This paper will be published in the 6th *Conference on File and Storage Technologies (FAST '08)*, San Jose, California, February 26–29, 2008.

Pergamum: Replacing Tape with Energy Efficient, Reliable, Disk-Based Archival Storage

Mark W. Storer Kevin M. Greenan Ethan L. Miller Kaladhar Voruganti
University of California, Santa Cruz *Network Appliance*

Abstract

As the world moves to digital storage for archival purposes, there is an increasing demand for reliable, low-power, cost-effective, easy-to-maintain storage that can still provide adequate performance for information retrieval and auditing purposes. Unfortunately, no current archival system adequately fulfills all of these requirements. Tape-based archival systems suffer from poor random access performance, which prevents the use of inter-media redundancy techniques and auditing, and requires the preservation of legacy hardware. Many disk-based systems are ill-suited for long-term storage because their high energy demands and management requirements make them cost-ineffective for archival purposes.

Our solution, Pergamum, is a distributed network of intelligent, disk-based, storage appliances that stores data reliably and energy-efficiently. While existing MAID systems keep disks idle to save energy, Pergamum adds NVRAM at each node to store data signatures, metadata, and other small items, allowing deferred writes, metadata requests and inter-disk data verification to be performed while the disk is powered off. Pergamum uses both intra-disk and inter-disk redundancy to guard against data loss, relying on hash tree-like structures of algebraic signatures to efficiently verify the correctness of stored data. If failures occur, Pergamum uses staggered rebuild to reduce peak energy usage while rebuilding large redundancy stripes. We show that our approach is comparable in both startup and ongoing costs to other archival technologies and provides very high reliability. An evaluation of our implementation of Pergamum shows that it provides adequate performance.

1 Introduction

Businesses and consumers are becoming increasingly conscious of the value of archival data. In the business arena, data preservation is often mandated by law, and data mining has proven to be a boon in shaping business strategy. For individuals, archival storage is being called upon to preserve sentimental and historical artifacts such as photos, movies and personal documents. In both of these areas, archival systems must keep pace with a growing need for efficient, reliable, long-term storage.

Many storage systems designed for long-term data preservation rely on sequential-access technologies, such as tapes, that decouple media from its access hardware. While effective for back-up workloads (write-once, read-rarely, newer writes supersede old), such systems are poorly suited to archival workloads (write-once, read-maybe, new writes unrelated to old writes). With as many as 50–100 tapes per drive, a requirement to keep tapes running at full speed, and a linear media-access model, random-access performance with tape-media is relatively poor. This conspires against many archival storage operations — such as auditing, searching, consistency checking and inter-media reliability operations — that rely on relatively fast random-access performance. This is especially important in light of the preservation and retrieval demands of recent legislation [23, 30]. Further, many data retention policies include the notion of a limited lifetime, after which data is securely deleted; selective deletion is difficult and inefficient in linear media. Finally, the separation of media and access hardware introduces the need to preserve complex chains of hardware; reading an old tape requires a compatible reader, controller and software.

Recently, hard drives have dropped in price relative to tape, making them a potential alternative for archival storage [33]. The availability of high-performance, low-power CPUs [4] and inexpensive, high-speed networks have made it possible to produce a self-contained, network-attached storage device [16] with reasonable performance and low power utilization: as little as 500 mW when both the CPU and disk are idle. The use of disks instead of tapes means that heads are packaged

with media, removing the need for robotics and reducing physical movement and system complexity. Using standardized communication interfaces, such as TCP/IP over Ethernet, also helps simplify technology migration and long-term maintenance. By using randomly-accessible disks instead of linear tapes, systems can take advantage of inter-media redundancy schemes. Unfortunately, many existing disk-based systems incur high costs associated with power, cooling and administration because of design approaches that favor performance over energy efficiency. However, recent work on MAIDs (Massive Arrays of Idle Disks) has demonstrated that considerable energy-based cost savings can be realized while maintaining high levels of performance [10, 32, 45], though such systems often favor performance over even greater energy savings.

Our design differs from that of existing MAID systems, which still have centralized controllers. Instead, our system, Pergamum, takes an approach similar to that used in high-performance scalable storage systems [36, 46, 48], and is built from thousands of intelligent storage appliances connected by high-speed networks that cooperatively provide reliable, efficient, long-term storage. Each appliance, called a Pergamum *tome*, is composed of four hardware components: a commodity hard drive for persistent, large-capacity storage; on-board flash memory for persistent, low-latency, metadata storage; a low-power CPU; and a network port. Each appliance runs its own copy of the Pergamum software, allowing it to manage its own consistency checking, disk scrubbing and redundancy group responsibilities. Additionally, the CPU and extensible software layer enables disk-level processing, such as compression and virus checking. Finally, the use of standardized networking interfaces and protocols greatly reduces the problem of maintaining complex chains of dependent hardware.

Pergamum introduces several new techniques to disk-based archival storage. First, our system distributes control to the individual devices, rather than centralizing it, by including a low-power CPU and network interface on each disk; this approach reduces power consumption by eliminating the need for power-hungry servers and RAID controllers. Systems such as TickerTAIP [8] used distributed control in a RAID, but did not include reliability checking and power management. Second, Pergamum aggressively ensures data reliability using two forms of redundancy: intra-disk and inter-disk. In the former, each disk stores a small number of redundancy blocks with each set of data blocks, providing a self-sufficient way of recovering from latent sector errors [6]. In the latter, Pergamum computes redundancy information across multiple disks to guard against whole disk failure. However, unlike existing RAID systems, Pergamum can stagger inter-disk activity during data recovery, minimizing peak energy consumption during rebuilding. Third, energy-efficient decentralized integrity verification is enabled by storing data signatures for disk contents in NVRAM. Thus, using just the signatures, Pergamum tomes can verify the integrity of their local contents and, by exchanging signatures with other Pergamum tomes, verify the integrity of distributed data without incurring any spin up costs. Finally, the Pergamum architecture allows disk-based archives to look like tape: an individual Pergamum tome may be pulled out of the system and read independently; the remaining Pergamum tomes will eventually treat this event like a disk failure and rebuild the "missing" data in a new location.

The goal of Pergamum, is to realize significant cost savings by keeping the vast majority, as many as 95%, of the disks spun down while still providing reasonable performance and excellent reliability. Our techniques allow us to greatly reduce energy usage, as compared to traditional hard drive based systems, making it suitable for archival storage. The use of signatures to verify data reduces the need to power disks on, as does the reduced scrubbing frequency made possible by the extra safety provided by intra-disk parity. Similarly, staggering disk rebuilds reduces peak power load, again allowing Pergamum to reduce the maximum number of disks that must be active at the same time. While we believe these techniques are best realized in a distributed system such as Pergamum—the use of many low-power CPUs is more efficient than a few high-power servers—they are also suitable for use in more conventional MAID architectures, and could be used to reduce power consumption in them as well.

The remainder of this paper is organized as follows. Section 2, places Pergamum within the context of existing research. Following that, Section 3 a detailed discussion of the systems components, including a discussion of the components in each Pergamum tome. Then, Section 4 details the system's design, including its redundancy and power management approaches. Section 5 contains our evaluation of Pergamum in terms of cost, long-term reliability and performance. Finally, following a discussion of future work in Section 6, we conclude the paper in Section 7.

2 Related Work

In designing Pergamum to meet the goals of energy-efficient, reliable, archival storage [7], we used concepts from various systems. These projects can be distinguished from Pergamum by identifying their intended workload, cost strategy and redundancy strategy. As Table 1 illustrates, many existing systems fulfill some of the goals of Pergamum, but none adequately address all of its concerns.

System	Media	Workload	Redundancy	Consistency	Power Aware
PARAID	disk	server clusters	RAID		Yes
Nomad FS	disk	server clusters	none		Yes
Google File System	disk	data-intensive apps	replicas	relaxed	No
EMC Centera	disk	archival	mirroring or parity	WORM media	No
Venti	disk	archival	RAID 5	content-based naming, type ids	No
Deep Store	disk	archival	selectable replication	content-based naming	No
Copan Revolution 220A	disk	archival	RAID 5	SHA 256	Yes
Sun StorageTek SL8500	tape	backup	N+1	WORM media	No
RAIL	optical	backup, archival	RAID 4	optional write verification	No
Pergamum	disk	archival	2-level erasure coding	algebraic signatures	Yes

Table 1: Overview of storage systems described in Section 2.

A number of systems have also sought to achieve cost savings through the use of commodity hardware [14, 45]. Typically, this strategy assumes that cheaper SATA drives will fail more often than server-class hardware, requiring that the solution utilize additional redundancy techniques. Recent studies, however, call this assumption into question, showing that SATA drives often exhibit the same replacement rate as SCSI and FC disks [37].

