Active Disks – Remote Execution for Network-Attached Storage

Erik Riedel[1]
Garth Gibson

December 1997
CMU-CS-97-198

School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213-3890

Abstract

The principal trend in the design of computer systems is the expectation of much greater computational power in future generations of microprocessors. This trend applies to embedded systems as well as host processors. As a result, devices such as storage controllers have excess capacity and growing computational capabilities. Storage system designers are exploiting this trend with higher-level interfaces to storage and increased intelligence inside storage devices. One development in this direction is Network-Attached Secure Disks (NASD) which attaches storage devices directly to the network and raises the storage interface above the simple (fixed-size block) memory abstraction of SCSI. This allows devices more freedom to provide efficient operations; promises more scalable subsystems by offloading file system and storage management functionality from dedicated servers; and reduces latency by executing common case requests directly at storage devices. In this paper, we push this increasing computation trend one step further. We argue that application-specific code can be executed at storage devices to make more effective use of device, host and interconnect resources and significantly improve application I/O performance. Remote execution of code directly at storage devices allows filter operations to be performed close to the data; enables support of timing-sensitive transfers and application-aware scheduling of access and transfer; allows management functions to be customized without requiring firmware changes; and makes possible more complex or specialized operations than a general-purpose storage interface would normally support.

This research is sponsored by DARPA/ITO through ARPA Order D306, and issued by Indian Head Division, NSWC under contract N00174-96-0002. The project team is 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 Inc., Compaq Corporation, Data General, Quantum Corporation, Seagate Technology, 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.

[1] Department of Electrical and Computer Engineering, Carnegie Mellon University

Keywords: active disks, network-attached storage, mobile code, distributed applications

1. Introduction

The cost and chip real estate needed for any particular computational need is dropping with ongoing VLSI trends. At some point, having significant additional computation power becomes a negligible cost. Disk array controllers today have central processors at most one generation behind state-of-the-art workstations. Individual disk controllers are further behind, but are catching up fast. The technology is at a point where it becomes feasible to combine the drive control microprocessor with the specialized ASIC responsible for the balance of the drive's processing into a single chip. As shown in Figure 1, this makes it possible to integrate a 200 MHz RISC core in the same die space as the current ASIC and still leave room for additional special-purpose hardware functions [Turley96]. Moreover, the purpose of most of the hardware in this ASIC is to keep the microprocessor off the critical path (per byte, per sector) processing of normal drive operations. So most of the additional cycles possible in the integrated ASIC are spare in normal operation. While the example in this figure is hypothetical, the trend is a reality.

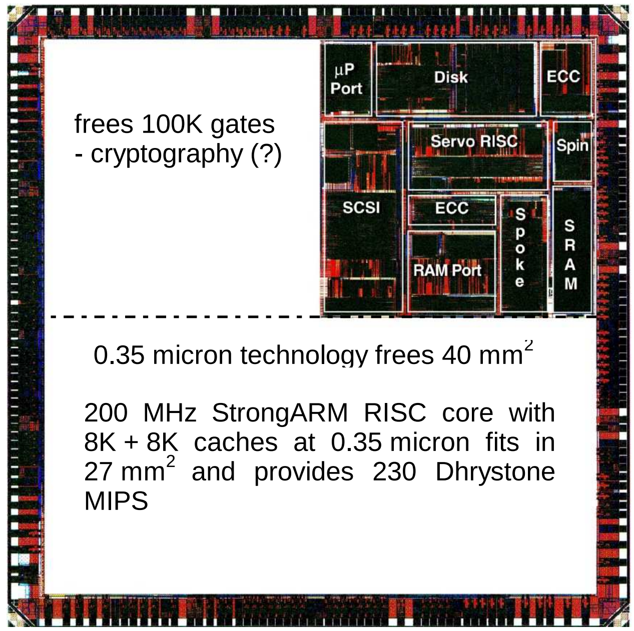

Figure 1 – On-Drive Processor. The ASIC in a typical SCSI drive that handles formatting, servo, ECC, and SCSI functions outlined in 0.68 micron process. Reducing this to next-generation 0.35 micron technology retains all the same function and provides 40 mm^2 of additional chip space. This is enough to include an integrated 200 MHz RISC core and an additional 100,000 gates of specialized processing for networking or security.

One use for the increasing computational power on disk controllers is to enrich their existing interface. For example, network-attached storage, as shown in Figure 2, integrates storage into the client network, rather than keeping it on a separate, server-attached peripheral bus. Eliminating server machines as a bottleneck for data transfers between storage and applications provides a significant opportunity for increased scalability. By not involving the server as a third party, common case transfers involve fewer store-and-forward copies and the number of requests that can be serviced at any given time is increased proportionally with the number of storage devices. The server remains responsible for overall file system functionality, but participates only infrequently when new access rights must be tested or cache consistency policies invoked [Gibson97].

