MIDAS: An Execution-Driven Simulator for Active Storage Architectures

Shahrukh Rohinton Tarapore[†] Clinton Wills Smullen, IV[‡] Sudhanva Gurumurthi[‡]

[†] Lockheed Martin Advanced Technology Labs
Cherry Hill, NJ 08002
starapor@atl.lmco.com

[‡] Department of Computer Science
University of Virginia
Charlottesville, VA 22904
{cws3k, gurumurthi}@cs.virginia.edu

Abstract

*Many applications today are highly data intensive and have stringent performance requirements. In order to meet the performance demands of these applications, we need to optimize both the processing and I/O subsystems. One promising approach to optimize performance is to use "Active Storage" systems, where we use disk drive controllers and storage array controllers as offload processors for the data intensive parts of an application and exploit Data-Level Parallelism (DLP) across the ensemble of processing components. From the architecture viewpoint, the design space of such active storage systems is large. There are a number of choices for the microarchitecture of the processors at the disk, storage array controller, and host, the organization of the electro-mechanical parts of the disk drive, and the characteristics of the interconnection network between these components. Since these design choices can have a significant impact on performance, we need a simulator that can give us detailed insights into the behavior of these systems. This paper presents the Modeling Infrastructure for Dynamic Active Storage (*MIDAS*). MIDAS is an accurate execution-driven simulator that captures both the processing and I/O behavior of active storage systems. We describe the design of MIDAS, providing details about the various simulation models and how they interact. We then present three case studies that illustrate how MIDAS can be used to study architectural design tradeoffs in active storage systems.*

1 Introduction

Computing today is highly data-driven. Many applications process large volumes of data for which a low turnaround time is required. These applications span several fields, such as, business analytics, bioinformatics, scientific data processing, unstructured data processing, and geographical information systems. However, efficiently processing such large volumes of data is a challenging problem [24]. For example, a business analyst often needs to interactively query a large database whose contents might be continuously changing due to round-the-clock customer transactions over the Internet. Such a query can take several hours to complete, even if the database is run on a high-performance cluster [19].

Due to the data intensive nature of many of applications, the "I/O Path" plays an important role in determining their overall performance. The I/O path is a hierarchy of components consisting of disk drives, array controllers, and interconnection networks. Due their impact on performance, a considerable amount of effort has been put into optimizing the I/O path, by using faster and higher bandwidth interconnects, faster disks, etc. However, power consumption poses a significant problem for these traditional performance scaling techniques [7, 12]. On the other hand, many of the I/O path components have processing capabilities that can be utilized for general-purpose computation. By tapping into this processing power, the multiple disks, each with their local processor and data, can now be viewed as Processing Elements (PEs) of a multiprocessor. These PEs can then be collectively used to exploit Data-Level Parallelism (DLP) in applications. This computing paradigm, referred to as "Active Disks" or "Active Storage", has been proposed at different points over the past three decades [3, 1, 16, 11].

Despite this research, the end result has been rather disappointing: active disks, due to their limited processing power, are capable of handling only very simple operations (e.g., data scans and counting [16, 1]). Anything more complex has been relegated to high-end storage clusters and appliances [8, 23], whose hardware, power, and cooling costs are significantly higher than disk drives. Indeed, commercial systems that perform some processing on the storage-side are built using such appliances [5, 23]. Although the prior work has motivated what active storage could do, enabling this technology requires *computer architecture support*.

The architecture space of active storage is large. For example, we can have disk drives of different RPMs, and disk, storage, and host processors with varying microarchitectural characteristics. Exploration of this rich design space

requires simulation tools that can provide detailed insights into the behavior of the system when running data intensive applications. In this paper, we introduce the *Modeling Infrastructure for Dynamic Active Storage (MIDAS)*. MIDAS provides execution-driven simulation of both the processing and I/O behavior of an active storage system. MIDAS also provides an API to facilitate the parallelization of applications to be simulated.

The organization of the rest of this paper is as follows. The next section presents an overview of the related work. Section 3 presents the design of MIDAS. The workloads used in this paper are described in Section 4 and Section 5 provides details about the validation. Section 6 gives the experimental setup. In Section 7, we show how MIDAS can be used for design space exploration of active storage systems. Section 8 concludes this paper.

2 Related Work

Active storage systems were first proposed as "database machines" to accelerate query processing [3]. Although this idea did not initially gain much traction, primarily due to cost reasons, there has been a resurgence of active storage research starting from the late nineties [1, 16, 11]. Acharya et al. [1] and Riedel et al. [16] investigated using the processor inside vanilla disk drives to exploit DLP. Keeton et al. [11], on the other hand, proposed replacing a traditional disk array with a storage server. The closest that these prior work came to investigating architecture alternatives is to evaluate the impact of different disk processor speeds on performance [1, 25, 13].

Another approach to storage-side computation is to use "Semantically Smart" disk drives [18], where the disk processor runs code that is application-aware and proactively optimizes the storage system. Both the active and semantically-smart techniques use the same underlying technique of running general-purpose computation on the disk processor. Their primary differences lie in how the processing capabilities are used (offloading an application vs. running a storage-side optimizer). Since our focus in this paper is on the design of the underlying hardware, we do not make an explicit distinction between these two techniques to optimize system performance.

