency components other than at the frequencies where correction is desired. This is essential to ensure convergence with iteration.

Example

Figure 8 illustrates the operation of such a filter on noisy, dc-biased position-error signal, $x(n)$, derived from a hypothetical 50-sector disk. For clarity, the first Fourier component, $x_{fund}(n)$, is plotted along $x(n)$ in Figure 8a.

For the case where only fundamental frequency runout rejection is desired, A_{fund} is nonzero, but A_{dc}, A_{2nd} and A_{3rd} are zero in formulating $h(n)$. This results in a simple sine wave correction signal, $y(n)$:

$$y(n) = A_{fund}A_1 \sin\left[\frac{2\pi(n-1)}{N} + (\phi_1 + \phi_{fund})\right]$$

The filter gain, A_{fund}, is set to unity, and ϕ_{fund} is set to 45 degrees lead. This can be seen in scaled $h(n)$ in Figure 8b.

The runout correction signal, $y(n)$, in Figure 8c can be seen to contain fundamental frequency runout information only and lacks the dc bias and high-frequency components of $x(n)$. Further, it is phase shifted by 45 degrees to the left compared to $x_{fund}(n)$, indicating the proper amount of phase lead. Figure 8d shows $x(n)$ with $y(n)$ injected into the servo system. As desired, the fundamental frequency runout is eliminated. The remaining position-error contains dc and high-frequency components only in this example.

Microprocessor Implementation Issues

Computational roundoff would seem to be a problem when adding an arbitrary number of sequences. However, the long-term effect of roundoff is not generally serious for applications where word length is significantly larger than the resolution of the data converters. If this is not the case, the problem can be easily solved by occasionally passing the last-computed correction signal through a Fourier component extracting filter. The filter passes only the desired correcting frequencies unattenuated and without phase shift. Eight-bit A/D and D/A converters and 16-bit fixed-point arithmetic have proven to offer acceptable performance for all laboratory implementations, though more resolution might be desirable.

This runout correction method generates a runout correction table whose size is the number of sectors times the number of heads or platters. For example, a drive with 50 sectors and 15 data heads would require a maximum of only 750 bytes, or words, of table lookup.

The amount of on-line computation is limited to table lookup. The computation involves adding one number to the D/A converter per sector, *independent of the number of runout frequency components being corrected*. Digital filtering may be done in a quasi off-line manner. In fact the first laboratory implementation used an Intel 8085 microprocessor, which did not even have multiplication in its instruction set. This indicates the low computational intensity and practical nature of this real-time system.

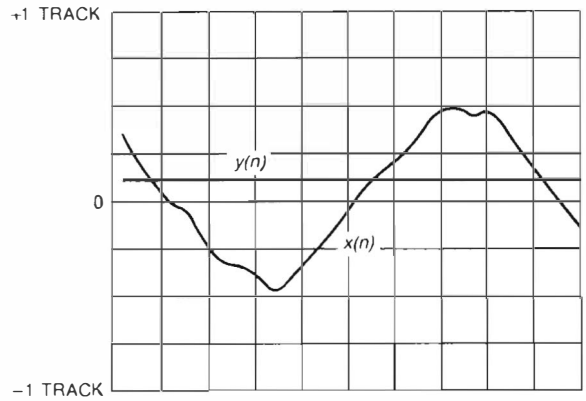
Adaptation may be triggered by a time scheduler in the drive microprocessor or by changes in drive temperature, growth in off-track position error, error-rate indicators, etc. An adaptation at power-up is usually a wise choice.

Results in Disk Products

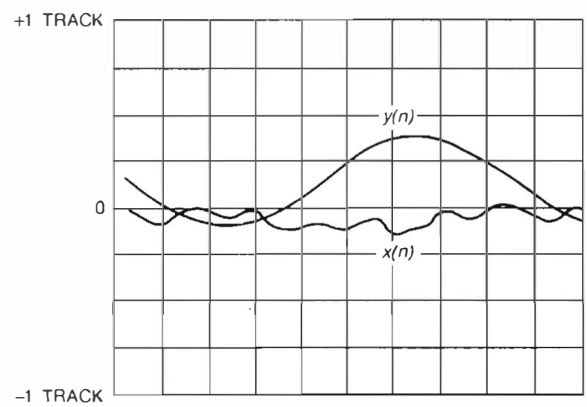
ARCS has been proved both effective in reducing repetitive runout and essential to the success of Digital's RC25 and RF30 disk drives and others still in development at this writing.

Experimental results using this method of runout correction indicate that two iterations take out about 95 percent of the remaining repetitive runout in the drive. Fundamental and first harmonic runout correction generally suffice to adequately remove the repetitive components.

Figure 9 shows how the runout correction system attenuates the fundamental frequency component and dc term of the position-error signal in a disk drive with large runout. In Figure 9a, $y(n)$ is inactive, and the position-error signal is seen to have substantial runout and high-frequency terms. The nonrepeatable component of runout can be clearly seen in this multirevolution time exposure. Figure 9b shows the position-error signal, $x(n)$, and active correcting signal, $y(n)$, compensating for both dc and fundamental runout.



(a) No Runout Correction



(b) With Runout Correction

NOTE:
2 mS PER DIVISION

Figure 9 Adaptive Runout Correction in a Disk Drive

ARCS Summary

A method for adaptive correction of repetitive runout in disk drives using digital signal processing techniques has been presented. The method has found practical application in commercial products

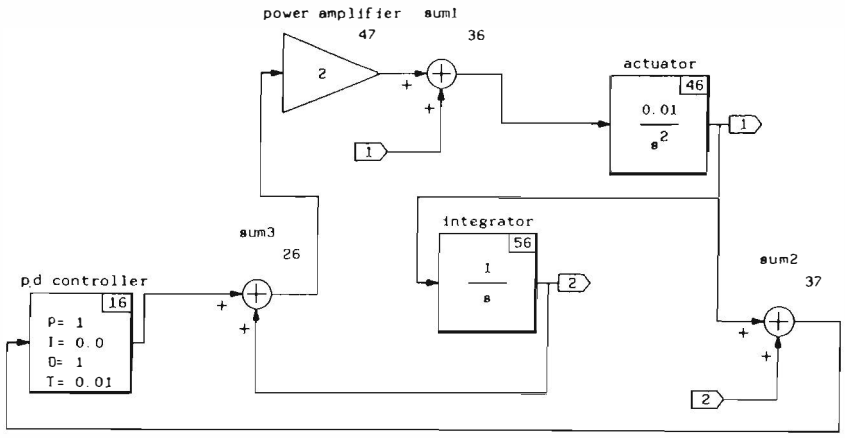


Figure 10 MATRIXx/WS Block Diagram Display

and is characterized by its fast convergence, accuracy, and very low computational intensity.

Unlike other runout correction schemes proposed in recent control systems literature, this system does not require careful selection of tuning parameters.[6] Because only selected runout frequencies are corrected, other side effects such as excitation of actuator resonances and response to nonrepeatable runout are eliminated.

Digital Signal Processors for Real-time Disk Servo Control

Digital's RA70 and RA90 disk drives utilize digital signal processors (DSPs) to control the head-positioning system. The value of digital control lies primarily in its repeatability and versatility. Ease of unit test and savings in printed circuit board space, especially critical for smaller drives, have also contributed to the proliferation of this technology in Digital's disk products.

The motivation for using digital signal processing in Digital's disk drives after adaptive runout correction and automatic bias force correction was to eliminate multistage analog notch filters. Such filters have been used in disk products to gain-stabilize lightly damped resonances that appear in the actuator plant transfer function.

In practice, the use of these devices has grown considerably beyond even these functions. The use of DSPs in disk servo control is probably best exploited in the RA90 drive. In this drive, a Texas Instruments TMS32020 DSP is additionally responsible for velocity estimation, quadrature track splicing, position interpolation, calibration of embedded channel gain and relative head offsets, velocity seek control, dedicated-embedded blending, servo fault detection, and more.

The hidden benefits of a real-time control processor for the design engineer include the ability to rapidly modify control parameters and control algorithms and immediately test the effects of the change in the laboratory. This flexibility is particularly important as the drive mechanics are modified during the product development cycle.

The future appears quite promising for this technology. The use of C compilers for commercially available DSPs is expected to substantially further reduce servo design cycle times for products under development. The performance of single-chip DSPs is increasing rapidly, and floating-point arithmetic will greatly decrease execution time, reduce design time, eliminate tedious scaling procedures, and make possible sophisticated control algorithms not easily coded into fixed-point processors.

Practical Servomechanical System Design Using Modern Control Tools

Advanced developers and product developers in Digital's Storage Systems Engineering groups are making extensive use of modern control design, simulation and analysis tools. MATRIXx/WS and DOE-MACSYMA have been introduced and enhanced by the Storage Systems Servo-Mechanical Advanced Development Group for Storage Systems' needs. An important part of drive technology advancement derives from the development of new computer design tools, and this is an important part of the advanced development focus.

DOE-MACSYMA is a symbolic math package originally developed at M.I.T. and now ported to VAX LISP by Paradigm Associates, Inc. It has found application

in control design, circuit analysis, and mechanical and magnetic design throughout Storage Systems Engineering. The package provides symbolic mathematical and graphical solutions to many engineering problems and is being used to form parametric state-space models from linear differential equations of physical plants.

MATRIXx/WS is a sophisticated control and signal processing package derived from Cleve Moler's MATLAB and licensed by Integrated Systems, Inc. The VAXstation version not only has facilities for state-space control design, but also for the simulation of dynamic systems which may be concurrently continuous, discrete-time, nonlinear, and time-varying. Figure 10 illustrates a very simple disk drive model as it is displayed on a workstation.

For rapid servomechanical control system design of plants containing resonant dynamics, it is vitally important to develop accurate models of the actuating system to be controlled. In this way, design can move from the laboratory to the workstation and proceed at a rapid pace. To facilitate this capability, interfaces between VAXstation-based MATRIXx and laboratory equipment such as Hewlett-Packard and Schlumberger frequency response analyzers[7] as well as other computer-aided engineering packages such as ANSYS have been developed at Digital.

References

1. M. Sidman, "Digital Servo Design for Rigid Disk Drives," Short course presented at Digital Equipment Corporation, 1987, 1988. Course notes.
2. M. Sidman, "Adaptive Misposition Correcting Method and Apparatus for Magnetic Disk Servo System," U.S. Patent Number 4,536,809 (August 20, 1985).
3. G. Franklin and J. Powell, *Digital Control of Dynamic Systems* (Reading: Addison-Wesley, 1980): 39.
4. M. Sidman, "Adaptive Runout Correction System for Disk Drives," *Proceedings of the Second ASME International Symposium on Robotics and Manufacturing Research* (1988).
5. A. Oppenheim and R. Schafer, *Digital Signal Processing* (Englewood Cliffs: Prentice-Hall, 1975): 105–109.
6. M. Tomizuka et al., "Discrete-Time Domain Analysis and Synthesis of Repetitive Controllers," *Proceedings of the American Control Conference* (1988).
7. F. DeAngelis, "Identification of Electromechanical Plants from Frequency Response Measurements," Master's thesis, M.I.T., 1988.

Alan B. Smith

Magnetic Domain Observations in Thin-film Heads Using Kerr Microscopy

In any thin-film recording head, the pole structure must be an effective conductor of magnetic flux. To achieve this goal, the magnetic domain configuration in the pole is critical. Engineers developing new head designs must determine how these domains are affected by changes in structure, materials, and processing. A technique, called Kerr microscopy, can be used to actually make these domains visible. The technique makes use of the rotation of polarized light which occurs upon reflection from the surface of a magnetized material. The rotation is very small, approximately 0.01 degrees, but can produce a contrast that makes domains visible if the proper equipment is used. In addition to a very high quality polarizing microscope, a sophisticated video image-enhancement system is essential. Digital's Magnetic Recording Head Development Group has constructed such a Kerr microscope setup and has found it to be a great aid in developing thin-film heads.

In many applications of magnetic materials, it is extremely important to know the detailed magnetization distribution on a microscopic basis. One would actually like to make visible any in the direction of magnetization. Until recently, the only practical way to achieve this goal was to coat the sample with "ferrofluid," a liquid containing small magnetic particles. This technique has various disadvantages, the most important of which is that the surface of the sample is contaminated by the liquid. Recently, another technique, Kerr microscopy, has been proposed which avoids all the major disadvantages of using ferrofluid.[1,2] We describe here the Kerr microscope constructed by the Magnetic Recording Head Development Group. As will be explained, the data that can be obtained with this equipment is extremely useful in the design and fabrication of heads for disk drives.

Thin-film Heads

In the RA90 disk drive, data is recorded and read using a thin-film recording head. These heads are fabricated using techniques similar to those used in the semiconductor industry. The result is a head having the small dimensions necessary to achieve the high data densities of the RA90 disk drive. For a further description of this head, see reference 3. For our purposes here, it is sufficient to consider only the main features of the head as shown in Figure 1.

As the figure shows, the head consists of magnetic "poles" that surround a coil. The disk (not shown in this drawing) is adjacent to the pole tips. During read, the flux emanating from the recorded data on

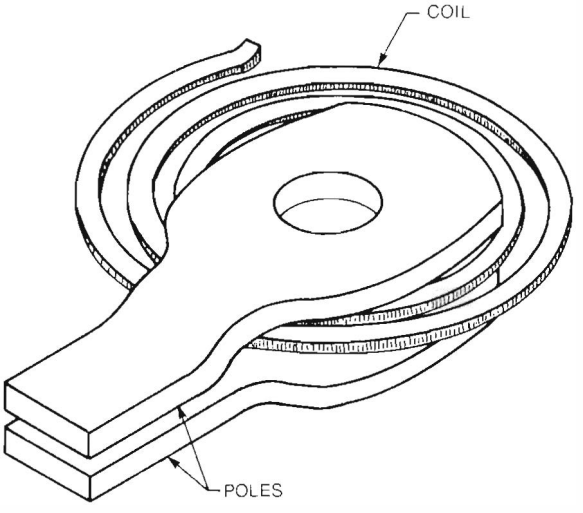


Figure 1 Basic Structure of a Thin-film Head

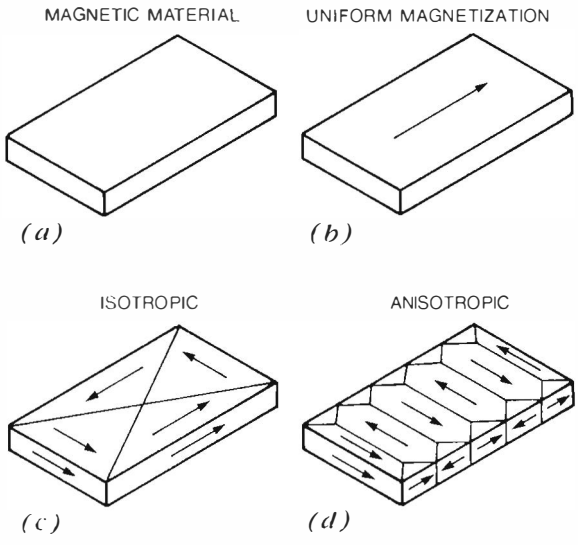


Figure 2 Examples of Domain Configurations in Thin, Rectangular Bars of Magnetic Material

the disk is picked up by the pole tips. This flux is conducted to the coil, producing an output voltage. For this read process to have the required efficiency, the whole magnetic structure must be a good conductor of magnetic flux. To achieve this goal, we require a detailed knowledge of the magnetic state of the poles. Our Kerr microscope provides the means for obtaining this data.

Domain Theory

To discuss the state of magnetization in a thin-film head, it is first necessary to talk about magnetization processes in general. Let us first consider a thin rectangular piece of magnetic material (Figure 2a). If this is a "soft" (i.e., easily magnetized) material, what will be its magnetized state in the absence of an applied field? The answer to this question is not as simple as it might first appear. The material will not be uniformly magnetized as indicated in Figure 2b. If it were, the material would behave like a small permanent magnet with a north pole on one end and a south pole on the other. Magnetic flux would leave the north pole and travel to the south pole, filling the surrounding space with magnetic lines of force. A large amount of stored energy would be represented by such a configuration. Therefore, the material will not be uniformly magnetized in this fashion. Instead it will spontaneously break up into areas with different magnetization directions.

If the material is isotropic, i.e., if there is no preferred direction of magnetization, it will take on the configuration shown in Figure 2c. Note that the magnetization goes head-to-tail around the outside edges, north pole to south pole. Hence very little flux exists outside, and the stored energy is low.

Often the material itself has a preferred direction of magnetization. (We shall later see why this might be a desirable situation.) In these cases, the magnetization pattern is distorted to favor that direction. An example of such an anisotropic material is shown in Figure 2d.

The differently directed magnetized regions in Figures 2c and 2d are called magnetic domains. The boundaries between these regions are called domain walls. (For further information on domain theory, the reader is referred to reference 4.)

In a disk drive, the head must respond to rapid changes in the flux received from the disk. These flux changes must be transmitted through the poles to the coil. To transmit these changes, the component of magnetization in the direction of the pole axis must change. Because the domain walls cannot move very rapidly, the required magnetization change cannot come from wall motion. The change must come, therefore, from rotation of the magnetization. Hence domains with magnetization already fully aligned along the axis are not of any use for this purpose. Only domains with magnetization transverse to the axis can contribute to flux conduction.

Bearing in mind this need for transversely directed magnetization, let us look at the domains in the pole tips of a thin-film head. Using the results from the previous figure, we have sketched these domains in Figure 3. The first part of this figure shows the configuration of a magnetically isotropic material; 3b shows the anisotropic case. The configuration of Figure 3b is seen to be much more desirable than 3a. Under the influence of the flux from the disk, the magnetization in Figure 3b can rotate as shown in Figure 3c.

It is thus very important to ensure that the head has a magnetic anisotropy that makes it resemble Figure 3b rather than 3a. The Kerr microscope can provide this vital information. Two examples of Kerr microscope pictures are shown in Figure 4. These examples are discussed further in the section Results; they are presented here to illustrate the pole tip domains. Figure 4a corresponds to 3a, and 4b to 3b.