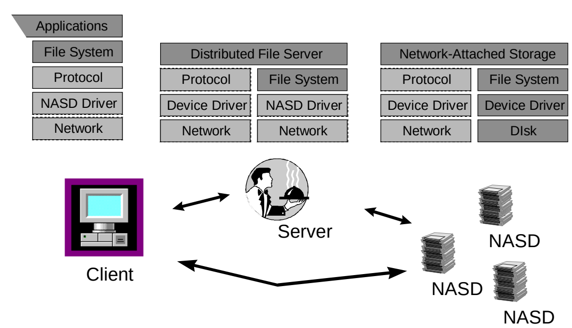

Figure 2 - Network-Attached Storage Architecture. NASD provides a fast-path for data transfer operations directly from clients to disks without requiring every byte to traverse the server and all the associated layers of processing. It also allows greater flexibility and higher levels of integration within the NASD itself.

As systems get faster and cheaper, people compute on larger and larger sets of data. Table 1 shows two contemporary systems with large storage requirements. The balance in

Microsoft TerraServer		Compaq ProLiant TPC-C	
4-CPU AlphaServer 4100		Four 200 MHz Pentium Pros	
4 x 400 MHz	1,600 MIPS	4 x 200 MHz	800 MIPS
2,048 MB	2,048 MB	4,096 MB	4,096 MB
1,100 MB/s	1,100 MB/s	540 MB/s	540 MB/s
320 SCSI Disks		113 SCSI Disks	
320 x 25 MHz	3,125 MIPS	113 x 25 MHz	2,800 MIPS
320 x 1 MB	320 MB	113 x 1 MB	113 MB
320 x 10 MB/s	3,200 MB/s	113 x 10 MB/s	1,130 MB/s

Table 1 - Two example systems with large storage requirements. The TerraServer is a geographic data server that provides access to satellite imagery of the entire world on the Internet [Barclay97]. The ProLiant system had the lowest $/tpmC rating for the week of 13 October 1997 [TPC97]. The table compares total processing power in MIPS, total memory in MB, and total transfer bandwidth in MB/s of disks vs. their host systems. The processor numbers are slightly unfair because both the Alpha and Pentium Pro systems are super-scalar, but even doubling the host values still keeps the aggregate drive power on par with the hosts. Bandwidth values for the hosts are peak advertised memory bandwidths.

these systems is typical of many large systems today. A large number of drives is necessary either to provide sufficient capacity for the target application or to provide sufficient aggregate throughput. We see that even with SCSI disks today (25 MHz processors and 10 MB/s sustained transfer rates), the potential combined processing power and transfer bandwidth at the disk drives is comparable to or exceeds that available on the host processors[2]. With 200 MHz disk processors and 30 MB/s disk transfer rates expected this decade, the potentially wasted power of a large disk farm is significant. Moreover, if storage is network-attached to a large collection of servers and clients, a cost-effective network is certainly the principle bottleneck in large-object processing.

This paper proposes Active Disks, next-generation disk drives that provide an environment for executing application code directly at individual drives. By partitioning processing across clients/servers and storage devices it may be possible to change the amount of data that crosses the network and to exploit cycles available at storage. Specifically, applications that apply simple filters or statistics to stored data or that make disk or network scheduling decisions based on current information at storage can improve the efficiency of an application's use of network and host resources and achieve more scalable and higher performance systems using Active Disks.

Section 2 outlines a range of candidate applications that can benefit from the availability of Active Disks. Section 3 presents a simple performance model promising significant performance advantages for two specific applications. Section 4 discusses the three major technology areas to be addressed to make Active Disks a reality. Section 5 discusses previous and related work and Section 6 concludes the paper.

2. Candidate Applications

There are several classes of applications that can derive significant benefits in performance, resource utilization, and functionality from the ability to execute code inside network-attached drives. The basic characteristics of successful remote functions 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, 3) execute a relatively small number of instructions per byte or 4) can make effective use of scheduling and management primitives inside the drives.

[2] The amount of memory at the host still exceeds that available at the disks, but the primary reason for this is that drives cannot easily take advantage of larger amounts of memory due to their current, limited APIs.

2.1. Filters

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 network 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 and improve overall throughput. 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 text search (e.g. `grep`), database select, image processing and extraction, association matching for data mining, and spatial data extraction, i.e. any relatively simple function with the potential to significantly decrease the amount of network traffic or host processing required.