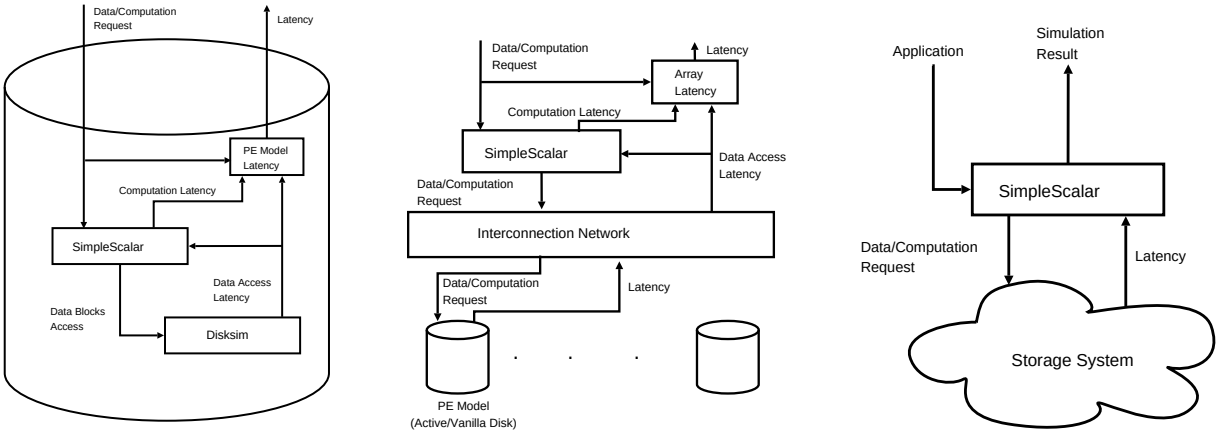
Prior evaluation methodologies in the area of active storage have used real system emulation [16] and software-based simulation [1, 25, 13]. The emulation approach, although useful for designing active disk algorithms, does not provide the flexibility of exploring various hardware alternatives. ADSim [1] is a simulator that uses a simplistic model of the storage system and a very coarse grained model of the processor. In this simulator, a trace is collected from a real system and is fed into ADSim to represent the execution of the processor. The effect of varying the disk processor clock frequency is simulated by scaling the timestamps of this input trace. DBSim [13] is another simulator that was specifically designed to simulate DBMS operations and its design is similar to ADSim. In this paper, we show that the microarchitectural design and data layouts can have a significant impact on performance and therefore it is important to consider these parameters when simulating active storage systems.

3 Design of MIDAS

MIDAS simulates a host system interacting with the I/O path via an interconnection network. The simulated I/O path can include disk drives with programmable processors (which we shall refer to hereafter as the *Disk Processing Unit* or *DPU*) and programmable storage controllers. The microarchitecture of each of these components is configurable. Using this framework, we can explore the effects of different processor microarchitectures, physical disk and network designs, and communication protocols on application performance. In this section, we give an overview of the MIDAS design, providing details about its constituent simulation models, and how they integrate to simulate entire systems. Before we explain the design of the simulator, we would like to clarify our terminology. We use the term *model* to refer to an implementation of a hardware component in the *simulator* and the term *instance* to mean a physical entity that is *simulated*.

The basic building block in MIDAS is the Processing Element (PE) model. The PE model consists of a processor, which is capable of running ad-hoc code, interacting with a disk drive. The processor and disk models are glued together via a layer called the *Space Manager*. The starting points for building MIDAS are Simplescalar [4] and Disksim [6], which simulate the processor and disk respectively. Simplescalar is an execution-driven simulator that allows one to study a wide variety of processor and cache organizations. Disksim is an event-driven simulator and has detailed parameterized models for disk drives, storage caches, and interconnects.

Using the PE as the basic building block, we can compose simulation models for higher-level system organizations, such as, disk arrays and even a host system interacting with the I/O path. An active disk is simulated by executing code on the PE processor (which corresponds to the DPU) and interacting with its corresponding PE disk, via the Space Manager. An "array of active disks" consists of multiple PE instances, one of which serves as the array controller. The array controller PE instance does not use its own disk model; it uses the MIDAS API to communicate with other PEs. Likewise, we can disable the processor model in a PE to simulate a vanilla disk drive that does not perform any local computation. MIDAS simulates and coordinates the execution of code across multiple PE instances via a Checkpointing Mechanism. An "active storage system" consists of multiple active disks or arrays communicating with a host machine. The host PE instance, like the array controller, uses the MIDAS API for its communication. The actual data transfers between all these hardware components is simulated by the Network Model. This hierarchical design is shown in Figure 1. We now describe how



(a) Processor Element Model (b) Array of Active Disks (c) Active Storage System

Figure 1. Overall Design of MIDAS.

each of these parts is implemented.

3.1 Processing Element (PE) Model

The PE model forms the core of MIDAS. In general, a PE instance consists of a processor that is interfaced with a local disk drive. Translation of memory addresses (used by the processor model) to physical disk blocks (used by the disk model) is handled by the Space Manager. From an implementation viewpoint, we use the Simplescalar `sim-outorder` simulator as our processor model, which we extend to simulate a finite-size main memory. This main memory serves as the on-board disk cache, an array controller cache, or the main memory of the host system, depending upon how a specific PE instance is configured for simulation. For simulating the PE disk, we use the Disksim disk drive timing model. This disk drive model captures the mechanical overheads of a disk access (e.g., seek time and rotational latency) based on characteristics, such as, the drive RPM and the number of disk zones.

The processor model interfaces with the disk model at three points, namely, instruction cache misses, data cache misses, and system calls. At anytime during the execution of an application, if the processor needs to access data that is not resident in its memory hierarchy, a disk access is made. This access is simulated by the processor model generating an I/O request to the disk model. The disk model uses information about the current state of the disk (e.g., location of the disk arm) and the I/O request, to determine the time for the disk access and returns this latency to the processor model. This procedure is illustrated in Figure 1(a). This latency is then applied to the instruction that generated the I/O request, thereby delaying its completion time. Care is taken to ensure that future instructions that depend on the delayed instruction realize a penalty and independent instructions do not, subject to any structural hazards in the processor pipeline. Interfacing the processor and storage models requires simulating file accesses and swapping. MIDAS handles these tasks via the *Space Manager*.

