Protecting Data Integrity with Storage Leases

Anjo Vahldiek, Eslam Elnikety, Ansley Post, Peter Druschel, Rodrigo Rodrigues

Max Planck Institute for Software Systems (MPI-SWS)

Technical Report: MPI-SWS-2011-008

Abstract

We present a storage primitive called a *storage lease*. Data stored under a lease cannot be written for a pre-determined period. During the lease period, online data is protected from corruption due to security breaches, software errors, or accidental data deletion. Storage leases fill an important gap in the spectrum of data protection options because they combine strong integrity for online data with the ability to eventually reclaim storage. We define the storage lease primitive, show how it can be implemented in storage device firmware, and discuss its applications. A simulation-based evaluation indicates that storage leases have a modest performance cost for most workloads on magnetic disks. Using a small amount of flash memory, this overhead can be reduced to near zero.

1 Introduction

Society's ever-increasing dependence on digital information underlines the importance of protecting data stored in a digital format. Corporations, governments, and other professional organizations are well aware of the need to protect their digital assets, since data loss can directly translate into operational disruption, financial losses, or the loss of irreplaceable cultural and historical artifacts. But even individuals increasingly store valuable digital information, such as personal financial and professional documents, as well as photos, videos, personal letters and other information of substantial sentimental value.

There are many threats to the durability and integrity of digitally stored data, ranging from failure or destruction of storage media and devices, bugs in various parts of the software stack (e.g., device drivers, storage system software, operating systems, or applications), operator error (either intentional or accidental), or deliberate manipulation by malicious intruders or malware running on a machine with access to the data.

Many of these threats can be mitigated by storing data redundantly. For instance, error-correcting codes can mask bit errors, striping or mirroring can mask block or device failures, and backup data copies can enable recovery from catastrophic storage system failures. However, threats from malicious intruders, viruses and worms, software bugs and operator errors may affect *all* copies of stored data that are on-line and writable. Redundancy alone is not sufficient to mitigate these threats.

To protect the data from such threats, backup copies must be maintained in either off-line or write-once storage. Enterprise storage solutions include such archival storage. Data can be backed up on tapes that are unmounted after a backup session. Tape cartridges, magneto-optical and ultra-density optical disks offer a write-once option, offering additional protection for existing data while the medium is mounted. Finally, high-end network-attached storage systems can be programmed to accept only the first write to a given storage block (WORM storage [10]), even though the underlying storage is based on conventional disk drives.

However, high-end solutions like tape robots and network-attached WORM storage systems are expensive; low-end write-once optical disks and magnetic tape drives are inexpensive, but require diligent and regular attention by an administrator. Home users and small businesses, in particular, may not have the staff, expertise, or budget to adequately protect their data.

In this paper, we introduce *storage leases*. A storage lease enabled device associates each data block with a lease period, during which the block cannot be written. Like off-line or write-once storage, leases protect data from (accidental or intentional) deletion or corruption, without requiring mechanical action and without committing storage indefinitely. Storage leases can be implemented within the firmware of a disk device. Therefore, the lease guarantee holds despite errors or compromise in the operating system, file or storage software.

In combination with a backup application, storage leases can bring enterprise-grade data protection to consumers, homes and small businesses. For instance, backup applications like Apple's TimeMachine [23] or Microsoft's Backup and Restore Center [27] automatically create and maintain period snapshots. By storing the snapshots under a lease for the snapshot's retention period (e.g., daily snapshots kept for a week, weekly snapshots kept for a months, etc.), the snapshots are protected from software errors [9], operator mistakes, malware and other system intrusions.

We provide additional background in Section 2. Section 3 discusses related work. Section 4 presents the storage lease abstraction, Section 5 describes implementations, and Section 6 discusses applications of storage leases. In Section 7, we report results of a simulation-based evaluation of storage leases, and Section 8 concludes.

2 Background

This section provides background on state-of-the-art techniques to protect data.

Data protection techniques

Creating redundant copies of data is the key to making it durable, but different levels of guarantees can be achieved depending on the way these copies are created and maintained. We break these down into the following four levels. First, *replicated file and storage systems* such as RAID [14] protect against storage device failure. Second, *on-line snapshots* protect against user error and application software faults. Third, creating *off-line or write-once copies* of data protects against storage system software and hardware errors, as well as security compromises that can affect on-line data. Lastly, creating additional *off-site copies* of the data protects against catastrophic failures that wipe out an entire site. We analyze how these different measures are usually applied in different environments.

