Active Storage For Large-Scale Data Mining and Multimedia Applications

Erik Riedel[1], Garth Gibson, Christos Faloutsos[2]

February 1998
CMU-CS-98-111

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

{riedel,garth,christos}@cs.cmu.edu

Abstract

The increasing performance and decreasing cost of processors and memory are causing system intelligence to move into peripherals from the CPU. Storage system designers are using this trend toward "excess" compute power to perform more complex processing and optimizations inside storage devices. To date, such optimizations have been at relatively low levels of the storage protocol. At the same time, trends in storage density, mechanics, and electronics are eliminating the bottleneck in moving data off the media and putting pressure on interconnects and host processors to move data more efficiently. We propose a system called Active Disks that takes advantage of processing power on individual disk drives to run application-level code. Moving portions of an application's processing to execute directly at disk drives can dramatically reduce data traffic and take advantage of the storage parallelism already present in large systems today. We discuss several types of applications that would benefit from this capability with a focus on the areas of database, data mining, and multimedia. We develop an analytical model of the speedups possible for scan-intensive applications in an Active Disk system.We also experiment with a prototype Active Disk system using relatively low-powered processors in comparison to a database server system with a single, fast processor. Our experiments validate the intuition in our model and demonstrate speedups of 2x on 10 disks across four scan-based applications. The model promises linear speedups in disk arrays of hundreds of disks, provided the application data is large enough.

[1] Department of Electrical and Computer Engineering
[2] On leave from the University of Maryland, College Park, MD 20742

This research is sponsored by DARPA/ITO through ARPA Order D306, and issued by Indian Head Division, NSWC under contract N00174-96-0002. Additional support was provided by NSF under grants EEC-94-02384 and IRI-9625428 and NSF, ARPA and NASA under NSF Cooperative Agreement IRI-9411299. We are indebted to generous contributions from the member companies of the Parallel Data Consortium. At the time of this writing, these companies include Hewlett-Packard Laboratories, Symbios Logic, Data General, Compaq, Intel, Quantum, Seagate Technology, Wind River Systems, and Storage Technology Corporation. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any supporting organization or the U.S. Government.

ACM Computing Reviews Keywords: B.4.2 Input/output devices, H.2.8 Database applications, D.4.7 Distributed systems, C.3.0 Special-purpose and application-based systems, B.4 Input/Output and Data Communications.

1. Introduction

In this paper we evaluate the performance advantages of exploiting the processors embedded in individual storage device for some of the data-intensive applications common in data mining and multimedia databases. This system is architecturally similar to the processor-per-disk database machines dismissed in the literature 15 years ago as expensive and unnecessary. In the intervening years, however, technology trends have made possible commodity storage devices with excess general-purpose computational power and application trends are emphasizing massive, complex data sets commonly processed with scans. It may soon be possible for collections of commodity storage devices to couple parallel processing and high-selectivity filtering to dramatically reduce execution time for many of these applications.

On the technology side, general purpose 32-bit microcontrollers with 100-200 MHz processing speeds are common in disk array controllers and being are already being incorporated into high-end commodity disk drives. Moreover, vendors of storage devices would welcome new uses for this largely underutilized processing power if it allows their products to compete on metrics beyond simple capacity and cost ($/MB). We propose a storage device called an *Active Disk* that combines in-the-field software downloadability and recent research progress in safe remote execution of code to execute application-level function directly at the device.

In this paper, we emulate an Active Disk with a six year old workstation and contrast host-resident to Active-Disk-assisted processing of four applications: nearest neighbor search in a high dimensionality database, frequent set counting to discover association rules, edge detection in images, and image registration in a medical database. These applications all process large volumes of data, ensuring substantial storage parallelism simply to accommodate the volume of data, and often operate with a relatively small number of instructions per byte of storage accessed. The processing in all these applications is scan-intensive, either due to the nature of the application and data, or because the high-dimensionality queries to be processed are not accelerated by traditional indices.

Active Disks benefit I/O-bound scans in two principle ways: 1) parallelism - massive amounts of data partitioned over many active disks allows “embarrassingly parallel” scans to convert a group of Active Disks into a programmable parallel-scan database machine. 2) bandwidth reduction - scans that filter data with a high degree of selectivity or compute only summary statistics transfer a very small fraction of the data’s size from the active disks to the host. For highly selective scans a group of Active Disks can process data at the aggregate disk rate in a machine whose interconnect bandwidth was designed for much less bandwidth-demanding applications.

Section 2 compares this work with past research on database processing performed at storage (i.e. database machines), discusses the trends in storage systems that have brought us to this point, and motivates the areas of data mining and multimedia as fertile ground for applications of Active Disks. Section 3 provides an analytical model to illustrate the potential benefit of using Active Disks and give some intuition on the speedups possible. Section 4 outlines the four representative applications we have chosen for detailed study. Section 5 describes our experimental setup and compares the performance of an existing server system to a prototype system using Active Disks.

Section 6 further explores issues of performance and the characteristics of applications that make them successful on Active Disks. Finally, Section 7 concludes and briefly discusses areas of future work.

2. Background

The prevailing counter-arguments to the database machines of the 80s were that 1) for a significant fraction of database operations (such as sorts and joins) simple select filters in hardware did not provide significant benefits, 2) special purpose hardware development increased the design time and cost of the storage device, and 3) a single general purpose host processor was sufficient to execute select at the full data rate of a single disk [DeWitt81, Boral83].

Boral and DeWitt concluded that aggregate storage bandwidth was the principle limitation of database machines. Fortunately, as shown in Table 1, in the intervening years aggregate storage bandwidth has dramatically improved. The improvement comes from disk array hardware and software that enable individual database operations to exploit disk parallelism [Livny87, Patterson88] and because databases are now large enough to justify hundreds of disks. Moreover, high-end disk data rates are now 15 MB/s sustained [Seagate97] and continue to grow at 40% per year [Grochowski96]. In place of raw disk bandwidth limitations, modern systems instead have a limited peripheral interconnect bandwidth, as seen in the system bus column of Table 1. We see that more MB/s can be read into the memory of a large collection of disk controllers than can be delivered to a host processor. In this case, the power of the host processor is irrelevant to the overall bandwidth limitation for large scans.