3.2 Space Manager

Simplescalar does not simulate the communication between the processor and the memory hierarchy that lies beyond the L1 and L2 caches. Disksim, on the other hand, provides detailed timing information about disk accesses for a given I/O request but does not simulate the hardware layers above the storage system. Due to this disconnect between the simulators, neither of them provide any form of simulation for system calls, file operations, and swap space management. The Space Manager is responsible for providing the basic functionality for allowing the processor and disks models to be integrated.

Note that the Space Manager is not intended to simulate an operating system. Instead, the Space Manager is meant to facilitate integrating the processor and disk models with sufficient detail so that one could conduct architectural/microarchitectural exploration of active storage systems. However, compared to evaluation infrastructures used in prior active storage research (described in Section 2), the level of simulation detail provided by MIDAS is significantly higher.

3.2.1 Simulating File Accesses

The Space Manager models a simple FAT-like [15] file system. This file system is exercised at the points where the processor and disk model interact, namely, during cache misses and system calls. We model the basic space management tasks of data layout management and disk block allocation. When the file system is initialized by the Space Manager, we layout the file data on disk using a user-specified configuration file. The configuration information consists of a list of files that are used by the simulated application, the simulated disks on which they are to be stored,

and the specific layout policy to be used by the Space Manager. These layout policies vary from contiguous allocation of the blocks of a file on disk, to those that allow files to be fragmented. When a processor makes a request for file data, the Space Manager translates the high-level file requests into logical disk block addresses, which are then presented to the disk model. The disk model converts these logical block requests into physical disk addresses, consisting of cylinder, surface, and sector numbers.

We implement disk block allocation as a bidirectional scheme that is a hybrid of the First-Fit and Best-Fit policies [17]. The block allocation scheme starts searching from the last block associated with the file on disk along two directions, going closer to the center of the platter and going farther from it, and gives preference to the free block which is on an outer track (since the outer tracks experience higher data rates).

Simplescalar does not provide detailed simulation of system calls. MIDAS captures the I/O behavior of system calls in detail. The Space Manager simulates the following UNIX-style system calls: `open`, `close`, `seek`, `read`, `write`, and `creat`.

3.2.2 Simulating Swapping

Simplescalar does not bound the size of the simulated main memory. This simulation methodology is not well suited for simulation of active storage systems in which applications process data that might not fit within the main memory of the processing devices involved in the computation. This is especially true for active disks, which are highly memory constrained.

In MIDAS, whenever code or data resident on a PE cannot completely fit within its own main memory, the Space Manager handles their swap to disk and their subsequent possible reloading into memory. For PE instances simulated as active disks, their local disk serves as their swap space. To handle PE instances that do not have a local disk, we can designate "swap areas" on other PE instances that have a local disk drive. Mappings between a PE instance and its corresponding swap area are specified via a configuration file. These mappings are used by the Space Manager to simulate swapping.

3.3 Checkpointing Mechanism

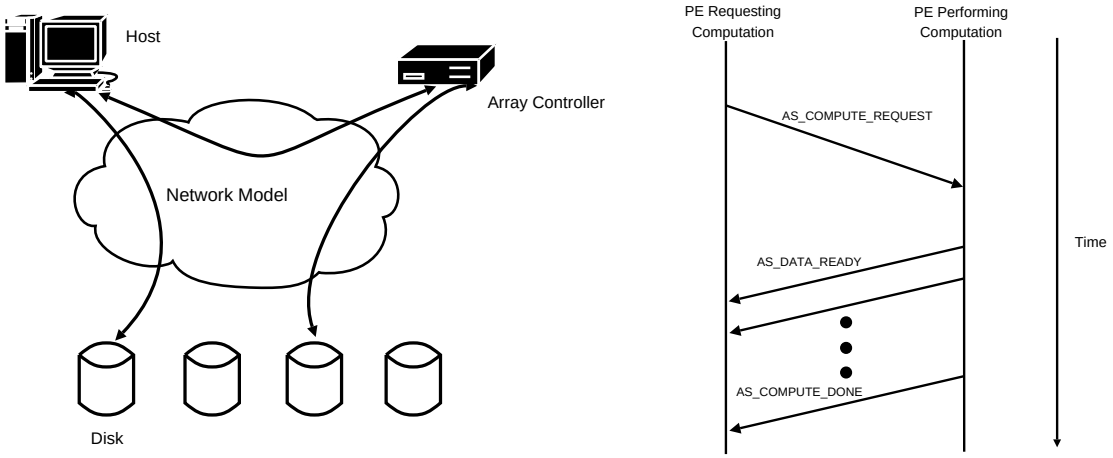
Since MIDAS simulates multiple PE instances, the state of the processor structures (both architected and microarchitected) and the contents of the entire storage hierarchy of each PE instance needs to be tracked on a per-cycle basis. In addition to the processor and memory states, state information pertaining to the disk model (e.g., the exact location of the disk arm) needs to be tracked as well. All this state information is tracked using a checkpointing mechanism. From the implementation viewpoint, the checkpointing mechanism allows an arbitrary number of PE instances to be simulated using a single PE model. This is achieved by maintaining an array of data structures, one for each PE instance, which contains the full PE state. When the simulation is in progress, for each PE instance, we restore its checkpoint from its previous cycle, simulate a single cycle on that PE, record its newly generated state, and then move onto the next PE instance.

3.4 Network Model

MIDAS simulates the communication between various PE instances via the Network Model. The possible communication paths simulated by the Network Model are shown in Figure 2(a). The Network Model is implemented as a pluggable module and exports a well-defined interface to the rest of MIDAS. This is primarily done so that we can incorporate a variety of interconnection network topologies and characteristics without having to modify other parts of the simulator.