2.2. Real-Time

Replay of stored multimedia must be coordinated by a system component that sees and controls all accesses. Individual drives may be the only place capable of accurately estimating the resources needed to meet a given real-time schedule. Assuming a drive with underlying real-time computing support, remote functions can implement reservation and application-specific transfer protocols that combine information about the data stream and the current drive state to provide optimal service.

2.3. Batching

Another class of applications that benefit from the capability to delay a response rather than satisfying a request immediately is anything that requires a large amount of work, but in which the order the work is done in is not important. Providing an aggregate description of a large amount of work to the drive (or perhaps across a number of drives) and allowing it (them) to schedule the work of the entire request in the most efficient manner possible can provide significant performance gain over executing the work as a series of simple requests [Kotz94]. This includes functions such as a drive-to-drive copy controlled at the drive, rather than through the client, and scatter/gather operations that distribute data across a large number of clients (collective I/O).

2.4. Self-Management

The existence of a remote execution environment at the drive makes it possible to provide management functions that are either more complex than drive firmware would normally allow or that are customized to the environment in which the drive is installed. For example, a backup function could be specialized to a particular environment or to an application-specific archive representation and a type-specific transfer protocol, all without interfering with other client work. Having such functions operate as remote functions also allows them to be extended and specialized by local management policy, rather than burned into the firmware. Possible functions include backup; layout optimization based on filesystem- or application-specific knowledge, or on usage patterns observed at the drive; defragmentation; and reconfiguration.

2.5. High-level Support

Specialized functions that require semantics not normally provided at drives could be provided by remote execution. This allows functions specialized to a particular environment or usage pattern to be executed at the lowest level of the system, where the semantics are most efficiently implemented, rather than requiring additional overhead at higher levels. Examples include a `READ/MODIFY/WRITE` operation for use by a RAID subsystem across NASDs; a drive-embedded web server; or an atomic `CREATE` that both creates a new file and updates the corresponding directory object, for optimization of a particular filesystem on NASD [Gibson97a].

2.6. Other

Researchers into I/O for large-scale scientific applications have noted a large variety in access patterns [Smirni96] that are often not good fits for general policies. Remote execution allows customization of the storage interfaces to meet different application needs. It also allows the implementation of a variety of storage optimizations that have been proposed in the context of custom systems, such as: AutoRAID [Wilkes96]; Semantic Caching [Dar96]; Progressive Browsing of Images [Prabhakar97]; TickerTAIP [Cao94]; optimal data layout; ADStar storage management [Cabrera95]; and Logical Disk [deJonge95].

3. Potential Benefits

We have taken a closer look at the benefits of Active Disks for two specific applications: database select and parallel sort. The following outlines a simple analytic model of how these applications might be partitioned in the presence of Active Disks and the performance benefits we might expect to see. The results here are intended to be illustrative rather than predictive.

Parameter	Value	Description
MIPS	200	Host CPU speed
NumDisks	8	Number of disks
DiskInst	5,000	Instr. to read a page from disk
PageSize	4096	Size of one data page (bytes)
NetBw	12.5	Network bandwidth (MB/s)
MsgInst	20,000	Instr. to send/recv a message
PerSizeMI	12,000	Instr. to send/recv 4096 bytes
Compare	2	Instr. to apply a predicate
HashInst	9	Instr. to hash a tuple
MoveInst	1	Instr. to copy 4 bytes
NumTuples	1,000,000	Number of tuples
TupleSize	64	Size of one tuple (bytes)
KeySize	4	Size of one key (bytes)

Table 2 - Parameters for Database Select. These values are from [Franklin96] and represent common values for typical client/server database systems today.

3.1. Database select

In order to evaluate the benefits possible using remote execution on drives, we modeled a database `SELECT` operation performed at drives rather than processing everything directly on the host. The basic modeling parameters are taken from [Franklin96] which studied the tradeoffs of performing hybrid query processing split between clients and database servers. We simply extended this notion to include Active Disks and moved portions of the computation directly to drives as our “servers”.