The Kerr Effect

Figure 5 shows the basic effect upon which the Kerr microscope is based. In this figure a beam of polarized light is incident on a magnetic material. The magnetization direction is indicated by the vector M, and the polarization direction by the vector E. The

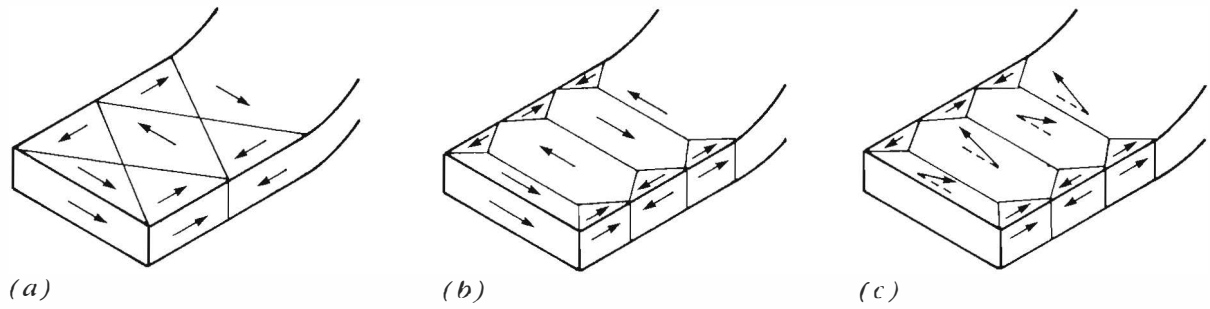


Figure 3 Domain Configurations in Pole Tip of Thin-film Head

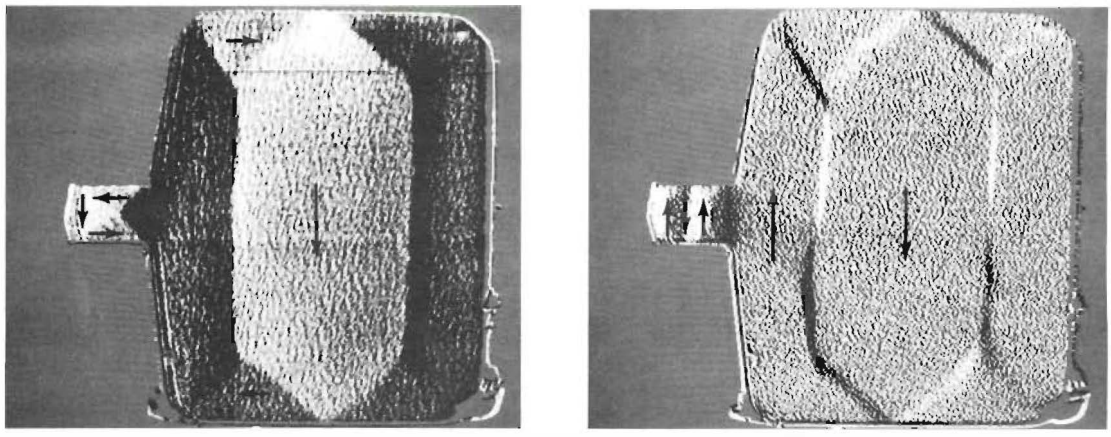


Figure 4 Kerr Micrographs of Thin-film Pole Tip Domain Structures (a) Conducts Flux Efficiently (b) Does Not Conduct Flux Efficiently

reflected beam has substantially the same polarization as the incident beam. However, there is also a small component (marked "Kerr") perpendicular to the main direction of polarization. This small component only exists with magnetic materials, and its direction depends on the direction of magnetization. The presence of this component is the "Kerr effect" discovered by John Kerr in 1888. Actually, there are several different manifestations of the Kerr effect, differing in the relative directions of the incident light and the magnetization.[5] We have shown in Figure 5 the "longitudinal" Kerr effect, which is the one most often used for Kerr microscopy.

We can now see how the Kerr effect makes possible the visualization of magnetic domains. The net polarization direction of the reflected beam in Figure 5 is the resultant of the main and Kerr polarizations. The direction is thus very slightly rotated (about 0.01 degree) from the incident polarization direction. If the magnetization direction is reversed, the direction of the rotation is reversed. Therefore, by placing a polarizer in the reflected beam, we can obtain different intensities depending on the direction of the magnetization. Thus the Kerr effect provides the basis for producing images in which differently magnetized domains have different intensities. The resultant contrast between the domains makes them visible (as in Figure 4, for example).

To produce pictures such as those in Figure 4 we need a setup such as the one in Figure 6. This figure shows the basic parts of a polarizing microscope. In one important respect, however, this microscope is used differently from the way a polarizing microscope is normally used. For Kerr observations, the incident light must hit the objective off center. This arrangement ensures that the light reaches the sam-

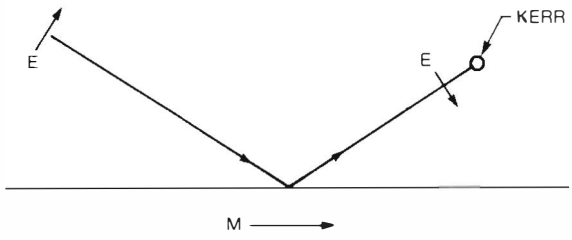


Figure 5 The Basic Kerr Effect

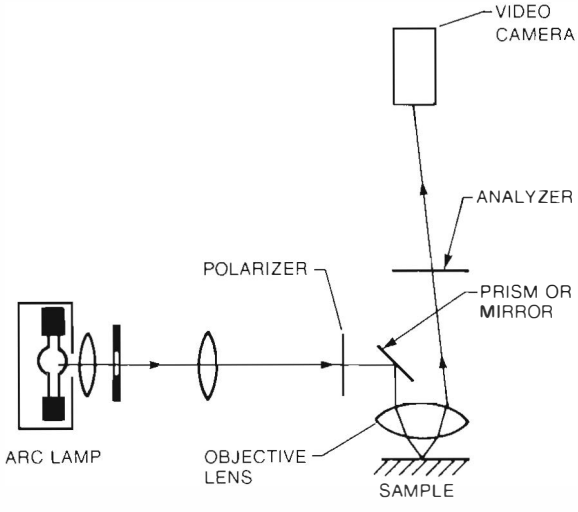


Figure 6 Basic Elements of Kerr Microscope

ple at an angle, as in Figure 5. (There is no Kerr effect if the incident beam meets the sample at a right angle.)

Note that Figure 6 shows a video camera where one would normally expect an eyepiece. The camera is used to view the Kerr-induced contrast between domains. Because the Kerr effect is so small, this contrast is very slight. In most cases the contrast cannot even be detected by the human eye. Sophisticated image processing is necessary to make the domains visible.

Image Processing

As stated above, the Kerr-induced contrast between domains is very weak. In fact, the contrast is much weaker than the normal sample features, such as surface graininess. The method by which this domain contrast is made visible is described in this section by reference to Figure 7.

Figure 7a depicts the picture on the video camera. In this illustration, we assume that a domain wall runs down the middle of the picture; i.e., the left side of the figure is magnetized up, and the right side is magnetized down. Because of the Kerr-induced contrast, the left side of the picture is darker than the right. (For the purposes of illustration, we have made the contrast much greater than it would normally be. As described above, the contrast would normally be so weak as to be undetectable by the eye.) In Figure 7a we have also included some little lines and spots to indicate the minor surface imperfections that are always present. The first step in enhancing the contrast is to increase the video gain as much as possible, i.e., do the equivalent of turning up the contrast on a television set. The amount of contrast enhancement that can be achieved in this manner is limited however. If too much gain is used, the relatively high-contrast surface features will overload the video amplifiers, obscuring the domain contrast. Hence we must turn to sophisticated digital processing techniques to achieve most of the required contrast enhancement.

The first step in processing is to digitize the image. The image is broken up into picture elements, or pixels. In our system, there are 640 pixels horizontally by 483 vertically, making a total of 309,120 pixels. Each pixel is assigned a number to indicate how light or dark that particular location is in the picture. Thus any shade from darkest black to lightest white can be represented by a number. This "gray scale" is divided into 256 levels, as indicated in Figure 7b.

Before the contrast can be enhanced digitally, another important digital processing step must be performed. We must remove the previously mentioned surface features from the picture. Until this is done, we have no hope of seeing the weak contrast between the domains. The surface features can be removed by the following subtraction process. First the picture in Figure 7a is digitized and stored in

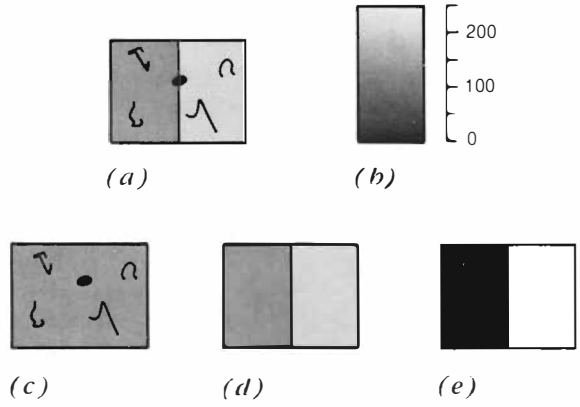


Figure 7 Television Image at Various Stages of Enhancement Process

memory. Then a magnetic field is applied to the sample, causing it to become uniformly magnetized. The resultant picture (Figure 7c) is also stored in memory. Next the corresponding pixels in Figure 7c are subtracted from those in 7a (and 100 is added to each resultant pixel to bring the result to the center of the gray scale). This operation produces the picture shown in Figure 7d. Note that the surface features are now absent. Because they were common to both of the original pictures, they were eliminated by the subtraction process. Figure 7d is therefore free of any contrast other than the desired domain contrast. Thus the image is now ready for digital contrast enhancement.

Figure 7d contains only shades of medium gray. Typically, every pixel in the picture has gray-scale values lying between 95 and 105. To enhance the contrast, we must expand this range of 10 proportionally until it fills the entire range from 0 to 255. So 95 becomes 0, 96 becomes 26, 97 becomes 51, and so on until 105 becomes 255. The result is the high-contrast image shown in Figure 7e. Here the domain configuration can readily be seen because all the area magnetized up appears black, whereas that magnetized in the opposite direction is white. The boundary between these areas is the domain wall.

A slightly different method of doing the subtraction leads to a somewhat different display. In Figure 7c the sample is completely saturated by a relatively large applied field. Suppose instead that we applied a small field that only moved the wall a little. The subtraction and enhancement process would then yield the result shown at the right of Figure 8a. We now have a white stripe on a black background, the stripe indicating the approximate position of the domain wall. If the polarity of the applied field is reversed, the domain wall moves in the opposite direction, as shown in Figure 8b. The resultant image is the negative of 8a. We will see examples of displays such as Figures 8a and 8b in the next section.

An important step in the image processing has not yet been mentioned. Because of the low light levels and the high video gains we use, some noise is introduced in the images we obtain. This noise looks like the "snow" seen on a television screen when the signal is weak. We can reduce this noise substantially by digitally averaging successive images before doing any of the other processing. In other words, we do not use a single video frame for Figures 7a or 7c. We take from 4 to 128 frames and digitally average them to produce each image. Since 30 frames are generated every second, this averaging process does not appreciably slow down the acquisition of data.

To accomplish the image processing described above, we utilize a Hamamatsu C1966 image processor under the control of a MicroVAX system. The MicroVAX system is programmed to initiate all the image processing functions as well as to switch the applied magnetic field at the appropriate time. Having everything controlled by the MicroVAX system is more than a convenience; it is essential because of the image subtraction process described above. For this process to be effective, the images in Figures 7a and 7c must be absolutely identical (except, of course, for the domains). Unavoidable mechanical drift in the microscope will cause the images to differ unless one is acquired very soon after the other. A delay of only a few seconds can cause problems. By using the MicroVAX computer rather than manual control, we ensure that the minimum time elapses and good images are obtained.

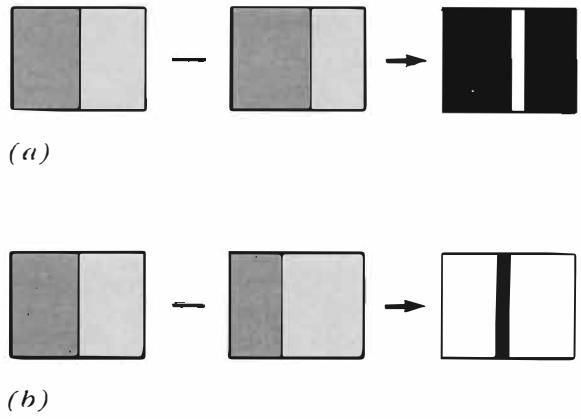
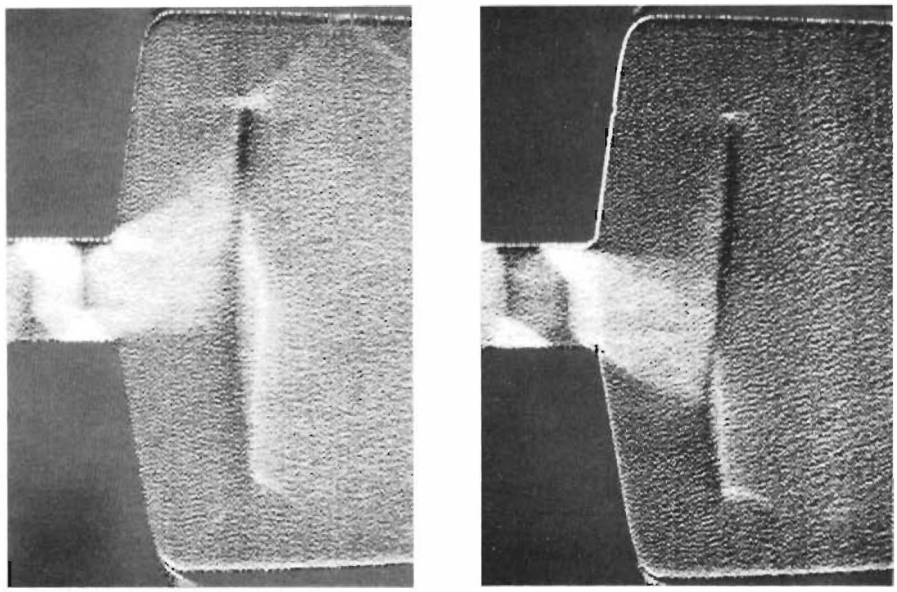


Figure 8 Image Obtained with an Applied Field Less than that Required to Saturate Sample

Results

We have already shown two Kerr images (Figure 4) as examples of domains in the pole tips of thin-film heads. We can now look at these pictures again in light of our discussion of image processing. They illustrate the two modes of presenting the data, i.e., Figure 7e versus Figure 8. The pole tip region in both Figures 4b and 4a are in the mode of Figure 7e. However the situation in the wide part of the pole is different in Figure 4a. The applied field is not sufficient to saturate in this case. The result is that we see the domain walls highlighted as in Figure 8. In Figure 4a a higher field was used, and we see a display like that of Figure 7e. Both methods of display show us the domain configuration, and both can provide the vital domain information needed in the design and manufacture of thin-film heads.

As indicated by the arrows in Figure 4, white and black indicate magnetization transverse to the pole



(a) (b)

Figure 9 Kerr Micrograph Illustrating "Flux-beaming" Effect

(i.e., vertical in the figure). Gray indicates magnetization along the pole axis. If the sample is rotated 90 degrees with respect to the incident light, this situation is reversed. Then axial magnetization shows up as white or black. This situation makes it easy to see axial components of magnetization and reveals some very interesting behavior where the pole tip joins the wider part of the pole. It turns out that the flux being conducted from the tip does not spread out immediately to fill the wide part of the pole. Instead the flux travels in a "beam" at an angle to the axis. This effect may be clearly seen in Figure 9a. Reversing the field direction (Figure 9b) causes the beam to switch to the other side of the axis. This flux beaming effect has important consequences for the design of thin-film heads and is the subject of a recent paper.[6]

One final example of Kerr microscopy is provided by Figure 10. As in Figure 9, the setup is such that magnetization along the pole axis produces white and black contrast. The area of interest in this picture is circled. The line represents a domain wall. The fact that it changes from white to black contrast means that the magnetization direction changes. In domain theory the wall is said to have a "Bloch point." The presence of a Bloch point is of scientific interest and may have some effect on how the walls behave with high frequency excitation.[7]

Conclusions

Kerr microscopy provides data that is extremely useful in the design and manufacture of thin-film heads. This new technique is much more convenient to use and gives better images than the alternative method (using ferrofluid).

Acknowledgments

We would like to acknowledge Alex Hubert of the University of Erlangen-Nurnberg for useful discussions on Kerr microscopy. We would like to thank

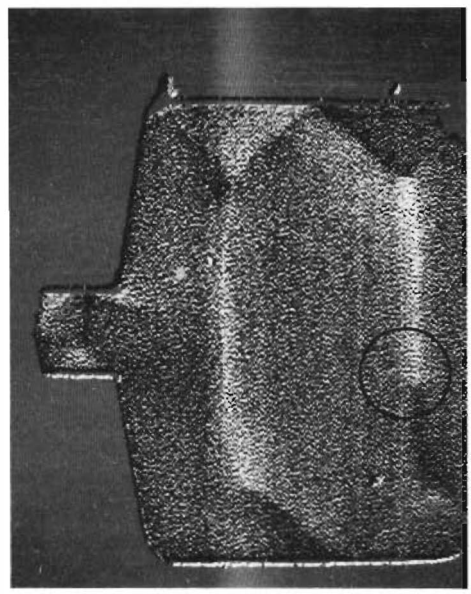


Figure 10 Kerr Micrograph Showing Domain Wall Containing a Bloch Point

Warren Goller for expert technical assistance in setting up and operating the Kerr microscope and taking the pictures presented in this paper. We are also grateful to the Digital Technical Systems Group and Paul Tesini for skillful programming of the MicroVAX system that controls the image processor.