The primary input parameter to the Network Model is the bandwidth. The Network Model is invoked either explicitly, using the MIDAS API, or implicitly through system calls from the host or array controllers that need to read or write data that may reside on remote PE instances. In either case, there are a set of basic operations required to simulate traffic over the network. The PE that wishes to access the network invokes a function called `network_latency`, which takes the following parameters: destination PE identifier, source PE identifier, and the size of the data to be transferred. The identifiers for the various PEs are assigned at the beginning of the simulation. Using this data and additional information about the current state of the network, `network_latency` calculates the time required to transfer a packet containing the data from the source to the destination. The implementation of the `network_latency` function depends on the specific network model that is plugged into MIDAS. The following discussion focuses on the default network model that is implemented in MIDAS. We plan to create a library of simulation models for more sophisticated networks and protocols in the future.

In the default network model, the communication time is calculated assuming that the network employs Frequency Division Multiplexing (FDM) to apportion the network bandwidth among the various devices. Each PE instance is assumed to be connected to a single full-duplex link, whose bandwidth is equal to the total network bandwidth divided by the number of PE instances. The network model captures contention and queuing at each of the links. If there are multiple requests pending for a particular PE instance, the requests are buffered on behalf of the destination PE and are serviced in FCFS order. If a PE instance is unable to transmit data due to insufficient bandwidth on its local pipe, the request is buffered within a queue on behalf of the sender. Since a single network packet is generated for an entire buffer worth of data, each pending request has to wait, in FCFS order, for sufficient bandwidth on its local pipe before it can attempt transmission. The network model uses information about the residual bandwidth of the sender's local pipe and the size of the network packet, to calculate the total transmission time. To accommodate cir-



(a) Possible communication paths between simulated devices.

(b) Inter-PE communication using the MIDAS API.

Figure 2. MIDAS Network Model and API.

cumstances where there is heterogeneity in the link capacities between certain PE instances, the network model allows the bandwidth of the links to be overridden, on a link by link basis, from the default FDM-based assignment.

3.5 MIDAS API

A key feature of active storage is the decomposition, orchestration, and mapping of computation to different processing sites on the I/O path. MIDAS provides an API to port parallel applications to the simulator. These API functions resemble Remote Procedure Calls. There are three API functions: `AS_COMPUTE_REQUEST`, `AS_DATA_READY`, and `AS_COMPUTE_DONE`. The usage of these functions is shown in Figure 2(b).

`AS_COMPUTE_REQUEST` is used to request a particular computation to be performed by a remote device. For example, the host can request all active disks in the system to scan through their local data. This function takes, as input, the destination PE identifier and the arguments for the remote procedure. The API call invokes the Network Model to simulate the latency for transmitting the request to the destination PE. The remote PE instance then invokes the appropriate procedure on its local data. Processed data is returned to the requester via the `AS_DATA_READY` API call. Each remote PE instance might send a stream of processed data back to the original requester, which is achieved through multiple invocations of the `AS_DATA_READY` function. Each remote computing device indicates the end of its data stream via the `AS_COMPUTE_DONE` API call.

4 Workloads

A key challenge in evaluating active storage is that traditional benchmark suites used in computer architecture research, such as SPEC [22], are not representative workloads. We require applications that do significant amounts of I/O as well as computation. Other desired application characteristics include data-parallelizability and high selectivity, which is a measure of the data reduction achieved as a result of storage-side processing. Developing a representative benchmark suite for active storage is an open research problem, which we do not attempt to tackle in this paper. Instead, we choose workloads that have been used in prior active storage research [1, 16]. (The workloads are compiled for the Simplescalar PISA ISA).

- **Image Edge Detection:** The objective of this workload is to detect the facial features of subjects (which are edges) in each image. The edge detection is performed using the Smallest Univalve Segment Assimilating Nucleus (SUSAN) algorithm [20]. Our dataset consists of a set of images from the MIT CBCL Face Recognition Database [14]. In the nonactive storage implementation of this workload, the host requests an image from the storage system, performs edge detection, and then requests the next image. Since the host requests one image at a time and the image sizes in the database are relatively small (around 10 KB per image), we do not stripe the disk blocks within individual files across multiple disks. However, we do distribute the contents of the database over multiple disks. In the active storage version of this procedure, each DPU performs the computation on its local set of images and streams the edges back to the host. The host itself does not perform any computation on the images.

- **Nearest Neighbor Search:** This workload implements a commonly used search procedure. This procedure works as follows: given a search criteria and non-negative integer k, the nearest neighbor search returns the k database records that match or come clos-

est to matching the criteria. We use a database from the National Oceanic and Atmospheric Administration (NOAA) National Hurricane/Tropical Prediction Center as our dataset [10]. This database contains information about tropical cyclones in the North Atlantic from the years 1851-2005, with storm progressions recorded at six-hour intervals. Each record in this database has several fields, including, the date and time of the storm, the latitude and longitude of the storm, etc. For a given (latitude,longitude) pair, we compute the k tropical storms that occurred closest to the given coordinate. The search procedure scans through the entire database, calculating the Euclidean distance between the search coordinates and the (latitude,longitude) pair of each stored record, maintaining a list of the k closest matches. We resort to a full scan of the database for each query, since scanning has been shown to be efficient for such multi-attribute searches [16]. The non-active storage implementation of this workload reads in the entire database from disk, performs the search, maintaining the k closest matches. Since the entire database needs to be scanned to compute the query result, we assume that we can transfer data to the host from the multiple disks, in parallel. The active storage implementation performs the search locally on each disk and returns its k closest matches to the host. The host then calculates the k closest matches from this reduced data set. In our experiments, we set k=3.

5 Validation

Validation is an important step in the design of any architecture simulator. We evaluated the accuracy of the simulator against real hardware. Although it is challenging to validate the simulator against a real active storage system (since programmable disks and controllers are not available), it is possible to validate against existing systems.