Enterprise environment

In an enterprise, data protection is one of the key responsibility of a professional IT department. To minimize the chances of data loss and to implement legally mandated data retention policies, a combination of techniques are employed. Often enterprises use a tiered storage model, where data is stored redundantly at multiple levels, and additionally moved to offline storage on a periodic basis. Servers often use some form of RAID configuration [14], and on-line snapshoting (e.g., NetApp WAFL [11], Sun's ZFS [28], or Windows VSS [26]). Off-line copies of the snapshot data are then created by copying the data onto tapes, and archived manually or using a tape robot. Lastly, tapes or disks are transported periodically to a safe location to create off-site copies.

Even though storage leases are not designed primarily for enterprise environments, they may help improve enterprise backup practices by (i) write-protecting data until it is has moved to offline storage, and (ii) lowering the cost of the solutions that are in place today by replacing expensive equipment like tape robots with the use of storage lease-enabled disks.

Home and small business environment

Increasingly, homes and small businesses have critical business and financial data, as well as personal data of great sentimental value to protect. However, they cannot afford the same professionally managed infras-

tructure as in the enterprise. Thus they are vulnerable to viruses, security attacks, or user error that could result in data loss.

Currently, diligent users install backup software (e.g., Apple's TimeMachine [23] or Microsoft's Backup and Restore [27]). However, unless a user is careful to take backup storage offline and avoid reconnecting it except when needed for a restore, this data is vulnerable to security compromises, like a virus or worm infection that destroys on-line data, or even to software bugs that silently erase backup data [9].

Some users may create off-line copies by periodically copying snapshots to write-once storage (e.g., DVDs). Since this process cannot be automated, it is burdensome and often not performed on a regular basis in the absence of professional IT staff.

Finally, some users rely on cloud storage to protect their data. Cloud storage does not ensure data integrity, because data can be deleted or corrupted by compromised client software or user error. Moreover, data stored in the Cloud can be lost [12]. Finally, in some cases, limited broadband upload bandwidth can lead to a high data transfer time and therefore increase the window of vulnerability during which data is not backed up. Despite these issues, Cloud storage can complement other forms of backup, because it provides an off-site replica of the data.

3 Related work

3.1 Replicated storage systems

In order to survive the failure of individual storage nodes, data must be replicated on multiple devices or media. Several commercial and research proposals offer various types of replicated storage systems.

RAID [14] makes multiple disks appear as a single device. RAID can be configured in multiple ways that alter the cost/reliability/performance tradeoffs. In the simplest case, data is written to two disks (known as mirroring), such that the failure of either disk does not result in the loss of data. More complicated RAID configurations allow the failure of more than one disk, or reduce the required redundancy using data coding techniques. Storage leases address the orthogonal problem of protecting the integrity of redundant data.

Among many proposed replicated file systems, Envy [3] is interesting in that it aims to protect users from file system bugs. It uses three separate file system implementations on top of a general block store interface, and uses majority voting when the results returned by each of the file systems disagree. Envy stores separate metadata for each file system replica, but shares the data blocks for space efficiency. Because Envy resides inside of a normal OS kernel, it is vulnerable to OS bugs and to attacks that compromise the OS. Storage leases do not attempt to mask file system bugs, but enable recovery from such bugs by protecting the integrity of online snapshots.

3.2 Versioning and backup

Many state-of-the-art file systems (e.g., NetApp WAFL [11], Sun's ZFS [28], or Windows VSS [26]) provide the ability to take consistent snapshots on demand. Taking frequent snapshots (daily or even hourly) is efficient, because data blocks are copied only when a file is modified, and each snapshot consists only of blocks modified since the previous snapshot. Unlike a snapshot file system, which only retains versions that exist at a time when a snapshot is taken, Files-11 [13] (the file system in OpenVMS), as well as Elephant [19] generally retain all versions of a file for some time.

Snapshots are commonly copied to another backup device. There are many possible ways of implementing backup [5], with a variety of tradeoffs in terms of cost, time, and space required. A number of back-up applications (e.g., TimeMachine [23] or Backup and Restore Center [27]) create periodic snapshots of the files on a client workstation or notebook, and store these snapshots on a separate disk or a remote server. Unlike those produced by a file system, the snapshots are not necessarily consistent. The snapshots are incremental at the file level, i.e., any modified file is copied in its entirety.