Table 2 shows the basic parameters used. This analysis considers only the total number of CPU cycles required to perform a particular computation. It does not model latency or contention in the network, or the details of request handling and scheduling inside the drives. This simple model trades off the sending of messages on the network in favor of processing at the drives. As the table indicates, one less page sent over the network saves 32,000 instructions while every page selected (hash + compare + copy) requires 1,728 instructions. We varied the percentage match of the relation and the processing power

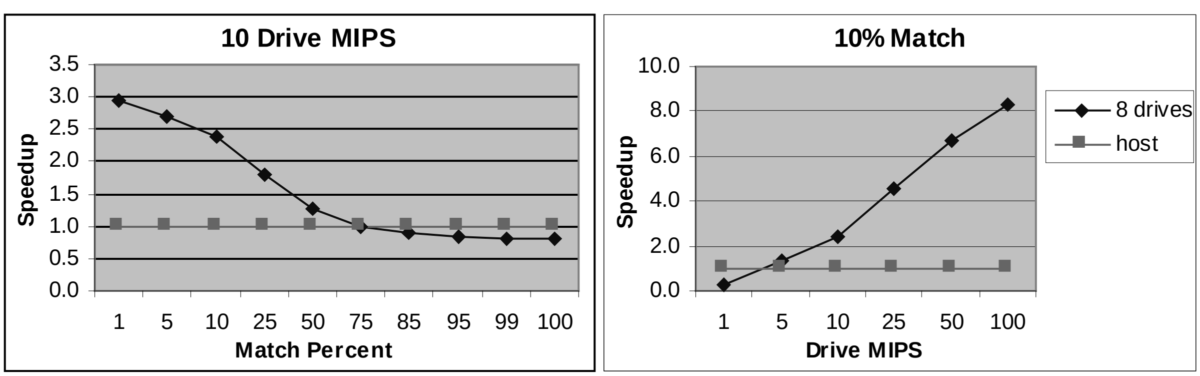

Figure 3 - Speedup over host-based processing. At low match percentages, significant improvements are possible even with drives only 1/20th the power of host processors.

available on the drives to obtain the charts in Figure 3. We see that at low match percentages (high degrees of selectivity), even drives with only a small fraction of the processing power of the host can provide significant improvements by eliminating expensive network processing operations, avoiding network bottlenecks, and taking advantage of the parallelism available across drives.

3.2. Parallel sort

The second application is a sorting algorithm based on parallel sample sort [Blelloch96] running across a network of workstations similar to the system described in [Arpaci-Dusseau97]. The processing in the normal case and in the presence of Active Disks is described in Table 3. The basic algorithm is a two-step process which first looks at a subset of the total data to be sorted and determines a rough distribution of key values. This information is then used to move the data to its final destination (client) where the records in a particular key range are sorted locally and the entire data set is written back in sorted order.

Step	Parallel Sample Sort	Sample Sort for Active Disks
1	Sample data	Sample data using `sample()` function on drives
2	Create distribution histogram	Create distribution histogram
3	Read data into clients from local disks	Read data into clients using `scan()` function
4	Distribute data among clients based on hist.	
5	Sort locally at each client	Sort locally at each client
6	Write back to local disks in sorted order	Write back to drives in sorted order

Table 3 – Processing for Sample Sort. The basic algorithm in a network of workstations with directly-attached disks and in a system with Active Disks. In cases where the sampled distribution is not completely accurate, a minor "shift" step may be necessary between Steps 5 and 6 to ensure that all clients write an equal amount of data and keep the sorted data balanced across disks. In order for the local sort to proceed most efficiently, the set of records to be sorted should fit into the client memory. This means that if a data set is larger than the aggregate memory of the clients, the entire algorithm will only sort a subset of the data and will be followed by a series of merge steps until all the data is sorted.

In the Active Disks system, two remote functions are shipped to the drive, `sample()` operates just like a normal `read()` operation, but returns only a randomly distributed subset (say 5%) of the data. With an understanding of the record structure of the data being processed, this operation is simply and efficiently implemented at the drive and may be able to take advantage of the fact that it doesn't care which data blocks are read, just that they are relatively randomly distributed, to optimize seek times. The second function is a `scan()` which also operates like `read()` but uses the results of the sample step to provide only data in a specific key range to the requesting client. The scan reads data sequentially from the disk surface, but returns only records in the key range for this client. All other records are stored in on-drive cache buffers until the client responsible for those records issues a request. If cache buffers must be freed before a client has made its scan request, the data is simply re-read from disk when that request eventually comes. Assuming the distribution determined by the sampling is relatively accurate, the load should be evenly distributed across clients and requests should arrive in a balanced fashion.

The most expensive step in the basic algorithm is Step 4 where data is exchanged among all the clients. This step is eliminated in the Active Disks algorithm. In the basic algorithm, all the data must traverse the "network" (either the local peripheral bus or the general interconnect network) three times, while in the Active Disks case it is moved only the absolute minimum of two times. Assuming that the entire execution is network-limited (which it will normally be if sorting and sending/receiving are properly overlapped) and that all "networks" are created equal, then Active Disks will improve processing time by 1/3. Since individual Active Disks should be significantly less expensive than general-purpose workstations, such an algorithm should provide a better cost/performance than one based solely on workstations and traditional SCSI disks. If we were to be more ambitious and expand the complexity of the drives functions, it would also be possible to perform the entire sort without the clients, drives would simply exchange the data among themselves and sort locally before writing, completely eliminated the need for the clients.