We ran the Image Edge Detection workload on three different systems: a high-end laptop, a slightly old desktop machine, and a high-end server. This workload was chosen because it performs a significant amount of both computation and I/O and can therefore exercise all parts of the simulator. The laptop had a 1.83 GHz Intel Pentium-M processor, 1 GB of main memory, and a 80 GB 5400 RPM SATA disk drive. The desktop was composed of a 350 MHz Pentium II processor, 384 MB of main memory, and a 40 GB 7200 RPM IDE disk drive. The server had a 2.0 GHz AMD Opteron processor, 2 GB of main memory, and a 160 GB 7200 RPM SATA disk drive. Each of these systems ran the Linux operating system, with the Ext3 file system.

We configured MIDAS to model the hardware of these three machines as closely as we could. We conducted independent runs on each of the real machines, rebooting the system between successive runs to ensure that caching effects do not skew our measurements. We used the confidence interval method to determine the number of independent runs required to get the execution time to be accurate within 1 second at the 95% confidence level. We then compared the results obtained from these runs to the simulated result reported by MIDAS. The results from this validation are given in Figure 3. For each machine, shown on the x-axis, the pair of bars give the execution time of the workload on the simulated and real system respectively. The standard error (in seconds) for the real system runs are shown above the corresponding bars.

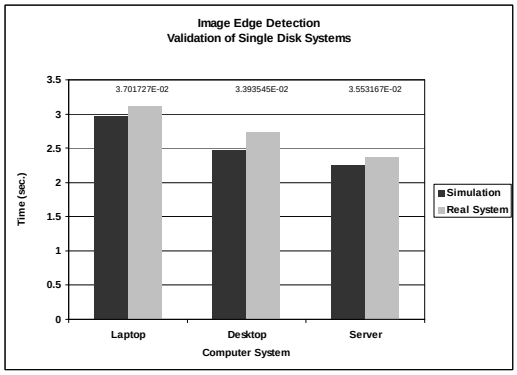


Figure 3. Execution time of the Image Edge Detection application on real systems and their simulated counterparts.

As we can see from the Figure, the execution time on the real machines and their simulated counterparts are very close. The maximum difference between the two is 8.9%, which is reported for the desktop machine. The timing numbers obtained from the real system runs are slightly higher than those reported by MIDAS primarily due to the operating system, whose execution overheads are not simulated in MIDAS. Nevertheless, the differences are small and MIDAS captures the behavior of real systems accurately.

6 Experimental Setup

The parameters used in the simulations are given in Table 1. We simulate the host machine as having an 1.6 GHz 8-wide issue processor, with a 32 KB L1 and 64 KB L1 d-cache and i-cache respectively (both 2-way set associative), a 4-way set-associative 512 KB unified L2 cache, and 512 MB of main memory. The DPU is modeled to resemble the ARM-based Intel XScale-based PXA255 processor [9]. Modern SCSI disk drives use disk processors that are similar to the PXA255 [2] and have 8-32 MB of DRAM for their disk cache. Since the PXA255 processor is available in three different clock frequencies (200, 300, and 400 MHz), we use these DPU speeds in our experiments. We choose the bandwidth of our simulated interconnection network to be 533 MB/s, which is similar to PCI Express.

Host Processor Parameters

Processor Clock-Frequency	1.6 GHz
Fetch/Decode/Issue/Commit Width	8
Fetch-Queue Size	8
Branch-Predictor Type	Bimodal Table-size of 2K entries
RAS Size	64
BTB Size	2K-entry 2-way
Branch-Misprediction Latency	7 cycles
RUU Size	128
LSQ Size	64
Integer ALUs	4 (1-cycle latency)
Integer Multipliers/Dividers	2 (3,20)
FP ALUs	2 (2)
FP Mult./Div./Sqrt.	1 (4,12,24)
L1 Cache Ports	2
L1 D-Cache	32KB, 2-way with 32B line-size (2)
L1 I-Cache	64KB, 2-way with 32B line-size (2)
L2 Unified Cache	512 KB, 4-way with 64B line-size (12)
Main Memory Size	512 MB
TLB Miss-Latency	30 cycles
Main Memory Latency	64 cycles

Disk Processor Parameters

200,300,400 MHz
1
1
Bimodal Table-size of 128 entries
8
128-entry direct-mapped
4 cycles
16
8
1 (1-cycle latency)
1 (3,20)
1 (2)
1 (4,12,24)
2
32KB, 32-way with 32B line-size (2)
32KB, 32-way with 32B line-size (2)
32 KB, 32-way with 32B line-size (12)
32 MB
20 cycles
64 cycles

Disk Drive Parameters

Disk Capacity	9.29 GB
Rotational Speed	10,000 RPM
Average Seek Time	4.48 ms
Platter Diameter	3.3 inches
# Platters	1

Interconnection Network Parameters

Total Bandwidth	533 MB/s
Transmission+Time of Flight Overhead	0.18 ms
Message Header Size	1 B

Table 1. Configuration parameters of the host, DPU (default values), the disk drive (data transfer system), and the interconnection network. Latencies of ALUs/caches are given in parenthesis. All ALU operations are pipelined except division and square-root.