References

1. F. Schmidt, W. Rave, and A. Hubert, "Enhancement of Magneto-optical Domain Observation by Digital Image Processing," *IEEE Transactions MAG-21* (1985): 1596–1598.

2. B. Argyle et al., "Optical Imaging of Magnetic Domains in Motion," *Journal of Applied Physics* 61 (1987): 4303–4306.

3. B. Crane, "Disk Drive Technology Improvements in the RA90," *Digital Technical Journal* (February 1989, this issue): 46–60.

4. B. Cullity, *Introduction to Magnetic Materials* (Reading: Addison-Wesley, 1972).

5. A. Hubert, *Magnetic Domains* (Heidelberg: Springer-Verlag, forthcoming).

6. M. Mallary and A. Smith, "Conduction of Flux at High Frequencies by a Charge-Free Magnetization Distribution," 1988 Joint Intermag-3M Conference, to be published in *IEEE Transactions MAG-24*.

7. B. Argyle, B. Petek, M. Re, F. Suits, and D. Herman, "Bloch Line Influence on Wall Motion Response in Thin-film Heads," *Journal of Applied Physics* 63 (1988): 4033–4035.

Reinhard Kretschmer
Siegbert Sadowski

Margin Analysis on Magnetic Disk Recording Channels

The recording channel of every magnetic disk drive comprises the heads, media, and the read/write electronics. Digital data is encoded into phase information on the media and is therefore subject to phase shift, an effect that significantly limits high-density recording. To reduce the raw error rate that results from phase shift and thereby increase disk drive reliability, Digital's engineers at Kaufbeuren, West Germany, developed a margin analysis system operating at channel level. This system provides direct information about actual disk drive performance. The efficient phase distribution analysis system was designed to be fast, comprehensive, and nondestructive to existing disk data. Statistical methods are applied to analyze the results.

Phase shift, also referred to as bit shift or peak shift, is the unwanted phenomenon of a shift of the readback phase from its ideal position. Errors due to phase shift are common to all disk drives. Solving the problem requires that we address the difficult problem of identifying which part of the drive system is responsible when error rates exceed acceptable limits. As part of the solution, Digital's engineers in Kaufbeuren, West Germany, developed the Phase Distribution Analyzer (PDA). The PDA is a fast and time-saving system for production that detects and analyzes potential failures in the magnetic recording channel long before problems occur.

This paper discusses the PDA and the margin analysis techniques used in the design and manufacture of Digital's disk drives. These analysis techniques serve to enhance drive reliability and data integrity.

Technical Background

In a disk drive there are several sources of phase shift, such as the heads/media, the read/write channel, and mechanical positioning system. Especially dominant are noise sources introduced by the magnetic media, the read/write head, and the preamplifier. (See Figure 1.) First, media noise is caused both by the finite volume density of magnetic particles in one bit cell and, more significantly, by the particles' nonuniform dispersion. Second, head impedance noise is in principle a kind of thermal noise. Finally, the decrease in the signal-to-noise (S/N) ratio caused by the preamplifier can be attributed to thermal and shot noise.[1]

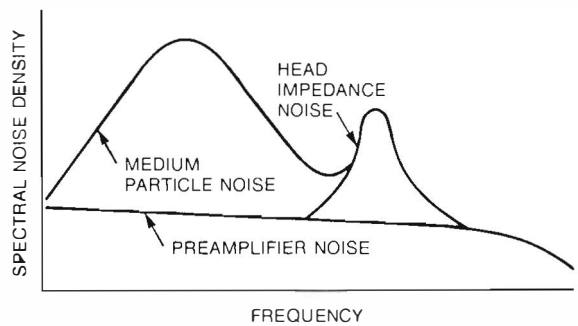


Figure 1 Noise Spectrum

Another major source of phase shift is the interference of the readback pulses through a combination of various effects. The electromagnetic interference of the bit cells distorts signals in phase and amplitude, as illustrated in Figure 2. The solid lines in this figure represent two pulses as if they were isolated; the dotted line shows the result of their superposition. Although sophisticated modulation coding is used to avoid this interference, the trade-off between desired bit density and tolerated interference still leaves us with pattern-dependent phase shifts. By expanding the bit patterns into Fourier series, we can understand their phase shift dependence as group delay distortions.

The signal conditioning process in the read channel is another contributor to phase shift. The analog filter and amplifier stages that compensate for the readback signal deviations may produce imperfections,

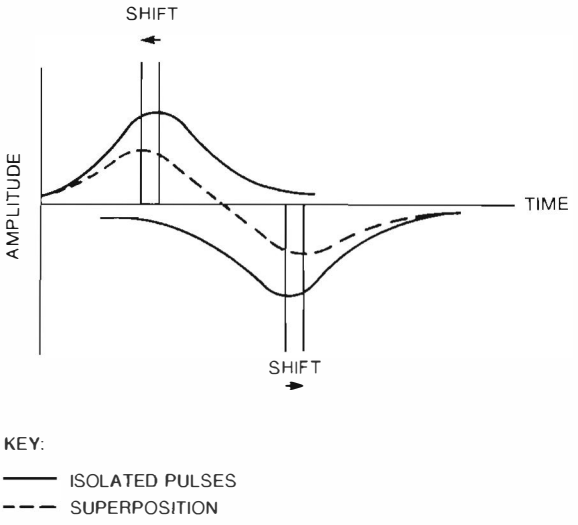


Figure 2 Phase Shift Diagram

such as improper pulse slimming or group delay distortion. In the qualifier that digitizes the pulses, the exact timing and thus the phase of a detected peak is influenced by its amplitude. The dynamics of the synchronizing phase-locked loop (PLL) are also critical aspects, for example, the PLL's tendency to lose lock towards decreasing S/N ratios. Imperfect overwrite, adjacent track pickup, and off-track error are additional sources for phase shift.

Phase margin analysis identifies the net result of all these influences and their impact on drive performance. To perform this analysis, we measure the proximity of the individual bit to the error limit and thus determine the error margin of the drive. This limit is represented by the boundaries of the data window which in turn is generated out of the data by the PLL. As long as a particular bit is in the proper data window, the bit can be synchronized to the clock, and no error is noted. When the data event crosses the boundary into the next window, the read decoder decodes false data. A read error immediately results. As a result of the fixed pass/fail criterion, the system as a whole appears to operate properly, although its performance is deteriorating. When a small variation starts moving bits out of the window, the error rate suddenly avalanches. Phase margin analysis takes into account the fluctuations of the bit position. As part of the analysis, we take several measurements of the individual location of each bit relative to the fixed time window. The result is a statistical distribution showing the measured system's safety margin, or phase margin, against the error limit.

Figure 3 shows schematically two different examples of such distributions. The shaded areas beyond the window limit represent the different actual error rates. Comparing now the two distributions at the same error rate (i.e., equal areas) yields different phase shift values. Phase margin is then defined as the difference between the window width and the phase shift at a specified and fixed error rate. This quantity is generally accepted as a measure of reliability.

An efficient and economical phase margin test system has to fulfill several requirements. The system must be able to

- Take high-speed measurements, evaluating every individual drive in a short time
- Analyze all flux changes in a given record/track, as well as every existing data pattern
- Perform all operations without destroying disk data or format
- Take measurements from the data separator's point of view
- Enable the user to analyze any physical location on the disk (cylinder, track, sector)

The development of the PDA system relative to these requirements is described in the following sections.

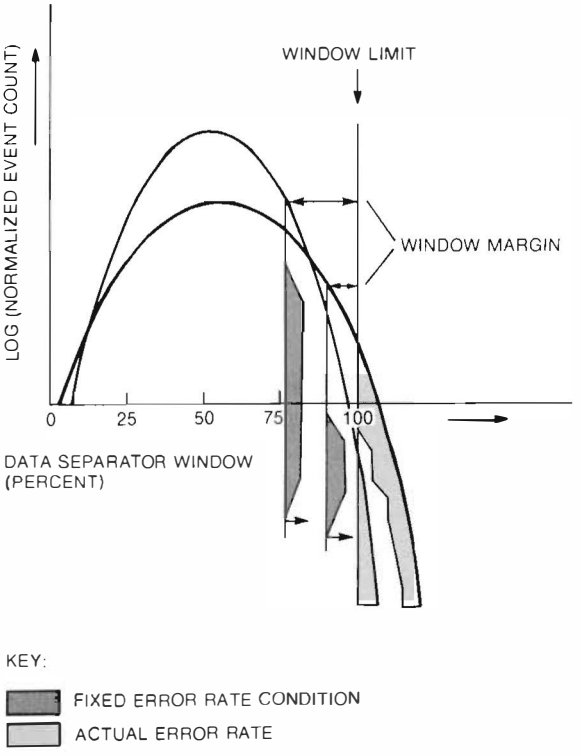


Figure 3 Phase Distribution Curve

Phase Distribution Measurement System

Phase margin analysis can be effected with a variety of architectures, some of which are commercially available.[2] However, these architectures did not meet several of our requirements. Notably, the commercially available equipment lacked sufficient measurement speed or destroyed disk data. Therefore, we decided to develop our own PDA.

The PDA is a high-precision, time-interval analyzer that measures the positions of individual bits relative to the beginning of the data window. As shown in Figure 4, the system consists of high-speed data acquisition hardware which connects by means of an IEEE interface with a customized set of commands to a MicroVAX system. The host controls the drive under test, for example, positioning the drive to the user-specified physical location. The host also controls the data acquisition hardware and processes the data. Test results can be sent over the host's local area network (LAN) interface, DECnet software, to larger VAX computers for more detailed statistical analysis.

When the data separator in a disk drive classifies readback pulses into data windows, it synchronizes data to the drive system clock. The phase margin at this point is most relevant to the performance of the magnetic recording channel. To address all potential error sources in the magnetic recording channel, the PDA measures the bit distribution in a normal production disk drive at this point of synchronization (i.e., phase shift from the data separator's view). The positions of the individual bits relative to the beginning of the data window are proportional to the phase angle of clock to data. The PDA divides the data window into 64 bit positions (bins) and categorizes every data event (flux change) into one of these positions. The resolution of every bin is approximately 600 picoseconds. Every bit position has a counter assigned that counts the number of bits detected at that position. Figure 5 illustrates the resolution and categorization of the data event. The capacity of each counter is 2^{18} data events. From the measurement of a large number of bits and the accumulation of their bit positions, a phase distribution function, or histogram, is produced. The analysis of these measurements is discussed in greater detail in the section Methodology for Data Evaluation.

To achieve accurate results, the PDA must analyze a high number of bits. Moreover, to measure every bit and keep test time within acceptable limits, the analyzer must absolutely take measurements at a very high speed. A short test time ensures that the margin analysis system is cost effective in the manufacturing environment. With the maximum measurement rate of 35 megasamples per second and the high bin-counter capacity, the PDA system is able to measure the phase histogram of all bits in a track within one disk revolution. All RA-family disk drives, for example, can be analyzed with the PDA.

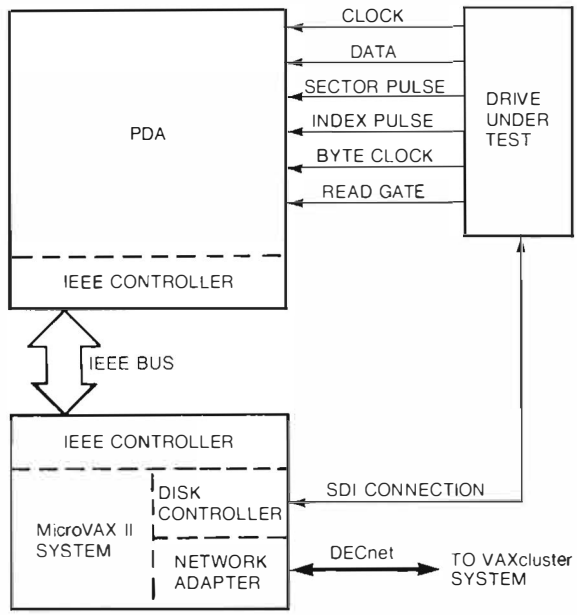


Figure 4 Phase Distribution Measurement System

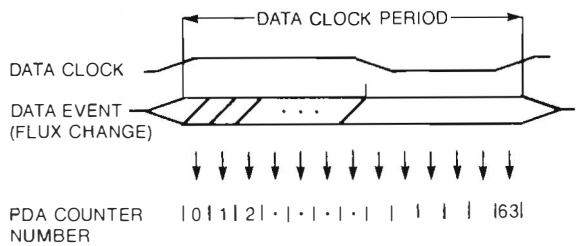


Figure 5 Data Window

In addition, the histograms of multiple measurement cycles can be averaged. In this mode, the PDA also calculates maximum and minimum event count histograms. Thus even a single bit out of millions falling out of the data window will be recognized, which allows the analysis of intermittent failures in the magnetic recording channel. Unlike other phase margin analyzers, the PDA does not need to write test patterns to the drive under test. Therefore the user can analyze any existing data pattern on the disk. This ability is especially suitable for the analysis of time-dependent parameter changes. Such

changes may occur over a longer period of product run-time as a result of degradation of the heads/media interface.

Methodology for Data Evaluation

As described above, phase distribution measurements determine the location of every single bit relative to a fixed time window. The time scale is partitioned into k bins, and the PDA system just counts the bits that fall into each time-bin.

Let $x_1, x_2, \ldots, x_k$ denote the midpoints of the time-bins, and $N_1, N_2, \ldots, N_k$ the corresponding bit counts. Graphically represented, this is a histogram showing the frequency distribution of the bits in the time window.

The next step is to extract the relevant information about the performance of magnetic recording systems. Two important performance measures are

- The error rate, which is the probability that a bit will be found outside the proper window
- The phase margin, which is the safe distance between the window width and the phase distribution curve at a given error rate

Due to the very low error rate, these figures are not directly available from the measured distribution. A suitable model is therefore used to extrapolate the measurements. Assume the bit positions are generated randomly according to a probability density function $f(x)$. The shape of this function is influenced by the pattern written on the disk and by several components, as discussed in the section Technical Background. Noise is also present, which can be modeled as a Gaussian distribution.[3] Assuming that several effects superimpose each other, $f(x)$ is expanded into several Gaussian densities $g_i(x)$.

Let a_i be the contribution of $g_i(x)$ to $f(x)$ thus

$$f(x) = \sum_i a_i g_i(x)$$

with

$$0 < a_i < 1 \text{ and } \sum_i a_i = 1.$$

The estimation of the model parameters and hence the determination of that function $f(x)$ which best fits the (standardized) histogram data are achieved with a minimization program: Find Gaussian density parameters and contributions a_i such that

$$\sum_{j=1}^{k} (N_j/N - f(x_j))^2/\Delta N_j = \text{minimal}$$

where ΔN_j specifies the measured standard error of N_j and

$$N = \sum_j^k N_j.$$

This method is known in physics under the notation *Chi-square-fit* and is equivalent to the usual *weighted least squares regression*. The method is also asymptotically equivalent to the well-known statistical method of maximizing the likelihood function.

The simple model described before is adequate for the general description of the system performance (e.g., error rate). Special care is taken to appropriately represent the tails of the distribution; these extreme values are often caused by spurious effects and determine the system margin, and ultimately, the reliability.

With the knowledge of $f(x)$, the performance measures are easily determined: The actual error rate of the drive is given by

$$\text{actual error rate} = \int_{\text{window limit}}^{\infty} f(x).$$

Since the error margin is the save distance between the window width and the phase distribution curve (compare with Figure 3), it is determined according to

$$\int_{\text{error margin}}^{\text{window limit}} f(x) + \int_{\text{window limit}}^{\infty} f(x) = \text{error rate (spec.)}.$$

The statistical estimation procedure also gives the relative errors of all parameters involved. Hence with error propagation formulae or a Monte Carlo simulation, it is possible to obtain confidence limits for the estimated performance measures.

The mathematical methodology described above reduces the many single measurements into a few significant parameters that give the relevant information. With this kind of extrapolation, we can accurately estimate the margin and the magnetic recording channel drive error rates.

System Integration

As described in the section Phase Distribution Measurement System, the PDA hardware system was designed to handle large amounts of data. For example, it measures about 10^7 data events per measurement cycle. The complete characterization of a typical disk drive requires about 10^3 such cycles. To be a valuable tool, the system must subsequently process the large data samples and perform the sophisticated statistical analysis quickly.

To meet the manufacturing process needs for high-volume throughput and automated testing, the PDA hardware system and the mathematical treatment of the data were integrated into one homogeneous and complete data measurement and analysis system. A menu-driven software package developed

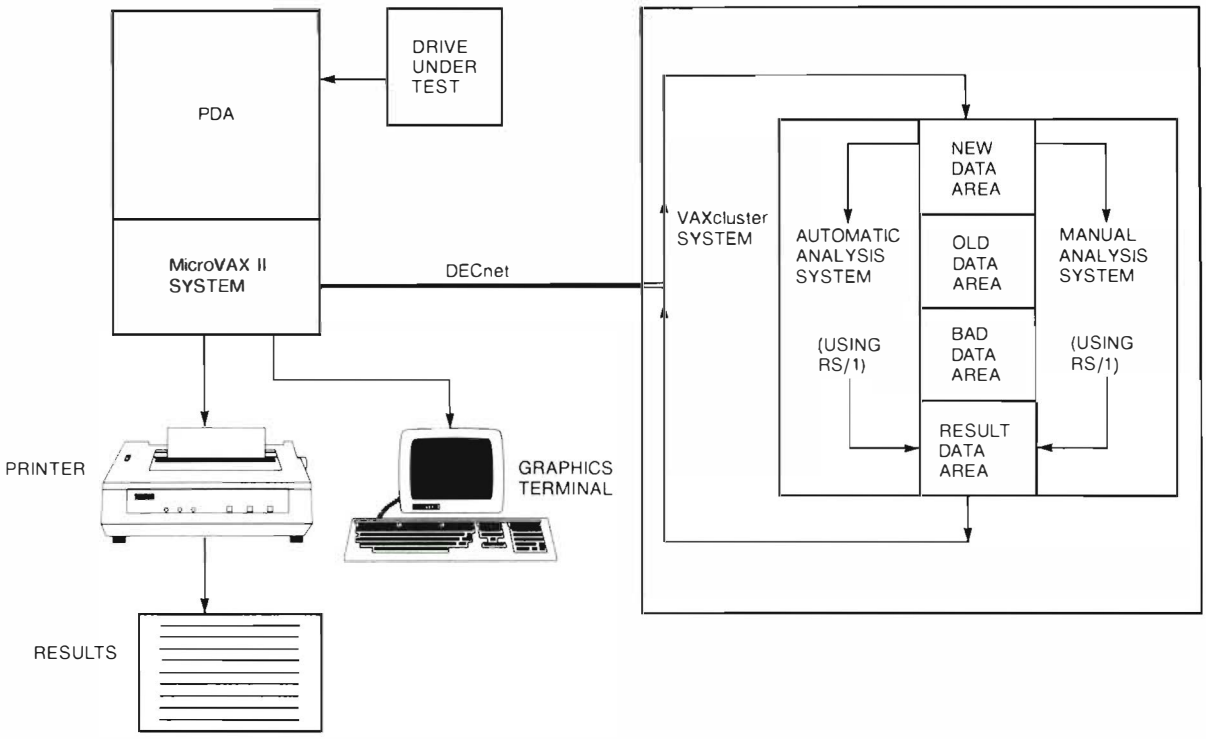


Figure 6 The Automatic and Manual PDA Data Analysis System AMPAS

for this purpose controls the entire data chain from data collection and reduction up to statistical analysis and interpretation. The process is illustrated in Figure 6.