If we consider the cost and complexity of special purpose hardware in database machines, technology trends again change the trade-offs. The increasing transistor count possible in inexpensive CMOS microchips today is driving the use of microprocessors in increasingly simple and

System	Component	Processor	On-Disk Processing	System Bus	Storage Throughput
Compaq TPC-C	Compaq ProLiant 7000 6/200	800 MHz	2,800 MHz	133 MB/s (?)	1,130 MB/s
	4 200 MHz Pentiums, 1 PCI	(4 x 200 MHz)			
	113 disks = 708 GB		(113 x 25 MHz)		(113 x 10 MB/s)
Microsoft TerraServer	Digital AlphaServer 4100	1,600 MHz	8,000 MHz	532 MB/s	3,200 MB/s
	4 400 MHz Alphas, 2 64-bit PCI	(4 x 400 MHz)			
	320 disks = 1.3 TB		(320 x 25 MHz)		(320 x 10 MB/s)
Digital TPC-C	Digital AlphaServer 1000/500	500 MHz	1,525 MHz	266 MB/s	610 MB/s
	500 MHz Alpha, 64-bit PCI				
	61 disks = 266 GB		(61 x 25 MHz)		(61 x 10 MB/s)
Digital TPC-D	Digital AlphaServer 4100	1,864 MHz	2,050 MHz	532 MB/s	820 MB/s
	4 466 MHz Alphas, 2 64-bit PCI	(4 x 466 MHz)			
	82 disks = 353 GB		(82 x 25 MHz)		(82x 10 MB/s)

Table 1: If we assume that current disk drives have the equivalent of 25 MHz of host processing speed available, large database systems today already contain more processing power on their combined disks than at the server processors. Assuming a reasonable 10 MB/s for sequential scans, we also see that the aggregate storage bandwidth is more than twice the backplane bandwidth of the machine in almost every case. Data from [TPC98] and [Barclay97].

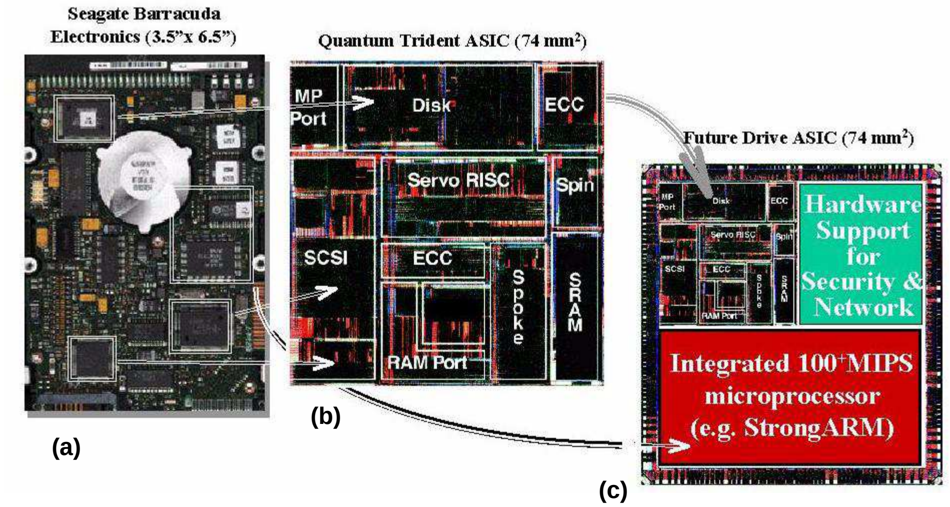

Seagate Barracuda
Electronics (3.5”x 6.5”)
Quantum Trident ASIC (74 mm^2)
MP Port
Disk
ECC
Servo RISC
Spin
SCSI
ECC
Spoke
SRAM
RAM Port
Future Drive ASIC (74 mm^2)
Hardware Support for Security & Network
Integrated 100$^+$MIPS microprocessor (e.g. StrongARM)
(a)
(b)
(c)

Figure 1: The trend in drive electronics is toward higher levels of integration. The Barracuda drive on the left contains separate chips for servo control, SCSI processing, ECC, and the control microprocessor. The Trident disk in the center has combined many of the individual specialized chips into a single ASIC, and the next generation of silicon makes it possible to both integrate the control processor and provide a significantly more powerful embedded core while continuing to reduce total chip count and cost [TriCore97].

inexpensive devices. Network interfaces, peripheral adapters, digital cameras, graphics adapters, array controllers and disk drives all have microcontrollers whose processing power exceeds the host processors of 15 years ago. For example, Quantum’s high-end disk drives already contain a 40 MHz Motorola 68000-based controller that provides the high-level functions of the drive.

In Figure 1 we show the effects of increasing transistor counts on disk electronics. Figure 1a reminds us that the electronics of a disk drive include all the components of a simple computer: a microcontroller, some amount of RAM, and a communications subsystem (SCSI), in addition to the specialized hardware for drive control. Figure 1b shows that this special control hardware has already been largely integrated into a single chip in current-generation disks. Extrapolating to the next generation of technology (from .68 micron to .35 micron CMOS in the ASIC), the specialized drive hardware occupies about one quarter of the chip and there is sufficient area to include a 200 MHz Digital StrongARM microprocessor [Turley96], for example. Commodity disk and chip manufacturers are already pursuing processor-in-ASIC technology. For example, Siemens has announced chips that offer a 100 MHz 32-bit microcontroller, up to 2 MB of on-chip RAM with up to 800 MB/s onchip bandwidth, external DRAM and DMA controllers and customer-specific logic (that is, die area for the functions of Figure 1b) in a .35 micron CMOS process [TriCore97]. Fundamentally, VLSI technology has evolved to the point that significant additional computational power comes at negligible cost.

Processing power inside drives and storage subsystems has already been successfully used to optimize underneath standardized interfaces such as SCSI. Such optimizations include many innovative optimizations for storage parallelism, bandwidth and access time [Patterson88, Drapeau94, Wilkes95, Cao94, Storagetek94] and for distributed file system scalability [Lee96, VanMeter96, Gibson97]. With Active Disks, excess computation power in storage devices is available directly for application-specific function in addition to supporting these existing optimizations. Moreover, instead of etching database functions into silicon as envisioned 15 years ago, Active Disks are programmed in software and use general purpose microprocessors.

Of course there are significant implications for language, safety, and resource management when downloading application code directly into devices [Riedel97]. However, a number of efficient and safe remote execution facilities are becoming available that provide innovative ways to ensure proper execution of code and safeguard the integrity of the drive [Gosling96, Necula96, Romer96, Bershad95, Small95, Wahbe93][1].

The third objection to database machines was the limited utility of full scan operations. However, a variety of emerging applications require sequential scanning over large amounts of data. We focus on two sets of applications: multimedia and data mining. In multimedia, applications such as searching by content [Flickner95, Virage98] are particularly good candidates. The user provides a desirable image and requests a set of similar images. The general approach to such a search is to extract feature vectors from every image, and then search these feature vectors for nearest neighbors [Faloutsos96]. The dimensionality of these vectors may be high (e.g. moments of inertia for shapes [Faloutsos94], colors in histograms for color matching, or Fourier coefficients). It is well-known [Yao85], but only recently highlighted in the database literature [Berchtold97], that for high dimensionalities, sequential scanning is competitive with indexing methods because of the "dimensionality curse." This is counter-intuitive since conventional database wisdom is that indices always improve performance over scanning. This is true for low dimensionalities, or for queries on only a few attributes. However, in high dimensionality data and nearest neighbor queries, there is a lot of "room" in the address space and the data points are far from each other. The two major indexing methods, grid-based and tree-based, both suffer in high dimensionality data. Grid-based methods require exponentially many cells and tree-based methods group similar points together, resulting in groups with highly overlapping bounds. One way or another, a nearest neighbor query will have to visit a large percentage of the database, effectively reducing the problem to sequential scanning. This is exactly the idea behind recent high-dimensionality indexing methods such as X-trees [Berchtold96] which deliberately revert to sequential scanning for high dimensionalities. In data mining, algorithms such as association discovery and classification also require repeated scans of the data [Agrawal96].