7 Using MIDAS to Explore Active Storage Architectures

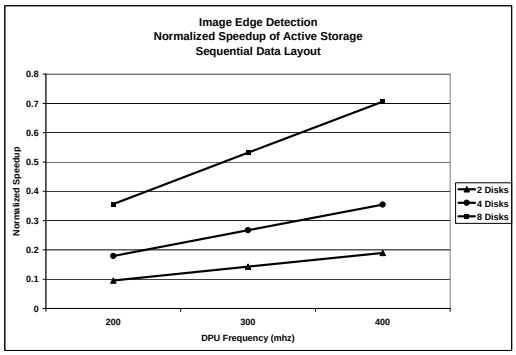
We now present three examples of how MIDAS can be used to explore architectural tradeoffs in active storage systems. First, we study the effect of varying the DPU clock frequency on the performance, assuming all I/O to be sequential. This first case study focuses solely on the processing aspects of active storage. In the second case study, we factor in the effects of the electro-mechanical parts of the disk drive by introducing random I/O. The final case study takes full advantage of the microachitectural simulation capabilities of MIDAS to evaluate the impact of varying the microarchitecture of DPUs. Please note that these examples are not intended to comprehensively explore the design space of active storage systems, nor meant to suggest optimizing a particular hardware component. Instead, they are intended to demonstrate the utility of the MIDAS simulator. We have conducted a more detailed evaluation of the design space of active storage architectures using additional benchmarks. The interested reader is referred to [21] for the details of this study.

The primary metric that we use is *speedup*, which is calculated as the ratio between the total execution time of an application on a given active storage configuration to that on the corresponding baseline configuration, which does not use any storage-side computation. In all the experiments, we configure the baseline system to use the same number of disks, data distribution, and disk block layout as their active storage counterparts.

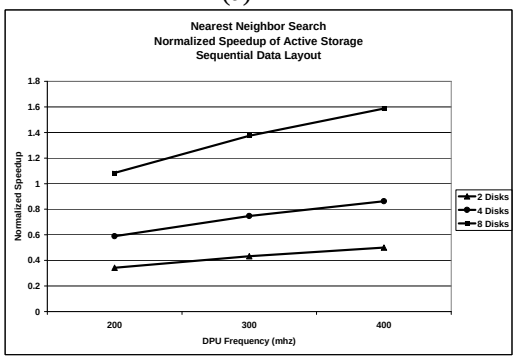
7.1 Usage Example 1: Studying the Impact of DPU Clock Frequency

In the first set of experiments, we evaluate the impact of DPU clock frequency on the performance of active storage. We vary the DPU clock frequency from 200 MHz to 400 MHz and consider storage systems with 2, 4, and 8 disks respectively. We assume that all data is accessed sequentially, and therefore disk seeks and head switches are minimal. All the disks send out a stream of data, either to their respective DPUs (in the case of active storage), or to the host system via the network (for the baseline). The results of this experiment are shown in Figure 4. The x-axis of each graph corresponds to a DPU clock frequency and the y-axis gives the speedup. Each of the curves correspond to a storage system with the given number of disks.

Looking at the trends across both the workloads, we observe that as we increase the number of disks, from 2 to 8, the performance of the active storage systems improve.



(a)



(b)

Figure 4. Impact of DPU clock frequency.

This is because, with an increasing number of active disks, the application can exploit more DLP. As expected, increasing the DPU clock frequency benefits performance as well. However, between the two workloads, we see contrasting behaviors. For the Image Edge Detection application, even the highest performance configuration (8-disk system with 400 MHz DPUs) is unable to do better than the baseline approach of doing all the computation at the host. On the other hand, the Nearest Neighbor Search application enjoys consistently higher performance than Image Edge Detection for the same number disks, and the 8-disk active storage configuration outperforms the baseline even with a 200 MHz DPU. We now explain why this happens.

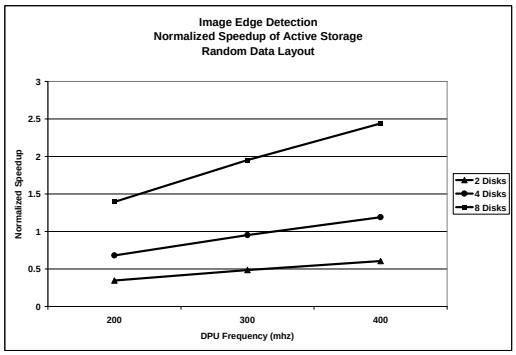
The Image Edge Detection workload fetches one image from disk, processes it, and then fetches the next image file. Therefore, in the baseline system, the host processes data in relatively small chunks. Moreover, the application code is predominantly composed of integer instructions, since the pixels of an image are encoded as integers. The host can handle this code efficiently given its relatively wide integer execution pipeline and its high clock frequency. Although the active storage configurations can process data in parallel and also have the added benefit of parallel I/O transfers, the small data footprint combined with the high computational power of the host favors the baseline approach. On the other hand, for the Nearest Neighbor Search application, the host scans the entire database to compute the query result. With sequential I/O and the parallel transfer of data from the disks, the host gets a large amount of data from the storage system at a high rate. The Nearest Neighbor Search also uses a significant amount of floating-point arithmetic, to compute the Euclidean distance of (latitude,longitude) pairs, which are represented as real numbers. However, the floating-point execution pipeline of the host processor is narrower than the integer pipeline. The large data footprint combined with fewer floating point processing resources impose a greater burden on the host system. However, as we increase the number of DPUs, we get both higher floating-point parallelism as well as data-level parallelism. Due to the large data footprint, exploiting DLP has a more significant impact on the performance of the Nearest Neighbor Search application. These factors end up favoring the active storage approach for this workload.