Pastiche [6] is a cooperative backup system that allows users to store encrypted copies of each others' data. In Friendstore [24], only friends within a social network store each others' data, which alleviates problems

associated with storing data on untrusted nodes, and is better aligned with users' incentives. Cumulus [25] uses the cloud as a store for backup data.

Storage leases address the orthogonal problem of ensuring integrity of the backup data.

3.3 Write-once storage

Storing backup data in write-once storage trivially rules out accidental or malicious corruption. The simplest form of write-once storage is a CD, DVD or Blu-ray disk. Venti [16] is a centralized storage service that implements a write-once interface and is intended as a storage back end for archival data. Internally, data is stored on a RAID disk group. Data blocks with identical content are coalesced prior to being stored.

Because data cannot be deleted in either of these systems, sufficient storage must be provided to keep all archived data permanently. Storage leases, on the other hand, allow the reclamation of backup storage after a pre-determined period.

3.4 Time protected storage

One mechanism for protecting the integrity of stored data is to take the storage device or media offline. One practical way of doing this is to rotate magnetic tape, where each tape is only used for some time and then taken offline for a specified period. There are a variety of schemes [5], which provide different guarantees depending on the pattern in which tapes are rotated. Storage leases, on the other hand, protect data integrity without requiring mechanical action or operator attention.

Self Securing Storage (S4) [22] implements a storage device that transparently retains shadow copies of all modified data for a pre-determined window of time (typically, on the order of weeks). The system is intended to prevent intruders from undetectably tampering with or permanently deleting stored data. Storage leases differ from S4 in two ways. First, storage leases permit fine-grained control over which data should be protected for what period of time. Second, storage leases can be implemented at the block device level; thus, the lease guarantee depends only on the integrity of the storage firmware.

NetApp's SnapVault [10] RAID storage server allows the creation of compliance volumes, which allow data to be retained for a specified time period for regulatory compliance. It comes in two types, one that prevents deletion of entire volumes within a specified period of time, and one that prevents deletion of individual files within a specified period of time. Storage leases bring a similar capability to low-end storage systems. Because leases operate at the block level, they can be implemented in the device firmware of consumer disks. Moreover, storage lease devices can generate secure receipts, enabling an end-to-end verification that data has been stored and protected for the desired period.

3.5 Extending disk functionality

Like storage leases, other work has looked at extending the functionality of storage devices. Self-encrypting hard disks [1] protect data from being accidentally leaked when a device is lost or stolen, by encrypting data before it is stored. Active disks [17] allow application-specific functions to be executed on storage devices for performance. Storage leases rely on the storage device firmware for the complementary purpose of protecting data integrity.

4 Storage leases

In this section, we present the design of storage leases. Storage leases provide a primitive to protect data from operator error, security breaches and most software errors. We begin this section by defining the abstraction and its properties.

4.1 The storage lease primitive

The main goal of storage leases is to protect stored data, for a pre-determined period of time, from accidental or malicious modification. Once written under a storage lease, a unit of data cannot be modified or deleted until the lease expires. During the lifetime of a SL, data stored under the SL is protected from modification or deletion due to human error, worms, viruses or other security breaches, and software bugs above the storage layer [9].

Storage leases provide the following guarantee. When a unit of data b is written to an SL-enabled device D with an expiration time of t, the device ensures that:

At any time smaller than t, the device rejects write requests for data block b.

Furthermore, once a storage lease is written, the device can issue a certificate c that attests to the storage commitment.

This guarantee holds despite arbitrary faults outside of the system component that implements the storage lease. For instance, in Section 5 we propose an implementation in the firmware of a disk drive. In this case, this guarantee holds despite any faults outside the device hardware or firmware (e.g., even if subsequently the host to which the device is attached becomes infected by malware).

Note that writing a block with a storage lease of infinity has the same effect as writing a block to write-once storage. Thus, write-once storage can be viewed as a special case of a storage lease.

4.2 Benefits of a write protection spectrum

Storage committed under a storage lease cannot be reclaimed during the lease period, but can be reclaimed anytime after the lease has expired. Thus, SL-enabled storage covers a spectrum from conventional read-write storage (which can be reclaimed at any time but does not protect data integrity) to write-once storage (which protects data but can never be reclaimed).

