N94-33816

Storage Media Pipelining: Making Good Use of Fine-Grained Media

Rodney Van Meter

ASACA Corporation
Tokyo, Japan
3-2-28 Asahigaoka, Hino-Shi, Tokyo, 191 Japan
rdv@alumni.caltech.edu

Abstract

This paper proposes a new high-performance paradigm for accessing removable media such as tapes and especially magneto-optical disks. In high-performance computing, striping of data across multiple devices is a common means of improving data transfer rates. Striping has been used very successfully for fixed magnetic disks, improving overall system reliability as well as throughput. It has also been proposed as a solution for providing improved bandwidth for tape and magneto-optical subsystems. However, striping of removable media has shortcomings, particularly in the areas of latency to data and restricted system configurations, and is suitable primarily for very large I/Os. We propose that for fine-grained media, an alternative access method, media pipelining, may be used to provide high bandwidth for large requests while retaining the flexibility to support concurrent small requests and different system configurations. Its principal drawback is high buffering requirements in the host computer or file server.

This paper discusses the possible organization of such a system, including the hardware conditions under which it may be effective, and the flexibility of configuration. Its expected performance is discussed under varying workloads, including large single I/Os and numerous smaller ones. Finally, a specific system incorporating a high-transfer-rate magneto-optical disk drive and autochanger is discussed.

1. Introduction

"Life does not give itself to one who tries to keep all its advantages at once."
Leon Blum
For Dr. Kim "Wombat" Korner, 1953-1993, a good and marvelously unconventional friend and teacher.

We propose that, for fine-grained media, a new access method, which we have dubbed **media pipelining**, can be used to dramatically increase the aggregate bandwidth available. Media pipelining operates much like pipelining in a CPU with multiple functional units[1], overlapping multiple requests (or portions of a single large request) to improve system throughput and resource utilization. Many of the analysis techniques applied to processor pipelines, including space-time diagrams and pipeline reservation tables can usefully be applied to media pipelining.

Pipelining can benefit single large jobs in a manner comparable to striping, while retaining the flexibility to accommodate smaller requests that striping may sacrifice. It also easily supports different system configurations, allowing the system to operate effectively with any number of drives. This flexibility also means that dynamically changing workloads can be handled effectively. The principal drawback to media pipelining is high buffering requirements in the file server or filesystem cache to achieve maximum throughput.

Section 2 presents some definitions for discussing the performance of such systems. Section 3 briefly explains striping for removable media. We present a contrasting discussion of pipelining in Section 4. Section 5 briefly covers host requirements for pipelining. Section 6 describes in detail one possible implementation, and section 7 presents our conclusions.

We define **granularity** as the ratio of the capacity of a medium to its transfer rate. The result is the amount of time it takes to read the entire medium. Some removable media have a very high capacity-to-bandwidth ratio. As an example, the ASACA AMD-1340NS HSMO disk drive, with a medium capacity of 600 MB per side and a transfer rate of 10 MB/s can read an entire medium in under one minute, which is approximately 4 times the time necessary for an autochanger to exchange the medium and a drive to perform its load and unload operations. We refer to such media as **fine-grained media**, as opposed to those whose read times may be on the order of hours (for example, the new optical tape has a capacity of one terabyte per reel, and a transfer rate of 3 MB/s, making a granularity of 3.3×10^5 seconds, or more than 900 hours), which we call **coarse-grained media.** Some example granularities of both common and experimental removable media are summarized in table 1 (the capacities for HSMO and ISO MO are for one side of a double-sided disk). (Note that this simple chart does not take into account drive type and host interface, which may result in different apparent granularities for the same medium.) It is easy to see that granularity varies by orders of magnitude. The impact of this feature on system design has not been fully explored.

Media Type	Capacity (MB)	Transfer Rate (MB/s)	Granularity (sec)
3480 tape	200	2	100
VHS T-120 tape	14,000	3	2,700
D-2 S tape	25,000	15	1,700
HSMO disk	600	10	60
ISO MO disk	300	0.6	500
optical tape	1,000,000	3	330,000

Table 1: Example Granularities of Removable Media

It is also sometimes desirable to talk about the effects of media granularity on the total performance of an autochanger system. In this case the important measure is the (dimensionless) ratio of the media granularity divided by the cartridge exchange time, G/t_x.

2. Assumptions and Definitions

We will discuss both striping and pipelining in the context of two different access patterns. The first is for an assumed linear scan of a very large ($>>c_m$) dataset. It will be analyzed primarily for its steady-state behavior rather than startup or latency. The principal metric is r_t, the total apparent throughput for a complete system.