7.2 Usage Example 2: Studying the Impact of Random I/O

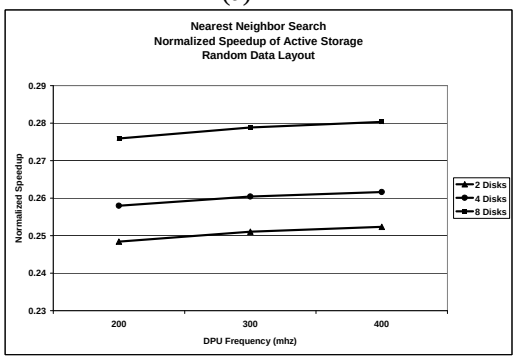
Having seen the performance of active storage under sequential I/O, we now factor in the effects of disk seeks and study the behavior under random I/O. We simulate such disk I/O behavior by choosing the starting block address of each data unit stored on disk to be at a random location. For the Image Edge Detection application, each image corresponds to an individual data unit. For the Nearest Neighbor Search workload, a set of consecutive database records that fit within a 512B disk block constitutes a data unit. We assume that all the disk blocks within a data unit are stored sequentially. Care is taken to ensure that no two data units have overlapping block addresses. The results from this experiment are shown in Figure 5.

With the introduction of random I/O, we now see a trend reversal between the two applications. For the Image Edge Detection application, even the 4-disk active storage system starts providing a speedup over the baseline and the 8-DPU 400 MHz configuration provides more than 2X speedup. On the other hand, the Nearest Neighbor Search workload suffers severe performance degradation. Neither the use of more DPUs nor higher DPU clock frequencies have appreciable impact on the speedup of this application. We shall now explain the reason for these trends.

For the Image Edge Detection application, the lack of parallel data transfers from disk has a detrimental impact on baseline performance. This is because, one of the biggest benefits of a parallel I/O system is the ability to hide the latency of seeks. In a parallel I/O system, we can allow data to be transferred from some disks while others might be seeking. In the baseline case, where code is executed sequentially, a physical seek operation would lead to a disruption in the I/O stream, during which the host processor is stalled waiting for the data to arrive. On the other hand, in an active storage system, even when one DPU is stalled due to a seek, the other DPUs can still process their local data and hence provide latency hiding. This is an interesting scenario where, in spite of the host not explicitly taking advantage of the parallelism of the I/O system, the active storage system implicitly taps into this *data-transfer* paral-



(a)



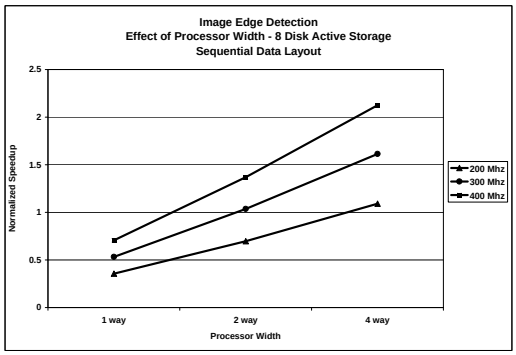
(b)

Figure 5. Impact of Random I/O.

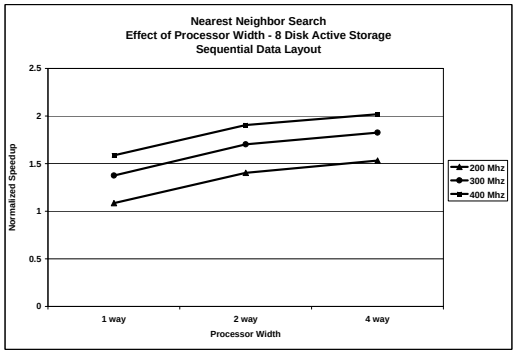
lelism. In the baseline case, we would expect to tap into the I/O parallelism from the host side of the system. For the Nearest Neighbor Search application, the parallel I/O system is available even for the baseline system. However, due to random I/O, the throughput of the storage system is lower than with sequential data access. As a result of this, the host receives the contents of the database at a lower data rate. Therefore, unlike with sequential I/O, the host gets data in smaller fragments, which it can process more efficiently. This tilts the balance towards the baseline.

7.3 Usage Example 3: Studying the Impact of Superscalar DPU Organizations

One of the main advantages of MIDAS is that we can study microarchitectural tradeoffs in active storage system design at a relatively fine granularity. Many of these tradeoffs are very challenging to study using emulation [16, 8] or the simplistic simulation techniques [1, 13] used in prior work. We now consider one such microarchitectural design parameter, namely, the width of the DPU pipeline. We explore how exploiting Instruction Level Parallelism (ILP) at the DPUs could affect the performance of an active storage system. We consider 2-way and 4-way out-of-order superscalar DPU microarchitectures. For each superscalar DPU configuration, we increase the processor resources, from those given in Table 1, to accommodate the higher pipeline widths. We assume that the data is accessed sequentially (as done in Section 7.1). The results from this experiment are shown in Figure 6 for an 8-DPU active storage system. (We conducted this experiment for the other storage system sizes as well and found the trends to be similar). For each graph in Figure 6, the leftmost point on the x-axis corresponds to the non-superscalar DPU organization that we used in the previous experiments. The y-axis indicates the speedup over the 8-disk (nonactive) baseline system. Each of the curves corresponds to a particular DPU clock frequency.



(a)



(b)

Figure 6. Impact of using superscalar processors for the DPUs.

This experiment reveals very interesting trends. First, we observe that the wider-issue superscalar DPUs improve the performance of both applications. For the Image Edge Detection workload, going in for even a 2-way superscalar DPU, with 300 or 400 MHz clock frequencies, now provides a speedup over the baseline. The wider integer execution pipeline at each of the DPUs allows a greater amount of ILP to be exploited. This higher processing parallelism on the storage side of the system now has a more dominant effect on performance than the higher clock frequency of the host, thereby tilting the balance towards active storage system. For the Nearest Neighbor Search workload, the superscalar DPUs boost performance as well, although the benefits are less pronounced than for Image Edge Detection. Although the superscalar DPUs benefit the integer instructions in the application, the width of the floating-

point pipeline increases only moderately. The number of floating-point adders and multiplier/divider units for the 2-way and 4-way DPUs are (1,1) and (2,2) respectively. The corresponding integer pipelines have (2,2) and (4,4) of these functional units for the 2-way and 4-way configurations respectively.