4. Implementation Issues for Active Disks

Having argued that disk drives have excess processing and transfer capability and that applications exist which can significantly benefit from being able to execute code directly at drives, we now discuss the major implementation issues for Active Disks.

4.1. Language Model

Probably the most basic question for a remote execution system is what programming model should be provided for user-defined functions. The principle issues are the language expressiveness, the risk and cost of collateral damage caused by faulty code, and the load-time and runtime costs of protecting against collateral damage. Where application needs do not require broad expressiveness, a well-defined and limited set of interfaces such as packet filters or SQL allows powerful control over the safety and efficiency of user-provided functions. While filter functions may lend themselves to this approach, full support for applications requires a more expressive language. Moreover, although processing at the drives is abundant enough to exploit, if it is squandered with safety checks, there will be little value in relocating function to drives.

Dynamic relocation hardware with protection checking is not out of the question on a drive, but it must be compared to the use of type-safe programming languages, compile-time, load-time, and runtime safety checks [Gosling96, McGraw97, Sirer96, Romer96, Wahbe93]. A particularly low overhead approach, proof-carrying code, moves the cost of demonstrating safety from the runtime system to the code producer [Necula96]. The runtime publishes a *safety policy* that all programs must adhere to and the code producer is responsible for providing a *proof* that their program meets the limitations of this policy. The runtime system simply has to verify that the program provided satisfies the proof provided -- a much less costly check than generating the proof or verifying the safety policy directly. This approach is promising, but has only been demonstrated for very small functions and very simple safety policies, while generating proofs for arbitrary programs is known to be intractable.

The most significant consideration in the tradeoff between the cost of accepting a remote function (compilation, translation, static checking) and the runtime safety checking costs is the lifetime of programs executing at the drive. Current drive interfaces provide for requests of very limited duration – typically a few to a few dozen milliseconds – in part because operations are limited, contiguous SCSI block transfers. The object interface for network-attached storage proposed in [Gibson97a] expands these bounds by allowing scatter/gather access to variable-length objects and by associating drive-managed attributes with objects. Associating translated programs with object attributes allows reuse and amortization of load-time work, favoring safety mechanisms with low runtime overhead. Table 4 shows some of the tradeoffs of the possible technologies based on how safety is ensured and where the protection is performed and verified.

Technology	Per Program		Per Invocation		Per Statement	
	Cost	Where	Cost	Where	Cost	Where
Compilation	High	Drive	None		None	
Pre-Compilation	High	Producer	None		None	
Sandboxing	None		High	Drive	Low	Drive
Interpreter	None		Medium	Drive	High	Drive
PCC	High	Producer	Low	Drive	None	

Table 4 - Costs of language/safety mechanisms. Translation and protection cost may be incurred for each new remote program that is created, for each invocation of the program at the device, or possibly for each statement in the program. Some technologies allow the majority of the burden for ensuring safe code to be moved to the code producer and allow efficient translation/execution at the drive.

The question of program duration is also tied to what types of state user-provided programs are allowed to maintain at the drive and how this can be shared among different programs and with the drive environment itself. As with processor cycles, memory at the drive will be inherently more limited than it is in host systems. This places a limit on the size of programs that can be efficiently executed and on the amount of state programs can maintain while running (unless on-disk storage is used).

4.2. Safety and Manageability

Providing a general-purpose execution environment on a drive challenges the ability of the external file manager to ensure filesystem integrity and management. Even a bug-free application program, if given total access to drive internal state, can effect changes important to filesystem integrity that are invisible to the filesystem outside of the drive. The most direct solution to this is to limit remote functions to the same or a very limited extension of the drive's normal storage interface. For example, the network-attached drive interface proposed in [Gibson97a] uses a capability system [Gobioff97] to authorize a specific set of operations on data objects on the drive. Protecting data integrity in the face of remote programs can be achieved by requiring remote functions to have capabilities and use the existing system to authorize their access to data objects on the drive.

However, some Active Disk applications want more than the normal storage interface. The challenge is to find the proper tradeoff between protecting drive integrity and manageability while providing sufficiently powerful interfaces. In order to provide sufficient power to all remote functions, the internal interface may need to support 1) the basic object interface provided to normal requests, 2) mechanisms to affect scheduling of requests and responses, 3) read access the internal drive state, and 4) mechanisms to modify drive state in controlled ways.