The second workload is small requests located randomly in the entire available dataspace. In any removable-media system, with an average request size $O(r_m * t_r)$, both p and r_t are low; a better metric is x_n, the number of requests that can be serviced in a given time. Flexibility and the ability to support dynamically varying workloads will also be discussed.

We assume throughout this paper that the number of media in use is much larger than the largest possible stripe set; typically hundreds of media in a single autochanger (or "cart machine"). We further assume that it is desirable for this entire collection of media to be composed into a single dataspace (or a small number of large dataspaces). For fine-grained media this appears to be a reasonable assumption, freeing applications from managing data in chunks that may be unnatural and result in wasted capacity. However, this does force fixed addressing, as demonstrated later, making compressing media or other media with highly

variable capacity poor candidates for pipelining. This also essentially constrains the management of the media to automated handlers, but fine-grained media are unlikely to be used for human-handled dataset import/export anyway. While removing or adding media from/to the middle of the dataspace is impossible once the addressing is fixed, simple expansion is straightforward -- new addresses are allocated past the end of the existing dataspace.

If a user process is transferring very large amounts of data, >> c_m, a fine-grained system is to a certain extent handicapped. The apparent aggregate transfer rate using one drive, r_a, is limited to $r_m * p$, where

$$p = G/(G + t_x) \quad \text{for} \quad t_x = t_u + t_r + t_l + t_s.$$

It is clear that for coarse-grain media this ratio p is close to 1, meaning that over the long term for very large requests (>>c_m) that the cost of media exchanges is negligible. However, for fine-grained systems such as the ASACA HSMO, p may be significantly less than one. Thus, low granularity would appear to be a significant handicap; can it be turned into an advantage?

Abbr..	Description
B	buffer space necessary
c_m	capacity of a single medium
c_s	capacity of a stripe set = S * cm
p	percentage of time a drive spends transferring data
r_m	transfer rate of a single drive
r_s	transfer rate of a stripe set
r_p	transfer rate of a pipeline configuration
r_a	aggregate transfer rate including media exchange times
r_t	total throughput for a multi-drive system
t_m	robot cartridge move time
t_l	drive load time
t_u	drive unload time
t_i	time for the robot handler to actually insert/remove a medium
t_d	data transfer time
t_s	seek time
t_r	robot round trip time, perhaps 2*tm
t_x	time to exchange a medium, including eject, setup, and seek
n_b	number of blocks per medium
n_d	number of drives in system
S	striping factor (ignoring ECC additions)
s_b	block size (in bytes) of media
s_s	logical block size for stripe set = S * sb
s_p	logical block size for pipeline set = sb
x_n	request (transaction) rate (dimension #requests/time)
G	granularity = c_m / r_m (dimension is time)

Table 2: Definitions

3. The Shortcomings of Striping

One possible way to increase system throughput for large requests is to stripe the data across multiple media, increasing the available data size and multiplying the data rate. This approach is fine for fixed disks with stable configurations, but in a more dynamic system with removable media it presents severe management and use problems and may substantially increase the vulnerability of the user's data to access problems or media failures.

Hard disk systems that perform some form of striping must take steps to ensure the integrity of the data. The simplest approach, simply copying the data across two disks, improves the safety of the data but does not help either the transfer rate or the capacity. Larger RAID systems improve all of the above by using more than two disks and designating one or more disks to store error correction information[2].

Striping of data across two or more removable media is being investigated[3]. The principal problem with striping of removable media, especially in a robotic system designed to reduce the latency of access to the data, is that the whole set (perhaps, depending on the management scheme, minus the error control tape) must be on-line at once to read or write. Figure 1 shows the logical block layout of two stripe sets laid back to back as a single address space. A minor consideration is that the block size also goes up by a factor of S.