Energy efficiency is an area that many designs have explored in pursuit of cost savings. Some reports state that commonly used power supplies operate at only 65–75% efficiency, representing one of the primary culprits of excess heat production, and contributing to cooling demands that account for up to 60% of data-center energy usage [17]. The development of Massive Arrays of Idle Disks (MAIDs) generated large cost savings by leaving the majority of a system's disks spun down [10]. Further work has expanded on the idea by incorporating strategies such as data migration, the use of drives that can spin at different speeds, and power-aware redundancy techniques [31, 32, 45, 49, 51]. While these systems realize energy savings, they are not designed specifically for archival workloads, instead attempting to provide performance comparable to "full-power" disk arrays at reduced power. Thus, they do not consider approaches that could save even more power at the expense of high performance. For example, some MAID systems, such as those built by Copan Systems [19], use a relatively small number of server-class CPUs and controllers that can control dozens of disks. However, this approach is still relatively power-hungry because the CPU and controllers are always drawing power, reducing energy efficiency. A Copan MAID system in normal use consumes 11 W/TB [19]; as shown in Table 2, this is comparable to the 11–13 W required by a spun-up Pergamum tome with a 1 TB drive. However, it is much higher than the 2–3 W/TB that Pergamum can achieve with 95% of the disks powered off.

Another class of systems relies on media such as tape or optical media rather than hard drives [41, 43]; such systems are typically used for archival or back-up workloads. While the raw media cost may be somewhat lower than that of disk, the cost savings of such media are often offset by the need for additional hardware, *e. g.*, extra drive heads and robotic arms. Additionally, the random access performance of these systems is often quite poor, introducing a number of correlated side effects such as limitations on the system's choice of redundancy schemes. For example, RAIL stores data on optical disks and utilizes RAID 4 redundancy, but only at a very high level: for every five DVD libraries, a sixth library is solely devoted to storing parity [43]. Other systems have used striped tape to increase performance [13]; later systems used extra tapes in the stripe to add parity for reliability [24].

A final class of storage systems is designed for an archival workload, but lacks a specific cost-saving strategy [2, 20, 34, 50]. Like many systems designed for primary tier storage, these systems favor performance over power-efficiency and cost savings. While they may offer fast random access performance, their lack of cost efficiency makes them ill-suited for the long-term preservation of large corpora of data. Other wide-area long-term storage systems, such as SafeStore [26], OceanStore [35] and Glacier [21], can provide data longevity, but do not take energy consumption into account. For example, SafeStore uses multiple remote storage systems to ensure data safety, but does not address the issue of reducing power consumption on the remote servers.

Pergamum also expands upon techniques found in systems spanning various usage models and cost strategies. Many systems have used hierarchical hashing as a means of ensuring file integrity [1, 2, 25, 27, 28, 34, 35]; Pergamum extends this technique by utilizing hash trees of algebraic signatures [39]. Additionally, we extend the use of hierarchical hashing to the power-efficient auditing and consistency checking of inter-device replication. Intra-disk redundancy strategies were first suggested for use in full-power RAID systems to avoid loss of data due to disk failure and simultaneous latent sector errors on a surviving disk [11, 12]. Finally, a number of hybrid

drives that combine flash memory with a hard drive have come to market [40]. However, in such units, the flash is used primarily as a read and write cache, in contrast to Pergamum, which uses flash memory on each Pergamum tome for metadata consistency, and indexing information, allowing Pergamum to reduce disk spin-ups while preserving high levels of functionality.

3 System Components

The design of Pergamum was driven by a workload that exhibits read, write and delete behavior that differs from typical disk-based workloads, providing both challenges and opportunities. The workload is write-heavy, motivated by regulatory compliance and the desire to save any data that *might* be valuable at a later date. Reads, while relatively infrequent, are often part of a query or audit and thus are likely to be temporally related. Deletes are also likely to exhibit a temporal relationship as retention policies often specify a maximum data lifetime. This workload resembles traditional archival storage workloads [34,50], adding deletion for regulatory compliance.

The Pergamum system is structured as a distributed network of independent storage appliances, as shown in Figure 1. Alone, each Pergamum tome acts as an intelligent storage device, utilizing block-level erasure coding to survive media faults and algebraic signatures to verify block integrity. Collectively, the storage appliances provide data reliability through distributed RAID techniques that allow the system to recover from the loss of a device, and inter-disk data integrity by efficiently exchanging hash trees of algebraic signatures. As we will show, this approach is so reliable that disk scrubbing [38] need not be done more frequently than annually. In addition, lost data can be rebuilt with lower peak energy consumption by staggering disk activity; this approach is slower, but reduces peak power consumption.

The next two sections discuss the design and implementation of Pergamum and implementation of these techniques. This section describes an individual Pergamum appliance, or *tome*, including its components, intra-appliance redundancy strategy, interconnection network, and interface. Section 4 then describes how multiple storage appliances work together to provide reliable, distributed, archival storage, including a description of the system's inter-appliance redundancy and consistency checking strategy.

3.1 Pergamum Tomes

A Pergamum tome is a storage appliance made up of four main components: a low-power processor, a commodity hard drive, non-volatile flash memory and an ethernet

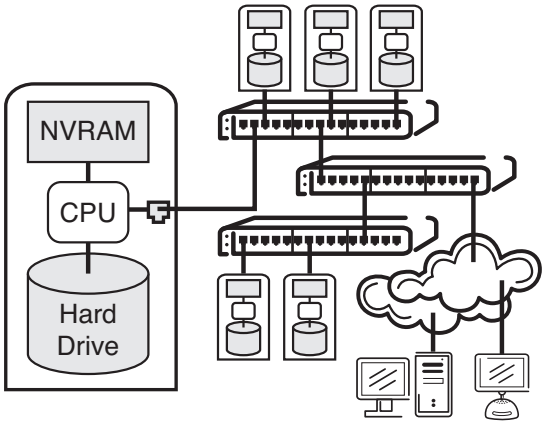

Figure 1: High-level system design of Pergamum. Individual Pergamum tomes, described in Section 3.1 are connected by a commodity network built from off-the-shelf switches.

Component	Power
SATA Hard Drive [47]	7.5 W
ARM-based board (w/ NIC) [4]	3.5 W
NVRAM	< 0.6 W

Table 2: Active power consumption (in watts) of the four primary components that make up a Pergamum tome.

controller. To protect against media errors, erasure coding techniques are used on both the hard drive and flash memory.

Each Pergamum tome is managed by an on-board, low-power CPU; a modern ARM-based single board computer consumes 2–3 W when active (using a 400 MHz CPU) and less than 300 mW when inactive [4]. The processor handles the usual roles required of a network-attached storage device [15,16] such as network communications, request handling, metadata management, and caching. In addition, each Pergamum tome's CPU manages consistency checking and parity operations for the local drive, responds to search requests, and initiates communications with other disks to provide inter-disk reliability. The processor can also be used to handle other operations at the device level, such as virus checking and compression.

Persistent storage is provided through the unit's SATA-class hard drive. The use of commodity hardware offers cost savings over more costly SCSI and FC drives while providing acceptable performance for archival workloads. By using both intra-disk redundancy and distributed redundancy groups, commodity SATA-class drives can provide excellent reliability for long-term archival storage [37].

While a single processor could manage multiple hard drives, Pergamum pairs each processor with a single hard

drive. This is done for performance matching, power savings, and ease of maintenance. As Section 5 details, low-power processors are not fast enough to run even a single disk at full speed, so there is little incentive to control multiple disks with a single CPU. Power savings is another issue: a faster CPU and multi-disk controller would consume more power than multiple individual low-power CPUs (cutting processor voltage in half results in half the clock speed but one fourth the power consumption). Finally, the pairing of a CPU with a single disk and network connection makes it simpler to replace a failed Pergamum tome. If any part of the Pergamum tome fails, the entire Pergamum tome is discarded and replaced, rather than trying to diagnose which part of the Pergamum tome failed to "save" working hard drives. The system then heals itself by rebuilding the data from the failed device elsewhere in the system. By reducing the complexity of routine maintenance, Pergamum reduces ongoing costs.

In addition to a hard drive, each Pergamum tome includes a pool of on-board NVRAM for storing metadata such as the device's index, data signatures and information about pending writes. The purpose of the NVRAM is to provide low-power, persistent storage; operations such as metadata searches and signature requests do not require the unit's drive to be spun up. While the use of flash-type NVRAM provides better persistency and energy-efficiency compared to DRAM, it does raise two issues: reliability and durability. Our system protects the flash memory from erroneous writes and media errors through the use of page-level protection and consistency checking [18], ensuring memory reliability. Flash memory is also limited in that the memory must be written in blocks, and each block may only be rewritten a finite number of times, typically 10^4–10^5 times. However, since the NVRAM primarily holds metadata such as algebraic signatures and index information, flash writes are relatively rare; flash writes coincide with disk writes. Because this typically occurs fewer than 1000 times per year, or 8000 times during the lifetime of a disk, even if the flash memory is totally overwritten each time, such activity will still be below the 10,000 write cycles that flash memory can support. Additionally, while the current implementation uses NAND flash memory, other technologies such as MRAM [44] and phase change RAM [9] could be used as they become available and price-competitive, further reducing or eliminating the rewrite issue.