This additional degree of freedom enables a continuous trade-off between data protection and storage commitment, which is particularly important in environments that lack professional system administration: on the one hand, we need write protection, because taking storage off-line by manual or mechanical means is not practical. On the other hand, we cannot backup all data in write-once storage, because many systems produce large amounts of scratch data that would be expensive and unnecessary to store permanently. Storage leases provide a degree of freedom that can be used to resolve this tension.

4.3 Certificate generation

When committing to a storage lease, the storage device signs a certificate, which provides unforgeable evidence that the data was written with a certain lease period to a specific device.

Certificates are useful when SL-enabled devices are attached to machines that may be compromised. For example, consider the following scenario. A user has two machines in a home or corporate network, m_1 and m_2, where m_2 is infected by a virus but m_1 is not. When m_1 tries to use m_2's SL-enabled device to back up data, the malicious software running on the machine might act is if it had stored the data but in practice it discards it, leading m_1 to wrongfully believe the data had been backed up. To prevent this attack, the backup agent can request a certificate from the SL-enabled device. This certificate provides a confirmation of the write operation, cryptographically signed by the SL-enabled storage device.

Furthermore, as we will explain in Section 6, certificates can enable other uses of storage leases, namely to provide proof of storage commitment in environments where multiple parties from mutually distrusting administrative domains interact, such as peer-to-peer storage systems or cloud storage.

4.4 Managing storage space-time

Storage leases prevent the corruption of stored data, even when a computer's operating system is compromised by an attacker. However, leases also introduce a new attack vector: an adversary (e.g., a virus that has infected a machine) can commit the resources of an attached SL-enabled device by storing large quantities

of useless data with long-term leases. The effect of such an attack is less serious than data corruption, as it is limited to preventing future use of the storage (denial-of-service). Nevertheless, it is a threat that must be addressed.

To understand how we can mitigate this attack, we must precisely characterize the resource that is being managed by an SL-enabled device. The resource commitment associated with a storage lease is proportional to both the amount of data and the period of the lease. This commitment is captured by the *storage space-time* $s_i = n_i * t_i$, where n_i is the amount of data covered and t_i is the period of lease i. The total space-time S committed in a device is $S = \sum_{i \in L} s_i$, where L is the set of existing leases (whether expired or not) in the device.

To mitigate *space-time filling attacks*, an SL-enabled device limits the rate at which space-time can be consumed by new leases, i.e., $dS/dt \leq R$. The device enforces that R can only be set once by the administrator installing the device; it should be chosen to limit the space-time an attacker can consume within a given period. To illustrate the usefulness of this rate limiting, assume that a virus infection or a security breach is detected after no more than T_{detect}, and that a device with capacity C has an expected lifetime of L, i.e., after L units of time the device will be filled with storage leases when used at its maximum rate. Then, we can ensure that an attacker can at most consume 1% of the device's total space-time by setting $R = 0.01\frac{C*L}{T_{detect}}$. Note that the attacker can choose to consume more data for a shorter time, or vice versa, but the damage is limited either way.

Choosing R involves a trade-off between limiting the damage an attacker can cause and admitting legitimate leases at a sufficient rate to cover peak demand. In practice, however, satisfying the latter concern is not as critical as it may seem. First, there is often high demand for space-time during the installation of a new storage device, e.g., when an operating system or a database is installed. We can make sure this initial demand is not subject to the rate limit by setting R *after* the initialization. Second, legitimate users of leases (e.g., file systems or backup applications) can work around the rate limit should they exceed it during peak demand; they can simply request a lease with a shorter period than desired, and extend the lease prior to its expiration. An attacker, on the other hand, only has limited time until detection, and can at most commit $R \cdot T_{detect}$ of space-time.

Nevertheless, if an attack is not detected for a long time, a large amount of space-time may be wasted. To enable recovery from such an event, SL-enabled devices may provide a hardware switch (DIP switch or jumper) on the circuit board of the device, which can be used to reset all leases and the rate R. The switch can be operated by an expert as part of the system recovery. Note that the guarantees of storage leases are not affected by the presence of such a switch, because it cannot be activated by any software action.

4.5 Storage lease interface

The storage lease interface is similar to that of a block device. Defining the storage lease interface at the block level gives more flexibility in how to implement it, e.g., at the device firmware level. The interface, as shown in Table 1, contains three key elements that depart from a traditional block device interface.

Lease period: The block write operation takes the expiration time t for the lease of the newly written block as an argument. A write fails if the target block is currently protected by a lease, or granting the lease would exceed the rate limit R. There is also an update operation used to extend the lease of an existing block, and an attest operation used to obtain the lease period of an existing block.