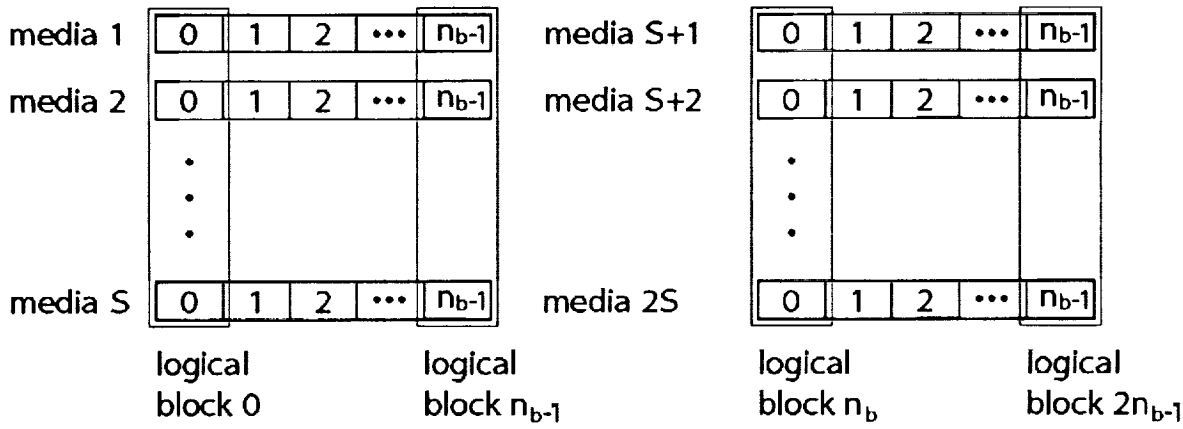


Figure 1: Logical Block Layout for Striped Media

Assuming intra-cart machine striping, bringing a stripe set online involves several operations by the robot to fetch multiple media, and forces the drives to sit idle while other drives in the set are loaded. Also, if the set consists of S striped tapes, if only S-1 drives are available at the time, the dataset may be unavailable. In a striped system drives must generally be allocated and used in sets of size S, meaning that the addition of a single drive does little good but the removal of a single drive may prevent access to data.

For a single cart machine servicing small requests, the maximum rate of requests that can be serviced, x_n, is $1/(t_r * S)$.

Inter-cart machine striping can be used to eliminate the increased latency for media access and increase the throughput of the system for small requests, but this again is very limiting in system configuration (requiring S cart machines exactly) and increases the vulnerability of the system to robot failures. It may be effective for small autochangers (10-tape stackers, for example) but becomes a very expensive solution for larger autochangers.

Figure 2 shows the access timing for a 4-way intra-cart machine stripe set.

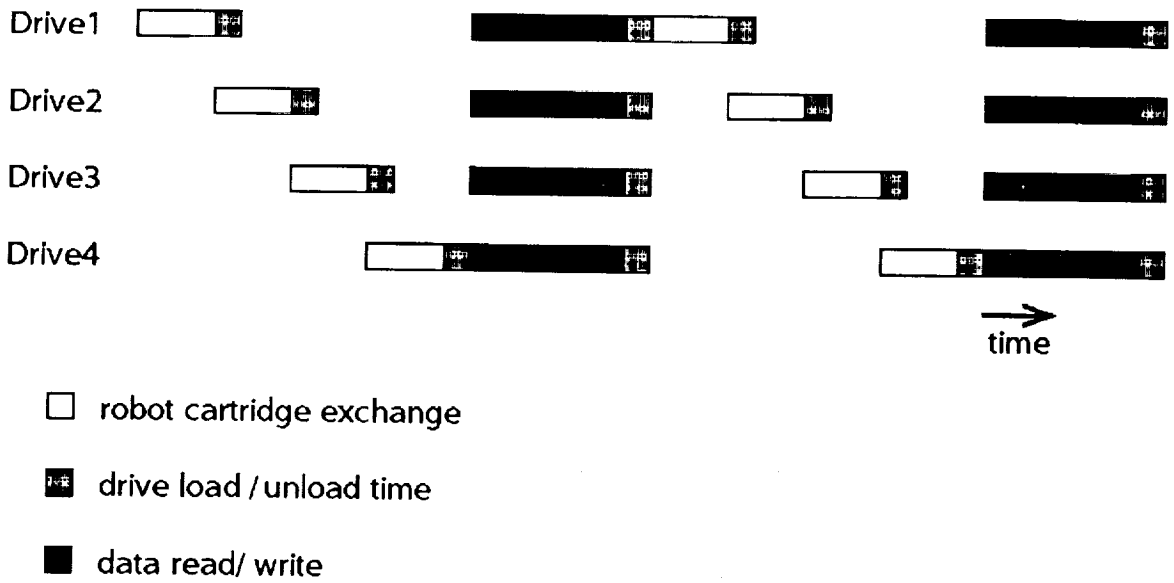


Figure 2: Striping of Removable Media

4. An Alternative Solution

One answer which appears to address some of the problems of striping for fine-grained media is **media pipelining**. It can use a higher percentage of available drive bandwidth, increasing total throughput for large requests, and retain the flexibility to accommodate small requests.

The logical block layout for media pipelining can be the "obvious" one, with blocks counting up on the first medium in the system and continuing consecutively across media boundaries, as shown in figure 3. A key assumption of this paper, as mentioned earlier, is the desirability of maintaining a single addressable space (the dataspace).