Finally, each Pergamum tome includes an Ethernet controller and network port, providing a number of important advantages. First, a network connection is a standardized interface that changes very slowly—modern Ethernet-based systems can interoperate with systems that are more than fifteen years old. In contrast, tape-based systems require a unique head unit for each tape format, and each of those devices may require a different interface; supporting legacy tapes could require the preservation of lengthy hardware chains. The use of a network also eliminates the need for robotics hardware (or humans) to load and unload media; such robots might need to be modified for different generations of tape media and must be maintained. Instead, the system can use commodity network interconnects, leaving all media permanently connected and always available for messaging.

3.2 Interconnection Network

Since Pergamum must contain thousands of disks to contain the petabytes of data that long-term archives must hold, its network must scale to such sizes. However, throughput is not a major issue for such a network—a modern tape silo with 6,000 tapes typically has fewer than one hundred tape drives, each of which can read or write at about 50 MB/s, for an aggregate throughput of 5 GB/s. Scaling a gigabit Ethernet network to support comparable bandwidth can be done using a star-type network with commodity switches at the "leaves" of the network and, potentially, higher-performance switches in the core. For example, a system built from 48-port gigabit Ethernet switches could use two switches as hubs for 48 switches, each of which supports 46 disks, with the remaining two connections going to each of the two hubs. This approach would support over 2200 disks at minimal cost; if the central hubs each had a few 10 Gb/s uplinks, a single client could easily achieve bandwidth above 5 GB/s. This structure could then be replicated and interconnected using a more expensive 10 Gb/s switch, allowing reasonable-speed access to any one of tens of thousands of drives, with the vast majority remaining asleep to conserve power.

The interconnection network must allow any disk to connect with any network-connected client. By using a standard Ethernet-based network running IP, Pergamum ensures that *any* disk can communicate with any other disk, allowing the system to both detect newly-connected disks and allowing them to communicate with existing disks to "back up" their own data.

The approach described above is highly scalable, with minimal "startup cost" and low incremental cost for adding additional disks. Further efficiencies could be achieved by pairing the Ethernet cable with a higher-gauge wire capable of distributing the 14–18 W that a spun-up disk and processor combination requires. Alternatively, the system could use disks that can spin at variable speeds as low as 5400 RPM [47], reducing disk power requirements to 7.5 W and overall system power needs to below 11 W, sufficiently low to use standard power-over-Ethernet. Central distribution of power has

several advantages, including lower hardware cost and lower cabling cost. Additionally, distributing power via Ethernet greatly simplifies maintenance—adding a new drive simply requires plugging it into an Ethernet cable. While the disks in the system will work to keep average power load below 5% utilization, a central power distribution system will allow the network switches themselves to guarantee that a particular power load will never be exceeded by restricting power distributed by the switch.

3.3 Pergamum Tome Interface

There are two distinct data views in Pergamum: a file-centric view and a block-centric view. Clients utilize the file-centric view, submitting requests to a Pergamum tome through traditional read and write operations. In contrast, requests from one Pergamum tome to another utilize the block-centric view of data based on redundancy group identifiers and offsets.

Clients access data on a Pergamum tome using a set of simple commands and a connection-oriented request and response protocol. Currently, clients address their commands to a specific device, although future versions of Pergamum will include a self-routing communications mechanism. Internally, files are named by a file identifier that is unique within the scope of a single Pergamum tome. The `new` command allocates an unused file identifier and maps it to a filename supplied by the user. This mapping is used by the `open` command to provide the file's unique identifier to a client. This file id, the device's `read` and `write` commands, and a byte offset are then used by the client to access their data.

Requests between Pergamum tomes primarily utilize a data view based on segment identifiers and block offsets, as opposed to files. There are four main operations that take place between Pergamum tomes. First, external parity update requests provide the a Pergamum tome storing parity with the delta and metadata needed to update its external redundancy data. Second, signature requests are used to confirm data integrity. Third, token passing operations assist in determining which devices to spin up. Finally, there are commands for the deferred (*foster*) write operations discussed in Section 4.3.1.

Management of Pergamum tomes can be done either with a centralized "console" to which each Pergamum tome reports its status, or in a distributed fashion where individual Pergamum tomes report their health via LED. For example, each Pergamum tome could have a small green LED that is on when the appliance is working correctly, and off when it is not. An operator would then replace Pergamum tomes whose light is off; this approach is simple and requires little operator skill. Alternatively, a central console could report "Pergamum tome 53 has failed," triggering a human to replace the failed unit. The Pergamum design permits both approaches; however, we do not discuss the tradeoffs between them in this paper.

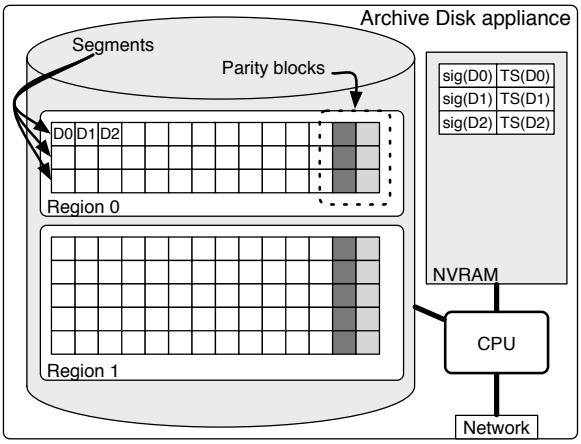

Figure 2: Layout of data on a single Pergamum tome. Data on the disk is divided into blocks and grouped into segments and regions. Data validity is maintained using signatures, and parity blocks are available to rebuild lost or corrupted data.

4 Pergamum Algorithms and Operation

A Pergamum system, deployed as described in Section 3 is highly decentralized, relying upon individual disks to each manage their own behavior and their own data. Each disk is responsible for ensuring the reliability of the data it stores, using both local redundancy information and storage on other nodes.

4.1 Intra-Disk Storage and Redundancy

The basic unit of storage in a Pergamum tome are fixed-size blocks grouped into fixed-size *segments*, as shown in Figure 2. Together, blocks and segments form the basic units of the system's two levels of redundancy encoding: intra-disk and inter-disk. Since the system is designed for archival storage, blocks are relatively large—128 KB–1 MB or larger—reducing the metadata overhead necessary to store and index them. This approach mirrors that of tape-based systems, which typically require data to be stored in large blocks to ensure high efficiency and reasonable performance.

The validity of individual blocks is checked using hashes; if a block's content does not match its hash, it can be identified as incorrect; this approach has been used in other file systems [27, 42]. Disks themselves maintain error-correcting codes, but such codes are insufficiently accurate for long-term archival storage because they have a silent failure rate of about 10^{-14}, a rate sufficiently high

to cause data corruption in large-scale long-term storage. To avoid this problem, each disk appliance stores both a hash value and a timestamp for each block on disk. Assuming a 64-bit hash value and a 32-bit timestamp, a 1 TB disk will require 96 MB of flash memory to maintain this data for 128 KB blocks. Keeping this information in flash memory has several advantages. First, it ensures that block validity information has a different failure mode from the data itself, reducing the likelihood that both data and signature will be corrupted. More importantly, however, it allows the Pergamum tome to access the signatures and timestamps without powering on the disk, enabling Pergamum to conduct inter-disk consistency checks without powering on individual disks.

The hash values used in Pergamum are *algebraic signatures*—hash values that are highly sensitive to small changes in data, but, unlike SHA-1 and RIPEMD, are not cryptographically secure. Algebraic signatures are ideally suited to use in Pergamum because, for many redundancy codes, they exhibit the same relationships that the underlying data does. For example, for simple parity:

$$d_0 \oplus d_1 \cdots \oplus d_{n-1} = p \Longrightarrow$$
$$\text{sig}(d_0) \oplus \text{sig}(d_1) \cdots \oplus \text{sig}(d_{n-1}) = \text{sig}(p) \quad (1)$$

While 64-bit algebraic signatures are sufficiently long to reduce the likelihood of "silent" errors to zero; they are ineffective against malicious intruders, though there are approaches to verifying erasure-coded data using signatures or fingerprints that can be used to defeat such attacks [22, 39].

As Figure 2 illustrates, each segment is protected by one or more parity blocks, providing two important protections to improve data survivability. First, the extra parity data provides protection against latent sector errors [6]. If periodic scrubbing reveals unreadable blocks within a segment, the unreadable data can be rebuilt and written to a new block using only the parity on the local disk. Second, while simple scrubbing merely determines whether the block is readable, the use of algebraic signatures and parity blocks allows a disk to determine whether a particular block has been read back properly, catching errors that the disk drive itself cannot [22, 39] and correcting the error without the need to spin up additional disks.