4.3. Resource Management

Once a programming model is established and there is a means for ensuring safety, the biggest question remaining is how to dynamically control the resources used by remote functions. Unlike a timesharing system, it is not obvious that a drive should accept any remote function and eventually execute it. Some functions will only benefit their application in certain circumstances. Some functions will never benefit their own application and may hurt others' performance. Because remote code consumes drive resources, it may be necessary to refuse new requests or eject ineffectively running functions.

In order to refuse a new remote function, an admission control policy may need some estimate of the cycle and memory/cache usage, either on a function-wide or a per-byte basis, needed for execution. The drive could then execute only functions that fit within its current resource constraints and would be able to eject running functions that exceeded their estimates. For at least some of the restricted functions that profit from Active Disks, this type of control might be guaranteed by compilation or proof techniques that try to bound a particular piece of code to only consume *n* cycles before yielding or *m* bytes of memory.

With current drive interfaces, allocation of resources is done in a relatively straightforward manner. Requests are not of widely varying amounts of work or duration and are selected fairly and run to completion. Dynamic optimizations such as read-ahead also compete for resources, but are limited to minimal interference with normal requests. The inclusion of long-running remote functions, even if they are being effective, impacts the fairness and responsiveness of this scheme.

One possibility for managing remote programs would be to apply the same technique as currently used for read-ahead and allow remote programs to execute only when the drive would otherwise be idle. The use of idle time to perform more expensive optimization work than simple read-ahead is used in storage systems such as AutoRAID [Wilkes96], which uses idle periods to reorganize data layout based on information gathered during normal operation. Such a mechanism requires a means of detecting idle time and requires relatively low start-up and shutdown times when new foreground work arrives [Golding95]. However, limiting remote programs to execute only in idle periods treats the associated applications as low priority, which, if their remote function reduces overall system work, is counter-productive for global resource management and throughput. In addition, remote programs that affect future activity, such as the use of remote execution to provide scheduling and synchronization for a streaming video application, would, almost by definition, be most effective during times of high foreground load.

5. Related Work

Active Networks provide a mechanism for running application code at network routers and switches to accelerate innovation and enable novel applications in the movement of data and network management [Tennenhouse96]. Two possible approaches for managing network programs are suggested - a *discrete* approach that allows programs to be explicitly loaded into the network and affect future packet processing and an *integrated* approach in which each packet consists of a program instead of simply "dumb" data. The primary tradeoff, as for Active Disks, is the amount of state that devices can be expected to maintain between requests and how many requests can be "active" at any given time. The implementation of the Active IP option [Wetherall96] describes a prototype language system and an API to access router state and affect processing. It does not address the resource management issues inherent in allowing more complex programs.

The Liquid Software project addresses advances in language technology that make it possible to support *mobile code* in different parts of a distributed system [Hartman96] with a focus on fast compilation and interpretation technology. Many results in this space will be applicable to the more limited domain of remote execution at storage devices.

Remote execution has existed for 15 years in the Postscript language used in printers which allows formatting to be described more compactly and flexibly as a program for drawing a page, rather than as a bitmap of the final result [Perry88]. Postscript is specialized for page description, but provides many features of a higher-level programming language and can even (with a bit of practice and patience) be written by users directly. Issues of safety and resource management are not paramount as a specific job has complete control of the device while it is active and there is a power switch to "preempt" misbehaving programs.

Extension and specialization of operating system functionality via application- or user-defined extensions is growing in popularity and sophistication. Simple mechanisms such as packet filters [Mogul87] provide ways to add user-defined functionality in a limited, yet powerful way. Allowing application code to be added directly into the operating system at specific points can improve performance over the normally rigid operating system interfaces [Bershad92]. At the extreme, moving policy decisions completely out of the operating system kernel and placing them under user control allows even greater flexibility to applications and promises further improvements in performance [Engler95] at the cost of losing control of some resource management.

Small and Seltzer [Small95] compare several techniques for safely extending operating system code and provide a taxonomy of available techniques. Several compiled technologies (Modula-3 and SFI) show promise, while interpreted languages impose too high an overhead for in-kernel processing. The authors divide extensions into three categories: *Streams* for applications that transform data or perform incremental computation (which we call Filters), *Prioritization* for applications which simply require choosing from a list of candidates (which come under our Batching and Real-Time headings), and *Black Boxes* which are provided with some input, produce a single output and are allowed to maintain some amount of internal state.

General-purpose database query languages such as SQL operate within a limited set of functions and semantics and the mechanisms for scheduling and optimizing such queries are well-established. The use of stored procedures and user-defined functions that execute directly on database servers is available today, but query optimization for such functions is often primitive.