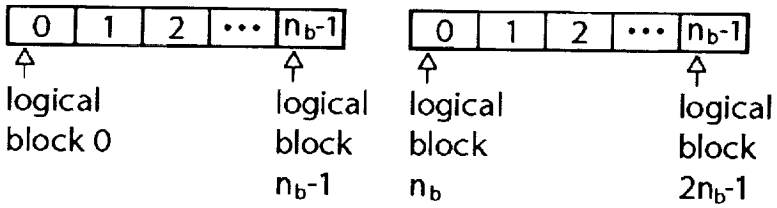


Figure 3: Logical Block Layout for Pipelined Media

The concept of media pipelining comes into play when requests move into the range of the media size c_m, on up to the terabyte range. How can multiple drives be used to increase the speed at which such requests are serviced? The simple answer is to recognize that the request spans a disk boundary, and preload the next disk in the request and have it ready to read when the first disk completes. This in itself is a simple form of pipelining, with the loading of a disk overlapping the reading of another. However, it commits two drives to the read and provides the effective bandwidth of one full drive. This we call **linear pipelining** (our definition is

somewhat different from that of Hwang and Briggs). Linear pipelining is illustrated in figure 4. Even in fine-grained systems, two drives should be sufficient to provide linear pipelining.

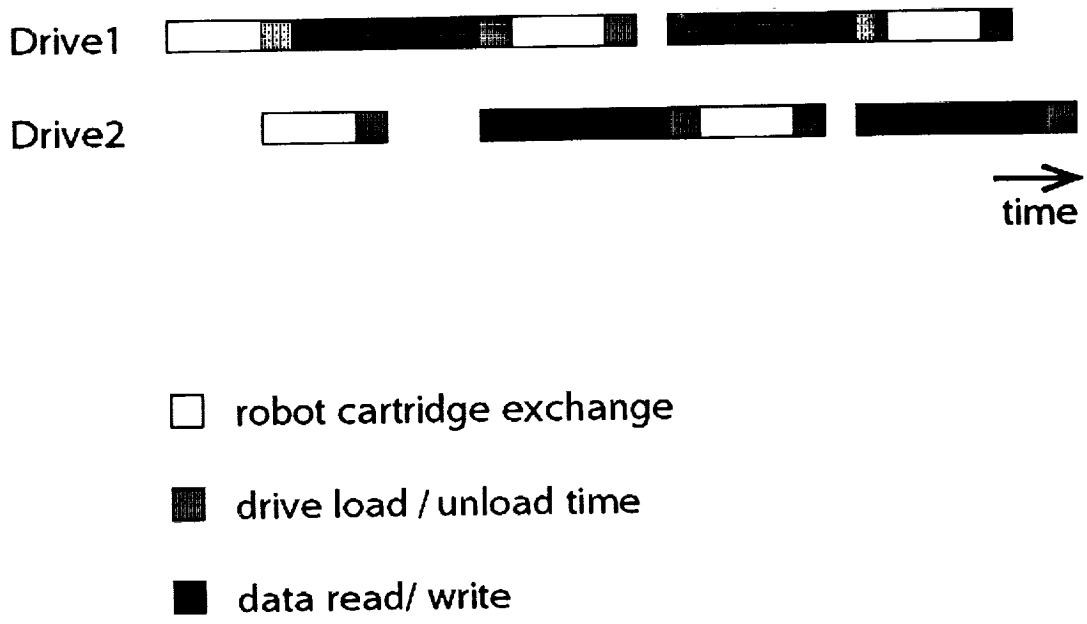


Figure 4: Linear Pipelining

It is possible, with a little care and appropriate driver software, to increase the drive utilization with pipelining by allowing the second drive to begin reading as soon as it comes on line. This we refer to as **superlinear pipelining**, as in figure 5. The diagram shows the overlapping reads, and below, a graph of the amount of data delivered to the application, assuming that the application is infinitely voracious but insists on data being delivered in order. It can be seen that both drives run at p efficiency, resulting in a total sustained throughput $r_t = 2 * r_a$.

The data that is read off of the second medium before the end of the first medium must be buffered (represented graphically as the gray area above the data delivered curve). It can be seen that this peaks right before the end of the first medium. The maximum amount of buffer space necessary is $B = (G - t_r) * r_m$. It should be readily apparent that this requirement can be reduced by having the system pause before starting the transfer from the second disk, reducing the total overlap. This obviously delays somewhat the delivery of the data, but may be acceptable depending on the rate at which the application is truly consuming data (in very long reads, the impact on average throughput would be negligible anyway). The most efficient configuration is of course device- and system-dependent.

As the size of the request grows, the full resources of the system can be brought to bear on the problem. For reads of $N * c_m$ or larger, we reach steady-state transfer rates utilizing the maximum percentage of the bandwidth of each drive.