Certificates: The storage lease interface allows clients to obtain a signed lease certificate after data has been written with a given retention period. Note that the certificate must cover the absolute expiration time (and not the lease duration) to prevent replay attacks.

Batching: To amortize the overhead of signature generation for certificates, a sequence of SL operations can be batched together and a single certificate issued for the batch. Reducing the number of signatures to one per batch instead of one per operation greatly improves the performance of, for instance, large file writes consisting of many block writes. Also, we must give upper layers control over how operations are delimited within a batch, so that the certificate can identify objects such as files or directories rather than blocks. Therefore, at any time within a batch, it is possible to obtain a hash of a series of data blocks operated on as part of the batch. These hash values are what is attested to when a certificate is subsequently generated.

Function	Description and Exceptions
`BId startBatch()`	starts a new batch and returns its id; fails if too many open batches
`void write(BId id, Bnum b, Expiration t,void *buf)`	writes data block #b from buffer buf with lease expiration time t; fails if b is lease protected, or space-time rate limit exceeded
`void read(BId id, Bnum b, void *buf)`	reads block #b and stores it in buffer buf
`void update(BId id, Bnum b, Expiration t)`	extends the lease of block #b to expiration time t; fails if t is earlier than b's current lease expiration, or space-time rate limit exceeded
`Expiration attest(BId id, Bnum b)`	returns the current lease expiration time for block #b, or zero if block is not covered by an active lease
`Hash hash(BId id)`	returns the accumulated hash in the batch, then resets the accumulator; this accumulator is updated upon read/write/attest.
`Certificate getCertificate(BId id)`	returns a certificate confirming the set of operations in the batch; the certificate is a signed hash of the set of successful operations in the batch, including their arguments and results
`void endBatch(BId id)`	terminates a batch; subsequent invocations of any operation with this batch id fail
`void setTime(TimeCerticate T)`	Advances the device's clock to the time provided in the certificate T; fails if T's signature can't be verified; ignored if T's time is less than or equal to the device's current time
`void setRate(long rate)`	Sets the rate for new lease admission by specifying the maximal rate of space-time commitment; only the first invocation succeeds

Table 1: Storage lease device interface. Additionally, the device supports normal read and write operations. However, write operations to a block with an active lease fail.

Therefore, certificates have the following format:
$[Hash(o_1 + o_2 + .. + o_n), D]_{\sigma_D}$ where $o_1, ..., o_n$ is a sequence of operations in a batch, as delimited by the application, + indicates concatenation, and D the SL device. Moreover, o_i is the concatenation of the operation identifier, arguments and results of the ith operation in the batch.

5 Implementing storage leases

Next, we describe an implementation of storage leases in the firmware of a disk device. Section 5.6 briefly discusses alternative implementations in the OS driver of a storage device, and as part of a cloud storage service.

Implementing storage leases at the firmware level has a significant advantage: the trusted computing base required to enforce the properties of storage leases is limited to the firmware and the device hardware. Software faults or compromises of the OS and applications do not affect the lease guarantees.

A potential challenge is that this implementation requires support from the device vendor. We believe that such support is feasible, because the cost for implementing leases is moderate and vendors have an incentive to include value-added features. E.g., disk vendors have recently added support for data encryption to disk drives [1].

Implementing storage leases requires four tasks, which we consider in turn: (1) Maintaining lease information for each data block and enforcing write protection; (2) keeping track of real time to decide when a lease has expired; (3) controlling the rate of space-time committed to leases; and (4) generating signed lease certificates on demand.

5.1 Keeping lease information and enforcing protection

The firmware sets aside a set of data blocks in non-volatile memory, which contain the lease expiration times for each of the primary data blocks (we call these lease blocks). Given that a 16-bit timestamp is sufficient for a lease granularity of one day, the overhead for storing an array of timestamps, one for each 4KB data block, is less than 0.05%.

Before writing to any block on the device, the respective lease block must be checked and the write denied if there is an unexpired lease protecting the block. To speed up this step, a cache of lease blocks is maintained in the disk controller's DRAM cache. Modified lease blocks are flushed to non-volatile memory before a flush operation completes or the certificate for the respective batch of writes is returned (only then can the application be sure that newly written blocks with a storage lease are protected).