Within the speedup profiles for each of the workloads, we observe interesting tradeoffs between clock frequency and ILP. For the Image Edge Detection workload, we observe that a superscalar DPU that runs at 200 MHz is able to out-perform single-issue DPUs that run at a higher clock frequency (the 2-way and 4-way organizations provide a greater speedup than the 300 MHz 1-way and 400 MHz 1-way configurations respectively). In fact, the 4-way 200 MHz DPU provides slightly better performance than even the 2-way 300 MHz counterpart. Similar tradeoffs can be observed for the Nearest Neighbor Search workload as well, although the differences are less pronounced.

This case study indicates that there is indeed scope for exploring DPU organizations in more detail. MIDAS provides the necessary capabilities for conducting such architecture-oriented research on active storage.

8 Conclusions

This paper has presented MIDAS, which is an accurate execution-driven simulator for studying the architecture of active storage systems. We have described the design of the simulator, providing details about the Processing Element and Network Models, the Space Manager, and the API for writing parallel applications. Using two data intensive workloads, we have demonstrated how MIDAS can be used to study various architecture tradeoffs in the processing and the data-transfer subsystems.

9 Acknowledgements

We thank Parthasarathy Ranganathan, Mustafa Uysal, and Kevin Skadron for their inputs. This research has been supported in part by NSF CAREER Award CCF-0643925, NSF grants CNS-0551630, CNS-0627527, and gifts from Intel, HP, and Google.

References

[1] A. Acharya, M. Uysal, and J. Saltz. Active Disks: Programming Model, Algorithms and Evaluation. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 81–91, October 1998.

[2] ARM Collaborates With Seagate For Hard Disc Drive Control, June 2002. ARM Press Release.

[3] H. Boral and D. DeWitt. Database Machines: An Idea Whose Time Has Passed? In *Proceedings of the International Workshop on Database Machines*, pages 166–187, September 1983.

[4] D. Burger and T. Austin. The SimpleScalar Toolset, Version 3.0. http://www.simplescalar.com.

[5] EMC Centera. http://www.emc.com/products/systems/centera.jsp.

[6] G. Ganger, B. Worthington, and Y. Patt. *The DiskSim Simulation Environment Version 2.0 Reference Manual*. http://www.ece.cmu.edu/ ganger/disksim/.

[7] S. Gurumurthi, A. Sivasubramaniam, and V. Natarajan. Disk Drive Roadmap from the Thermal Perspective: A Case for Dynamic Thermal Management. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 38–49, June 2005.

[8] L. Huston, R. Sukthankar, R. Wickremesinghe, M. Satyanarayanan, G. Ganger, E. Riedel, and A. Ailamaki. Diamond: A Storage Architecture for Early Discard in Interactive Search. In *Proceedings of the USENIX Conference on File and Storage Technonologies (FAST)*, April 2004.

[9] Intel PXA 255 Processor. http://www.intel.com/design/pca/prodbref/252780.htm.

[10] B. Jarvinen, C. Neumann, and M. Davis. A Tropical Cyclone Data Tape for the North Atlantic Basin, 1886-1983: Contents, Limitations, and Uses. Technical Report NWS NHC 22, National Oceanic and Atmospheric Administration (NOAA), 1984.

[11] K. Keeton, D. Patterson, and J. Hellerstein. The Case for Intelligent Disks (IDISKs). *SIGMOD Record*, 27(3):42–52, September 1998.

[12] E. Kim and et al. Energy Optimization Techniques in Cluster Interconnects. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 459–464, August 2003.

[13] G. Memik, M. Kandemir, and A. Choudhary. Design and Evaluation of Smart Disk Architecture for DSS Commercial Workloads. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, pages 335–342, August 2000.

[14] MIT Center for Biological and Computational Learning (CBCL) Face Recognition Database. http://cbcl.mit.edu/ software-datasets/heisele/facerecognition-database.html.

[15] S. Mitchell. *Inside the Windows 95 File System*. O'Reilly & Associates, Inc. (ISBN: 1-56592-200-X), 1997.

[16] E. Riedel, G. Gibson, and C. Faloutsos. Active Storage for Large-Scale Data Mining and Multimedia. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 62–73, August 1998.

[17] A. Silberschatz, P. Galvin, and G. Gagne. *Operating Systems Concepts - Sixth Edition*. John Wiley and Spns Inc. (ISBN: 0-471-25060-0), 2003.

[18] M. Sivathanu, V. Prabhakaran, F. Popovici, T. Denehy, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. Semantically-Smart Disk Systems. In *Proceedings of the Annual Conference on File and Storage Technology (FAST)*, March 2003.

[19] H. Smith. Delivering on the Promise of Business Analytics, Technology Roundtable IV Talk, November 2005.

[20] S. Smith and J. Brady. SUSAN - A New Approach to Low Level Image Processing. *International Journal of Computer Vision*, 23(1):45–78, May 1997.

[21] C. Smullen, S. Tarapore, S. Gurumurthi, P. Ranganathan, and M. Uysal. Active Storage Revisited: The Case for Power and Performance Benefits for Unstructured Data Processing Applications. In *Proceedings of ACM Computing Frontiers (CF)*, May 2008.

[22] SPEC - Standard Performance Evaluation Corporation. http://www.spec.org/.

[23] The Netezza Performance Server System. http://www.netezza.com/products/products.cfm.

[24] The Office of Science Data-Management Challenge - Report from the DOE Office of Science Data-Management Workshops, March-May 2004. http://www.sc.doe.gov/ascr/Final-report-v26.pdf.

[25] M. Uysal, A. Acharya, and J. Saltz. Evaluation of Active Disks for Decision Support Databases. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pages 337–348, January 2000.