A key feature of pipelining is its flexibility of configuration, allowing dynamically varying drive allocation depending on user needs and work load. For example, in a four-drive system, the first user to begin using the system may receive data at the full transfer rate, just as in a striped system. A second user entering the striped system may find his access to his data blocked, while the pipelined system can reconfigure (presumably on a medium boundary in the first transfer) dynamically, reallocating the drives 2-2 or 3-1 in favor of either user, allowing both to continue working in a manner very analogous to dynamic multiprocessor configurations. The system should autoconfigure, allowing up to as many users as there are

drives available (at one drive per user, this would be sublinear pipelining, equivalent to a "stalled" processor pipeline). It is not even a requirement that the different operations be the same kind of operation; one may be an excellent candidate for pipelining, one may be a concentrated series of operations within a single medium, while another is randomly placed reads and writes throughout the entire dataspace.

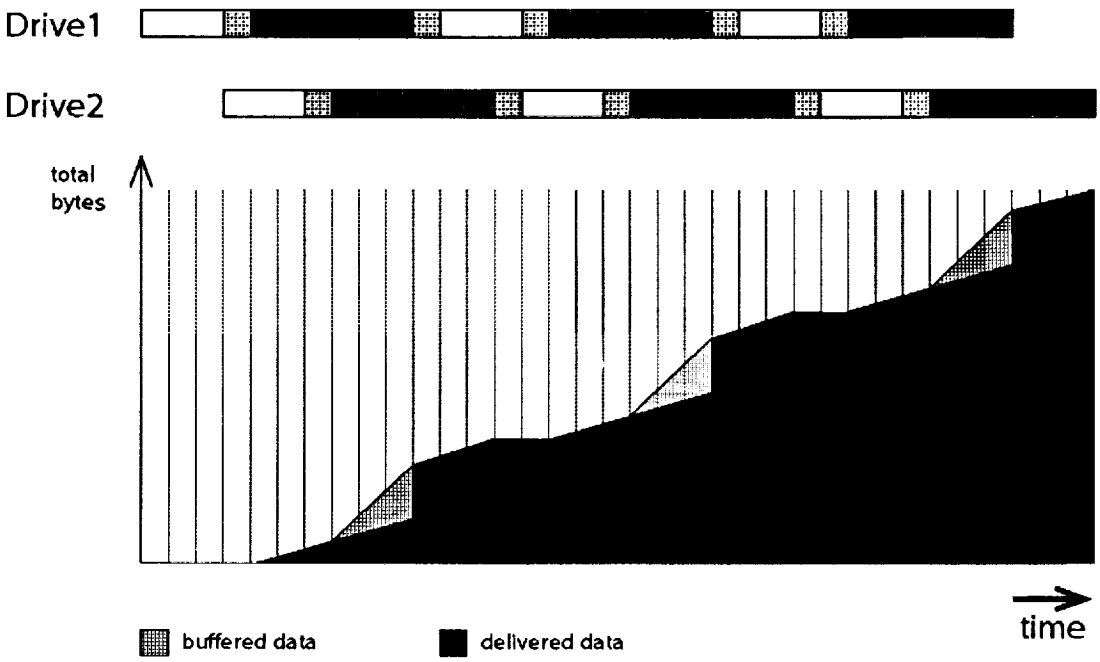


Figure 5: 2-Way Superlinear Pipelining

The pipeline overlap can be decreased when necessary (i.e. when the host cannot afford such a large buffer) for the simple tradeoff of reduced throughput for pipelined requests.

The system hardware configuration is extremely flexible with respect to the number of drives in the system. Unlike a striped system, a pipelined system can run comfortably with any number of drives, allowing drives to be moved, allocated, or maintained without necessarily forcing the unavailability of the entire system.

For small reads and writes (where "small" in this context may be less than a few hundred megabytes), $<< c_m$, the probability of the entire read residing on a single medium is high. In this case, the operation will be dominated by the media load time rather than the transfer time (as mentioned above, the load time becomes substantially longer in a striped system), negating any advantage in improved transfer rate from a stripe set. A single drive may be allocated to the request, leaving the other drives free for other operations, thus allowing the system to concurrently process as many requests as it has drives.

5. Operating System and File System Requirements

The operating system and file system have numerous demands made of them in a pipelined system. Both must be able to address large spaces. The device driver must be able to support multiple physical devices and manage and reassemble data as it comes in. Naturally the system should support transfers at such high rates.

If the requests to be pipelined arrive at the driver as a linear collection of smaller requests rather than a single huge request, the driver should still be able to pipeline the requests by prefetching data. If the file system provides "hints" it eases the process of determining when a

prefetch and pipeline setup will be worth the extra trouble. As discussed above, the system must be able to provide adequate buffer space, which should be readily available in a configuration with a minisupercomputer as a file server for a supercomputer.