4.2 Inter-Disk Redundancy

While intra-disk parity guards against latent sector errors, Pergamum can survive the loss of an entire Pergamum tome through the use of inter-tome redundancy encoding. Segments on a single disk are grouped into *regions*, and a *redundancy group* is built from regions of identical sizes on multiple disks. To ensure data survival,

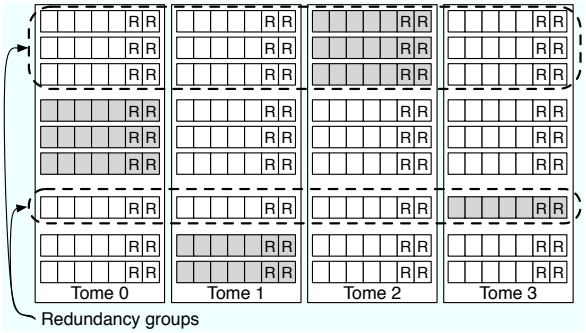

Figure 3: Two levels of redundancy in Pergamum. Individual segments are protected with redundant blocks on the same disk—those labeled with an **R**. Redundancy groups are protected by the shaded segments, which contain erasure correcting codes for the other segments in the redundancy group. Note that segments used for redundancy still contain intra-disk redundant blocks to protect them from latent sector errors.

each redundancy group also includes extra regions on additional disks that contain erasure correction information to allow data to be rebuilt if any disks fail. These *redundancy regions* are stored in the same way as data regions: they have parity blocks to guard against individual block failure and the disk appliances that host them store their algebraic signatures in NVRAM.

The naïve approach to verifying the consistency of a redundancy group would require spinning up all the disks in the group, either simultaneously or in sequence, and verifying that the data in the segments that make up the regions in the group is consistent. Pergamum dramatically reduces this overhead in two ways. First, the algebraic signatures stored in NVRAM can be exchanged between disks in a redundancy group and verified for consistency as described in Section 4.1. Since the signatures are retrieved from NVRAM, the disk need not be spun up during this process as long as changes to on-disk data are reflected in NVRAM. If inconsistencies are found, the timestamps may be used to decide on the appropriate fix. For example, if a set of segments is inconsistent and a data segment is "newer" than the newest parity segment, the problem is likely that the write was not applied properly; depending on how writes have been applied and whether the "old" data is available, the parity may be fixed without powering up the whole set of segments.

While this approach only requires that signatures, rather than data, be transmitted, it is still very inefficient, requiring the transmission of nearly 100 MB of signatures for each disk to verify a redundancy group's consistency. To further reduce the amount of data and computation that must be done, Pergamum uses hash trees [29] built from algebraic signatures, as shown in Figure 4. Using signatures of blocks as d_i in Equation 1 shows that

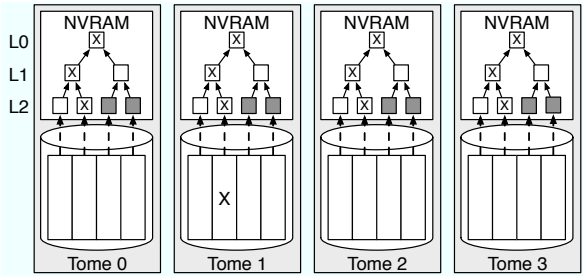

Figure 4: Trees of algebraic signatures. Tomes in a redundancy group exchange the roots of their trees to verify consistency; in this diagram, the signatures marked with an X are inconsistent. The roots (L0) are exchanged; since they do not match, the nodes recurse down the tree to L1 and then L2 to find the source of the inconsistency. "Children" of consistent signatures (signatures shaded in gray at L2) are not fetched, saving transmission and processing time. The inconsistent block on tome 1 is found by checking the intra-segment signatures on each block; only those on tome 1 were inconsistent. Note that only tome 1's disk need be spun up to identify and correct the error if it is localized.

signatures of sets of signatures follow the same relationships as the underlying data; this property is maintained all the way up to the root of the tree. Thus, the signatures at the roots of each disk's hash tree for the region should yield a valid erasure code word when combined together. If they do not, some block in the redundancy group is invalid, and the disks recurse down the hash tree to find the bad block, exchanging the contents at each level to narrow the location of the "bad" block. This approach requires $O(k)$ computation and communication when the group is correct—the normal case—and $O(k \log n)$ computation and communication to find an error in a redundancy group with a total of n blocks across k disks. Since redundancy groups are not large ($k \leq 50$, typically), high-level redundancy group verifications can be done quickly and efficiently.

4.3 Disk Power Management

Reducing power consumption is a key goal of Pergamum; since spinning disks are by far the largest consumer of power in a disk appliance, keeping the disk powered off ("spun down") dramatically reduces power consumption. In contrast to earlier systems that aim to keep 75% of the disks inactive [19], Pergamum tries to keep 95% or more of the disks inactive all of the time, reducing disk power consumption by a factor of five or more over existing MAID approaches. This goal is achieved with several strategies: sequentially activating disks to update redundancy information on writes, low-frequency scrubbing, and sequentially rebuilding regions on failed disks.

To guard against too many disks being spun up at once, Pergamum uses *spin-up tokens*, which are passed from one node to another to allow spin-up. If multiple nodes require a token simultaneously, the node currently holding the token (which may or may not be spun up at the time) calculate need based on factors such as a unit's oldest pending request, the types of requests it has pending, the number of pending requests and the last time the disk was spun up.

4.3.1 Reading and Writing Data

When a client requests a data read, the device from which data is to be read is spun up. This process takes a few seconds, after which data can be read at full speed. While a Pergamum tome is somewhat slower than a high-power network-attached disk, its performance, discussed in Section 5, is sufficient for archival storage retrieval. Moreover, since the data is stored on a disk rather than a tape, random access performance is significantly better than that of a tape-based system.

As with reads, archive writes require a spun-up disk. Pergamum clients choose the disks to which they write data; Pergamum does not impose a choice on users. This is done because some clients may want to group particular data on specific disks: for example, a company might choose to archive email for an individual user on one drive. On the other hand, a storage client may query Pergamum nodes to identify spun-up nodes, allowing it to select a disk that is already spun up.

Since writes require the eventual update of distributed data, they are more involved than reads. First, the target disk is spun up if it is not already active. Next, data is written to blocks on the local disk. However, existing data blocks are not overwritten in place; instead, data is written to a new data block, allowing the Pergamum tome to calculate "deltas" based on the old and new block. These deltas are then sent to the Pergamum tomes storing the redundancy regions for the old block's segment. On the local device, the segment mapping is updated to replace the old block with the new block. It is important to note, however, that the old block is retained until it has been confirmed that all external parity has been updated.

On the Pergamum tomes storing the redundancy information, the deltas arrive as a parity update request. Since the redundancy update destination knows how the erasure correcting code is calculated, it can use the delta from the data target disk to update its own redundancy information; it does not need both the old and new data block, only the delta. Because the delta may be different for different parity disks, however, the Pergamum tome that received the original write request must keep both old and new data until all of the parity segments have

been updated. However, doing updates this way ensures that a write requires no more than two disks to be active at any time; while the total energy to write the data is unchanged—a write to an (m,n) redundancy group must still update $n-m+1$ disks—the peak energy is dramatically reduced from $n-m+1$ disks active to 2 disks active, resulting in an improvement for any code that can correct more than one erasure.

One problem with allowing writes directed to a specific Pergamum tome is that the disk may not be spun up when the write is issued. While the destination disk may be activated, an alternate approach is to write the data to *any* currently active disk and later copy the data to the "correct" destination. This approach is called *surrogate writing*, and is used in Pergamum to avoid spinning disks up too frequently. Instead, writes are directed to an already-active disk, and the Pergamum tome to which data will eventually be sent is also notified. The data can the be transferred to the correct destination lazily.

4.3.2 Scrubbing and Recovering Data

To ensure reliability, disks in Pergamum are occasionally scrubbed: every block on the disk is read and checked for agreement with the signature stored in NVRAM. This procedure is relatively time-consuming; even at 10 MB/s, a 1 TB disk requires more than a day to check. However, Pergamum tome's use of on-disk redundancy to guard the data in a segment, described in Section 4.1, greatly reduces the danger of data loss from latent sector errors, so the system can reduce the frequency with which it performs full-disk scrubs. Instead, a Pergamum tome performs a "limited scrub" each time it is spun up, either during idle periods or immediately before the disk is spun down. This limited scrub checks a few hundred randomly-chosen locations on the disk for correctness and examines the drive's SMART status [5], ensuring that the disk is basically operating correctly. If the drive passes this check, the major concern is total drive failure, either during operation or during spin-up, as Section 5.2 describes.

Complete drive failures are handled by rebuilding the data on the lost drive in a new location. However, since fewer than 5% of the disks in Pergamum may be on at any given time and redundancy groups that may contain data and parity on 15–40 disks for maximal storage efficiency, it is impractical to spin up all of the disks in a redundancy group to rebuild it. Instead, Pergamum uses techniques similar to those used in writing data to recover data lost when a disk fails. The rebuilding algorithm begins by choosing a new location for the data that has been lost; this may be on an existing disk (as long as it is not already part of the redundancy region), or it may be on a newly-added disk. Pergamum then spins up the disks in the redundancy region one by one, with each disk sending its data to the node on which data is being rebuilt. The node doing the rebuilding folds the incoming data into the data already written using the redundancy algorithm; thus, it must write each location in the region m times and read it $m-1$ times (the first "read" would result in all zeros, and is skipped).