The PDA MicroVAX system creates files in ASCII format. These files are then transferred to a large VAXcluster system where they are converted into RS/1 tables and processed through various RS/Explore procedures. (RS/Explore and RS/1 are designed for data management, analysis, and statistical interpretation.)

The procedures are written in Digital Command Language (DCL), PASCAL, and RPL (a programming language implemented in RS/Explore). RPL was chosen because it provides full access to the statistical and graphical subroutines of RS/Explore. A set of RPL programs performs the curve fitting procedures, limit checking, and error calculations, and generates summary tables.

The benefits of the system to users are as follows:

- The menu-driven design allows automated measurement and data analysis routines to be created for the manufacturing environment.
- The design gives engineers the flexibility to create special engineering tests and to make individual investigations.
- Several users can use the system at the same time.
- The summary tables are stored in a database. The margin history of every individual HDA (head-disk assembly) and magnetic recording channel is maintained for the whole product lifetime.

Range of Application

The useful application of the PDA can be successfully demonstrated in several areas:

- Process improvement, control, and reliability
- Design and component qualification
- Long-term product behavior

This new test tool was integrated in Digital's Kaufbeuren manufacturing process and is now being used as a process control instrument and failure analysis tool. It is also used for specific engineering tasks at engineering sites in Colorado Springs and Kaufbeuren.

Manufacturing in particular derives great benefit from the PDA system. In the manufacture of high-technology products, two of the most important requirements are process stability and total quality control. Although strict tolerances for the assembly

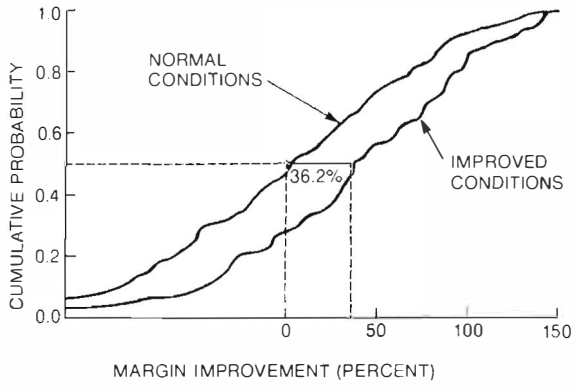


Figure 7 Result of Process Optimization

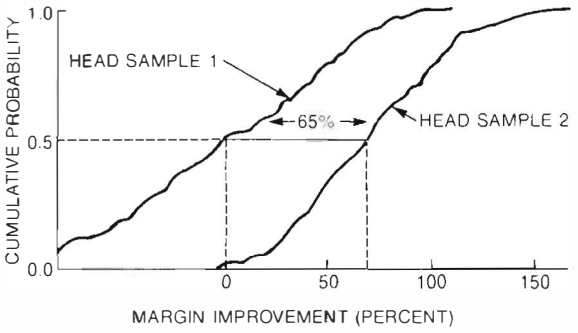


Figure 8 Comparison of Different Head Samples

process are required, the interaction of complex system parameters may induce problems that result in lower product quality and poor long-term manufacturing efficiency. Such problems can occur even if fluctuations are minimal. Therefore knowledge of the impact of critical process steps on product performance is the key to successful process design and control.

A graph showing the PDA contribution to process reliability and improvement is given in Figure 7. The process used in this comparison is servo writing, which is the critical process step in disk drive manufacture. In this process, specific patterns are written on the media; these patterns accurately regulate the positioning and seeking of the read/write heads. Figure 7 compares measurements made before and after tests to improve this critical process step by adjusting write currents and mechanical properties of the servo write tool. The system performance shows significant margin improvement of about 36 percent.

As shown in the Figure 7 example, the PDA can measure and quantify the effect of the servo writing process on system performance. Moreover the PDA provides reliable engineering data within minutes about the success of the optimization. Measuring those effects with traditional test methods is impractical in terms of time and the number of drives needed to accumulate the necessary run-time to check drive error rate differences. The rapid determination of drive performance and error rate is the key to the PDA success. Ensuring an error rate of less than 10^{-11}, for example, would take several days with traditional methods. This compares to a few minutes with PDA margin testing. As a consequence of its speed, this methodology allows certification of every individual disk drive at a relatively low cost.

Another example of the use of the PDA for design and component qualification is the investigation of different read/write heads. To determine the most efficient head for a given disk drive design, different head technologies can be compared, as shown in Figure 8. The data provided by the PDA enables engineers to quantify the differences very quickly and to attribute system performance to the responsible components and their parameters.

Another application for the PDA system is the investigation of performance stability. For this purpose, margin measurements of drives with different loads and run-times are compared for changes over time. Adequate statistical and physical models that simulate the dependencies are then used to extrapolate measurements and predict the reliability performance over time. To ensure that a disk drive is not subject to heads, media, and mechanical degradation, application of the margin methodology is prerequisite, because even very small effects, if they occur, must be detected and quantified. The alternative to this approach is to measure the raw error rate — an experiment that can take several weeks and requires the use of many drives. In conclusion, margin measurements help us to provide sufficient product and process margin over the whole product lifetime.

Summary

Margin analysis at channel level is an efficient method to determine performance and reliability of high-density magnetic recording systems. The high-speed Phase Distribution Analyzer has been extended into a comprehensive test and analysis system for useful application in Digital's manufacturing environment. The margin analysis system — data collection, analysis software, and a sophisticated methodology — makes a significant contribution to data integrity in Digital's disk drives.

Acknowledgment

The Phase Distribution Analysis System is the result of work supported by many individuals in several engineering groups at Digital Equipment International, Kaufbeuren. The authors wish to thank everyone who contributed to the project, in particular Stephan Bolz, Hermann Eiting, Margit Eschbaumer, Dr. Wolfgang Fehse, Hans Grapenthin, Manfred Haug, Dr. Matthias Heiden, Heinz Herbrik, Dr. Hans-Dieter Klein, and Franz Kuess.

References

1. Y. Tahara, Y. Miura, and Y. Ikeda, "Peak Shift Caused by Gaussian Noise in Digital Magnetic Recording," *Electronics and Communications in Japan,* vol. 59-C (10) (1976): 77–86.

2. N. Mackintosh, "A Margin Analyzer for Disk and Tape Drives," *IEEE Transactions on Magnetics,* vol. Mag-17, no. 6 (1981): 3349–3351.

3. E. Katz and T. Campbell, "Effect of Bit Shift on Error Rate in Magnetic Recording," *IEEE Transactions on Magnetics,* vol. Mag-15, no. 3 (1979): 1050–1053.

T.K. Rengarajan
Peter M. Spiro
William A. Wright

High Availability Mechanisms of VAX DBMS Software

VAX DBMS software manages large, complex databases and ensures high system availability. In a VAXcluster environment, DBMS allows concurrent, multiple-node database access. Cooperating nodes and user processes communicate through VMS lock value blocks. DBMS simulates cluster-wide shared memory by using node global sections, shared disks, and the VMS lock manager. Recovery from system failures is coordinated by database monitor processes executing on different nodes. A sophisticated lock protocol enables failed transactions to be recovered on surviving nodes. Long-term database availability is achieved by after-image journaling (AIJ), which enables recovery from media errors. In addition, an on-line backup facility allows concurrent database access during database backup.

VAXcluster systems offer a more available and extensible computer configuration than standalone computers. Because individual computers can join and leave a VAXcluster system without stopping other cluster members, the system provides high computer availability. Moreover, the computing power of these closely coupled, distributed systems can be easily extended by the addition of standard Digital computers.[1] This combination of high availability and increased CPU power along with the benefits of shared, inter-node access to a set of disks makes VAXcluster systems an excellent computing environment.

VAX DBMS Environment

VAX DBMS software is Digital's CODASYL database management system. It provides complete database services in the centralized VAXcluster environment in a seamless and transparent manner.[2] This environment, illustrated in Figure 1, consists of three basic entities:

- The database itself — A DBMS database consists of a root file and multiple data files, the number and size of which depend on the database design. The root file contains database metadata and transaction control information. Actual user data is stored in the data files. All these files contain specific DBMS formats and are accessed by means of the $QIO system service (which queues an I/O request to a device).[3,4] For the database to be accessible from all nodes in the VAXcluster system, all the database files must reside on cluster-wide disks.
- The database users — A VAX DBMS user is any VMS process executing an application that accesses a DBMS database. The application's executable image includes calls to the VAX DBMS run-time system. As a result, DBMS coordinates the database page buffers and database locks of these individual users in order to maintain data integrity.
- The DBMS monitor processes — On each node, a process, called the DBMS monitor, is started at system startup time. The monitor's major responsibility is detecting failures and initiating the necessary recovery processes on behalf of failed, incomplete transactions.

The Challenges of VAXcluster Systems

The distributed character of VAXcluster systems complicates the implementation of full database management capabilities. Nevertheless, VAX DBMS, facilitated by the VMS lock manager, overcomes these difficulties. For example, DBMS transparently recovers a failed node's outstanding database activity on one of the remaining nodes. Complete data consistency and integrity are assured.

This paper is divided into three major sections. Each describes the mechanisms used by the VAX DBMS software to solve a difficult database management challenge posed by the VAXcluster system.[2] These are

- Sharing data between processes across nodes, a mechanism that is a prerequisite to resolving the following two items

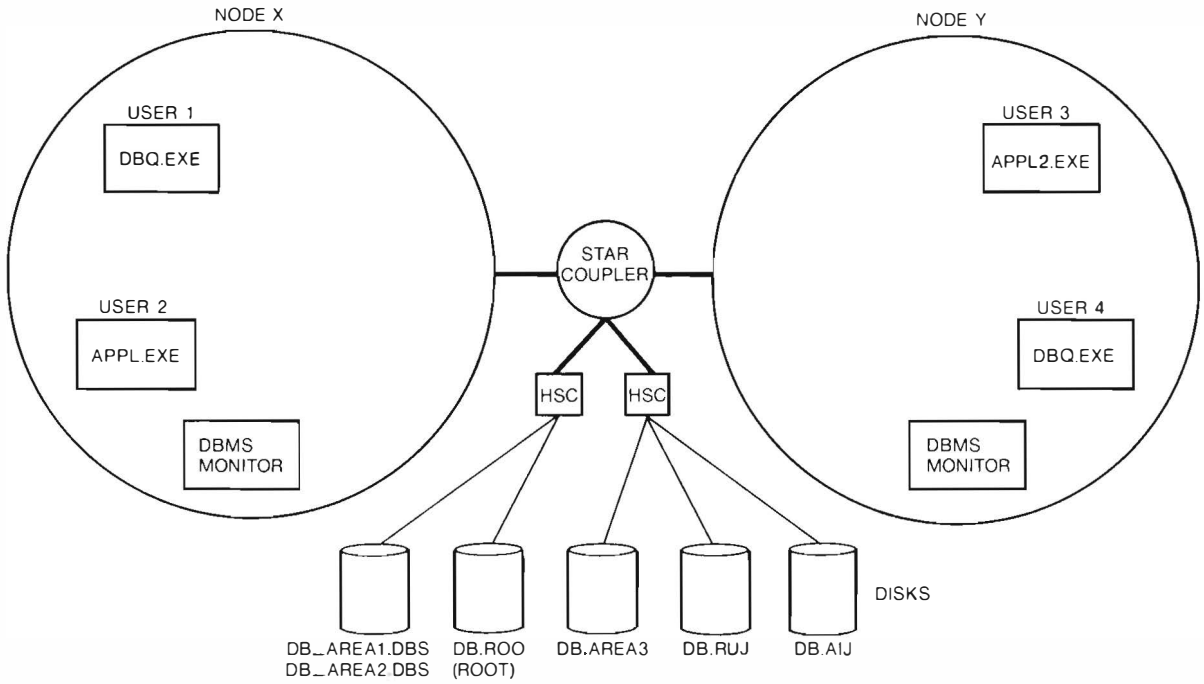


Figure 1 VAX DBMS Environment

- Transparent cluster failover and database recovery, which allow high database availability and data integrity
- After-image journaling (AIJ) and database backup, which ensure high availability in the presence of database media errors

Fulfillment of these requirements is necessary in a fully functional database management system. The mechanisms are implemented in the KODA subsystem of the VAX DBMS product.

Data Sharing in a VAXcluster System

Before VAXcluster systems were introduced, VAXDBMS software used VMS global sections extensively for sharing data. VAX DBMS users shared the contents of the database root file by mapping the file to a global section. Updates to this global section were synchronized by the VMS operating system. Once an update was made, all users immediately saw the updated information.[3]

In a VAXcluster system, however, there is no shared memory between nodes. Therefore global sections alone cannot be used to share data among users on multiple nodes. Instead, DBMS relies on two methods for sharing data between nodes: shared disks and lock value blocks provided by the VMS lock manager. Where possible, DBMS uses lock value blocks.[3,4] However, for large packets of data, shared disk access is used. In the next two sections, we describe each method.

Sharing Disks Using Cluster-wide Global Sections

Without cluster-wide global sections, a simple solution for sharing data would be to always access shared disks. However, this is inefficient in a transaction processing environment. What is needed is a way to optimize the accessing of shared data on disk. The VAX DBMS engineering group devised a technique of caching shared information on each node. This technique utilizes global sections per node, cluster-wide disks, and the VMS lock manager. Figure 2 illustrates this technique.

When the first user on a node attaches to a database, DBMS copies the database root file information to a VMS page-file global section, called the TROOT (temporary ROOT). All users of the same database on one node share the same TROOT global section. Since each node in the cluster can have its own TROOT global section, the problem lies in keeping all these TROOT copies consistent. This is the classic cache coherency problem. Figures 3 and 4 illustrate how DBMS solves this problem.

The coordination is accomplished by the DBMS object manager. The root file is broken up by data structure into objects. Each object has a VMS lock