Query optimization mechanisms are being extended to work across distributed systems, taking into account the most appropriate location for a particular bit of computation [Franklin96]. Application-specific extensions to the type system and processing in object-oriented databases such as DataBlades [Stonebraker96] provide flexibility and promise improved performance for specialized data types.

Several mechanisms exist to improve the safety of extension code. The use of type-safe languages and compilers that certify the safety of a piece of code [Bershad92] provides a way to ensure integrity, but does not protect efficiency. Using a type-safe language and limiting access to system resources through interface constraints [Gosling96] allows greater control, but does not address all safety and efficiency concerns [McGraw96]. Proof-Carrying Code [Necula96] promises a mechanism to prove that a piece of user-provided code meets a specific safety policy and may also allow the checking of resource constraints, assuming certain restrictions on the generality of the extension code.

There has been considerable work on optimizing safe languages such as Java through the use of Just-In-Time compilation [Gosling96] or translation [Proebsting97]. Small-footprint Java implementations are becoming available for embedded devices. A recent product announcement goes so far as a smart card that provides a Java Virtual Machine in 4K of ROM and can run bytecode programs up to 8K in size for a significant subset of the language [Schlumberger97]. This demonstrates that it is possible to implement a workable subset of the JavaVM in a very limited resource environment. Other systems such as Inferno [Inferno97] are specifically targeted for embedded, low-resource environments.

6. Conclusions and Future Work

Technology trends are making possible disk drives with significant amounts of available processing power. With such drives, the balance of the aggregate computing power in systems may often be in the storage devices, rather than in the host systems. We propose a way to take advantage of these developments with Active Disks - advanced disk drives that provide an environment for executing application-specific code directly inside the drive.

Active Disks provide computational power that scales with storage capacity and allow remote functions at drives to combine knowledge of application details and drive internal state to improve performance and scalability. Executing remote functions directly on drives can significantly reduce network traffic and host system processing and can enable novel applications that are not possible within the simple interface provided by today’s drives. Applications that could benefit from this technology exhibit critical components that filter, schedule, batch, manage data, or extend the semantics of the storage interface.

Using an analytic model of database select, we show possible speedups of factors of two or more for highly selective processing of large relations. Similarly, for sorting of large databases, we show how a complete network copy step can be eliminated through the use of Active Disks.

Our future work will focus on providing answers to the questions raised here on what language technology is most appropriate, how the internal interfaces for Active Disks should be designed, and how to properly partition applications across hosts and drives.

7. Acknowledgements

All the members of the NASD project contributed their thoughts and experience to the ideas presented here, including Khalil Amiri, Fay Chang, Howard Gobioff, Tara Madhyastha, David Nagle, David Rochberg and Jim Zelenka. We also thank all the other members of the Parallel Data Lab who work in the background to make our research possible. We thank our industry partners for helping us to understand the issues, problems, and opportunities surrounding Active Disks, including John Wilkes and Richard Golding of Hewlett-Packard, Satish Rege and Mike Leish of Quantum and Dave Anderson of Seagate.

This research is sponsored by DARPA/ITO through ARPA Order D306, and issued by Indian Head Division, NSWC under contract N00174-96-0002. The project team is 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 Inc., Compaq Corporation, Data General, Quantum Corporation, Seagate Technology, 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.

8. References

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

[Arpaci-Dusseau96] Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H., Culler, D.E., Hellerstein, J.M., Patterson, D.A. "High-Performance Sorting on Networks of Workstations" *International Conference on Management of Data (ACM SIGMOD '97)*. June 1997.

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

[Bershad92] 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" *Fifteenth ACM Symposium on Operating System Principles*. December 1995.

[Blelloch96] Blelloch, G.E., Leiserson, C.E., Maggs, B.M., Plaxton, C.G., Smith, S.J., Zagha, M. "An Experimental Analysis of Parallel Sorting Algorithms" *Communications of the ACM*. To appear.

[Cabrera95] Cabrera, L.F., Rees, R., Steiner, S., Penner, M. and Hineman, W. "ADSM: A Multi-Platform, Scalable, Backup and Archive Mass Storage System" *IEEE COMPCON*. March 1995.

[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.

[Dar96] Dar, S., Franklin, M.J., Jonsson, B.P., Srivastava, D. and Tan, M. "Semantic Data Caching and Replacement" *22nd International Conference on Very Large Databases*. September 1996.

[Engler95] Engler, D.R., Kaashoek, M.F., O'Toole, J. "Exokernel: An Operating System Architecture for Application-Level Resource Management" *Fifteenth ACM Symposium on Operating System Principles*. December 1995.