5 Experimental Evaluation

Our experiments with the current implementation of Pergamum were designed to measure several things. First, we wanted to evaluate the cost of our system in order to ensure that our solution was economically feasible. Second, we wanted to confirm that Pergamum can provide long-term reliability through a strategy of multiple levels of parity and consistency checking using algebraic signatures. Finally, we wanted to measure the performance of our implementation to show that Pergamum is suitable for archival workloads and to identify potential bottlenecks.

The remainder of this section proceeds as follows. First, we first present an analytical evaluation of the system's cost. Then, using a series of simulations, we examine the system's long-term reliability. Finally, we present the results of our performance tests with the current implementation of Pergamum.

5.1 Cost

An archival system's cost can be broken down into two primary areas: static (initial costs) and operational. The first figure describes the cost to acquire the system, and the second figure quantifies the cost to run the system. Examining both costs together is important because low static costs can be overshadowed by the total cost of operating and maintaining a system over its lifetime.

We do not consider personnel costs in any of the systems we describe; we assume that all of the systems are sufficiently well automated that human maintenance costs are relatively low. However, this assumption is somewhat optimistic, especially for large tape-based systems that use complex hardware that may require repair. In contrast, Pergamum is built from simple, disposable components—a failed Pergamum tome or network switch may simply be thrown out rather than repaired, reducing the time and personnel effort required to maintain the system.

Static costs reflect the expenses associated with acquiring an archival storage solution, and can be calculated by totaling a number of individual costs. One is the system expense, which totals the base hardware and software costs of a storage system with a given capacity for storage media. This cost is paid at least once per

storage system, regardless of how much storage is actually required. Media cost, in dollars per terabyte, is a second expense. Large archival storage systems may require several "base" systems; for example, an archival system that uses tape silos and robots might require one silo per 6,000 tape cartridges, even if the silo will not be filled initially.

Operational costs reflect those costs incurred by day to day operation of an archival storage system. This cost can be measured using a dollars per operational period figure, normalized to the amount of storage being managed. Some of the primary contributors to a system's total operational expenses include power, cooling, floor space and management. As described above, we omit management cost, both because we assume it will be similar for different storage technologies, and because it is extremely difficult to quantify. We also omit the cost of floor space since it is highly variable depending on the location of the data center. However, an important, but often omitted, aspect of operational costs includes the expenses related to reliability: expected replacement costs for failed media and the operational cost associated with parity operations. This cost, along with power and cooling, forms the basis of our comparison of operational costs.

The static and operational costs must include the cost for any redundant hardware or storage. However, since existing solutions vary in their reliability, even within a particular technology, we have not attempted to quantify the interplay between capacity and reliability. Instead, we assume that a system that requires mirroring simply costs twice as much to purchase and run per byte as a non-redundant system. In this respect, Pergamum is very low cost: the storage overhead for a system with segments using 62 data and 2 parity blocks and redundancy groups with 13 data disks and 3 parity disks is $\frac{64}{62} \times \frac{16}{13} - 1 = 0.27$ times usable data capacity. In such a system, 1 TB of raw storage can hold 787 GB of user data.

All of these factors—static cost, operational cost, and redundancy overhead—are summarized in Table 3. Static costs are approximations based on publically available hardware prices. For operational costs, We have used a constant rate of $0.20/kWh for electricity to cover both the direct cost of power and the cost of cooling. Table 3 shows the costs for a 10 PB archive for each technology, including sufficient base systems to reach this capacity. While the costs reflected in the table are approximate, they are useful for comparative purposes. Also, we note that some systems have ranges for redundancy overhead because they can be configured in several ways to ensure sufficient reliability; we chose the least expensive reliability option for each technology. For example, the EMC Centera [20] can be used with mirroring; doing so might increase reliability, but will certainly increase total cost.

The results summarized in Table 3 illustrate a number of cost-related archival storage issues. First, as shown by PARAID, even energy-efficient, non-archival systems are too expensive for archival scenarios. Second, media with low storage densities can become expensive very quickly because they require a large amount of hardware to manage the high numbers of media. For example, RAIL uses UDO2 optical media that only offers 60 GB per disk and thus the system requires numerous cabinets and drives to handle the volume of media. Using off-the-shelf dual-layer DVDs, with capacity under 10 GB per disk, would reduce the media cost, but would increase the hardware cost by a factor of six because of the added media; such an approach would require 100 DVDs per terabyte, making the cost prohibitive. Third, the Copan and Centera demonstrate two different strategies for cost effective storage: lower initial costs versus lower runtime costs. Finally, we see that Pergamum is competitive in cost to Sun's StorageTek SL8500 system while providing functionality, such as inter-archive redundancy, that tape-based systems are unable to provide.

An understanding of the costs associated with reliability is important because it assists in matching the data to be protected with an economically efficient reliability strategy. Unfortunately, because it is largely dependent on the data itself, the economic impact of lost data is difficult to calculate. Moreover, many of the costs resulting from data loss are, at best, difficult to quantify. For example, the cost to replace data can vary from zero (don't replace it) to nearly priceless (how much is bank account data worth?). Another factor, opportunity costs, expresses the cost of lost time; every hour spent dealing with data loss is an hour that is not spent doing something else. In a professional setting, data loss may also involve mandatory disclosures that could introduce costs associated with bad publicity and fines. While we do not quantify these costs, we note that long-term archive reliability is a serious issue [7].

5.2 Reliability

There are many tradeoffs that influence the reliability of an archival storage system. Factors such as stripe size, both on an individual disk and between disks, disk failure rate, disk rebuild time and the expected rate of latent sector errors must be considered when building a long-term archival system. Our analysis considers these factors along with strict power-management constraints to compute the expected mean time to data loss (MTTDL) of a deployed Pergamum system.

Table 4 shows the parameters used in our analysis. In the absence of an archival workload for our reliability

System	Media	Static cost	Oper. cost	Redundancy
Sun StorageTek SL8500	T10000 tape	$4,250	$60	None
EMC Centera	SATA HD	$6,600	$1,800	parity
PARAID	SCSI HD	$37,800	$1,200	RAID
Copan Revolution	SATA HD	$19,000	$250	RAID-5
RAIL	UDO2	$57,000	$225	RAID-4 (5+1)
Pergamum	SATA HD	$4,700	$50	2-level

Table 3: Comparison of system and operational costs for 10 PB of storage. All costs are in thousands of dollars and reflect common configurations. Operational costs were calculated assuming energy costs of $0.20/kWh (including cooling costs).

Parameter	Value
Disk Fail Rate (λ_D)	$1/100000$ hours
Disk Repair Rate (μ_D)	$1/100$ hours
Latent Sector Fault Rate (λ_S)	1/13245 hours
Scrub Rate (μ_S)	$1/8640$ hours

Table 4: Simulator and model parameters.

analysis, we assume that each active device transfers a constant 2 MB/s, on average. Given a byte error rate of 1×10^{-14}, the on-disk sector error rate is approximately $1/13245$ hours. Due to the incremental nature of our rebuild algorithm, we approximate the time to rebuild a single device in our system to be roughly 100 hours, or 3 MB/s. Finally each disk in the system fails at a rate of $1/100000$ hours and is subject to a full scrub every year or 8640 hours. We consider these estimates to be liberal and provide a near-worst-case MTTDL of our system.

In order to determine the reliability of our system, we developed a discrete event simulator in Python using the SimPy module. There are four core events in our simulator: `DiskFail`, `DiskRebuild`, `SectorFail` and `Scrub`. Values for disk failure time, sector failure time and disk scrub are all drawn from an exponential distribution, while disk rebuild takes place in simulation time at 3 MB/s. We model the effects of disk spin-up by subtracting 10 hours from the life of a disk every time it is spun up [38]; this may well be pessimistic, resulting in an MTTDL that is shorter than in a real system. Each iteration of the simulator runs until a data loss event is reached and the current time is recorded. Although we found that around 100 iterations is sufficient, we calculate the MTTDL of a single configuration by running 1000 iterations with that configuration.