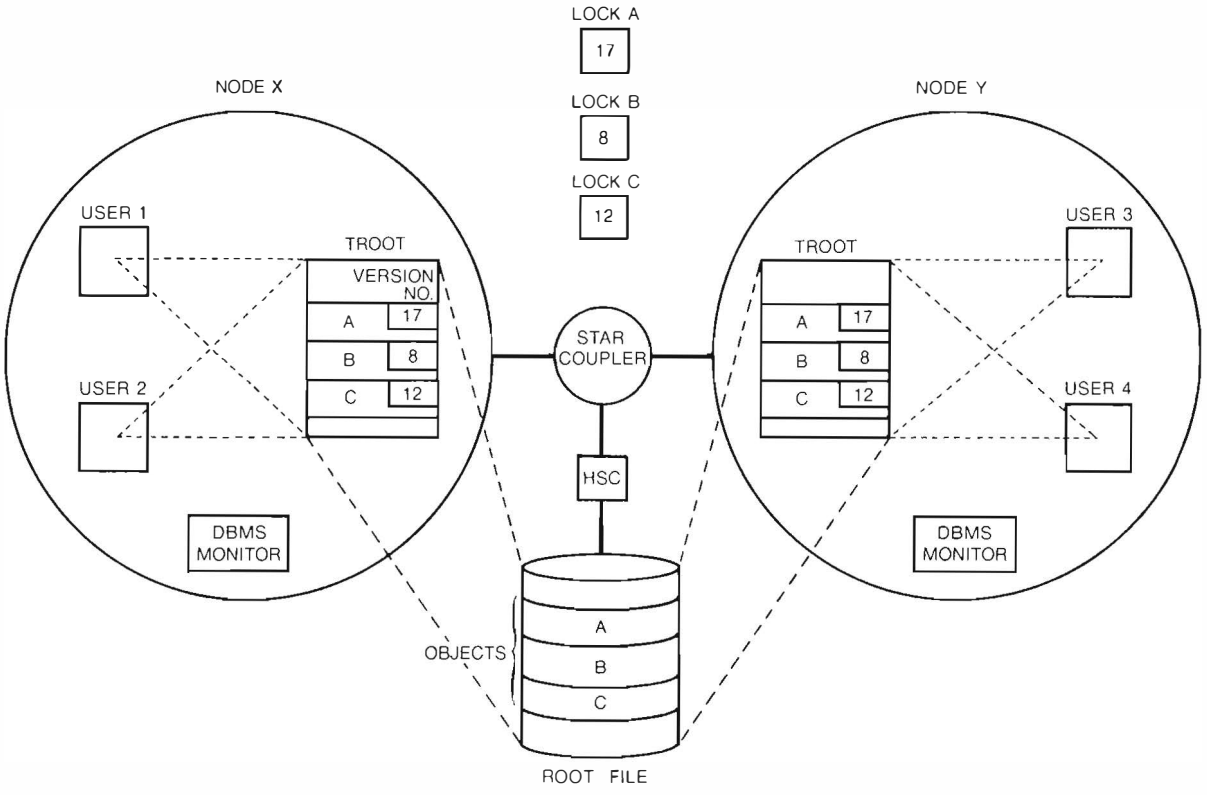


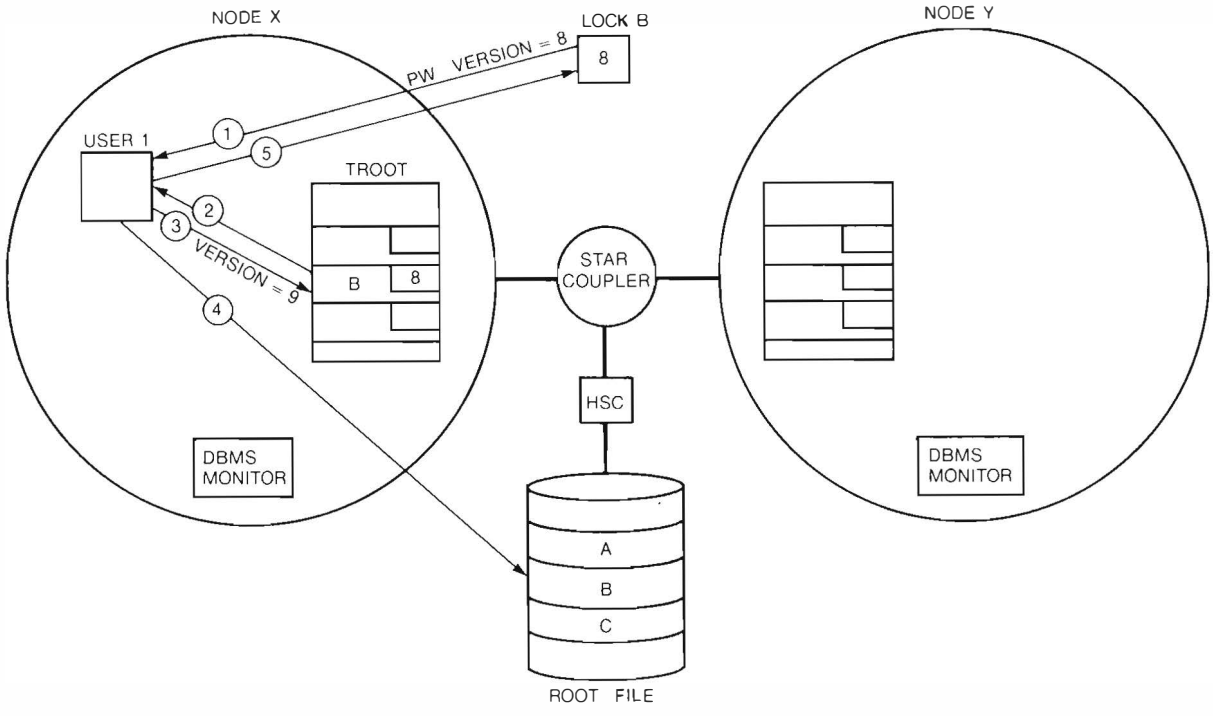
Figure 2 Cluster-wide Global Sections

associated with it. The object manager uses this lock to synchronize the reading and writing of a root file object. The object manager ensures that the latest copy of each object exists in the root file on disk. As a result, the other nodes are always able to access the latest copy of the object from the root file when their copies need to be refreshed. To read an object, the object manager acquires the object's lock in protected read (PR) mode. Its lock value block contains the object's current version number. This version number corresponds to the latest copy of the object in the root file. The TROOT contains a copy of the object and a version number corresponding to that copy. If the versions are the same, then the TROOT object is up to date. If not, the object manager reads the object from the root file into the TROOT. Then, it updates the TROOT object's version number to the version number that is in the lock value block. This indicates the TROOT object is up to date. DBMS demotes the object's lock at this point. Figure 4 illustrates the case of a node refreshing its version of an object from the root file because an update on that object took place.

To update an object, the object manager requests the object's lock in protected write (PW) mode. Once granted, the object manager checks to make sure the TROOT copy on this node is up to date. If not, it reads the new object in from the root file. Then, DBMS makes the changes to its copy of the object in the TROOT and writes these modifications out to the root file on disk. Finally, the object manager increments the object's version number in both the TROOT and the object's lock value block. The lock is then demoted. The updated version number in the lock value block renders the TROOT copies of this object on the other nodes obsolete. Figure 3 illustrates this update operation.

Sharing Data in Lock Value Block

To share small data in a VAXcluster system, VAX DBMS software uses the VMS lock manager and lock value blocks. A 16-byte value block associated with each resource functions as a small piece of global memory that is atomically updated.[5] However, lock value blocks are volatile; for example, their contents can be lost (become invalid) when a process holding this lock in either PW or exclusive (EX) mode terminates abnormally.[3] When problems such as this occur, DBMS must be able to reestablish the values from other sources — usually from data on disk. In



KEY:

(1) Acquire object B's lock in PW mode. Read lock value block.

(2) Compare lock's version to TROOT's object version. The same.

(3) Make changes to TROOT object. Also increment its version number.

(4) Write object back to root file.

(5) Increment lock value block and release.

Figure 3 Updating the Cluster-wide Global Section

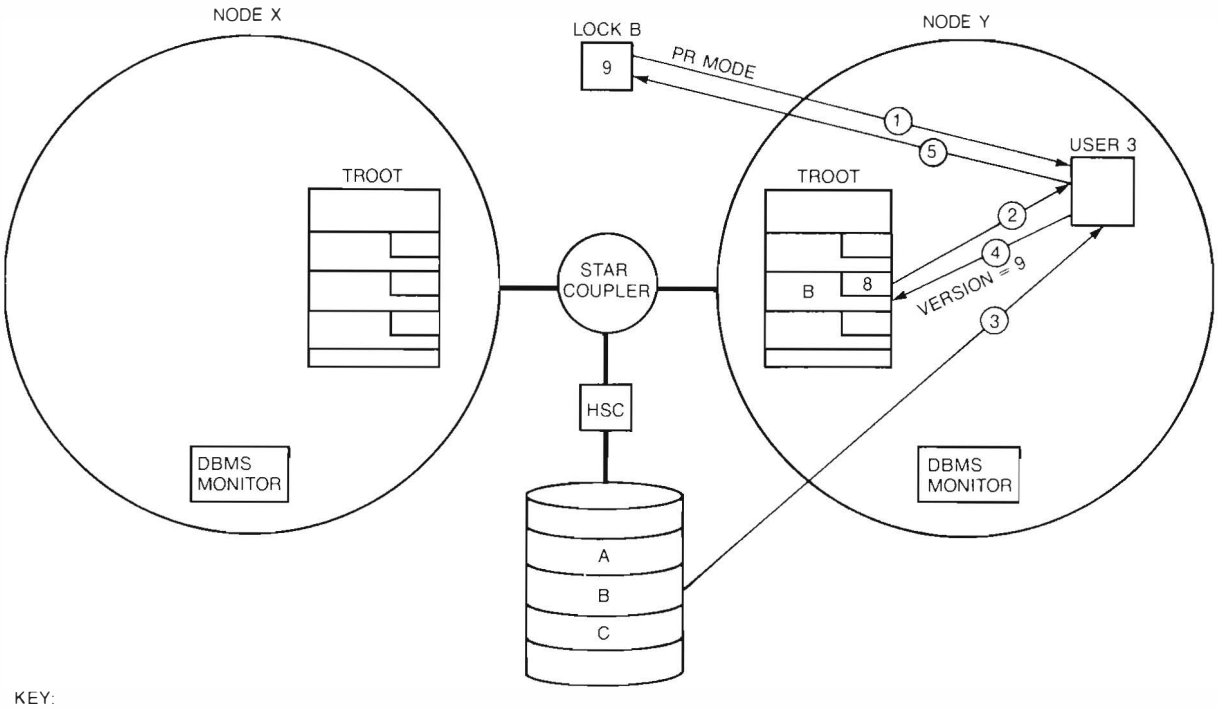
fact, recovering from the loss of value blocks is an integral part of making these algorithms robust.

A primary example of DBMS's use of lock value blocks is the way in which it distributes time as a monotonically increasing number. VMS time and date stamps are not synchronized across the VAXcluster nodes. Therefore, the VAX DBMS development group devised a cluster-wide, monotonically increasing clock mechanism called the transaction sequence number (TSN). TSNs identify database transactions and are used extensively throughout DBMS. For example, each record in the after-image journal file (discussed later) is marked with the TSN of the transaction in which the update occurred.

The next TSN to be allocated to an update transaction resides in the database root file. To avoid the obvious bottleneck of reading and updating the root file every time a TSN is needed, VAX DBMS software acquires a block of TSNs, currently eight TSNs, each time it accesses the root file's TSN object. This is done by incrementing the root file's TSN object by eight instead of by one.

A DBMS lock, called the RWROOT lock, is used to synchronize the distribution of this block of TSNs. The lock value block holds the next TSN to be given out. If the lock value block is lost, no damage occurs; only the remainder of the current group of TSNs is wasted. DBMS simply reads the next group of TSNs from the root file and refreshes the TSN value in the lock value block with this new TSN.

When a new update transaction begins, DBMS acquires the RWROOT lock in PW mode. This makes the distribution of TSNs single-threaded. The current TSN in the lock value block is assigned to this transaction. DBMS increments the TSN in the lock value block. If this value is still within the current group of TSNs, DBMS demotes the lock so another user can acquire a TSN. Otherwise, the current group of TSNs is exhausted. DBMS reads the next group of TSNs from the root file and refreshes the



KEY:

(1) Acquire object B's lock in PR mode. Read lock value block.

(2) Compare lock's version to TROOT's object version. Different.

(3) Read newer version of object from the root file on disk in TROOT's object. This is the data to use.

(4) Update TROOT object's version number.

(5) Release the lock. The read is complete.

Figure 4 Reading the Cluster-wide Global Section

current TSN in the lock value block before demoting the RWROOT lock.

Database Recovery

In the previous sections, we described two general mechanisms for communication used by VAX DBMS software in a VAXcluster environment. Both these mechanisms are used in the VAX DBMS recovery strategy to provide the properties of atomicity, concurrency, isolation, and durability for database transactions. These properties are referred to as ACID properties in the literature on this topic.

Inclusion of these properties means that no transaction sees uncommitted updates made by other transactions. All updates to the database performed by committed transactions are guaranteed to be in the database. The various software and hardware components of the system may fail at any point in time, in which case the uncommitted transactions are rolled back.

Following is a simple scenario of how failures can compromise transaction properties in a database system. If two users on two nodes want to update the same record in the database, the first user is granted the lock on the record, and the second user waits for the same lock. Suppose the first node fails at this time. The VMS lock manager releases the lock held by the first user and grants the released lock to the second user. The second user may now see uncommitted updates to the database made by the first user. This is a violation of the ACID properties of transactions.

VAX DBMS prevents such inconsistencies. Database recovery in a VAXcluster consists of four distinct steps.

1. Failure is detected.
2. All user locking activity on the database is stopped.
3. The updates made by failed transactions are undone.
4. All users continue after recovery is complete.

In the sections that follow, we describe the steps in the VAX DBMS recovery strategy.

Step 1: Failure Detection

VAX DBMS maintains the integrity of the database in the event of several types of failures possible in a VAXcluster system. First, such failures must be detected; they cannot be predicted. There are three basic types of failures.

- Abnormal termination of a user image. For example, the user may type control-Y at any time to stop the execution of the application program. Or the VMS process may halt for a variety of reasons.
- Node crash. A VAXcluster node may crash as a result of hardware failure.
- Cluster crash. The whole VAXcluster system may crash as a result of hardware or power failure.

We describe in the next three sections how these three kinds of failures are detected.

Detecting Abnormal User-image Termination

User-image termination is detected by using the VMS lock manager and the DBMS termination lock. When a user process attaches to a database, it acquires its termination lock in PW mode. The DBMS monitor process on the same node also enqueues a request for the same lock in PW mode, which is not granted since it is incompatible.

Normally during a user-image rundown, the VAX DBMS software releases all DBMS locks owned by the user. However, in the event of an abnormal image termination, the transaction must be recovered before the release of the database locks. Therefore, at image rundown, a kernel mode routine is invoked; the termination lock is demoted to null (NL) mode, and a request is queued for promotion to PW mode again. As soon as the lock is demoted to NL mode, the monitor is granted the PW mode lock that was pending. Thus the monitor detects the failure of the user process. Since the monitor now has the PW mode lock and the user process has enqueued a request for the same lock in PW mode, the user process is temporarily stalled. While the user process is still stalled, the monitor proceeds with the rest of recovery. We refer to this monitor that coordinates the recovery as the recovery monitor.

Note that recovery of abnormal user-process termination is handled in the same way as abnormal user-image termination, since VMS performs image rundown before process deletion.

Detecting Node Crash in a VAXcluster System

The method described above to detect process failures cannot be used to detect node failures. In a VAXcluster system, no mechanism can stall the release of locks held by the processes on the failed node. When a VAXcluster node fails, all locks held by the processes on the failed node are released in an uncontrolled manner by the VMS lock manager. This characteristic may compromise database integrity. Hence, node failures must be detected by all user processes of the database. Note that the crash of a monitor process on any node is considered to be the same as the failure of that node for recovery purposes.

The DBMS deadman lock enables all user processes and other monitor processes to detect the failure of a monitor process. A special option of the VMS lock manager is used to make sure that the deadman lock is released earlier than the other database locks.

There is a deadman lock for every monitor process attached to the database. A unique monitor identifier (MONID) is assigned to the monitor when it attaches to the database. The MONID is recorded in the MEMBIT bitmap in the database root file.

The name of the deadman lock comprises a prefix which denotes the lock as being a deadman lock and the MONID of the monitor that is attached to the database. This is represented as <DEADMAN,MONID>. Each monitor seizes an EX mode lock on its deadman lock when it attaches to a database. To detect failure of a monitor with MONID=M1, all user processes and other monitor processes attached to the same database request a PR mode lock on the lock <DEADMAN,M1>. This lock is not granted until the monitor with MONID=M1 fails and releases the EX mode lock. But as soon as monitor M1 fails, all user processes and monitors are granted the PR lock, since a PR lock is compatible with another PR lock. Therefore, when a user process is granted the <DEADMAN,M1> lock in PR mode, the process assumes the monitor M1 has failed.

The selection of the recovery monitor is decided with the help of a DBMS lock called the membership lock. There is only one membership lock for a database. Holding this lock in PR mode means that the monitor knows the current contents of the MEMBIT bitmap. During normal operation, all monitors have a PR mode lock on this lock. Whenever a monitor attaches to or detaches from a database, the monitor must obtain this lock in PW mode to include its MONID in the MEMBIT bitmap.

Upon detection of a node failure, all the monitors that detect the failure demote their membership lock for the database to NL mode. They then try to obtain a PW mode lock on the same lock. Only one of them can obtain the lock, and that monitor becomes the recovery monitor. The recovery monitor identifies all other monitors attached to the same database by checking the MEMBIT bitmap. It then

determines whether each of the monitors attached to the same database is active by checking the deadman lock for each monitor. The recovery monitor checks the list of all active database users in the root file for the failed monitors and starts a database recovery (DBR) process to recover each of these users.

Detecting VAXcluster Crash A crash of the whole cluster cannot be detected until the VAXcluster system is rebooted. The detection of the VAXcluster failure comes only after user lock activity has been stopped implicitly by the crash. When the first user tries to use the database, the user process sends a mail request to the monitor on its node to attach to the database.

The VAXcluster crash is detected only when the first monitor attaches to the database. The recovery monitor is the first monitor that attaches to the database after the VAXcluster reboot.

As in the case of a failed node, the recovery monitor determines all failed monitors from the MEMBIT bitmap. In this case, all the monitors that were attached to the database prior to cluster failure must have failed. It then creates DBR processes one by one for all failed users.

Step 2: Database Freeze

After failure is detected, the recovery monitor starts a DBR process for each failed user transaction. The DBR process uses the DBMS freeze lock to begin a database freeze. We define a database freeze as the state when no more locking activity is allowed on the database. There is one freeze lock per database. Database freeze is understood to be in effect if a user process cannot obtain a PR mode lock on the freeze lock.

Whenever a DBR process starts execution, it tries to freeze the database by enqueueing a request for a PW mode lock on the database freeze lock. (A database freeze is always requested by a DBR process except in the case of a VAXcluster crash. When a cluster crashes, the recovery monitor process prevents all user lock activity simply by not granting any request to attach to the database until recovery is complete.) All active database user processes are given a blocking asynchronous system trap (AST) by VMS; user processes then demote their freeze lock to NL mode and thus cooperate in ensuring the freeze. The database freeze is in effect after that request is granted.

A database freeze can happen at any time. Therefore, every user process checks to see if it still has the freeze lock in PR mode after obtaining each database record lock. If a database freeze is in effect, the user process releases the record lock and stalls until the unfreeze.

Step 3: Undo of Uncommitted Updates

After the database freeze has been established, the DBR process performs the recovery of transactions. The recovery procedure is exactly the same as executing the ROLLBACK command to voluntarily abort a transaction. VAX DBMS uses the "undo/no-redo" recovery strategy to ensure the integrity of the database.[6] Key to this strategy is the recovery unit journal (RUJ) file. One RUJ file exists for every database attach. The RUJ file must be available to recovery processes on any node and therefore must be on a disk accessible from all VAXcluster nodes.

During normal operation, the RUJ file is updated by a user process in the following manner. Before an update takes place, a process locks the record in the database to be updated. The process then writes the current value of the record to the RUJ file and updates the record in memory; only then can the updated record be flushed to the database. The logging of before-images of records to the RUJ file is optimized such that only one I/O is needed for typical OLTP (on-line transaction processing) transactions.

The DBR process reverses the transaction updates to the database by looking through the RUJ file and installing the before-images of updated records. The operation of installing the before-images of records is idempotent; that is, the operation can be executed any number of times and provides the same result as executing the operation once. Therefore, if a failure occurs during such a recovery operation, the recovery process can safely start again.