High-speed interfaces are required to make such a system work. HIPPI and SCSI-2 and SCSI-3 are examples. The peak supported I/O burst rate (sum for the I/O busses used for pipelining; there is no requirement that all the drives be on the same bus) must be at least $n_d * r_m$ in order to maximize pipelining.

6. A Specific MO System

Evaluating real-world systems is of course substantially more complex than the abstract concepts presented above. Probably pipelining is best suited to removable disks, since their capacity is fixed, although uncompressed 3480 tape, with its low granularity, is also a possibility.

Asaca has developed the world's fastest magneto-optical disk drive, the AMD-1340N,[4] with a native data transfer rate of 12.24 megabytes/second, and a cartridge capacity of 1.2 GB (600 MB/side). These represent, respectively, twenty times and two times the values for most ISO-standard 5.25" MO drives. The speed advantage comes primarily from ASACA's 4-beam head technology, in which two heads each focus four lasers, for a total of eight beams lifting data off the disk concurrently. This tremendous speed improvement results in an entire side of a disk being readable in only 50 seconds. Using the SCSI-2 fast-wide interface, sustained transfer rates of approximately 10 MB/s are expected, and this is the number used throughout this paper.

The Asaca ADL-450 HSMO library contains 450 disks, 900 sides, 14,790 sectors of 40,448 bytes each, for a total capacity of 538 GB. It can hold up to four AMD-1340NS drives. Its potential in mass storage has already been discussed[5].

With the ASACA cart machine, the first disk comes on-line in approximately 15 seconds, and the second in approximately 23. The disk handler can hold two disks, meaning that during steady-state striping operations, the handler can prefetch the next disk while a drive is finishing a read and ejecting the disk, and have the second disk ready to load when the first is ejected. Thus, although the round-trip exchange time t_r is 15 seconds, the load time t_l is 8 seconds, the unload time t_u is 3 seconds, and the insert time is approximately two seconds, a new disk can be online in approximately 15 seconds $t_x = t_u + t_l + 2 * t_i$).
For $p = G/(G + t_x)$, using $G = 60$, $p = 0.8$, so during steady-state pipelining, we can expect to receive approximately 80% of the drive bandwidth.

Assuming that the driver makes the intelligent choice of mounting the disk with the most data on it first (a pipelining reordering operation), the worst case for a one gigabyte read is when the data is split 500 MB each on two disks. In that case, the read should be completed in 50 seconds of read plus the 23 to mount the disks, for an average aggregate throughput of 12 MB/s.

This transfer rate amounts to an aggregate of 32 MB/sec. across the four drives in a complete system. The difficulty is in managing the data, drives, and robot to provide fast access in a relatively transparent manner. A crucial part of the problem is coordinating multiple drives so that a user may take the best advantage of all the resources the system has to offer.

7. Conclusion and Future Work

We have presented here a new concept, the granularity of media, and shown how fine-grained media can be used in a method called media pipelining, which offers some advantages over striping for fine-grained removable media. It offers additional flexibility and improved response time compared to striping for small-request and dynamic workloads. For very large

requests, it can offer improved throughput compared to striping at the cost of high buffering requirements.

Further work calls for simulations and an implementation to verify predicted performance, increased formalization of the analytic model, and possibly extensions to the concept to allow pipelining to be used for coarse-grain media as well. The interaction of media pipelining with the host operating system also offers challenging work.

---

[1]Hwang, Kai and Briggs, Faye' A., *Computer Architecture and Parallel Processing*, McGraw-Hill, 1984, pp. 145-212.

[2]David A. Patterson, Garth Gibson, and Randy H. Katz, "A Case for Redundant Arrays of Inexpensive Disks" *Proc. ACM SIGMOD*, June 1988.

[3]Drapeau, Ann L. and Katz, Randy H., "Striped Tape Arrays," *Proc. Twelfth IEEE Symposium on Mass Storage Systems*, Monterey, CA, April 1993.

[4]Nakagomi, Takashi et al, "Development of High Speed Magneto-optical Disk Drive Using 4 Beam Optical Head," *IEEE Translation J. on Magnetics in Japan*, Vol. 6, No. 3, p. 250, March 1991.

[5]Nakagomi, Takashi, et all, "Re-Defining the Storage Hierarchy: An Ultra-Fast Magneto-Optical Disk Drive," *Proc. Twelfth IEEE Symposium on Mass Storage Systems*, Monterey, CA, April 1993.