We also use Markov models to compute the reliability of single, double and triple disk fault tolerant codes. These models only capture disk failure and rebuild, and thus serve two purposes. First, the models give us a straightforward way to verify the behavior of the simulator. Most importantly, the MTTDL computed from each model serves as an approximation to a system that has

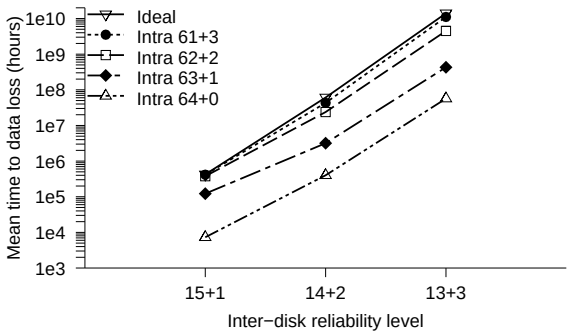

Figure 5: Mean time to data loss in hours for a single 16 disk group. 61+3 intra-disk parity is nearly equivalent to the "ideal" system, in which latent sector errors never occur. Note that MTTDL of 10^{10} hours for 16 disks corresponds to a 1000 year MTTDL for a 10 PB Pergamum system.

the ability to handle any number of latent sector errors.

Recent work has shown that latent sector errors make a non-trivial contribution to system reliability [6]. We thus modeled data loss in our system for configurations with 1, 2, and 3 parity segments per redundancy group under several different assumptions: levels of intra-disk parity protection ranging from 0–3 parity blocks per segment, and an "ideal" analytical model which assumed *no* latent sector errors occurred and considered only whole-disk failures. The results of our modeling using a scrub rate of once per year for each disk, shown in Figure 5, indicate that latent sector errors do indeed cause data loss if nothing is done to guard against them. The distance between the top curve ("ideal" MTTDL without latent sector errors) and bottom curve (no intra-disk parity) is approximately two orders of magnitude, showing that Pergamum must guard against data loss from latent sector errors. However, by using intra-disk erasure coding, the effect of latent sectors on MTTDL is nearly eliminated. In essence, we are trading disk space for a longer scrub interval, saving power in the process. Figure 5 shows that a configuration of 3 intra-disk parity blocks per 64 block segment provides nearly two or-

ders of magnitude longer MTTDL than no protection at all, approaching the "ideal" situation where latent sector errors never exist. With the exception of the configurations with 3 inter-disk parity elements and the "ideal" case, all of the MTTDL values in the graph are based on 1000 iterations of the simulator; we were only able to capture tens of MTTDL numbers for the configurations involving 3 inter-disk parity elements and 1 or 2 intra-parity elements, and the simulation for 3 inter-disk parity and 3 intra-disk parity elements took a great deal of time to run and only resulted in a few data points. This lack of data is due to the extremely high reliability of these configurations—the simulator modeled many failures, but so few caused data loss that the simulation ran very slowly. This behavior is precisely what we want from an archival storage system: it can gracefully handle many failure events without losing data. Even though we captured fewer data points for the triple inter-parity configuration, we believe the reported MTTDL is a reasonable approximation. As we see in the graph, the use of 3 intra-disk parity elements is close to the "ideal" situation across all inter-parity configurations.

Our simulation and modeling show that a configuration of 3 inter-disk parity segments per 16-disk reliability group and 3 intra-disk parity blocks per segment will result in an MTTDL of approximately 10^{10} hours. If each disk has a raw capacity of 1 TB, a Pergamum system capable of storing 10 PB of user data will require about 800 such groups, resulting an MTTDL of 1.25×10^7 hours, or about 1,400 years. Should this MTTDL for an entire archive be too low, we would recommend using more inter-disk parity—3 parity blocks per 64 block segment can correct most of the latent sector errors.

5.3 Performance

The current Pergamum prototype system consists of approximately 1,400 lines of Python 2.5 code, with an additional 300 lines of C code that were used to implement performance-sensitive operations such as data encoding and low-level disk operations. Our implementation includes the core system functionality, including internal redundancy, external redundancy, and a client interface that allows for basic I/O interactions. In its current state however, the implementation relies upon statically assigned redundancy groups and it does not include scrubbing or consistency checking.

For testing, all systems were located on the same gigabit Ethernet switch with little outside contention for computing or network resources. Communication between the Pergamum tome and the client used standard TCP/IP sockets in Python. For maximum compatibility, we utilized an MTU size of 1500 B.

Each Pergamum tome was equipped with an ARM 9 CPU running at 400 MHz, 128 MB of DDR2 SDRAM and Linux version 2.6.12.6. The client was equipped with an Intel Core Duo processor running at 2 GHz, 2 GB of DDR2 SDRAM and OS X version 10.4.10. The primary storage on each Pergamum tome was provided by a 7200 RPM SATA drive formatted with XFS. For read and write performance experiments, we utilized block sizes of 1 MB and 64 blocks per segment. Persistent metadata storage utilized a 1 GB USB flash drive and Berkeley DB version 4.4. The workload consisted of randomly generated files, all several megabytes in size.

Test	Client	Server
Raw Data Transfer	20.02	20.96
Raw Data Write	9.33	9.98
Unsafe Pergamum Write	4.74	4.74
XOR Parity Pergamum Write	4.72	3.25
Reed Solomon Pergamum Write	4.25	1.67
Fully Protected Pergamum Write	3.66	0.75
Pergamum Read	5.77	5.78

Table 5: Read and write performance for a single Pergamum tome to client connection. XOR parity writes used 63 data blocks to one parity block segments. Reed Solomon writes used 62 data blocks to two parity block segments. Fully protected writes utilize two level of Reed Solomon encoding and the server throughput reflects time to fully encode and commit internal and external parity updates.

5.3.1 Read and Write Throughput

Our first experiment with the Pergamum implementation was an evaluation of the device's raw data transfer performance. As Table 5 shows, the maximum throughput of a single TCP/IP stream to a Pergamum tome is 20 MB/s at the device. Further tests showed that, the device could copy data from a network buffer to an on-disk file at about 10 MB/s. Together, these values serve as an upper limit for the write performance that could be expected from a single client connection over TCP/IP.

Write throughput using the Pergamum software layer was tested at varying levels of write safety. The first write test was conducted with no internal or external parity updates. As shown in Table 5, writes without data protection ran at 4.74 MB/s. While no redundancy encoding was performed in the unsafe write, the system did incur the overhead of updating segment metadata and dividing the incoming data into fixed-size blocks.

Testing with internal parity updates enabled was performed using both simple XOR-based parity and more advanced Reed Solomon encoding. In these tests, the client-side and server-side throughput differ, as Pergamum utilizes parity logging during writes. Thus, while

the client views throughput as the time taken to simply ingest the data, the Pergamum tome's throughput includes the time to ingest the data and update the redundancy information. The first test utilized simple XOR-based parity in a 63+1 (63 data blocks and 1 parity block) configuration. This arrangement achieved a client-side write throughput of 4.72 MB/s and a Pergamum tome-side throughput of 3.25 MB/s. As Table 5 shows, using Reed Solomon in a 62+2 configuration results in similar client side throughput, 4.25 MB/s. However, the extra processing and parity block updates results in a server throughput of 1.67 MB/s.

The final write test, fully protected Pergamum tome writes, utilizes both inter- and intra-disk redundancy. Internal parity utilized Reed Solomon encoding in a 62+2 configuration. External redundancy utilized Reed Solomon with 3 data regions to 2 parity regions. In this configuration, client throughput is reduced to 3.66 MB/s as the CPU is taxed with both internal and external parity calculations. This is evident in the server throughput which is reduced to 0.75 MB/s. However, this does reflect the time required to update both internal and external parity and thus reflects the rate at which a single Pergamum tome can protect data with full internal and external parity.

Profile data obtained from the test runs indicates the system is CPU-bound. The performance penalty for the Pergamum tome writes appears to be based largely on two factors. First, as shown in the difference between a raw write and an unsafe Pergamum tome write in Table 5, Python's buffer management imposes a performance penalty, an issue that could be remedied with an optimized, native implementation. Second, as seen in the difference between the XOR Pergamum tome write and the Reed Solomon write, data encoding imposes a significant penalty for lower power processors. This is further evident by the results of our read throughput tests. Since a read operation to the Pergamum tome involves less buffer management and parity operations, throughput is correspondingly faster. We were able to achieve sustained read rates of 5.78 MB/s.

While the performance numbers in Table 5 would be inadequate for most high-performance workloads, even our current, prototype implementation of Pergamum is capable of supporting archival workloads. For example, 1000 Pergamum tomes and a spin-up rate of only 5% can provide a system-level ingestion throughput in excess of 175 MB/s, ingesting a terabyte in 90 minutes and fully protecting it in 8 hours. At this rate such an archive built from 1 TB disks could be filled in a year.

Encode Operations	ARM9	Core Duo
XOR parity	20.02	201.41
Reed Solomon; 5 data, 2 parity	3.13	33.68
Data signature (64-bit)	57.44	533.33

Table 6: Throughput, in MB/sec, to encode 50 MB of data using the Pergamum tome's 400 MHz ARM9 board drawing 2-3 W and a desktop class 2 GHz Intel Core Duo drawing 31 W.

5.3.2 Data Encoding

One of the primary functions of each Pergamum tome's processor is data encoding for redundancy and signature generation. Thus, we wanted to confirm that the low-power CPUs used by Pergamum to save energy are actually capable of meeting the encoding demands of archival workloads.