In addition to supporting complex queries, trends are toward larger and larger database sizes. One hour of video requires approximately 1 GB of storage and video databases can easily contain over 1 TB of data such as daily news broadcasts [Wactlar96]. Such databases can be searched by content (video, text, or audio) and require both feature extraction and a combination of the searching algorithms mentioned above. Medical image databases also impose similarly heavy data requirements [Arya94]. In data mining applications, point-of-sale data is collected over many months and years and grows continually. Telecommunication companies maintain tens of TB of historical call data.

[1] Another, simpler, option would be to add standard memory management hardware at the drive and provide protected address spaces for applications executing inside the drive. The relative merit of these approaches is clearly a topic of further research.

3. Basic Approach

The basic characteristics of successful remote functions for Active Disks are that they 1) can leverage the parallelism available in systems with large numbers of disks, 2) operate with a small amount of state, processing data as it “streams past” from the disk, and 3) execute a relatively small number of instructions per byte.

In this section we develop an analytical model for the performance of such applications. The purpose of this model is to develop an intuition about the behavior of Active Disk systems relative to processing in a traditional server[1]. We use the following symbols in our model:

Application Parameters	System Parameters	Active Disk Parameters
N_{in} = number of bytes processed	s_{cpu} = CPU speed of the host	s_{cpu}' = CPU speed of the disk
N_{out} = number of bytes produced	r_d = disk raw read rate	r_d' = active disk raw read rate
w = cycles per byte	r_n = disk interconnect rate	r_n' = active disk interconnect rate
t = runtime for traditional system		
t_{active} = runtime for active disk system		

Traditional vs. Active Disk Ratios

$$\alpha_N = N_{\text{in}}/N_{\text{out}} \qquad \alpha_d = r_d'/r_d \qquad \alpha_n = r_n'/r_n \qquad \alpha_s = s_{\text{cpu}}'/s_{\text{cpu}}$$

To keep the model simple, we assume that our applications have the three characteristics mentioned above and that disk transfer, disk computation, interconnect transfer and host computation can be pipelined and overlapped with negligible startup and post-processing costs, and that interconnect transfer rates exceed disk read rates.

Starting with the traditional server, overall runtime is the largest of the individual pipeline stages: disk read time, disk interconnect transfer time, and server processing time which gives:

$$t = max\left(\frac{N_{\text{in}}}{d \cdot r_d}, \frac{N_{\text{in}}}{r_n}, \frac{N_{\text{in}} \cdot w}{s_{\text{cpu}}}\right) \text{ and } \text{throughput} = \frac{N_{\text{in}}}{t} = min\left(d \cdot r_d, r_n, \frac{s_{\text{cpu}}}{w}\right)$$

For the Active Disks system, we derive the comparable times for disk read, interconnect transfer, and on-disk processing stages:

$$t_{active} = max\left(\frac{N_{\text{in}}}{d \cdot r_d}, \frac{N_{\text{out}}}{r_n'}, \frac{N_{\text{in}} \cdot w}{d \cdot s_{\text{cpu}}'}\right) \text{ and } \text{throughput}_{\text{active}} = \frac{N_{\text{in}}}{t_{active}} = min\left(d \cdot r_d', r_n' \cdot \frac{N_{\text{in}}}{N_{\text{out}}}, d \cdot \frac{s_{\text{cpu}}'}{w}\right)$$

Each of these throughput equations is a minimum of three limiting factors: the aggregate disk bandwidth, the storage interconnect bandwidth, and the aggregate computation bandwidth.

[1] We use a single host processor as the “traditional system” as a matter of convenience. This parameter could also be the equivalent performance of a multiprocessor, but the additional cost of the more complex system would have to be considered.

If we rewrite the equation for throughput with Active Disks in terms of the parameters of the traditional server and the ratios between traditional and Active Disk system parameters - the total data moved (the selectivity α_N), the disk bandwidth (α_d, which should be 1), the interconnect bandwidth (α_n), and the relative CPU power (α_s), we have:

$$\text{throughput}_{\text{active}} = min\left(\alpha_d \cdot (d \cdot r_d), \alpha_N \cdot \alpha_n \cdot (r_n), d \cdot \alpha_s \cdot \left(\frac{s_{\text{cpu}}}{w}\right)\right)$$

This equation captures the basic advantages of active disks: applications with high selectivity (large α_N) experience less restrictive interconnect limitations, and configurations with many disks ($d \cdot \alpha_s > 1$) can achieve effective parallel processing.

3.1. Estimating System Ratios

The kinds of applications we discuss here exhibit selectivities (α_N) of 100 to 10^8 or more, providing throughput possible only with essentially infinite interconnect bandwidth in the traditional system. Practically, this allows system cost to be reduced with lower bandwidth interconnects while maintaining a high throughput for these types of applications, so we allow for slower Active Disk interconnects $0.1 < \alpha_n < 1.0$. Active Disk processors will be slower than traditional server CPUs. In order to make this difference significant for purposes of the model, we assume that Active Disks cannot scan data at disk rates. That is, we assume $s_{\text{cpu}}' / w < r_d'$.

The final and critical system parameter is the ratio of Active Disk to server processor speed. We expect 100 and 200 MHz microcontrollers in near-term high-end drives, and individual server CPUs of 500 to 1,000 MHz in the same time frame, so a ratio of about $\alpha_s = 1/5$ may be practical. In this case, Active Disk processing power exceeds server processing power once there are more than about 5 disks working in parallel.

3.2. Implications of the Model

Figure 2 illustrates basic trade-offs for Active Disk systems. The slope of line A represents the raw disk limitation in both systems. Because we expect Active Disks cannot keep up with their disk transfer rates for many applications ($s_{\text{cpu}}' < w \cdot r_d'$), their aggregate throughput will have the somewhat lower slope shown by line B on the chart.

Active Disks saturate their interconnects at line C, with $\text{throughput} = r_n' \cdot \alpha_N$. Because interconnect bandwidth is typically greater than single disk bandwidth and Active Disks cannot keep up with their raw read rate in saturation, the number of disks must be larger than the selectivity of the application before this limit sets in, as shown to the right of point Z on the chart. With the large selectivities of the applications discussed here, we would expect our perfect overlap assumption to fail (Amdahl's Law) before this point is reached.