Step 4: Database Unfreeze

After the DBR process has recovered the failed transaction, the unfreeze is triggered. The DBR process demotes the freeze lock for the database to PR mode. After this demotion, all the user processes of the database are granted their outstanding requests for PR mode locks on the freeze lock. The unfreeze is then complete and the user processes can now proceed with database activity.

After-image Journaling

We have described how DBMS utilizes an RUJ log to recover from abnormal process termination and system failure. However the recovery mechanisms only back out uncommitted transactions, they do not allow the system to recover from media errors.

At any point in time, the collection of data files on disk constitutes the state of a DBMS database. That is, the disk files reflect all committed updates made to the database. If one of the database disks becomes corrupt, the database system must ensure that all committed data is once again reflected in the files on that disk.

As a simple example of the problem, consider an automatic teller machine (ATM) system which serves a large city. Assume the database is backed up each morning at 3:00 A.M., when no one is accessing the system. This backup is an exact copy of all committed data in the database at that moment in time. Throughout the course of the day, people use the ATMs to perform a variety of transactions (debit, credit, transfer, etc.). If there is a disk crash at 11:45 A.M., all the updates committed between 3:00 and 11:45 could be lost.

DBMS recovers from this scenario by keeping a log (AIJ file) of all committed database updates. After a media failure, it is possible to reconstruct the database by first restoring the database backup and then reapplying all the committed transactions stored in the AIJ log. The reapplication of transactions from the AIJ file is called roll-forward.

In DBMS, there is one AIJ file per database. Before a transaction can commit, all its updates must be reflected in AIJ records which are stored in the AIJ file. In addition, DBMS flushes a commit record for the transaction to the AIJ file. Consequently, the AIJ file will always contain the necessary information to redo any completed transaction.

To simplify the roll-forward operation, the AIJ mechanism utilizes one log for all database users.

Considerations for Journaling in the VAXcluster

All database updates are logged to one AIJ file. The efficient coordination of a single file among many users presents a number of difficulties. We outline the concerns here and describe below the DBMS implementation that addresses these.

First, the record management services (RMS) of the VMS operating system do not support shared sequential files efficiently enough to allow high throughput for the database system. That is, since each update transaction must write AIJ records, the I/Os needed for flushing AIJ records must be minimized. (In fact, at the time DBMS implemented VAXcluster support, RMS did not even allow this capability.)

Second, it is desirable to have a design in which each process attached to the database shares the responsibility of flushing data to the AIJ file. This design eliminates a single point of failure. For example, if there is one "journaling process," the database system is dependent on that process being available and could be forced to shut down in the event of a node failure.

At high transaction rates, DBMS can generate an enormous amount of AIJ log data. Consequently, it is desirable to store the AIJ log on tape for long-term storage. The problem posed by VAXcluster systems is that tape devices cannot be mounted cluster-wide. As a result, DBMS cannot flush directly to tape. Hence it is necessary to have a method for transferring the AIJ information from disk to magnetic tape.

We describe the solutions to these problems in the following sections.

Shared Sequential Log Files

VAX DBMS implemented its own version of shared sequential files using global sections on a per node basis, SQIOs, and the VMS lock manager. This mechanism uses one global AIJ lock and one local AIJ lock per node.

The global AIJ lock is used to single-thread access to the AIJ file. DBMS uses the lock value block to maintain the file context (current block and end of file [EOF]). There is one global AIJ lock per database. A process must acquire the global AIJ lock in PW mode before it can flush AIJ records to the AIJ file.

The local AIJ lock is used to single-thread access to a local AIJ cache, which is located in the TROOT. A process must acquire the local AIJ lock in PW mode before accessing the AIJ cache. The AIJ cache is used as a temporary repository for formatting AIJ records before flushing to the AIJ file. DBMS does not maintain any data in the lock value block of the local AIJ lock; it is simply a mutual exclusion mechanism. There is one local AIJ lock and one AIJ cache per node in the database.

Because lock value blocks can be lost, DBMS needs to be able to reconstruct the lock value block of the global AIJ lock. DBMS deals with this problem by initializing the AIJ file with set bits whenever it is created or extended. This initialization allows DBMS to rebuild the lock value block by searching backwards through the AIJ file until an uninitialized block is encountered. The uninitialized block indicates that the block contains data. Thus DBMS can reset the current block and EOF in the value block of the global AIJ lock.

Group AIJ Flush

When a transaction commits, DBMS ensures that all the AIJ records associated with the transaction are flushed to the AIJ file. However, by using the AIJ cache and a queue (also located in the TROOT), DBMS allows one process to format and flush AIJ records for other processes on that node, thereby amortizing the I/O among many different transactions. This mechanism is called group flush.

This feature prevents the AIJ file from becoming a database hot spot, which could restrict throughput. For example, if each transaction required one I/O to flush to the AIJ file and the AIJ disk could perform 25 I/Os per second, the AIJ file and disk would limit

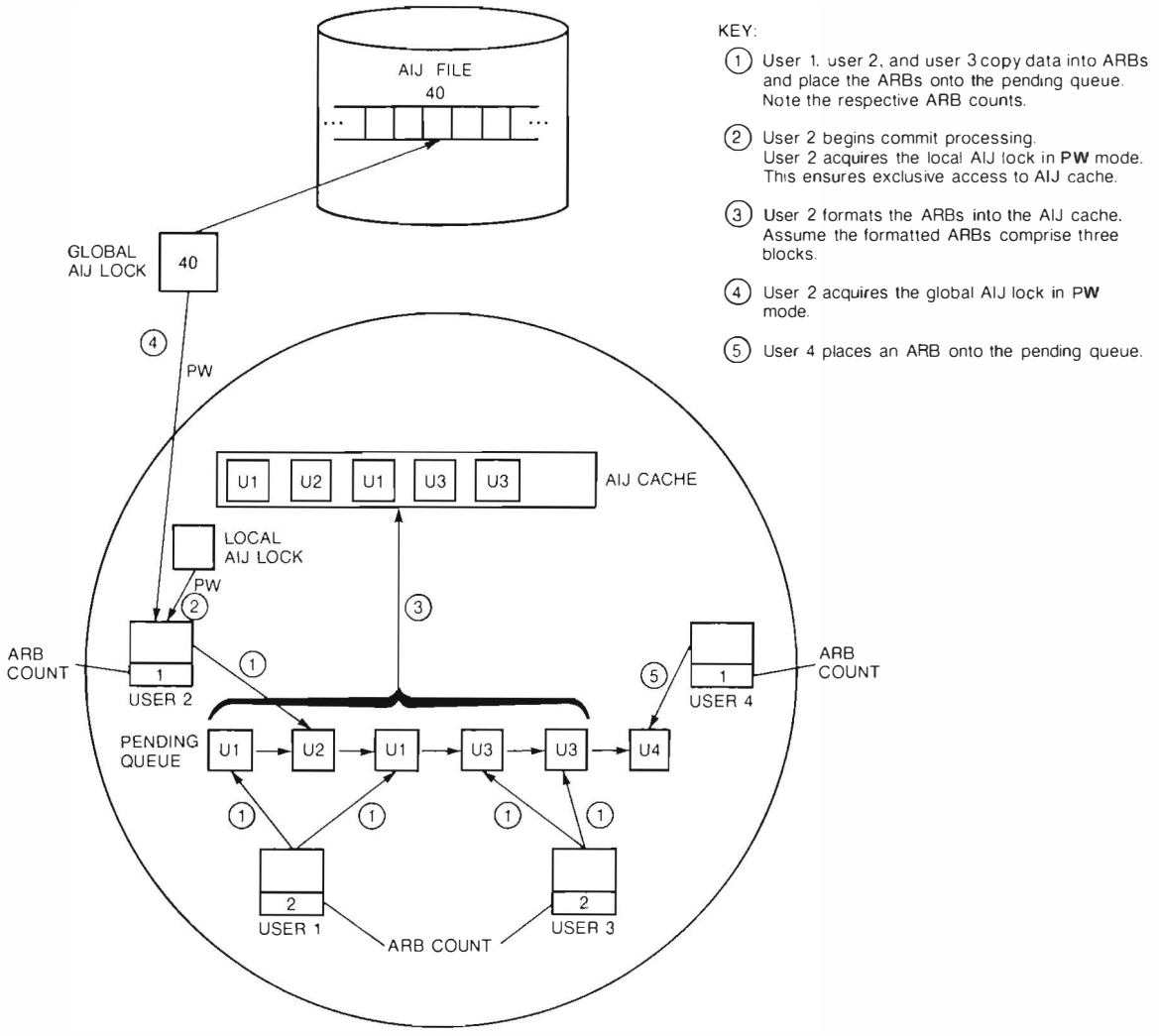


Figure 5 The After-image Journaling Sequence (Part A)

the database system to 25 transactions per second. This throughput is not acceptable for OLTP applications, hence the group flush optimization.

As a process modifies data in the database, it copies the record after-images to a process-private buffer. When the private buffer fills up, the process allocates an AIJ request block (ARB) and then copies the data to the ARB. The process then increments its ARB count (ARB_CNT) by using the VAX/VMS interlocked add instruction, and the process places the ARB onto a pending queue using the VAX/VMS interlocked queue instruction.

The ARBs, the pending queue, and the ARB_CNT of each process are all located in the TROOT. Therefore these data structures are accessible to all user processes attached to the database from the same node.

At commit time, a process first checks its ARB_CNT, which maintains the count of ARBs that the process currently has residing on the pending queue. If ARB_CNT is zero, another process has cooperatively flushed all the AIJ records created by this transaction, and the process simply returns. If the ARB_CNT is not zero, the process enqueues a PW lock request for the local AIJ lock. Once it acquires the local AIJ lock, the process again checks its ARB_CNT. While the process was waiting on the lock request, another user may have flushed some or all of the ARBs on the queue. If this process still has ARBs on the ARB queue, it is forced to flush its AIJ records to disk.

If forced to flush, the process formats as many ARBs as possible from the queue into the AIJ cache, which is 64 blocks long. Because of the VAX inter-

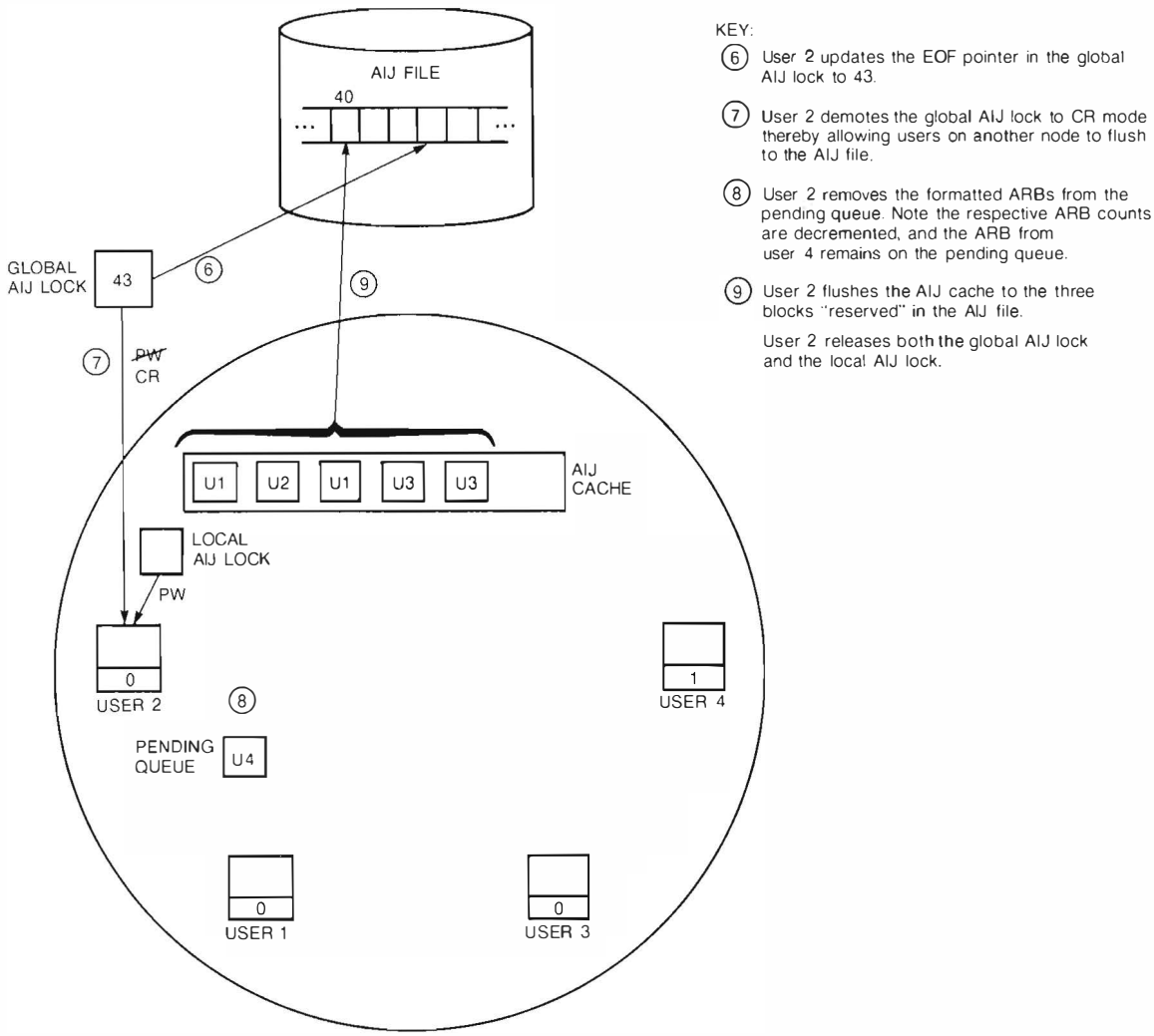


Figure 5 The After-image Journaling Sequence (Part B)

locked queue instructions, other processes can add new ARBs to the queue while one process is formatting data. Once the AIJ cache is full, or the queue is empty, the process enqueues a PW request for the global AIJ lock.

The process increments the current pointer in the lock value block by the number of blocks it formatted in the AIJ cache, thereby reserving this space in the AIJ file. Then the process demotes the global AIJ lock to concurrent retrieval (CR) which allows a process on another node to continue its flush sequence. The lock demotion allows a degree of concurrency since the global AIJ lock is held only for a short period of time to reserve space in the AIJ file.

Since all the ARB data has been formatted into the AIJ cache, the flushing process removes the ARBs from the pending queue. During removal, the process decrements the ARB_CNT of each process for which an ARB has been formatted. Thus the flushing process indicates to other processes that their data has been flushed. The process then issues a $QIO to flush the cache to the "reserved" virtual block in the file. Finally, the process releases both the global AIJ lock and the local AIJ lock. This sequence is illustrated in Figure 5.

Note that the ARBs are freed up after formatting into the AIJ cache, but before the $QIO is issued. If the process performing the flush aborts, the DBR process flushes the AIJ cache to disk. The DBR process must complete an aborted flush since there can be records from different transactions in the AIJ cache.

The group flush sequence, described above, performs quite well. In benchmark tests that process about 50 transactions per second, DBMS requires only 0.2 I/Os to flush the AIJ information for each transaction.

AIJ to Tape

Even though processes can cooperatively flush to an AIJ file on a cluster-wide disk, the problem of storing the AIJ data on tape remains. To solve this problem, DBMS uses a separate despooling process that transfers data from the AIJ file on disk to a tape device. This occurs while normal database activity continues.

When activating the despooling process, a user can specify the amount of AIJ data to be transferred and the frequency at which the data is to be copied. Once initiated, the process transfers blocks from the AIJ file on disk to the AIJ file on tape. After the data has been copied, the despooling process acquires the global AIJ lock in EX mode for a brief period in order to truncate the AIJ file and update the file context.

The despooler can be started or stopped on any node in a VAXcluster system. Failure of the despooler process simply stops the archiving activity to tape. DBMS continues writing AIJ data to disk.

On-line Backup Facility

In the previous section, we described the mechanism for recovering from media errors. This recovery depends on the existence of a complete database backup as the starting point before all committed transactions in the AIJ file are reapplied.

Since a database may be accessed continuously, there is no time to perform normal database backups. As a result, DBMS provides a way of backing up a database while users are still accessing it. This facility, called the on-line backup facility, allows high database availability. The on-line backup facility uses the snapshot capability of DBMS to achieve a consistent picture of the database even when update transactions are active.[7]

The snapshot mechanism allows the on-line backup to access only records that have been committed by the time its read-only transaction starts. In this context, DBMS uses TSNs to mark the "age" of records. Update transactions date modified records using their TSNs and then copy old versions of the records to a snapshot file which maintains a chain of record versions. Read-only transactions then use the TSNs to decide which record version in the snapshot chain is visible to them. As a result, all database activity continues normally while the on-line backup process produces a consistent archived copy of the database.

Summary

VAX DBMS software incorporates novel mechanisms to operate a centralized database in the VAXcluster environment. In general, these mechanisms solve the problem of efficient sharing of data in a cluster. DBMS solves this problem by making use of lock value blocks provided by the VMS lock manager, global sections, and shared disks. The DBMS recovery mechanisms successfully avoid single points of failure. In particular, DBMS immediately recovers from abnormal termination of users and node failures as long as one node remains active. After a cluster failure, DBMS recovers all uncommitted transactions before allowing normal access to the database. The on-line backup and AIJ facilities combine to ensure efficient recovery from media failures.

The transaction recovery and media recovery mechanisms supported by DBMS preserve the high availability of the VAXcluster environment.

Acknowledgments

Many engineers participated in the design, implementation, and optimization of the VAX DBMS software described here. We wish to thank all of them. In particular, we would like to thank Steve Klein, consulting software engineer, who is principally responsible for the techniques described in this paper.

References

1. N. Kronenberg et al., "The VAXcluster Concept: An Overview of a Distributed System," *Digital Technical Journal* (September 1987): 7–21.

2. *VAX DBMS Maintenance and Performance Guide* (Maynard: Digital Equipment Corporation, Order No. AA-Y313D-TE, 1988).

3. *VAX/VMS Introduction to System Services Manual* (Maynard: Digital Equipment Corporation, Order No. AA-LA68A-TE, 1988).