We have explored two approaches to storing the lease information. One is to store the lease values in dedicated blocks on the magnetic disk, and the other is to store this information in the flash memory of a hybrid disk drive [20].

Storing leases on the magnetic disk: Here, the lease blocks are grouped on the physical disk into extents of at least the same size as the cache line of the on-disk cache. Therefore, once the disk head is positioned for a desired lease block, an entire extent can be read or written efficiently to/from the cache, taking advantage of likely spatial locality. The lease block extents are placed equidistantly on the disk, such that an extent contains leases for the data blocks closest to the extent. This choice minimizes, subject to the choice of lease block extent size, the seek distance between a data block and the lease block containing the associated lease. For instance, if the block size is 4KB (containing 2048 lease values) and a cache line has 64 blocks, then a lease block extent of 64 blocks follows every extent of $64 * 2048 = 128K$ data blocks.

Storing leases in flash memory: Here, we maintain a B+Tree in flash memory, which maps extents of contiguous disk blocks to the associated lease value. Updates to the tree are written to an append-only log in flash. An index stored in the disk's DRAM maps each tree node identifier to the most recent copy of the node in the log, which allows fast tree lookups. Additionally, we maintain a small cache of recently accessed flash pages in DRAM. During power-up, the DRAM index is recovered by reading the log tail in reverse order. To bound the recovery time, a snapshot of the DRAM index is periodically written to the log. Log cleaning is performed to reclaim space used by stale tree nodes, while ensuring even wear of the flash erase blocks.

Dealing with corrupted lease information: When data blocks storing lease information become corrupted, the disk can no longer determine whether a write to the corresponding data block should be allowed. This case can be handled in different way, namely: (i) replicate lease blocks, which would increase storage overhead, (ii) optimistically allow such writes, risking data loss, or (iii) deny such writes, effectively enforcing a lease on the block for the lifetime of the disk. Because a data block corruption is a relatively rare event and option (iii) is always safe, we decided for this strategy. Moreover, option (iii) only consumes capacity compared to (i) performance or (ii) lease guarantee.

5.2 Keeping real time

Keeping track of real time to determine when leases expire presents a design challenge. We cannot depend on the real-time clock of the attached computer, because that clock could be compromised by an attacker. Alternatively, adding a hardware real-time clock with the required battery to the circuit board of a storage device would add significant component and maintenance cost.

Our solution is based on two observations. First, it is sufficient for the device to maintain a time that is earlier or at most equal to the actual real time. Having an earlier time means that leases are enforced longer than needed, which is conservative. Second, the passage of time can be tracked at a coarse granularity, because leases are for long periods, e.g. months. Thus, a coarse precision on the order of one day is acceptable.

Given these observations, it is sufficient to have the operating system periodically provide a signed time certificate generated by a trusted time server to the device. A fault or compromise affecting the OS can at worst delay the flow of valid, up-to-date time certificates, which would delay the expiration of leases.

As mentioned, delaying the expiration is conservative and causes no harm to data (though it delays the reclamation of storage).

The requirements for the trusted time server are straightforward in terms of precision and query load. As mentioned, a precision as low as one day is sufficient. Serving a cached certificate even to hundreds of millions of devices once every hour is straightforward with current server technology. For instance, a commercial secure time service could be used [7], or commercially available trusted servers could be operated by vendors of SL-enabled storage devices or another trusted organization.

5.3 Rate-controlling space-time

As described in Section 4.4, the rate at which space-time is allocated to new leases is limited to prevent a compromised system from filling the disk with long-term lease commitments. The parameter R can be set once on a new disk (or after a manual reset) using the SetRate operation. Until R is set, no leases can be updated.

Whenever a new lease is requested or an existing lease extended, the firmware checks the sum of space-time commitments due to new or extended leases during the past 24 hours, and rejects the request if the rate exceeds R. Of course, the system can retry a rejected request with a shorter lease period. Note that a request may be denied because the firmware's notion of real-time is behind. In this case, providing the disk firmware with a recent time certificate resolves the problem.

5.4 Generating lease certificates

The device firmware must generate signatures for certificates, and verify the signatures of time certificates. The crypto primitives must be strong enough to prevent attacks on the signing keys. Because the keys need only be secure for the lifetime of a storage device (i.e., on the order of 10 years), we do not foresee the need for updates of the crypto libraries or keys on a device. The necessary keys can be assigned, and their validity verified once the devices are deployed, by the device manufacturer; crypto algorithms can be upgraded with each new device generation.