Traditional server systems are likely to exhibit both interconnect and server CPU bottlenecks, represented by line D in the figure. If the server CPU is the principle bottleneck, the number of

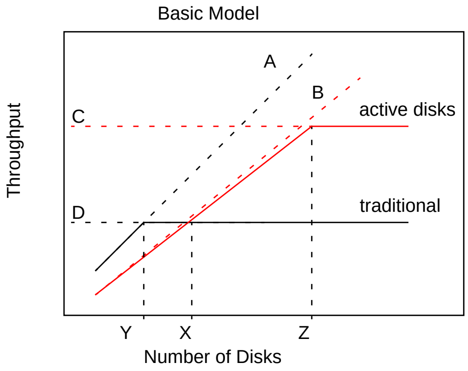


Figure 2: Simple model of the throughput of an application running in an active disk system compared to a traditional single server system. There are several regions of interest, depending on characteristics of the application and the underlying system configuration. The raw media rate of the disks in both cases is plotted as line A. The raw computation rate in the active disk system is line B, which varies by application. Line C shows the saturation of the interconnect between the disks and host, which varies with the selectivity of the application processing. Line D represents the saturation of the server CPU in the traditional system, above which no further gain is possible as disks are added. To the left of point Y, the traditional system is disk-bound. Below the crossover point X, the active disk system is slower than the server system due to its less powerful CPU. Above point Z, even the active disk system is network-bottlenecked and no further improvement is possible.

disks at which an active disks system crosses over to higher throughput is $1/\alpha_s$ as discussed above and shown by point X in the figure. However, if the principle bottleneck is the interconnect, then the crossover occurs at a lower throughput and therefore a lower number of disks.

If we combine all of the above analysis and define speedup as Active Disk throughput over server throughput, depending on the traditional system's principle bottleneck, we have:

if the traditional system is interconnect limited for $1/\alpha_s < d < \alpha_N$ speedup is

$$\frac{d \cdot (s_{\text{cpu}}'/w)}{r_n} > 1$$

if the traditional system is CPU limited, then for $1/\alpha_s < d < \alpha_N$ speedup is the ratio of the CPU bottlenecks

$$\frac{d \cdot (s_{\text{cpu}}'/w)}{(s_{\text{cpu}}/w)} = d \cdot \alpha_s$$

when active disks hit saturation above point Z in the chart, speedup is

$$\frac{(r_n' \cdot \alpha_N)}{min\left(r_n, \frac{s_{\text{cpu}}}{w}\right)} = max\left(\alpha_N \cdot \alpha_n, \alpha_N \cdot \alpha_s \cdot \left(\frac{w \cdot r_n'}{s_{\text{cpu}}'}\right)\right)$$

$$> \alpha_N \cdot max(\alpha_n, \alpha_s)$$

With our assumption above of $\alpha_s = 1/5$, the speedup at the double saturation point is at least a factor of 20, for the applications discussed here.

We do not consider the "slowdown" of Active Disks when $d < 1/\alpha_s$ (the area to the left of point X in the figure) because this condition is independent of the application parameters, so a query optimizer can determine *a priori* when to prefer traditional execution of the scan for a particular system configuration, rather than executing the scan at the drives.

Finally, if we consider the prevailing technology trends, we know that the processor performance (B) improves by 60% per year and the disk bandwidth (A) by 40% per year. This will cause the ratio of processing power to disk bandwidth in both systems to increase by 15% per year, narrowing the gap between line A and B and bringing active disks closer to the ideal total storage bandwidth.

We now look in greater detail at some specific applications that benefit from active disks.

4. Applications

In this study, we examine four real-world data-intensive data mining and multimedia applications that meet the assumptions of our Active Disks model.

4.1. Database - Nearest Neighbor Search

Our first application is a variation of a standard database search that determines the *k* items in a database of attributes that are closest to a particular input item. We use synthetic data from the Quest data mining group at IBM Almaden [Quest97] which contains records of individuals applying for loans and includes information on nine independent attributes: `<age>`, `<education>`, `<salary>`, `<commission>`, `<zip code>`, `<make of car>`, `<cost of house>`, `<loan amount>`, and `<years owned>`. In searches such as this across a large number of attributes, it has been shown that a scan of the entire database is as efficient as the building of extensive indices [Berchtold97], so an Active Disk scan is appropriate. The basic application takes a target record as input and processes records from the database, always keeping a list of the *k* closest matches so far and adding the current record to the list if it is closer than any already in the list. Distance for the purpose of comparison is the sum of the simple cartesian distance across the range of each attribute. For categorical attributes we use the hamming distance, a distance of 0.0 is assigned if the values match exactly, otherwise 1.0 is assigned.

For the Active Disks system, each disk is assigned an integral number of records and the comparisons are performed directly at the drives. The central server sends the target record to each of the disks which determine the ten closest records in their portions of the database. These lists are returned to the server which combines them to determine the overall ten closest records. In terms of our model parameters, the application reduces the records in a database of arbitrary size to a constant-sized list of ten records, so selectivity is arbitrarily large. Finally, the state required at the disk is simply the storage for the list of ten closest records.

4.2. Data Mining - Frequent Sets

The second application is an implementation of the Apriori algorithm for discovering association rules in sales transactions [Agrawal95]. We again use synthetic data generated using a tool from the Quest data mining group at IBM Almaden [Quest97] to create databases containing transactions from hypothetical point-of-sale information. Each record contains a `<transaction id>`, a `<customer id>`, and a list of `<items>` purchased. The purpose of the application is to extract rules of the form “if a customer purchases item A and B, then they are also likely to purchase item X” which can be used for store layout or inventory decisions. It does this in several passes, first determining the items that occur most often in the transactions (the *1-itemsets*) and then using this information to generate pairs of items that occur often (*2-itemsets*) and larger groupings (*k-itemsets*). The threshold of “often” is called the *support* for a particular itemset and is an input parameter to the application (e.g. requiring support of 1% for a rule means that 1% of the transactions in the database contain a particular itemset). Itemsets are determined by successive scans over the data, at each phase using the result of the *k*-itemset counts to create a list of candidate *(k+1)*-itemsets, until there are no k-itemsets above the desired support.

For the Active Disks system, the counting portion of each phase is performed directly at the drives. The central server produces the list of candidate *k*-itemsets and provides this list to each of the disks. Each disk counts its portion of the transactions locally, and returns these counts to the server. The server then combines these counts and produces a list of candidate *(k+1)*-itemsets which are sent back to the disks. The application reduces the arbitrarily large number of transactions in a database into a single, variably-sized set of summary statistics - the itemset counts - that can be used to determine relationships in the database. The state required at the disk is simply the storage for the candidate *k*-itemsets and their counts at each state.

4.3. Multimedia - Edge Detection

For image processing, we looked at an application that detects edges and corners in a set of grayscale images [Smith95]. We use real images from Almaden's CattleCam [Almaden97] and attempt to detect cows in the landscape above San Jose. The application processes a set of 256 KB images and returns only the edges found in the data using a fixed 37 pixel mask. The intent is to model a class of image processing applications where only a particular set of features (e.g. the edges) in an image are important, rather than the entire image. This includes tracking, feature extraction, and positioning applications that operate on only a small subset of the original images data. This application is significantly more computation-intensive than the comparisons and counting of the first two applications.

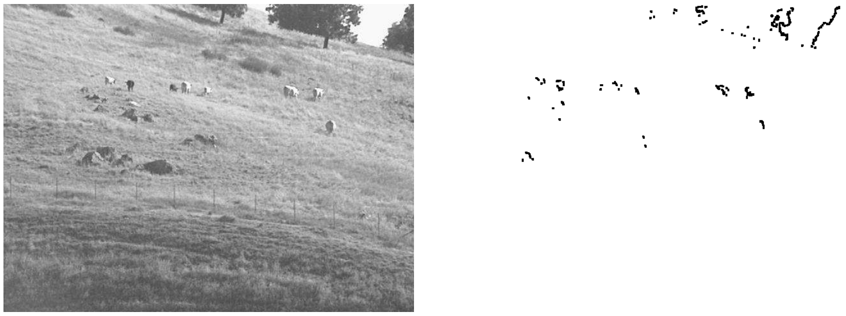


Figure 3: Edge detection in a scene outside the IBM Almaden Research Center. On the left is the raw image and on the right are the edges detected with a brightness threshold of 75.