The Trend to Parallel, Object-Oriented DBMS

David J. DeWitt
Professor and Romnes Fellow
Computer Sciences Department
University of Wisconsin
1210 W. Dayton Street
Madison, WI 53706
Phone: (608) 263-5489 / (608) 262-1204
FAX: (608) 265-2635

1

Background

- Supercomputers users and vendors are finally discovering the importance of I/O!
- Recently I read a paper titled "Satisfying the I/O Requirements of Massively Parallel Supercomputers"
- Nice paper, but not a single reference to any work in the parallel database system field
- I found this:

2

AMAZING!

3

Why Amazing?

- Parallel DBMS community has been working on this problem for 15 years and essentially has it solved
- Teradata has systems in the field with over 300 processors and 1000 disk drives!
- Other vendors include NCR/Sybase, Tandem, IBM SP2, & DEC (soon)
- All vendors use the same basic architecture
- None of the supercomputer vendors use it.

4

What am I going to talk about?

I wondered the same thing when I saw the program for this conference

5

Talk Outline

- Hardware and software architectures used by today's relational DBMS products
- DBMS trends - the transition from relational to object-oriented
- What is an object-oriented (OO) DBMS?
- OODBMS and standards such at NetCDF
- Future OODBMS directions
- 1 slide sales pitch

6

Database Systems Today

- Relational data model and SQL dominate
- Targeted at commercial applications
- A relational database: set of relations
- A relation: a set of homogenous tuples

Telephone_Book

Name	Address	Number
Jones	110 Main St	255-4834
Smith	2164 Lake Lane	238-5936
Smith	5 Roby Rd	746-0192

- SQL used to create, update, and query

```
SELECT Number
FROM Telephone_book
WHERE Name = "Smith"
```

7

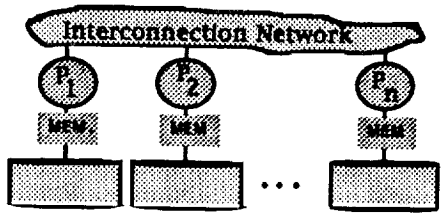
Relational Database Systems

- SQL is easy to optimize and parallelize
- Terabyte databases, consisting of billions of records, are becoming common
- Databases of this size require the use of parallel processors
- Teradata and other commercial parallel DBMS employ what is termed a shared-nothing architecture

8

Shared-Nothing

- Each memory and disk is owned by some processor that acts as a server for that data

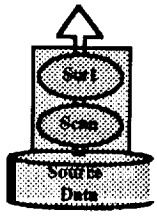


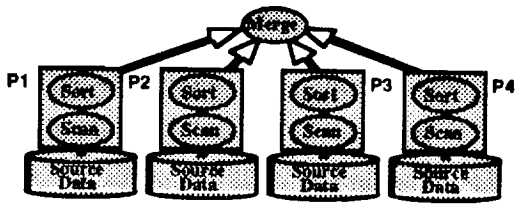
- Teradata, NCR 3600, Tandem, IBM SP2
- Actual interconnection network varies: trees, hypercubes, meshes, rings, ...

9

Relational DBMS Parallelism

- 3 key techniques: pipelining, partitioned execution, & data partitioning
- Pipelined parallelism
- Partitioned execution





Telephone_Book Relation

10

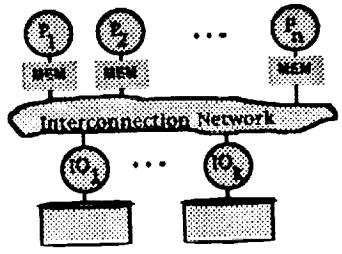
Relational DBMS Summary

- Shared-nothing approach has proven to scale very successful providing both linear speedup and scaleup on I/O intensive applications
- The largest Teradata systems have over 300 processors and 1000 disk drives (1 terabyte)
- While NASA has lots of satellite data, K-Mart and WalMart have lots of cash registers!
- Despite all the talk about an I/O bottleneck, the vendors supplying parallel processors to the scientific community are not following suit.
- What are they doing?

11

Dedicated I/O Nodes (Shared-Disk)

- Each processor has a private memory and access to all disks



- CM-5, Intel Paragon, Cray T3D, IBM 3090
- DBMS community have rejected such architectures
  - coordinating access to shared data is complex
  - extra cost required for I/O nodes and their network interfaces

12

Common Gripes about Shared-Nothing

- **Packaging problems**
  - **total nonsense. Teradata is sufficient proof**
  - **by 2000, 1", 1 gigabyte drives will be common**
- **Interference with application code**
  - **Assume:**
    - » 50 MIP cpu
    - » 20 ms. to do a disk I/O
    - » Each "remote" disk request consumes 3000 instructions locally (1000 to accept message, 1000 to start I/O, 1000 to send page back to requestor)
  - **So every 20 ms., 3000 instructions are stolen from application. These 3000 instructions account for 0.3% of the available CPU cyles!!!**
- **The future of parallel computing may be commodity computers connected by commodity networking technology (e.g. ATM)**