We assume that the disk controller has a low-end crypto co-processor, which efficiently computes cryptographic data hashes and RSA signatures. Such crypto chips are inexpensive and widely used in devices like broadband wireless routers. Finally, the state associated with a batch does not require more than a few hundred bytes, and can be stored in the device controller's existing RAM.

5.5 Firmware updates

To prevent an attacker from bypassing the lease protection, firmware updates must be protected. Therefore, the disk firmware must accept only updates that are signed by the device vendor or another trusted authority. The crypto infrastructure already in place for verifying time certificates can be reused for this purpose.

5.6 Alternative implementations

The storage lease abstraction can also be instantiated in a variety of alternative implementations with different properties.

Device driver. Providing this functionality as part of the storage driver is straightforward; however, a driver implementation increases the computing base trusted to enforce storage leases to include the OS kernel.

Storage enclosure. Low-end enclosures such as RAID boxes are now inexpensive enough even for home environments. Again, the implementation path is simple, and in this case only the storage enclosure firmware becomes part of the trusted computing base.

Cloud storage service. In this case, the cloud service enforces the guarantees of storage leases. Here, the trusted computing base for storage leases comprises the entire cloud service. While this is substantial, it is professionally managed and can be designed to have a narrow interface that minimizes the attack

surface. Moreover, the faults it may suffer are unlikely to be correlated with a fault or compromise in the customer's computing base. Thus, a combination of client-side SL-enabled storage devices and an SL-enabled cloud service provides strong protection from operator error, software faults, security compromises, and even catastrophic site failures.

6 Applications of storage leases

In this section, we sketch applications of storage leases. A storage lease enabled device appears to the operating system as a normal block device with an augmented interface. As such, the device is fully backward compatible and an unmodified operating system can use it as if it was a normal device. To make use of storage leases, however, an operating system, file system or application has to be modified to use the extended interface, and it has to be prepared to deal with the semantics of blocks protected by a lease.

In the following, we sketch four applications. The first two, a backup/restore application and a lease-aware snapshotting file system, are representative of different degrees of integration with the existing operating system and file system. The other two, peer-to-peer storage and cloud storage, highlight interesting uses of lease certificates.

6.1 Backup application

Implementing a backup and restore application that uses storage leases is relatively straightforward. A dedicated SL-enabled device (or a dedicated partition on such a device) is used for storing backup snapshots. The application mounts the partition or device as a raw device. It uses the usual block operations to read and write blocks, as well as the extended lease interface via IOCTL calls.

The application can chose any data and metadata format it desires to store backup snapshots, but has to respect the semantics of leases. First, all data and metadata associated with a snapshot must be stored under a lease with a period that equals the intended retention time of the snapshot. Second, any metadata that is shared among snapshots must be append-only, because a metadata block stored under a lease cannot be modified until the lease expires.

Apart from the application itself and minor modification to a disk device driver to support the SL-related IOCTL calls, no modifications are required to operating systems or file system. We have implemented a backup and restore application; its design and evaluation is presented in one author's dissertation [15].

6.2 Lease-aware file system

A tighter integration of lease-based storage requires a lease-aware file system. For instance, a lease-aware snapshotting file system could use leases to protect the integrity of in-place, copy-on-write snapshots, obviating the need to copy data to a dedicated device as part of a backup.

Starting from a snapshotting file system like WAFL or ZFS, the required changes are relatively modest. First, the file system must be extended to request leases for all data and metadata blocks associated with a given snapshot. Second, the file system allocator must be made aware that leased blocks cannot be reclaimed until the lease expires. Lastly, metadata stored under a lease must be append-only. The changes are very similar to those required for Flash memory aware file systems, where data blocks cannot be modified and reclaimed freely and individually. A detailed design of a lease-aware file system is beyond the scope of this paper.

6.3 Peer-to-peer storage

Lease certificates can be used as proof of a storage commitment in cooperative storage systems, to ensure that all users contribute storage. For instance, in a cooperative peer-to-peer backup system [6], peers might pretend to contribute their fair share of resources, while in reality they are free-loading. With storage leases, peers can demand lease certificates from peers that are supposed to store their data, and exclude those that

Workload	Description	I/O (MB)	ID.
Sequential	Sequential read/write	400	SR/SW
Local	All reads/writes to only N blocks	400	LR/LW
Random	Random read/write	400	RR/RW
Bittorrent	Download a movie	3087	BT
Game	Play a strategy game	659	G
Install	Download & install Firefox	647	IN
Movie	View a movie	1000	M
Web	Surf & read email	591	W

Table 2: Workloads used in the evaluation

fail to produce appropriate certificates. Note that a lease certificate proves not only that a peer currently stores the data, but that the peer will store the data for the lease period. (The peer has no way to delete the data, subject to the integrity of the storage device software and hardware.)