For the Active Disks system, the edge detection for each image is performed directly at the drives and only the edges are returned to the central server. A request for the raw image in Figure 3 returns only the data on the right, which can be represented much more compactly. The application reduces the amount of data transferred to the server by a large fraction (from 256 KB to 9 KB for this particular image). The state required on disk is the storage for a single image that is buffered and processed as a whole.

4.4. Multimedia - Image Registration

Our second image processing application performs the image registration portion of the processing of an MRI brain scan analysis [Welling98]. Image registration determines the set of parameters necessary to register (rotate and translate) an image with respect to a reference image in order to compensate for movement of the subject during the scanning. The application processes a set of 384 KB images and returns a set of registration parameters for each image. This application is the most computationally intensive of the ones study, since the algorithm performs a Fast Fourier Transform (FFT), determines the parameters in Fourier space and computes an inverse-FFT on the resulting parameters. In addition to this, the algorithm requires a variable amount of computation since it is solving an optimization problem which may require a variable number of iterations to converge to the correct parameters. Unlike the applications discussed above, the per byte cost of this algorithm varies significantly with the data being processed[1]. The average computation cost of each of the algorithms discussed in this section is shown in Table 2 in the next section.

For the Active Disks system, this application operates similarly to the edge detection. The reference image is provided to all the drives and the registration parameters for each processed image is performed directly at the drives with only the final parameters (1.5 KB for each image) returned to the central server. The application reduces the amount of data transferred to the server by a large, fixed fraction to a single, fixed amount of data. The state required at the disk is the storage for the reference image and the current image that must be buffered and processed as a whole.

5. Prototype / Experiments

Our experimental testbed contains ten prototype Active Disks, each one a 6 year old DEC Alpha 3000/400 (133 MHz, 64 MB, Digital UNIX 3.2g) with two 2.0 GB Seagate ST52160 Medalist disks. For the server case, we use a single DEC AlphaStation 500/500 (500 MHz, 256 MB, Digital UNIX 3.2g) with four 4.5 GB Seagate ST34501W Cheetah disks on two Ultra-Wide SCSI busses. All these machines are connected by an Ethernet switch and a 155 Mb/s OC-3 ATM switch (DEC Gigaswitch/ATM).

Our experiments compare the performance of a single server machine with directly-attached SCSI disks against the same machine with network-attached Active Disks, each of which is a workstation with two directly-attached SCSI disks in our prototype.

5.1. Database - Nearest Neighbor Search

Figure 4 compares the performance of the single server system against a system with Active Disks as the number of disks is increased from 1 to 10. As predicted by our model, we see that for a small number of disks, the single server system performs better. The server processor is four times as powerful as a single Active Disk processor and can perform the computation at full disk rate. We see that the server system CPU saturates at 25.7 MB/s and performance no longer improves as additional disks are added, while the Active Disks system continues to scale linearly to 58 MB/s with 10 disks. Our prototype system was limited to 10 disks by the amount of hardware we had available, but if we extrapolate the data from the prototype to a larger system with 60 disks, the smallest of the systems in Table 1, we would expect throughput near the 300 MB/s that our model predicts for this configuration.

[1] The other algorithms also do not have absolutely constant compute time per record due to properties of the data (the transactions in the frequent sets data are variable-sized for example) and cache effects, but the variation is much less.

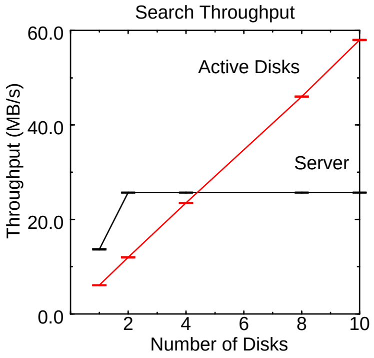
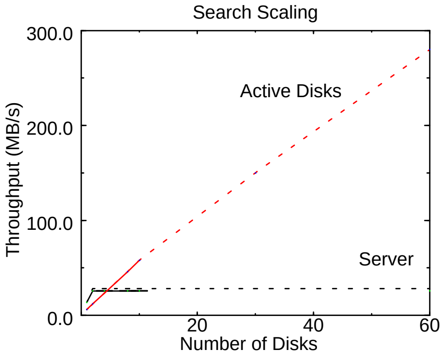


Figure 4a: The search application shows linear scaling with number of disks up to 58 MB/s, while the server system bottlenecks at 26 MB/s.

Figure 4b: Because of the high selectivity of this search,we would not expect the Active Disks system to saturate for at least a few hundred disks.



5.2. Data Mining - Frequent Sets

In Figure 5, we show the results for the first two passes of the frequent sets application (*1-itemsets* and *2-itemsets*). We again see the crossover point at four drives, where the server system bottlenecks at 23.5 MB/s and performance no longer improves, while the Active Disks system continues to scale linearly to 60.5 MB/s. Figure 5b shows an important property of the frequent sets application that affects whether or not a particular analysis is appropriate for running on Active Disks. The chart shows the memory requirements across a range of input support values on two different databases. The lower a support value, the more itemsets are generated in successive phases of the algorithm and the larger the state that must be held on disk. We expect the support value tend toward the higher values since it is difficult to deal with a large number of rules, and the lower the support, the less compelling the generated rules will be, but for very low values of the support, the limited memory at Active Disk may become an issue. Modern disk drives today contain between 1 MB and 4 MB of cache memory, so we might expect 4 - 8 MB in the timeframe that Active Disks become a reality. This means that care must be taken in designing algorithms and in choosing when to take advantage of execution at the disks.

5.3. Multimedia

Figure 6 shows the results for the image processing applications. As we saw in Table 2, the image processing applications require much more CPU time than search or frequent sets, leading to much lower throughputs on both systems. The edge detection bottlenecks the server CPU at 1.7 MB/s, while the Active Disks system scales to 4.2 MB/s with 10 disks. Image registration is the

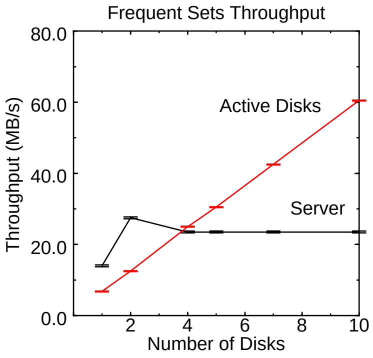
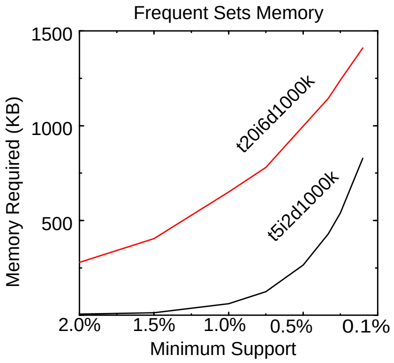


Figure 5a: The frequent sets application shows linear scaling to 60 MB/s while the server system bottlenecks at 24 MB/s.