13

Another Observation

- **DB community has totally accepted message passing for both parallel computation & parallel I/O**
- **Scientific community has accepted message passing as the standard communication mode (though HPF may hide a lot of ugly details)**
- **But, is holding on to a "shared-disks" architecture for the parallel I/O system**

14

Talk Outline

- Hardware and software architectures used to today's relational DBMS products
- DBMS trends - the transition from relational to object-oriented
- What is an object-oriented (OO) DBMS?
- OODBMS and standards such at NetCDF
- Future OODBMS directions
- 1 slide sales pitch

15

The Transition from Relational to Object-Oriented

- Why?
- Relational DBMS:
  - Modeling capabilities too limited:
    - » Tuples (records) of base types only!
    - » No arrays let alone polygons or polylines
    - » No nested tuples or structure-valued attributes
  - Application interface (i.e. SQL with cursors) is simply wrong for manipulating scientific or CAD data
    - » CAD applications love to chase pointers around
  - No support for tertiary storage

16

What is an Object-Oriented DBMS?

- The marriage of a modern programming language such as C++ and a modern DBMS.
- From the programming language world:
  - Rich type system including classes with encapsulation and inheritance
  - Computational completeness
- From the DBMS world:
  - Persistence
  - Bulk types (sets, lists)
  - Transactions (concurrency control and recovery services)
  - Associative queries (balance < $100)
- Transparent Access to Persistent Objects

17

Example

- Given the following type definition

```
class raster_data {
    int     time;
    int     frequency;
    float   image[4096][4096];
}
```

- Can declare persistent variables of this class:

```
persistent raster_data X, Y;
```

- Transparent access to data on disk:

```
for (i=0;i<4096;i++)
    for (j=0;j<4096;j++)
      Y[i][j] = f(X.image[i][j]);
```

- A year's worth of data:

```
persistent raster_data GeosDataSet[365];
```

18

OODBMS & Standards like NetCDF

- Data model provided by a typical OODBMS is much more general than that provided by a standard
  - Typical OODBMS, in addition to arrays and records, provide sets, lists, and relationships as type constructors
- Transparent access to persistent data makes manipulation of data residing on secondary and tertiary storage trivial
  - Current products have sufficient performance to satisfy even the most demanding CAD applications
- Persistent objects are strongly typed with their type descriptors stored as persistent objects in the database

19

Talk Outline

- Hardware and software architectures used to today's relational DBMS products
- DBMS trends - the transition from relational to object-oriented
- What is an object-oriented (OO) DBMS?
- OODBMS and standards such at NetCDF
- Future OODBMS directions
- 1 slide sales pitch

20

Future OODBMS Directions

- **Standardization via either ODMG or SQL3**
- **Integrated support for tertiary storage**
- **Extension to parallel processors**
  - **Current products architected for client-server environments**
  - **Only "small" databases supported: 10s of gigabtyes and not terabytes**
  - **Two possible directions**
    - » Relational products will adopt a richer type system such as SQL3
    - » OODBMS products will be extended to operate on parallel processors
    - » Joint project between KSR and Intellitic to parallelize Matisse is the first such effort

21

Parallel-Sets (ParSets)

- **Proposed by Kilian, basis of Matisse/KSR effort**
- **Employs a data-parallel approach to object-oriented parallel programming**
- **ParSets extend set type constructor as follows:**
  - **ParSets are partitioned across multiple processors/disks to facilitate CPU and I/O parallelism**
  - **Provide 4 basic operations:**
    - » Add()
    - » Remove()
    - » SetApply() - invokes a method on all the objects in parallel
    - » Reduce() function - calculates a single value from all objects in the ParSet
- **Most promising proposal to date. Can be extended to other bulk types such as arrays, lists, trees, etc.**

22

Sales Pitch - What we are doing

- Shore - public domain, object manager (ARPA)
  - Data model based on ODMG standard
  - Support for client-server and parallel processors
  - Parallelism via ParSets
- Paradise (NASA)
  - Parallel information system for managing large EOSDIS data sets
  - Uses GEO as a front-end
  - Uses Shore object manager for storing persistent data

23

Conclusions

- Parallel relational database systems are a proven technology with solutions to the I/O bottleneck problem
- Future is likely to witness a merger of technologies developed for relational and object-oriented database
- The use of object-oriented database systems with typed persistent objects reduces the needs for restrictive standards such as NetCDF
- Extensions to HPF to provide transparent access to persistent arrays (termed "out-of-core" arrays) are on the drawing boards

24