4. *VAX/VMS System Services Reference Manual* (Maynard: Digital Equipment Corporation, Order No. AA-LA69A-TE, 1988).

5. W. Snaman et al., "The VAX/VMS Distributed Lock Manager," *Digital Technical Journal* (September 1987): 29–44.

6. P. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems* (Reading: Addison-Wesley, 1987).

7. *VAX DBMS Database Design Guide* (Maynard: Digital Equipment Corporation, Order No. AA-Y311C-TE, 1988).

Ashok M. Joshi
Karen E. Rodwell

A Relational Database Management System for Production Applications

VAX Rdb/VMS software, Digital's relation database management system for VAX/VMS systems, was designed primarily for use in ad-hoc query-intensive applications. For its third major release, VAX Rdb/VMS has been enhanced to support the requirements of large, complex production applications as well as the requirements of end-user data access. These improvements include an additional access method, specifically, hash indices; new database structuring capabilities to significantly reduce I/O bottlenecks; record placement control; a query optimizer which exploits the new placement and index methods; an on-line, high-performance backup utility; and several utility enhancements. This paper describes these features and presents examples to demonstrate their utility.

Digital began a serious effort to enter the on-line transaction processing (OLTP) market in the fall of 1986. We recognized that in order to meet the performance requirements of the OLTP market, we would have to minimize the I/O bottleneck. Based on our experiences in the development of the VAX DBMS software, we decided to incorporate the physical structuring capabilities of CODASYL systems into Rdb/VMS software. These structuring capabilities allow a database administrator (DBA) to spread I/O over multiple disks thus eliminating any I/O bottlenecks. We also decided to implement hash indexing which is superior to the B-tree indexing in terms of I/O and locking requirements. The structuring and placement capabilities as well as details of the hash index implementation are discussed later in this paper.

Another major goal was to implement a high-performance, high-data-integrity, on-line, backup facility. This facility, described at the end of this paper, is available in version 3.0 as are a verify utility, statistics enhancements, and physical restructuring support.

Version 3.0 is completely compatible with earlier versions. The physical structuring attributes of the database are only visible by means of the utility interfaces. To exploit the new features, DBA intervention and careful schema design are required. However, old application programs can work without any changes. Users who are satisfied with the performance of earlier versions can continue to use the product with the same level of expertise and time investment as before. They need only increase that investment when and if they use some of the newer features. Migration from older versions is easy because of the availability of excellent conversion utilities.

In the following sections, we present several examples to illustrate the utility of these new features. Our discussion begins with an explanation of the Rdb/VMS architecture.

Rdb/VMS System Architecture

The Rdb/VMS system is designed in a layered, modular fashion. The software is layered on the VMS operating system and makes extensive use of VMS system services and file management facilities. The two major layers of the Rdb/VMS architecture are the relational data manager and the record storage system. Shown in Figure 1, this architecture is unique in that the record storage system is common to both the Rdb/VMS system and Digital's CODASYL database management system. The result is a highly maintainable system. Further, both products can exploit enhancements made to the lower layers. We are very pleased with the benefits of this architecture.

Relational Data Manager

The relational data manager is made up of several components. Collectively, they provide an interactive user interface, a callable interface for integration into application programs, a catalog manager, a query parser, a semantic analyzer, an optimizer, and

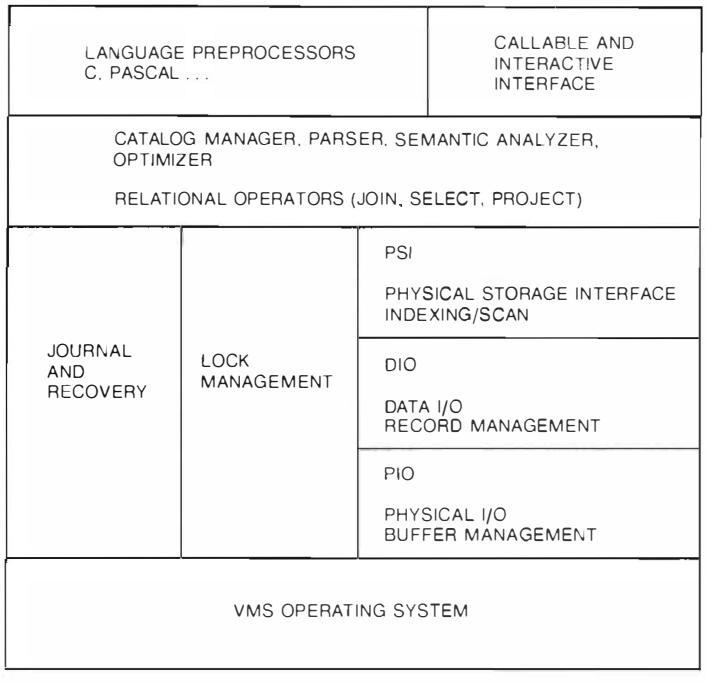


Figure 1 Overview of Rdb/VMS System Architecture

a query executor. In addition, Rdb/VMS language preprocessors generate procedure calls according to the Digital standard relational interface (DSRI). Rdb/VMS utilities also backup and restore the database, monitor system performance, and restructure the database.

A request or query is processed in several stages, namely, compilation, access path selection, code generation, and execution. The database query language is translated into BLR (binary language representation, the internal representation defined in DSRI). BLR describes the interaction between the user application and the query processing component of the Rdb/VMS product, which provides a complete set of relational operators (such as join, restriction, and projection). Based on each BLR request, the compilation stage builds data structures for use during query execution. A query or request can be compiled once and executed several times within a session.

Database Monitor

The database monitor is a watch-dog process that keeps records of user sessions on the database, coordinates recovery, and handles abnormal termination of user processes. The monitor also detects cluster transitions and initiates recovery on behalf of users on the failed node.

Record Storage System

The record storage system (RSS) provides the components used in common by Rdb/VMS and VAX DBMS software. (VAX DBMS is Digital's CODASYL database management system.) In a cluster, RSS allows users equal access to the database from any node. In addition, if a node in the cluster fails, only users on that node are affected. Using intricate protocols, RSS can recover from a cluster transition with minimal penalty to users on healthy nodes.

The following sections explain some of the RSS subsystems in greater detail. For a detailed discussion on failure recovery in a cluster, see reference 1.

Modules and Subsystems

As shown in Figure 1, RSS is made up of several subsystems, each performing a set of logically related functions. Three of these subsystems are implemented in a layered fashion.

The lowermost subsystem, the physical I/O (PIO), is layered on the VMS system. This subsystem, or layer, fetches pages into the database system buffers and flushes them to disk. It manages the buffer pool on a least-recently-used basis. The PIO layer also manages the physical files that contain database pages. At this level, a page is a set of contiguous disk blocks (512 bytes on the VMS system). RSS supports variable page sizes in multiples of 512 bytes.

The data I/O (DIO) subsystem is primarily concerned with fetching, storing, modifying, and erasing logical records. A record is a sequence of bytes with a type identifier. A record is identified by a database-wide unique identifier called a DBKEY. The internal structure of the record is invisible at the DIO level. A record may be fragmented over several pages if it does not fit on a page. This layer's responsibility is to reassemble the fragmented record in virtual memory (if necessary) before making it visible to its client. DIO holds and releases record locks to enforce consistency. In addition, DIO uses the services of the journaling and recovery subsystems to log changes to data records.

The next layer above the DIO is the physical storage interface (PSI). This subsystem implements access methods such as B-tree and hash indexing.[2] PSI also maintains retrieval and update scans on sets of records. Using the services of the DIO layer, PSI manages B-tree nodes and hash index buckets. PSI allows single-attribute as well as multiattribute indexing.

In addition to the three layers described above, RSS contains modules that enforce lock protocols, perform redo and undo logging and recovery, and utilities for maintaining the database.

The lock subsystem implements two-phase locking using the services of the VMS lock manager.[3] Two-phase locking protocols guarantee consistency of the database by ensuring that concurrent users only see correct and committed changes to the database. The subsystem also maintains lock hierarchies to reduce conflicts.[4]

For logging changes to the database, the journaling subsystem implements the write-ahead-log protocol.[4] It performs both redo and undo journaling.

The undo journaling subsystem, called RUJ, writes before-images to disk on behalf of a user performing updates. These before-images are written to a journal file using efficient algorithms to minimize the number of I/O operations. The RUJ also ensures that before-images are flushed to disk before dirty and uncommitted data is written to disk.[1]

RSS also provides a redo-journaling facility, called AIJ for after-image journaling, to recover from media failures. AIJ maintains a single log of all the updates to the database. AIJ log writes use group-commit protocols to reduce the I/O activity and contention at transaction commit time. The cost of using the group-commit technique is a small addition to the response time of a transaction.

Finally, RSS utilities perform several miscellaneous functions:

- Create and modify the database
- Dump information about the database
- Verify and reformat the database

In addition, RSS has a very powerful statistics and monitoring package that can be used to monitor the performance and behavior of various subsystems.

Hash Indices

Rdb/VMS version 3.0 provides hash indexing as an alternate access method. Hashing is a better retrieval method for exact-match queries than B-tree index scans or sequential retrievals. In a majority of cases, this method accesses data records in two I/O operations, regardless of the number of records in the relation. In special cases, such as clustered indices, the data records can be accessed in only one I/O operation. (An example is given in the section Clustered Indices.) This represents a significant improvement over B-tree index access, which requires three to four I/O operations to fetch data records. (The exact number of I/O operations performed in B-tree index access is a function of the logarithm of the number of records in the relation.)

Hash indices also perform better than B-tree indices with respect to locking contention during updates due to the randomizing nature of the hash function. In addition, fewer locks are necessary to manage update operations on hash indices.

The randomizing behavior of the hash function also affects the utilization of the database buffers. There is almost no locality of reference for hash index buckets, even if there is locality in the index key space. Thus, hash indices exhibit poor caching characteristics. Consequently, an index access leads to an I/O operation in a majority of the cases. B-tree indices, on the other hand, exhibit much better caching characteristics.

The Hash Index

Hash indices are an extension of main memory hashing techniques to secondary storage. A hash index can be defined over a field (or set of fields) of a relation. This field (or set of fields) is called the hash index key (or simply index key). A hash index uses a hash function to maintain a map between index key values and the DBKEYs of records that contain these values. This mapping is stored in a file on disk.

A hashing function maps the index key space into a set of hash buckets. Typically this is a many-to-one function; that is, the function maps more than one key value to the same bucket. The mapping does not maintain the key sequence of the hash index keys. A good choice of the hashing function ensures with a high probability that the key space will be uniformly distributed over the number of available buckets.[5]

A hash bucket is a data structure that contains a list of hash elements. A hash element maintains

	HASH ELEMENT	HASH ELEMENT	HASH ELEMENT
HEADER	SMITH	JONES	McDONALD

Figure 2 Structure of a Hash Bucket

information about one hash index key and the DBKEY of record containing that key value, denoted as < hash index key, DBKEY of record >. The number of hash buckets is equal to the number of pages in the file that contains the hash index. This number is typically much smaller than the number of values of the hash index key. Therefore, more than one hash index key will map to a bucket — a situation referred to as a collision. By choosing the number of pages in the hash index file appropriately, it is possible to reduce the number of collisions and consequently the size of the hash bucket. A good hashing function will result in roughly equal-sized buckets, with the number of hash elements per bucket being relatively constant. Figure 2 illustrates the structure of a hash bucket. In this example, keys Smith, Jones, and McDonald hash to the same bucket.

Algorithms

We now look at how a hash index can be used to retrieve records with a minimal number of accesses to disk. Assume that a hash index is defined on the EMP_ID field in the Employees relation of Figure 3. Also assume we are storing a new employee record in the database with the following values.

{10101, James Smith,
10 Tara Blvd. Nashua NH,
Senior Engineer, 432}

The record is assigned a DBKEY of 55. The hash element for this record should be the pair < 10101, 55 >. To make an entry in the hash index, the key value 10101 is hashed. Assume that this results in the selection of bucket number 69. The index insertion algorithm stores the pair < 10101, 55 > in the bucket on page 69 of the hash index file.

To retrieve the data record containing the key value 10101, the hash index search algorithm works as follows. The key value 10101 is hashed, resulting in page number 69. This page is now read in. (This is the first I/O operation in the retrieval.) The bucket on that page is searched for the hash element that contains the key value 10101. (Due to collisions, many other values may also be in this bucket.) The fetch algorithm uses this DBKEY (55) to retrieve the record. (This is the second I/O operation of the retrieval.) Thus, the index can be used to retrieve records in two I/O operations, independent of the number of records in the relation.

Unsuccessful searches require only one I/O operation to the hash index file. Assume that the Employees relation does not contain the record with key value 769. If a query is made to determine whether the key value 769 exists in the database, the hash index can be used as follows. As before, the key value is hashed to generate a bucket number, e.g., 498. The index search fetches page 498 and searches the bucket for a hash element containing the value 769. The hash element is not found, and the search is terminated.

So far our discussion has assumed there is only one data record with a given key value. To maintain hash indices on relations where there is more than one record containing a given key value, a hash element is allowed to contain a set of DBKEYs of records that contain the key value. This set of DBKEYs is stored in a duplicates node instead of in the hash

EMPLOYEES:

EMP_ID	EMP_NAME	EMP_ADDRESS	EMP_TITLE	DEPT_ID

DEPARTMENTS:

DEPT_ID	DEPT_NAME	DEPT_LOCATION	DEPT_MANAGER

DEPENDENTS:

EMP_ID	DEP_NAME	DEP_AGE	CLASSIFICATION

Figure 3 Sample Schema

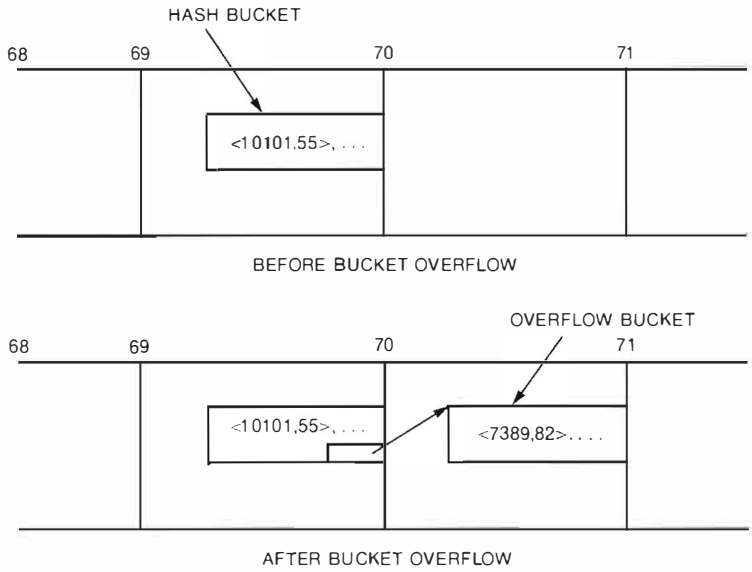


Figure 4 Hash Bucket Overflow

element. The decision to use duplicates nodes results in considerable simplification of the algorithms that maintain index scans. By placing the duplicates nodes on the same data page as the associated hash element, it is possible to avoid extra I/O operations to fetch them. However, if the number of duplicates for some key value is very large, duplicates nodes may "spill over" to overflow pages. In this case, more than one I/O operation may be required.

A bucket could grow beyond the size of a database page. This undesirable condition can occur for a variety of reasons, such as large numbers of duplicates, skewed key values, or poor choice of hash function. Hash index performance can degrade rapidly as the number of overflows increases. The hash index algorithms handle overflows by splitting the overflowing bucket into two or more buckets that are linked together and placed on pages close to the original page. (See Figure 4.) Of course, the head of this list of buckets must remain on the same page as before. Database monitoring utilities can be used to measure the number of overflows in hash indices. If the number is high, it is possible to restructure the index by using the backup and restructuring utilities.

Design Trade-offs

During the design phase, we had to choose between static and dynamic hash indexing. In static hashing, an explicit reorganization of the index is necessary when there is substantial performance degradation (due to overflows). This reorganization is usually performed off-line. In contrast, dynamic hashing schemes are self-reorganizing. For version 3.0, we chose to implement a static hash indexing scheme based on design and implementation simplicity. Our experience so far indicates that with careful physical design of the index storage area, the index need not be reorganized for substantial periods of time. Further, static hashing may result in disk space savings as compared with some dynamic hashing schemes such as extendible hashing.[6]

Structuring and Partitioning Capabilities

In earlier versions, an Rdb/VMS database was constrained to a single VMS file. The bound volume set capability of the VMS system allowed the file to be striped across several disks. However, data striping was not enough to distribute the I/O operations over a large number of disks. The result was performance bottlenecks (called the I/O bottleneck), since the database system performance was limited by the I/O transfer rate of a single disk.

As noted earlier, one of the goals of the Rdb/VMS project was to give the DBA complete control over the placement of data files on disks. With this new version, a DBA can exploit the structuring capabilities to eliminate the I/O bottleneck. This feature does, however, require careful physical design and file placement at database design time.

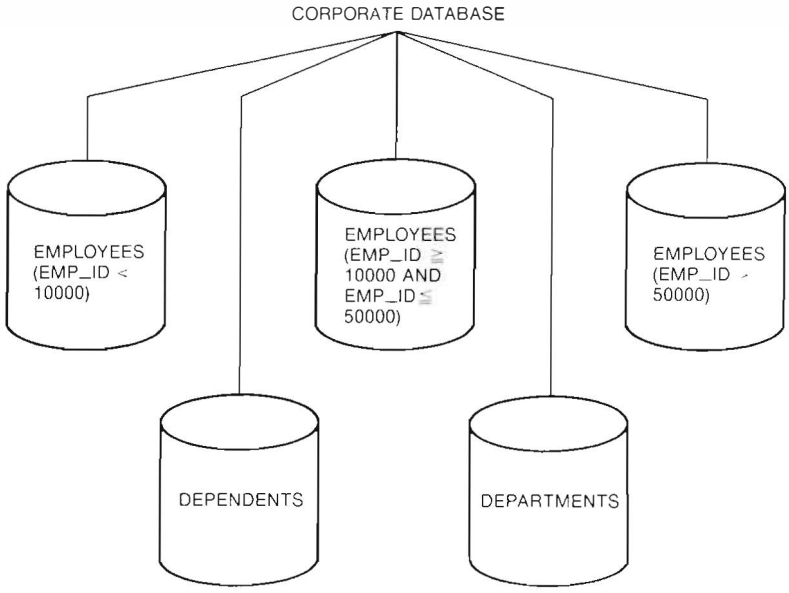


Figure 5 Partitioning a Relation

A database can consist of several VMS files, called storage areas. Each storage area can be independently assigned to a single disk. Storage areas can be mapped to disks (or bound volume sets) at the physical design stage. This mapping is referred to as the multifile database capability.