Figure 5b: The amount of memory necessary for the frequent sets application increases as the level of support required for a particular rule decreases. Very low support values may require multiple megabytes of memory at each Active Disk.



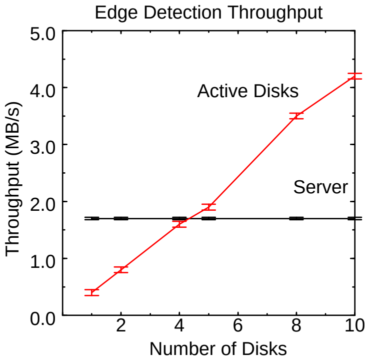


Figure 6a: The edge detection application shows linear scaling with number of disks while the server system bottlenecks at about 1.7 MB/s.

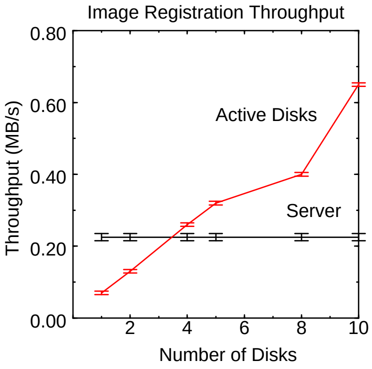


Figure 6b: The image registration application also scales linearly, but requires almost a factor of ten more CPU cycles, reducing throughput in both systems.

most CPU-intensive of the applications we have considered, it achieves only 225 KB/s on the server system, and scales to 650 KB/s with 10 Active Disks.

5.4. Model Validation

In terms of our model's parameters, this testbed has $\alpha_s = 133/500 = 1/3.8$ (easily calculated from the clock rates because the processors are the same basic chip, and the code is identical for both cases), Ideally, we would have $\alpha_d = \alpha_n = 1$ for our tests, but this was not possible in our testbed. Instead $r_d = 14\text{ MB/s}$, $r_d' = 7.5\text{ MB/s}$, $r_n = 60\text{ MB/s}$ and $r_n' = 10\text{ MB/s}$.

Estimating the applications' selectivity was a straightforward exercise of counting bytes. For our four applications, α_N was 100,000; 14,000; 175; and 230 respectively. We estimate the number of cycles per byte by instrumenting the application to determine the number of cycles spent for the entire computation on the server system, and then divide this by the total number of bytes processed. All of the applications are reasonably optimized versions of the respective codes and these values should represent a good approximation to the time these algorithms would require in a production system. The main difficulty is in accounting for the cycles spent in the operating system and filesystem code on behalf of the application, and this introduces some imprecision in our measurements. The parameters for all four applications are summarized in Table 2.

application	computation (cycles/byte)	memory (KB)	selectivity	parameter
Search	17	0.6	100,000	k=10
Frequent Sets	15	4	14,000	s=0.25%
Edge Detection	394	64	175	t=75
Image Registration	2387	768	230	-

Table 2: Costs of the applications presented in the text. Computation time per byte of data, memory required at each Active Disk, and the selectivity factor in the network. The parameter values are variable inputs to each of the applications.

Figure 7 combines the results for all four applications and superimposes the predictions of the model based on these system and application parameters. The basic trends are consistent with the results from the prototype. The server system bottlenecks at a relatively low throughput while the Active Disks system scales close to linearly as additional disks are added. The predicted throughputs deviate by up to 50% in either direction for the search and frequent sets applications, which

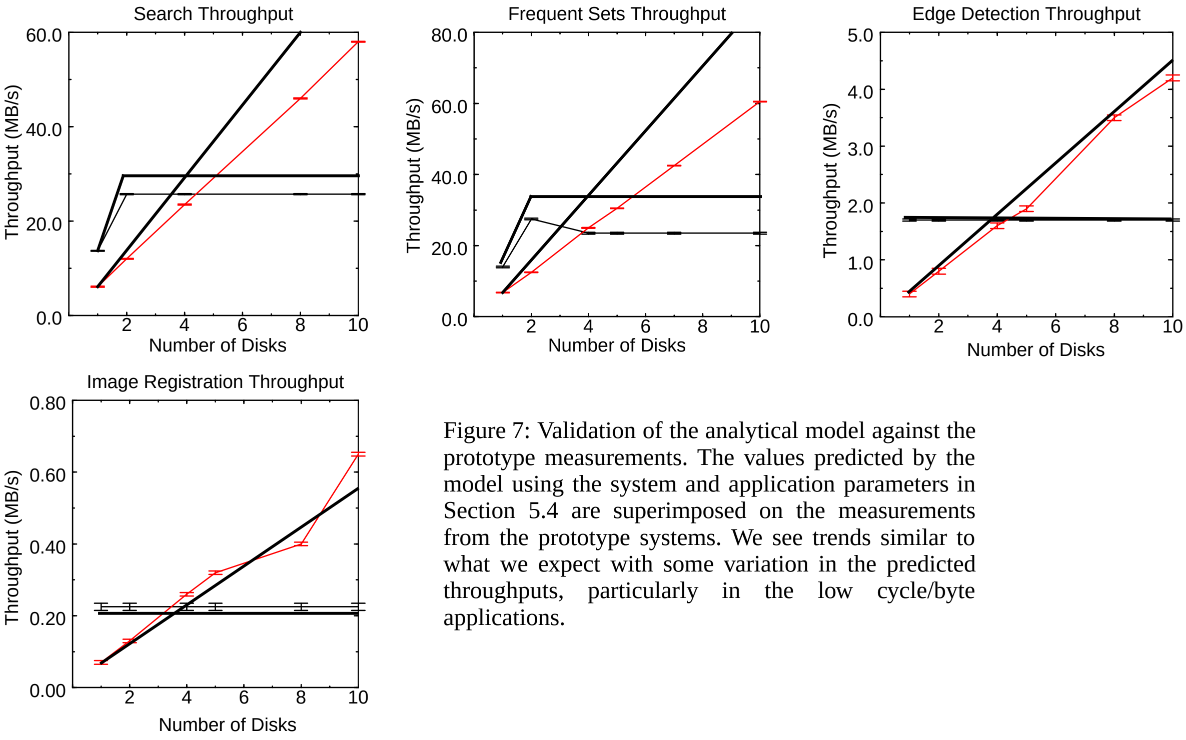


Figure 7: Validation of the analytical model against the prototype measurements. The values predicted by the model using the system and application parameters in Section 5.4 are superimposed on the measurements from the prototype systems. We see trends similar to what we expect with some variation in the predicted throughputs, particularly in the low cycle/byte applications.

we attribute to measurement error in the parameters[1] rather than a breakdown of the assumptions made in the model. Given our goal of using the model to develop intuition about the performance of Active Disks systems, these results are certainly encouraging.