In our first data encoding test, we measured the throughput of the XOR operation by updating parity for 50 MB of data. We were able to achieve an average encoding rate of 20.79 MB/s on the tome's CPU. For reference, a desktop class processor using the same library was able to encode data at 201.41 MB/s. However, this performance increase comes at the cost of power consumption; the Intel Core Duo processor consumes 31 W compared to the tome's ARM-based processor which consumes roughly 2.5 W for the entire board.

A similar result was achieved when updating parity for 50 MB of data protected by a 5+2 Reed Solomon configuration. As Table 6 summarizes, the processor on the Pergamum tome was able to encode the new parity blocks at a rate of 3.13 MB/s. For reference, the desktop processor could encode at average rate of 33.68 MB/s. Again, we notice an order of magnitude throughput increase at the cost of over an order of magnitude power consumption increase.

Our final encoding experiment involved the generation of data signatures. Our current implementation of Pergamum generates data signatures using $GF(2^{32})$ arithmetic in an optimized C-based library. Generating 64 bit signatures over 32 bit symbols, we achieved an average signature generation throughput of 57.44 MB/s. For reference, the same library on the desktop-class client achieved a rate of 533.33 MB/s.

Our results indicate that the low-power processor on the Pergamum tome is capable of encoding data at a rate comparable to its power consumption. Additionally, we believe that is capable of adequately encoding data for an archival system's write-once, read maybe usage model. While our current performance numbers are reasonable, our experience in designing and implementing the Pergamum prototype has shown that low-power processors greatly benefit from carefully optimized code.

Our early implementations provided more than adequate performance on a desktop class computer but were somewhat slow on the Pergamum tome's low-power CPU.

6 Future Work

While Pergamum demonstrates some of the features needed in an archival storage system, work remains to turn it into a fully effective, evolving, long-term storage system. In addition to the engineering tasks associated with optimizing the Pergamum implementation for low-power CPUs, there are a number of important research areas to examine.

Storage management in Pergamum, and archival storage in general, is an open area with a number of interesting problems. Management strategies play a large part in cost efficiency; many believe that management costs eclipse hardware costs [3]. As a long-term data repository, the effectiveness of archival storage is increased as management overhead is decreased and, ideally, automated; the easier it is to store long-term data, the more ubiquitous it will become. Thus, Pergamum must address how extensibility can be handled in an automatic way, without sacrificing its distributed nature, or the independence of its Pergamum tomes. This overall question includes a number of facets. How are redundancy groups populated? How does the system know if a Pergamum tome is nonfunctional as opposed to temporarily off-line? How can the system's capacity be expanded while still providing adequate reliability?

In our current implementation, users interact with Pergamum by submitting requests to specific Pergamum tomes using a connection-oriented protocol. In future versions, the use of a simple, standardized `put` and `get` style protocol, such as that provided by HTTP, could allow storage to be more evolvable and permit the use of standard tools for storing and retrieving information. Further, techniques such as distributed searching that take into account data movement and migration could greatly simplify how users interact with the system.

While the trade-off between redundancy and storage usage is well acknowledged, there is still work to be done in understanding the interplay of redundancy, storage overhead and power consumption. We have chosen a relatively small set of points in this space; future work could explore this space more completely. This could include an examination of which redundancy codes are best suited to the unique demands and usage model of archival storage.

Long-term storage systems must assume that no single device will serve as the storage appliance for the data's entire lifetime. Thus, data migration in a secure and power-efficient manner is another requirement for Pergamum, and is a critical area for research. This research direction also has implications for reliability; a policy of device refreshment could be an integral part of a long-term reliability strategy.

The optimality of the choice of one CPU and network connection per disk is also an open question; our choice is based on both quantitative and qualitative factors, but other arrangements are certainly possible. Additionally, it has always been assumed that client machines would include modern desktop level CPUs that could be leveraged for pre-processing. Similarly, determining the best network to use to connect thousands of (mostly idle) devices is an interesting problem to consider.

7 Conclusions

We have developed Pergamum, a system designed to provide reliable, cost-effective archival storage using low-power, network-attached disk appliances. Reliability is provided through two levels of redundancy encoding: intra-disk redundancy allows an individual device to automatically rebuild data in the event of small-scale data corruption, while inter-disk redundancy provides protection from the loss of an entire device. Fixed costs are kept low through the use of a standardized network interface, and commodity hardware such as SATA drives; since each Pergamum tome is essentially "disposable", a system operator can simply throw away faulty nodes. Operational costs are controlled by utilizing ultra-low-power CPUs, power-managed disks and new techniques such as local NVRAM for caching metadata and redundancy information to avoid disk spin-ups, intra-disk redundancy, staggered data rebuilding, and hash trees of algebraic signatures for distributed consistency checking. Finally, Pergamum's performance is acceptable for archival storage: the use of many low-power CPUs instead of a few server-class CPUs results in disks that can transfer data at 3–5 MB/s, with faster performance possible through the use of optimized code.

At 2–3 W/TB and under $0.50/GB for a full system, Pergamum is far cheaper and more reliable than existing MAID systems, though the techniques we have developed may be applied to more conventional MAID designs as well. Moreover, a Pergamum system is comparable in cost and energy consumption to a large-scale tape archive, while providing much higher reliability, faster random access performance and better manageability. The combination of low power usage, low hardware cost, very high reliability, simpler management, and excellent long-term upgradability make Pergamum a strong choice for storage in long-term data archives.

Acknowledgments

We would like to thank our colleagues in the Storage Systems Research Center (SSRC) who provided valuable feedback on the ideas in this paper, helping us to refine them. We would also like to thank our shepherd Doug Terry and the anonymous reviewers for their insightful comments that helped us improve the paper.

This research was supported by the Petascale Data Storage Institute under Department of Energy award DE-FC02-06ER25768, and by the industrial sponsors of the SSRC, including Los Alamos National Lab, Livermore National Lab, Sandia National Lab, Agami Systems, Data Domain, Digisense, Hewlett-Packard Laboratories, IBM Research, LSI Logic, Network Appliance, Seagate, Symantec, and Yahoo!.

References

[1] ADYA, A., BOLOSKY, W. J., CASTRO, M., CHAIKEN, R., CERMAK, G., DOUCEUR, J. R., HOWELL, J., LORCH, J. R., THEIMER, M., AND WATTENHOFER, R. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)* (Boston, MA, Dec. 2002), USENIX.

[2] AGARWALA, S., PAULY, A., RAMACHANDRAN, U., AND SCHWAN, K. e-SAFE: An extensible, secure and fault tolerant storage system. In *Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)* (2007), pp. 257–268.

[3] ANDERSON, E., HOBBS, M., KEETON, K., SPENCE, S., UYSAL, M., AND VEITCH, A. Hippodrome: running circles around storage administration. In *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)* (Monterey, CA, Jan. 2002).

[4] ARCOM, INC. `http://www.arcom.com/`, Aug. 2007.

[5] ATA SMART feature set commands. Small Form Factors Committee SFF-8035. http://www.t13.org.

[6] BAIRAVASUNDARAM, L. N., GOODSON, G. R., PASUPATHY, S., AND SCHINDLER, J. An analysis of latent sector errors in disk drives. In *Proceedings of the 2007 SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (June 2007).

[7] BAKER, M., SHAH, M., ROSENTHAL, D. S. H., ROUSSOPOULOS, M., MANIATIS, P., GIULI, T., AND BUNGALE, P. A fresh look at the reliability of long-term digital storage. In *Proceedings of EuroSys 2006* (Apr. 2006), pp. 221–234.

[8] CAO, P., LIN, S. B., VENKATARAMAN, S., AND WILKES, J. The TickerTAIP parallel RAID architecture. *ACM Transactions on Computer Systems 12*, 3 (1994), 236–269.

[9] CHEN, Y. C., RETTNER, C. T., RAOUX, S., BURR, G. W., CHEN, S. H., SHELBY, R. M., SALINGA, M., RISK, W. P., HAPP, T. D., MCCLELLAND, G. M., BREITWISCH, M., SCHROTT, A., PHILIPP, J. B., LEE, M. H., CHEEK, R., NIRSCHL, T., LAMOREY, M., CHEN, C. F., JOSEPH, E., ZAIDI, S., YEE, B., LUNG, H. L., BERGMANN, R., AND LAM, C. Ultra-thin phase-change bridge memory device using GeSb. In *International Electron Devices Meeting (IEDM '06)* (Dec. 2006), pp. 1–4.

[10] COLARELLI, D., AND GRUNWALD, D. Massive arrays of idle disks for storage archives. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing (SC '02)* (Nov. 2002).

[11] DHOLAKIA, A., ELEFTHERIOU, E., HU, X.-Y., ILIADIS, I., MENON, J., AND RAO, K. Analysis of a new intra-disk redundancy scheme for high-reliability RAID storage systems in the presence of unrecoverable errors. In *Proceedings of the 2006 SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (2006), pp. 373–374.