[Franklin96] Franklin, M.J., Jonsson, B.P. and Kossmann, D. "Performance Tradeoffs for Client-Server Query Processing" *International Conference on Management of Data (ACM SIGMOD '96)*. 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" *Int'l Conference on Measurements and Modeling of Computer Systems (SIGMETRICS '97)*. June 1997.

[Gibson97a] Gibson, G., Nagle, D., Amiri, K., Chang, F., Gobioff, H., Riedel, E., Rochberg, D., and Zelenka, J. "Filesystems for Network-Attached Secure Disks" *Technical Report CMU-CS-97-112*. March 1997.

[Gobioff97] Gobioff, H., Gibson, G. and Tygar, D. "Security for Network Attached Storage Devices" *Technical Report CMU-CS-97-185*. October 1997.

[Golding95] Golding, R., et al. "Idleness is not sloth" *USENIX Technical Conference*. 1995.

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

[Hartman96] Hartman, J., Manber, U., Peterson, L. and Proebsting, T. "Liquid Software: A New Paradigm for Networked Systems" *Technical Report 96-11*. University of Arizona. 1996.

[Inferno97] "Inferno: Tomorrow's Full Service OS…Today" *www.lucent.com/inferno*. November 1997.

[deJonge93] deJonge, W., Kaashoek, M.F. and Hsieh, W.C. "The Logical Disk: A New Approach to Improving File Systems" *Fourteenth ACM Symposium on Operating System Principles*. December 1993.

[Kotz94] Kotz, D. “Disk-directed I/O for MIMD Multiprocessors” *First Symposium on Operating System Design and Implementation*. November 1994.

[McGraw97] McGraw, G. and Felten, E.W. *Java Security: Hostile Applets, Holes, and Antidotes*. John Wiley & Sons, 1997.

[Mogul87] Mogul, J.C., Rashid, R.F. and Accetta, M.J. “The Packet Filter: An Efficient Mechanism for User-level Network Code” *ACM Symposium on Operating System Principles*. November 1987.

[Necula96] Necula, G.C. and Lee, P. “Safe Kernel Extensions Without Run-Time Checking” *Second Symposium on Operating System Design and Implementation*. October 1996.

[Perry88] Perry, T.S. “’PostScript’ prints anything: a case history” *IEEE Spectrum*. May 1988.

[Prabhakar97] Prabhakar, S., Agrawal, D., Abbadi, A.E., Singh, A. and Smith, T. “Browsing and Placement of Images on Secondary Storage” *IEEE International Conference of Multimedia Computer Systems*. June 1997.

[Proebsting97] Proebsting, T.A., Townsend, G., Bridges, P., Hartman, J.H., Newsham, T. and Watterson, S.A. “Toba: Java For Applications A Way Ahead of Time Compiler” *Technical Report TR97-01*. University of Arizona. January 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 VII*. October 1996.

[Schlumberger97] Schlumberger Limited “First-Ever Java-Based Smart Card Demonstrated by Schlumberger” *www.slb.com/ir/news/et-java0497.html*. April 1997.

[Sirer96] Sirer, E.G., Savage, S., Pardyak, P., DeFouw, G.P. and Bershad, B.N. “Writing an Operating System in Modula-3” *Workshop on Compiler Support for System Software*. February 1996.

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

[Smirni96] Smirni, E., Aydt, R.A., Chien, A.A., and Reed, D.A. “I/O Requirements of Scientific Applications: An Evolutionary View” *Fifth IEEE Conference on High Performance Distributed Computing*. August 1996.

[Stonebraker97] Stonebraker, M. “Architectural Options for Object-Relational DBMSs” *White Paper*. Informix Software, Inc. 1997.

[Tennenhouse96] Tennenhouse. D.L., et al. “A Survey of Active Network Research” *SIGOPS 96*. 1996.

[TPC97] TPC-C Rev. 3.3 Rating for a Compaq ProLiant 7000 6/200 Model 1S. Transaction Processing Performance Council. *www.tpc.org*. October 1997.

[Turley96] Turley, J. “ARM Grabs Embedded Speed Lead” *Microprocessor Reports 2,10*. February 1996.

[Wahbe93] Wahbe, R., Lucco, S., Anderson, T.E. and Graham, S.L. “Efficient Software-Based Fault Isolation” *Fourteenth ACM Symposium on Operating System Principles*. December 1993.

[Wetherall96] Wetherall, D.J. and Tennenhouse, D.L. “The ACTIVE IP Option” *7th ACM SIGOPS European Workshop*. September 1996.

[Wilkes95] Wilkes, J., Golding, R., Staelin, C. and Sullivan, T. “The HP AutoRAID Hierarchical Storage System” *Fifteenth ACM Symposium on Operating System Principles*. December 1995.