6. Discussion

The applications discussed above take advantage of the inherent parallelism available in Active Disk systems. In addition to the parallelism benefits, filtering and improved batching/scheduling made possible by the ability to execute code directly at disk drives can improve a larger class of applications. We will discuss the potential benefits of each of these approaches in turn and outline the applications that can take advantage of them.

6.1. Parallelism

The largest single benefit from using Active Disks, as discussed above, is the parallelism available in large storage systems. In cost-effective systems built for a variety of large data applications, the number of disk drives will always greatly exceed the number of server processors. This means that although processing power on disk drives will always be less than on top-of-the-line server CPUs, there will very often be more aggregate CPU power in the disks than the server. Applications that can be partitioned to take advantage of this parallelism, and can be “split” across the server and drive CPUs, have available a much higher total computational power than applications running only on the server.

[1] Particularly the cycles per byte, where 4 or 5 additional cycles per byte would eliminate the majority of the error.

6.2. Filtering

In many systems in use today, interconnect bandwidth is at a premium compared to computational power. If an application is scanning large objects in order to select only specific records or specific fields within records, a large fraction of the data moved across the interconnect will simply be discarded. Allowing such filter functions to operate directly at drives, close to the data, and returning only the relevant fraction of the data to the client, can significantly reduce network utilization possibly avoiding this cost-critical bottleneck altogether. In addition to reducing network traffic, applications that scan large objects looking for specific properties or gathering statistics (e.g. counts of matching values) can take advantage of the computational power available at drives by performing these simple operations directly on the drives, thereby offloading the host and increasing the aggregate system power. Applications that fall into this category include search, database select, image processing and extraction, association rules for data mining, and feature extraction, i.e. any relatively simple function with the potential to significantly decrease the amount of network traffic or host processing required.

6.3. Batching/Scheduling

Throughout this paper we have concentrated on the parallelism and filtering advantages of Active Disks. In storage systems research, however, the most common application-specific optimizations are scheduling, batching and prefetching [Kotz94, Patterson95, Bitton88, Ruemmler91] and Active Disks can be expected to execute these types of remote functions as well.

For database operations, disk scheduling has typically been done at much higher levels of query processing and data layout. However, interconnect scheduling may be useful to database operators such as parallel sort or hash join [Kitsuregawa83] which inherently communicate among a large set of nodes during processing. While network scheduling cannot be expected to yield benefits like we have seen so far in this paper, it is intriguing to explore Active Disk functions for such complex operations. The key observation is that for data that is going to move through the network after it is read from disk, it may be possible to send it to the right place under Active Disks control, reducing network traffic through scheduling at the disk, rather than sending it to the “wrong place” and then communicating data among processing nodes.

We consider more closely an application similar to the NowSort developed at Berkeley [Arpaci-Dusseau97] which uses a network of workstations to accomplish the world’s fastest sort according to the 1997 Datamation benchmark [Gray97]. This benchmark sorts a set of 100-byte records with randomly distributed 10-byte keys using a parallel sort algorithm across a large number of nodes.

Our algorithm for an Active Disk sort is based on a parallel sample sort composed of a sample phase and a sort phase [Blelloch97]. During the short sample phase, a subset of the total data is read and a histogram is created which allows us to divide the key space into *n* buckets of roughly equal size. In the server version of this sort across a number of processing nodes, the entire data set is then 1) read into the nodes from their local disks, 2) exchanged across the network according to the key space distribution, 3) sorted locally at each node, and 4) written back to local disks.

For the Active Disks system, we remove the need for step 2 by having the drives perform the read and distribution operations at the same time. Instead of sending all data to a particular node, the drive is given the key ranges determined in the sample phase and responds to a request from client *n* with only the data "belonging" to client *n* as its portion of the key space. This means that data destined for a particular node will get to that node as soon as possible, and will never need to be exchanged among nodes at some future point. This reduces the number of transits of all the data across the network from three to two. In systems where the network is the bottleneck resource, this will improve overall performance of the algorithm by up to one-third.

From this preliminary discussion, it is certainly not clear how important Active Disks will be to operations like sort and join, but it does illustrate that these class of algorithms are valuable for further exploration.

7. Conclusions and Future Work

Commodity disks drives with an excess of computational power are visible on the horizon. Given recent advances in safe remote execution technology and languages, it is possible to provide an execution environment for application-specific code inside individual disk drives to produce what we call Active Disks. This allows applications to take advantage of the parallelism in storage to greatly increase the total computational power available to them. We have demonstrated an important class of applications that will see significant gains - with linear scaling in the number of devices added to the system - from the use of Active Disks. We have provided an analytical model for estimating speedup in the presence of Active Disks. We have presented numbers from a prototype system that sees speedups of more than 2 over a comparable single server system with 10 disks and is limited only by the amount of hardware we have available to us. Our system should easily scale to speedups of more than 10 in reasonably-sized systems similar to those already in use for large databases today.

Emerging applications such as data mining, multimedia feature extraction, and approximate searching involve the huge data sets, on the order of 100s of GB or TB, necessary to justify large numbers of Active Disks. Many of these applications have small CPU and memory requirements and are attractive for execution across Active Disks.

There are a variety of areas to be explored before the benefits presented here can be put into practice. Providing a safe environment for application code inside the drive in order to protect both the integrity of data on the drive and ensure proper function in the presence of misbehaved application code is critical. The issue of resource management becomes considerably more complex as the computation becomes more distributed. Programming such systems and optimizing the placement of different components of the execution [Franklin96] requires further study. Active disks will need to make more complex scheduling decisions than disk drives do today, but they also open new areas for optimization based on richer interfaces than storage devices currently provide.

8. Bibliography

[Agrawal95] Agrawal, R. and Srikant, R. ‘‘Fast Algorithms for Mining Association Rules’’ *VLDB*, September 1994.

[Agrawal96] Agrawal, R. and Schafer, J. "Parallel Mining of Association Rules" *IEEE Transactions on Knowledge and Data Engineering* 8,6. December 1996.

[Almaden97] Almaden CattleCam, IBM Almaden Research Center *www.almaden.ibm.com/almaden/cattle/home_cow.htm*, January 1998.

[Arpaci-Dusseau97] Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H., Culler, D.E., Hellerstein, J.M. and Patterson, D.A. "High-Performance Sorting on Networks of Workstations" *ACM SIGMOD*, June 1997.

[Arya94] Arya, M., Cody, W., Faloutsos, C., Richardson, J. and Toga, A. "QBISM: Extending a DBMS to Support 3d Medical Images" *International Conference on Data Engineering*, February 1994.

[Barclay97] Barclay, T. "The TerraServer Spatial Database" *www.research.microsoft.com/terraserver*, November 1997.

[Berchtold96] Berchtold, S., Keim, D.A. and Kriegel, H. “The X-tree: An Index Structure for High-Dimensional Data” *VLDB*, 1996.

[Berchtold97] Berchtold, S., Boehm, C., Keim, D.A. and Kriegel, H. “A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space” *ACM PODS*, May 1997.

[Bershad95] Bershad, B.N., Savage, S., Pardyak, P., Sirer, E.G., Fiuczynski, M.E., Becker, D., Chambers, C. and Eggers, S. "Extensibility, Safety, and Performance in the SPIN Operating System" *SOSP*, December 1995.