[12] DHOLAKIA, A., ELEFTHERIOU, E., HU, X.-Y., ILIADIS, I., MENON, J., AND RAO, K. Analysis of a new intra-disk redundancy scheme for high-reliability RAID storage systems in the presence of unrecoverable errors. Tech. Rep. RZ 3652, IBM Research, Mar. 2006.

[13] DRAPEAU, A. L., AND KATZ, R. H. Striped tape arrays. Tech. Rep. CSD-93-730, University of California, Berkeley, 1993.

[14] GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.-T. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)* (Bolton Landing, NY, Oct. 2003), ACM.

[15] GIBSON, G. A., NAGLE, D. F., AMIRI, K., BUTLER, J., CHANG, F. W., GOBIOFF, H., HARDIN, C., RIEDEL, E., ROCHBERG, D., AND ZELENKA, J. A cost-effective, high-bandwidth storage architecture. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (San Jose, CA, Oct. 1998), pp. 92–103.

[16] GIBSON, G. A., AND VAN METER, R. Network attached storage architecture. *Communications of the ACM 43*, 11 (2000), 37–45.

[17] GREEN GRID CONSORTIUM. The green grid opportunity, decreasing datacenter and other IT energy usage patterns. http://www.thegreengrid.org, Feb 2007.

[18] GREENAN, K. M., AND MILLER, E. L. Reliability mechanisms for file systems using non-volatile memory as a metadata store. In *6th ACM & IEEE Conference on Embedded Software (EMSOFT '06)* (Seoul, Korea, Oct. 2006), ACM.

[19] GUHA, A. Solving the energy crisis in the data center using COPAN Systems' enhanced MAID storage platform. Copan Systems white paper, Dec. 2006.

[20] GUNAWI, H. S., AGRAWAL, N., ARPACI-DUSSEAU, A. C., ARPACI-DUSSEAU, R. H., AND SCHINDLER, J. Deconstructing commodity storage clusters. In *Proceedings of the 32nd Int'l Symposium on Computer Architecture* (June 2005), pp. 60–71.

[21] HAEBERLEN, A., MISLOVE, A., AND DRUSCHEL, P. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)* (Boston, MA, May 2005), USENIX.

[22] HENDRICKS, J., GANGER, G. R., AND REITER, M. K. Verifying distributed erasure-coded data. In *Proceedings of the 26th ACM Symposium on Principles of Distributed Computing (PODC 2007)* (Aug. 2007).

[23] Health Information Portability and Accountability Act, Oct. 1996.

[24] HUGHES, J., MILLIGAN, C., AND DEBIEZ, J. High performance RAIT. In *Proceedings of the 19th IEEE Symposium on Mass Storage Systems and Technologies* (Apr. 2002), pp. 65–73.

[25] KALLAHALLA, M., RIEDEL, E., SWAMINATHAN, R., WANG, Q., AND FU, K. Plutus: scalable secure file sharing on untrusted storage. In *Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST)* (San Francisco, CA, Mar. 2003), USENIX, pp. 29–42.

[26] KOTLA, R., ALVISI, L., AND DAHLIN, M. SafeStore: a durable and practical storage system. In *Proceedings of the 2007 USENIX Annual Technical Conference* (June 2007), pp. 129–142.

[27] LI, J., KROHN, M., MAZIÈRES, D., AND SHASHA, D. Secure untrusted data repository (SUNDR). In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)* (San Francisco, CA, Dec. 2004).

[28] MANIATIS, P., ROUSSOPOULOS, M., GIULI, T. J., ROSENTHAL, D. S. H., AND BAKER, M. The LOCKSS peer-to-peer digital preservation system. *ACM Transactions on Computer Systems 23*, 1 (2005), 2–50.

[29] MERKLE, R. C. A digital signature based on a conventional encryption function. In *Advances in Cryptology - Crypto '87* (Berlin, 1987), Springer-Verlag, pp. 369–378.

[30] OXLEY, M. G. (H.R.3763) Sarbanes-Oxley Act of 2002, Feb. 2002.

[31] PINHEIRO, E., AND BIANCHINI, R. Energy conservation techniques for disk array-based servers. In *Proceedings of the 18th International Conference on Supercomputing* (June 2004).

[32] PINHEIRO, E., BIANCHINI, R., AND DUBNICKI, C. Exploiting redundancy to conserve energy in storage systems. In *Proceedings of the 2006 SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Saint Malo, France, June 2006).

[33] PRESTON, W. C., AND DIDIO, G. Disk at the price of tape? an in-depth examination. Copan Systems white paper, 2004.

[34] QUINLAN, S., AND DORWARD, S. Venti: A new approach to archival storage. In *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)* (Monterey, California, USA, 2002), USENIX, pp. 89–101.

[35] RHEA, S., EATON, P., GEELS, D., WEATHERSPOON, H., ZHAO, B., AND KUBIATOWICZ, J. Pond: the OceanStore prototype. In *Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST)* (Mar. 2003), pp. 1–14.

[36] SAITO, Y., FRØLUND, S., VEITCH, A., MERCHANT, A., AND SPENCE, S. FAB: Building distributed enterprise disk arrays from commodity components. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2004), pp. 48–58.

[37] SCHROEDER, B., AND GIBSON, G. A. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)* (Feb. 2007), pp. 1–16.

[38] SCHWARZ, T. J. E., XIN, Q., MILLER, E. L., LONG, D. D. E., HOSPODOR, A., AND NG, S. Disk scrubbing in large archival storage systems. In *Proceedings of the 12th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '04)* (Oct. 2004), IEEE, pp. 409–418.

[39] SCHWARZ, S. J., T., AND MILLER, E. L. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *Proceedings of the 26th International Conference on Distributed Computing Systems (ICDCS '06)* (Lisboa, Portugal, July 2006), IEEE.

[40] SEAGATE TECHNOLOGY LLC. Momentus 5400 psd. `http://www.seagate.com/docs/pdf/marketing/ds_momentus_5400_psd.pdf`, Aug 2007.

[41] SUN MICROSYSTEMS. Sun StorageTek SL8500 modular library system. `http://www.sun.com/storagetek/tape_storage/tape_libraries/sl8500/`.

[42] SUN MICROSYSTEMS. Solaris ZFS and Red Hat Enterprise Linux EXT3 file system performance. `http://www.sun.com/software/whitepapers/solaris10/zfs_linux.pdf`, June 2007.

[43] TANABE, T., TAKAYANAGI, M., TATEMITI, H., URA, T., AND YAMAMOTO, M. Redundant optical storage system using DVD-RAM library. In *Proceedings of the 16th IEEE Symposium on Mass Storage Systems and Technologies* (Mar. 1999), pp. 80–87.

[44] TEHRANI, S., SLAUGHTER, J. M., CHEN, E., DURLAM, M., SHI, J., AND DEHERRERA, M. Progress and outlook for MRAM technology. *IEEE Transactions on Magnetics 35*, 5 (Sept. 1999), 2814–2819.

[45] WEDDLE, C., OLDHAM, M., QIAN, J., WANG, A.-I. A., REIHER, P., AND KUENNING, G. PARAID : A gear-shifting power-aware RAID. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)* (Feb. 2007).

[46] WEIL, S. A., BRANDT, S. A., MILLER, E. L., LONG, D. D. E., AND MALTZAHN, C. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)* (Seattle, WA, Nov. 2006), USENIX.

[47] WESTERN DIGITAL. WD Caviar GP 1 TB SATA hard drives. `http://www.westerndigital.com/en/library/sata/2879-701229.pdf`, Aug. 2007.

[48] WILCKE, W. W., GARNER, R. B., FLEINER, C., FREITAS, R. F., GOLDING, R. A., GLIDER, J. S., KENCHAMMANA-HOSEKOTE, D. R., HAFNER, J. L., MOHIUDDIN, K. M., RAO, K., BECKER-SZENDY, R. A., WONG, T. M., ZAKI, O. A., HERNANDEZ, M., FERNANDEZ, K. R., HUELS, H., LENK, H., SMOLIN, K., RIES, M., GOETTERT, C., PICUNKO, T., RUBIN, B. J., KAHN, H., AND LOO, T. IBM Intelligent Bricks project—petabytes and beyond. *IBM Journal of Research and Development 50*, 2/3 (2006), 181–197.

[49] YAO, X., AND WANG, J. RIMAC: a novel redundancy-based hierarchical cache architecture for energy efficient, high performance storage systems. In *Proceedings of EuroSys 2006* (Oct. 2006), pp. 249–262.

[50] YOU, L. L., POLLACK, K. T., AND LONG, D. D. E. Deep Store: An archival storage system architecture. In *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)* (Tokyo, Japan, Apr. 2005), IEEE.

[51] ZHU, Q., CHEN, Z., TAN, L., ZHOU, Y., KEETON, K., AND WILKES, J. Hibernator: Helping disk arrays sleep through the winter. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)* (Brighton, UK, Oct. 2005), ACM.