A storage area can contain database relations or indices or both. It is possible to store more than one relation per storage area. For this reason, the Rdb/VMS software maintains space-management data structures, called SPAM pages, which can be used to accelerate sequential scans on relations. SPAM pages maintain information about free space in the storage area. In addition, SPAM pages can be used to determine whether a certain page contains records belonging to a specified relation. The proper design of SPAM pages and associated algorithms is critical to ensure that these pages do not become a hot spot during periods of high update activity.

To further alleviate the I/O bottleneck, Rdb/VMS software also allows partitioning of relations and indices over several storage areas as explained below.

There are two options for partitioning a relation. Rdb/VMS software can uniformly spread the records of the relation over several storage areas by using a randomizing function. This type of partitioning is useful when user transactions access single records in the relation. However, Rdb/VMS does not allow an index to be partitioned in this manner.

The second way to partition the relation is by specifying a partitioning predicate. For example, consider the Employees relation in Figure 3. Assume there are three storage areas over which we wish to spread the Employees relation. As illustrated in Figure 5, we can specify a partitioning predicate to indicate that employee records with values of EMP_ID less than 10000 are placed in the first storage area; records with values between 10001 and 50000 are placed in the second storage area; and records with values greater than 50000 are placed in the third storage area. The partitioning predicate must specify disjoint partitions of data so that every data record is allowed to exist in at most one partition. This type of partitioning is particularly useful in cases where users' access to the data records reflects the partitioning criteria. This technique can also be used to partition indices. The multifile and partitioning capabilities of Rdb/VMS software guarantee that no single disk can become an I/O bottleneck during database activity.

The partitioning of relations over several storage areas also introduces an additional level of locking granularity. If a transaction locks only one partition of the relation, concurrent access to other partitions is possible. The optimizer uses this property to allow a higher level of concurrency for transactions that access different partitions.

The structuring and partitioning of the relations require careful attention during the physical database design phase. Proper design can result in an order of magnitude improvement in execution performance for a majority of queries. It is also impor-

tant to note that the multifile and partitioning features are optional; users who are satisfied with the performance of the earlier versions can continue to use the new version of Rdb/VMS software without any performance degradation or DBA effort.

Record Placement and Clustering

Most transactions involve operations on more than one record. In a large percentage of the cases, the records that are accessed or updated belong to more than one relation but are related to each other. For example, a transaction may access a specific department record and all its related employee records. The operation of accessing a record as well as its related records is referred to as a join. The fields (of the related relations) that are used to relate the records to each other are called the join keys. For example, the DEPT_ID field in the Employees relation relates the Employees records to the corresponding Department record. Hence, the DEPT_ID field is the join key.

In almost all commercial relational database systems, a database page can only contain records from a single relation. Most systems are even more restrictive; they allow only one relation per storage area (or file). To perform the join operation, therefore, data pages from more than one file have to be accessed. This process almost always involves more than one I/O operation. (If the data is buffered, it may be possible to save an I/O.)

To avoid the penalty of performing extra I/O operations, Rdb/VMS version 3.0 allows records from more than one relation or index to be stored on the same page. A user can specify that a record and all its related records be placed on the same page, or if there is an overflow, on nearby pages. This technique of placing related records physically close together is referred to as interrelation clustering. During retrievals, the join operation can be speeded up considerably because of the reduction in I/O.

Behavior of Interrelation Clustering

Referring again to our example in Figure 3, consider the Departments and Employees relations whose records are related to each other. On average, we will assume there are 10 employees in each department. To place related records together, we will have to store groups of 11 records — 1 department record and 10 related employee records. Assume that the page size and record sizes allow one such group to fit on a database page. A page therefore contains only 1 department record and about 10 employee records. A query that requires the names of employees in a specified department need only perform one I/O operation. If interrelation clustering were not used, at least two I/O operations would be required. Hence, the number of I/O operations is reduced by 50 percent in the above example.

Now consider a query that prints the addresses of the various departments in the organization. This query needs to access only the department records. Such a query is referred to as a sequential scan. In this case, almost every retrieval of a department record will result in an I/O operation since a page only contains one department record. This example illustrates an important point: interrelation clustering will not improve the performance of all queries. In fact, some queries may suffer performance degradation. The choice of whether to use interrelation clustering or not is highly dependent on which queries the DBA wants to optimize. When a DBA chooses to cluster related records together, the retrieval performance is optimized for join accesses at the cost of sequential scan access.

An additional consideration in interrelation clustering is deciding which records to cluster together. Interrelation clustering works best when only one join key is used to cluster relations. Consider the case where the Dependents records are related to the Employees records by the EMP_ID fields (Figure 3). It is impossible to cluster Departments with the associated Employees and at the same time cluster Employees and related Dependents since the join key is different for the two relationships. The problem can be easily solved by including the DEPT_ID field in the Dependents relation (during the logical design phase). To simplify the rest of this discussion, let us restrict the number of related relations to two.

Placing Related Records Together

For records in two relations to be placed together, Rdb/VMS software requires an index on each relation on the join key field. For example, to store employee records close to the department record, an index must appear on the DEPT_ID field of the Employees relation as well as the DEPT_ID field of the Departments relation. The index can be either a hash index or a B-tree index. The relative position of the index entry within the index storage area guides the placement of the data record in its storage area. We explain this placement further in the following sections.

The placement capabilities and the multifile and partitioning capabilities provide the DBA a very powerful set of tools for physical schema design. To fully exploit these capabilities, the DBA must understand the access patterns of typical transactions. Using this information, the DBA can choose the best physical schema. In the following sections, we illustrate the complexity of physical schema design process.

Clustered Indices

In this example, we show how a relation's records can be placed on the same page as the index nodes or buckets. Assume there is a hash index on the EMP_ID field of the Employees relation. In addition, the Employees relation and the hash index are stored in the same file. As discussed earlier, the hashing algorithm chooses the bucket (and the page) into which the index entry is to be placed. The same page is used to store the data record. This placement algorithm ensures that the hash bucket and the associated record are placed on the same page (with high probability). Consequently when a user accesses an Employee record by means of the index, only one I/O operation need be performed.

Placement by means of a B-tree index can be specified in an analogous manner. However, due to the hierarchical structure of the B-tree, data records cannot be stored close to the corresponding leaf-level nodes of the B-tree. Thus, placement by means of a B-tree index is, at best, marginally useful when the relation and the B-tree index are in the same file. Instead, a related technique, called shadow clustering, can be used to store data records in the approximate sorted order of the join key.

To perform shadow clustering, the relation is stored in a file different from its associated index. The records in the relation are placed in the file based on the relative order of the entries in the leaf levels of the B-tree. Since the entries in the leaf-level nodes of the B-tree are maintained in sorted order, the records in the relation will also be stored in the sorted order of the index key. Note that it is not feasible to maintain the sorted order of the relation during random insertions, since this presumes displacement of a large number of records. Hence, shadow clustering works best when the records of the relation are sorted externally before being bulk-loaded into the relation. The "almost" sorted order of the data records results in considerable performance benefits when the user performs range retrievals based on the index key.

To place related records together, a combination of placement by means of hash index and shadow clustering can be used. For example, if we wish to place the Employee records on the same page as the Department record, we can elect to do the following:

- Define a hash index HE on the DEPT_ID field of Employees, as well as a hash index HD on the DEPT_ID field of the Departments relation.
- Store the two hash indices and the relations in the same file.
- Specify that the Employee records be placed near the corresponding hash bucket entries of hash index HE and that Department records be placed near the corresponding hash bucket entries of HD.

Since the hash indices HE and HD use the same hash function, we are assured that the same page will be chosen for identical values of DEPT_ID of Employee records as well as Department records. Figure 6 shows a page containing related Employee records, Department records, and the corresponding hash index entries.

Query Optimization

The query optimizer enhancements exploit the placement techniques and index methods to choose the best retrieval path for the user-specified query. The optimizer's decision is contingent on (1) whether the index is a hash index or a B-tree index, and (2) whether the data records and index nodes (or buckets) are placed on the same page.

The optimizer has also been improved in other ways. To reduce I/O operations, multiattribute selection attempts to perform DBKEY list intersection before fetching data pages. Improved join techniques reduce the complexity of the code as well as better utilize the database buffers. In the following list, we describe some of these features.

- Record placement. The optimizer attempts to choose an index that is defined on the same field as the field that is used to define record placement. The approach requires fewer I/O operations to do indexed retrievals because data records and index information are stored together.
- Hash index retrieval support. In version 3.0, the optimizer recognizes the new index type. Hash indexing is the most effective method for exact match queries. When a query can be answered simply by examining the entries in the index, the hash index is the best choice for answering the query.
- Multiattribute retrieval. An extension of the existing index retrieval technique, multiattribute retrieval reduces I/O operations by examining multiple indices, eliminating records that do not match all of the selection criteria, and finally retrieving the records that do match. When several indices are defined on the fields of a relation, the optimizer has several choices of index for a query that has a multiattribute selection predicate. In operation, the multiattribute retrieval algorithm generates lists of DBKEYs of records by

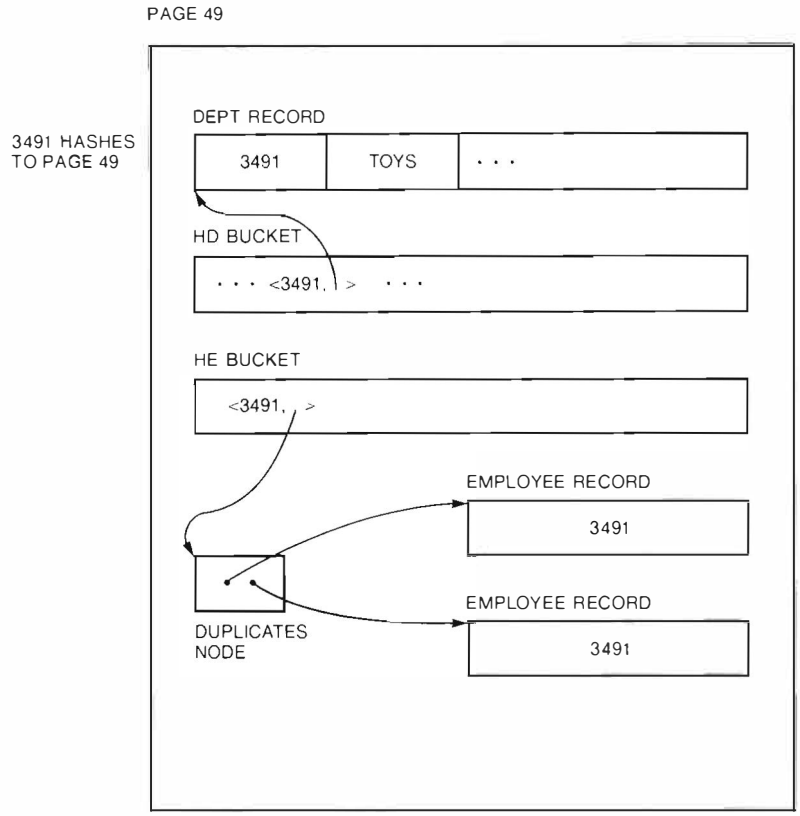


Figure 6 Example of Interrelation Record Clustering

using each relevant index. These lists are then intersected to generate the set of DBKEYs of records that must be retrieved. The intersection is performed in an optimized manner that guarantees that the most restrictive clause is used first. Rdb/VMS now retrieves the data records using this list of DBKEYs and applies any selection clauses that did not involve indices. This reduces the number of comparison operations that have to be performed.

- Index cardinality for nonunique indices. The optimizer maintains a count of the number of unique key values in an index on a relation. The optimizer can then choose indices based on cardinality information. Preference is given to indices with greater record selectivity. Cardinality information is maintained in an efficient manner so that it does not become a hot spot during periods of high update activity.
- Join optimization strategy. When the system performs an indexed equijoin, time can be wasted searching the entire inner index for a match for each key of the outer relation.[2] In Rdb/VMS version 3.0, the optimizer maintains context information that indicates the current position of the search in the inner relation's index. Thus the software saves the step of initiating a full search on the inner index for every key of the outer relation. The savings are more pronounced when the indices do not fit into the database buffers.

Fast Backup Utility

The performance of the backup and restore utility determines the maximum practical size of a database. The Rdb/VMS backup and restore utility can backup a 50 gigabyte database in less than 8 hours. We achieve this level of performance by applying a parallel processing paradigm to this traditionally sequential application.

The backup and restore operations are multithreaded. There are two kinds of threads: a reader and a writer. Reader and writer threads perform I/O operations on the database or the backup medium by using asynchronous overlapped I/O requests. The only synchronization points between the reader and a writer are the operations they have performed on the shared data buffers.

The backup utility starts one reader thread per storage area. There is one writer thread per backup medium (either a disk or, more often, a tape). This mode of operation is analogous to the well-known producer-consumer paradigm. Readers fetch pages from the database area using overlapped, asynchronous I/O. At any point in time, a large number of read requests are outstanding. As the I/O operations complete, the pages read in are queued in the appropriate order for the writer thread. The backup utility uses synchronized data structures to share buffers between readers and writers.

The backup operation can also be performed on-line. The snapshot capability of Rdb/VMS makes this operation possible. Snapshots ensure that read-only transactions can see a consistent view of the data while update transactions modify records. This Rdb/VMS implementation is one of the few relational database systems with this capability. The additional overhead of maintaining snapshots is nominal.

The on-line backup utility starts a read-only transaction on the database to ensure that it sees the data records as they were at the time the transaction was started. On-line backup is a feature critical in making Rdb/VMS a highly available system.

A restore utility is required to re-create a database from a backup. This utility is often used after a media failure when one or more of the database files is corrupted. After the database has been restored, the redo journal can be reapplied to restore the database to the latest consistent state. In a manner similar to the backup utility, the restore utility uses multiple reader threads to read the backup and multiple writer threads to write database pages to the database. The restore utility also needs to rebuild SPAM pages and related data structures that manage free space in the database files. To rebuild efficiently, the utility builds the SPAM pages in memory while the data pages are being written to the database. This approach saves an additional pass on the database to build the SPAM pages. An auxiliary pass reads some SPAM pages to reconstruct data structures that are used to simplify sequential scans on relations. Note that the auxiliary pass only reads the SPAM pages. Since the number of SPAM pages is typically less than one percent of the database size, this overhead is not very high.

Other Rdb/VMS Features

Rdb/VMS version 3.0 also includes "group-by" enhancements and statistics and analysis utilities.

A group-by expression in the SQL language divides a record stream into sections to allow for computation from each section. For example, a group-by aggregate might divide employees into subsets by department number and return the average, total, maximum, and minimum salaries for each department.

Earlier versions of Rdb/VMS have provided group-by functionality by performing multiple passes over the same stream. Version 3.0 supports group-by with a single pass over the relevant data.

The statistics utility gathers on-line statistics about

- I/O operations
- Lock usage
- Record fragmentation
- Index fetches and overflows
- Transaction durations
- System throughput

These statistics can be studied on-line or analyzed later to tune the database.

The analyze utility can be used to generate static statistics on data placement and storage utilization. This utility is useful for analyzing hash index behavior, file utilization and related aspects.

Summary

We have discussed several features that contribute to making Rdb/VMS a high-performance database system. We have discussed the multifile, partitioning, and record placement capabilities that play a fundamental role in eliminating the I/O bottleneck. Good database design and careful analysis are necessary to exploit these features. In the discussion, we have also highlighted trade-offs that a database designer is required to make.

Hash indexing and optimizer enhancements have a direct impact on the performance of queries at execution time. Hash indexing reduces the number of I/O operations as well as the locking activity in the system. The optimizer generates improved query plans that result in path length reduction, better use of system resources (such as buffers), and fewer I/O operations.

Finally, we presented some details of the fast backup and restore utility. We believe that the multithreaded design and implementation is a novel feature.

Developing Rdb/VMS into an industry leadership product is an ongoing process. We are extremely happy with the performance improvements of the current version. We will continue to design and implement new features that satisfy the needs of our customers.

Acknowledgments

As with any major product, Rdb/VMS version 3.0 is the result of the efforts of numerous engineers over the past few years. It is impossible to mention them all. We would like to thank Lou Dimino for his help with the section on the backup utility. Susan Hillson reviewed earlier drafts of this paper and made several suggestions for improving the presentation. Any errors and omissions in this article are entirely the responsibility of the authors.

References

1. T. Rengarajan, P. Spiro, and W. Wright, "High Availability Mechanisms of VAX DBMS Software," *Digital Technical Journal* (February 1989, this issue): 88–98.

2. C. Date, *An Introduction to Database Systems,* Vol. I (Reading: Addison-Wesley, 1985).

3. W. Snaman and D. Thiel, "The VAX/VMS Distributed Lock Manager," *Digital Technical Journal* (September 1987): 29–44.

4. J. Gray, "Notes on Database Operating Systems," *Computer Science Research Report RJ2188* (San Jose: IBM Research Laboratory, February 1978).

5. J. Carter and M. Wegman, "Universal Classes of Hash Functions," *Journal of Computer and System Sciences,* vol. 18, no. 2 (April 1979): 143–154.

6. R. Fagin et al., "Extendible Hashing — A Fast Access Method for Dynamic Files," *ACM Transactions on Database Systems,* vol. 4, no. 3 (September 1979): 315–344.

digital™

ISSN 0898-901X