[Bitton88] Bitton, D. and Gray, J. “Disk Shadowing” *VLDB*, 1988.

[Blelloch97] Blelloch, G.E., Leiserson, C.E., Maggs, B.M., Plaxton, C.G., Smith, S.J. and Zagha, M. "An Experimental Analysis of Parallel Sorting Algorithms" *Communications of the ACM*, To Appear.

[Boral83] Boral, H. and DeWitt, D.J. “Database Machines: An Idea Whoe Time Has Passed?” *International Workshop on Database Machines*, September 1983.

[Cao94] Cao, P., Lim, S.B., Venkataraman, S. and Wilkes, J. "The TickerTAIP Parallel RAID Architecture" *ACM Transactions on Computer Systems* 12 (3), August 1994.

[DeWitt81] DeWitt, D. and Hawthorn, P. “A Performance Evaluation of Database Machine Architectures” *VLDB*, September 1981.

[Drapeau94] Drapeau, A.L., Shirriff, K.W., Hartman, J.H., Miller, E.L., Seahan, S., Katz, R.H., Lutz, K., Patterson, D.A., Lee, E.K. and Gibson, G.A. “RAID-II: A High-Bandwidth Network File Server”, *ISCA*, 1994.

[Faloutsos94] Faloutsos, C., Barber, R., Flickner, M., Hafner, J., Niblack, W., Petkovic, D. and Equitz, W. “Efficient and Effective Querying by Image Content", *Journal of Intelligent Information Systems* 3 (4), July 1994.

[Faloutsos96] Faloutsos, C. *Searching Multimedia Databases by Content*, Kluwer Academic Inc., 1996.

[Flickner95] Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D. and Yanker, P. “Query by Image and Video Content: the QBIC System”, *IEEE Computer*, September 1995.

[Franklin96] Franklin, M.J., Jonsson, B.P. and Kossmann, D. "Performance Tradeoffs for Client-Server Query Processing" *ACM SIGMOD*, June 1996.

[Gibson97] Gibson, G., Nagle, D., Amiri, K., Chang, F., Feinberg, E., Gobioff, H., Lee, C., Ozceri, B., Riedel, E., Rochberg, D. and Zelenka, J. "File Server Scaling with Network-Attached Secure Disks" *ACM SIGMETRICS*, June 1997.

[Gosling96] Gosling, J., Joy, B. and Steele, G. *The Java Language Specification*. Addison-Wesley, 1996.

[Gray97] Gray, J. “Sort Benchmark Home Page” *www.research.microsoft.com/barc/SortBenchmark*, 1997.

[Grochowski96] Grochowski, E.G. and Hoyt, R.F., “Future Trends in Hard Disk Drives,” *IEEE Transactions on Magnetics* 32 (3), May 1996.

[Hsiao79] Hsiao, D.K. “DataBase Machines Are Coming, DataBase Machines Are Coming!” *IEEE Computer,* March 1979.

[Kitsuregaw83] Kitsuregawa, M., Tanaka, H. and Moto-Oka, T. “Application of Hash To Data Base Machine and Its Architecure” *New Generation Computing* 1, 1983.

[Kotz94] Kotz, D. "Disk-directed I/O for MIMD Multiprocessors" *OSDI*, November 1994.

[Lee96] Lee, E.K. and Thekkath, C.A., “Petal: Distributed Virtual Disks”, *ASPLOS*, October 1996.

[Livny87] Livny, M., “Multi-disk management algorithms”, *ACM SIGMETRICS*, May 1987.

[Necula96] Necula, G.C. and Lee, P. "Safe Kernel Extensions Without Run-Time Checking" *OSDI*, October 1996.

[Patterson88] Patterson, D.A., Gibson, G. and Katz, R.H., “A Case for Redundant Arrays of Inexpensive Disks”, *ACM SIGMOD*, June 1988.

[Patterson95] Patterson, R.H. et al., “Informed Prefetching and Caching”, *SOSP*, 1995.

[Quest97] Quest Project, IBM Almaden Research Center “Quest Data Mining Project” *www.almaden.ibm.com/cs/quest*, December 1997.

[Riedel97] Riedel, E. and Gibson, G. “Active Disks - Remote Execution for Network-Attached Storage” *Technical Report CMU-CS-97-198*, December 1997.

[Romer96] Romer, T.H., Lee, D., Voelker, G.M., Wolman, A., Wong, W.A., Baer, J., Bershad, B.N. and Levy, H.M. "The Structure and Performance of Interpreters" *ASPLOS,* October 1996.

[Ruemmler91] Ruemmler, C. and Wilkes, J., “Disk Shuffling”, *HP Labs Technical Report HPL-CSP-91-30*, 1991

[Seagate97] Seagate Technology “Cheetah: Industry-Leading Performance for the Most Demanding Applications”, *www.seagate.com*, 1998.

[Small95] Small, C. and Seltzer, M. "A Comparison of OS Extension Technologies" *USENIX Technical Conference*, January 1996.

[Smith79] Smith, D.C.P. and Smith, J.M. “Relational DataBase Machines” *IEEE Computer*, March 1979.

[Smith95] Smith, S.M. and Brady, J.M. “SUSAN - A New Approach to Low Level Image Processing” *Technical Report TR95SMS1c*, Oxford University, 1995.

[StorageTek94] Storage Technology Corporation, “Iceberg 9200 Storage System: Introduction”, *STK Part Number 307406101*, 1994.

[TPC98] Transaction Processing Performance Council "TPC Executive Summaries" *www.tpc.org*, 1998.

[TriCore97] TriCore News Release "Siemens' New 32-bit Embedded Chip Architecture Enables Next Level of Performance in Real-Time Electronics Design" *www.tri-core.com*, September 1997.

[Turley96] Turley, J. "ARM Grabs Embedded Speed Lead" *Microprocessor Reports* 2 (10), February 1996.

[VanMeter96] Van Meter, R., Holtz, S. and Finn G., "Derived Virtual Devices: A Secure Distributed File System Mechanism", *Fifth NASA Goddard Conference on Mass Storage Systems and Technologies*, September 1996.

[Virage98] Virage "Media Management Solutions" *www.virage.com*, February 1998.

[Wactlar96] Wactlar, H.D., Kanade, T., Smith, M.A. and Stevens, S.M. "Intelligent Access to Digital Video: Informedia Project" *IEEE Computer*, May 1996.

[Wahbe93] Wahbe, R., Lucco, S., Anderson, T.E. and Graham, S.L. "Efficient Software-Based Fault Isolation" *SOSP*, December 1993.

[Welling98] Welling, J. "Fiasco: A Package for fMRI Analysis" *www.stat.cmu.edu/~fiasco*, January 1998.

[Wilkes95] Wilkes, J., Golding, R., Staelin, C. and Sullivan, T. "The HP AutoRAID hierarchical storage system" *SOSP*, December 1995.

[Yao85] Yao, A.C. and Yao, F.F. "A General Approach to D-Dimensional Geometric Queries" *ACM STOC*, May 1985.