7 Evaluation

In this section, we evaluate the costs and performance of storage leases. There are two main sources of overhead: the storage cost of leases and the performance costs of checking leases, updating leases, and producing lease certificates.

As mentioned in Section 5, the raw storage overhead of a 16-bit lease value for each 4KB block is less than 0.05%. Lease values must be checked during each block write. They must be updated after a batch of block writes with storage leases. A lease certificate may need to be generated when a batch terminates. The cost for these operations depends on the placement of lease values and the policy for caching lease values in the disk controller's DRAM cache; the cost of lease update and certificate generation also depend on the batch size. Our evaluation seeks to quantify these overheads, and their performance impact on real applications.

We use an implementation of storage leases in the DiskSim [4] disk simulator to explore different lease value placements and caching policies under different workloads.

7.1 DiskSim evaluation

7.1.1 Experimental setup

To add support for storage leases in DiskSim, we implemented the `startBatch`, `write`, `read`, and `endBatch` commands, and added support for certificates as described in Section 5.

We use a validated disk model included in the DiskSim distribution. This modeled disk is a 15K RPM Seagate 15K.5 drive, with a capacity of 146GB and an on-disk DRAM cache size of 16MB. We configured the disk model to use segmented LRU cache replacement and a write back cache policy for disk writes. We used this particular disk model (a somewhat dated SCSI disk introduced in 2007), as it was distributed and validated by the DiskSim developers. In Section 7.3, we discuss the performance of leases on more recent disks, and on pure solid-state disks.

Pure disk (D). In the storage lease implementation on a pure magnetic hard disk drive, we reserved a region of the disk for lease blocks, and added support for lease block caching.

Hybrid disk (HD). We also added support for hybrid disks, which have a small amount (typically a few GB) of flash memory within the disk device assembly; such hybrid disks are already on the market, e.g. [20]. We simulate a Micron SLC NAND flash memory chip [21], with a page size of 2KB and an erase block size of 128KB. A page read/write takes 0.025ms/0.3ms, respectively, while an erase operation takes 2ms. The size of the simulated flash memory is configurable.

In the hybrid disk implementation of storage leases, we store lease values and the associated index in a portion of the flash memory, as described in Section 5.1. A flash implementation of a B+Tree (similar to [2])

maps extents of contiguous disk blocks to the associated lease value. The tree node size is 1KB, allowing 42 extent-to-lease mappings in a leaf node, and a fanout of 63 in the interior nodes. (These parameters were chosen empirically to achieve a good balance between lookup speed and flash memory consumption.)

To achieve good sequential access performance in the hybrid-disk implementation, we additionally implemented a lease pre-fetch optimization. When the controller observes sequential accesses to consecutive disk blocks, it pre-fetches the lease value for the next extent from flash into the DRAM cache while completing the disk writes for the current extent. As we will see, this optimization preserves the full sequential write throughput of the disk.

Workloads. Table 2 list the workloads used in the evaluation. We use the DiskSim synthgen module to generate three different synthetic workloads. The *random* workload generates a sequence of single-block writes to random blocks on the entire disk. The *local* workload generates single-block writes to random block numbers within a range of $\pm 40,000$ consecutive blocks from the current position. This range spans about 80 cylinders out of a total of about 72K cylinders on the disk. The *sequential* workload generates single-block writes to consecutive block numbers. In each workload, a total of 400MB of data is being written.

In addition, we gathered several I/O traces from an author's MacBook Pro notebook with 135GB of storage and the HFS+ file system. The traces were gathered using the `iosnoop` script, which is a wrapper around the DTrace [8] utility, and post-processed into a format usable by DiskSim. During each of the trace collections, the described activity was performed, while normal system and user background tasks were running (e.g., updating the mail database, or reading configuration files for Skype). The I/O operations and times at which they occurred were recorded in the trace for each task. Unless otherwise stated, we ignored the think times between consecutive disk requests when re-playing the traces in order to obtain maximal throughput results.

7.1.2 Lease lookup overhead

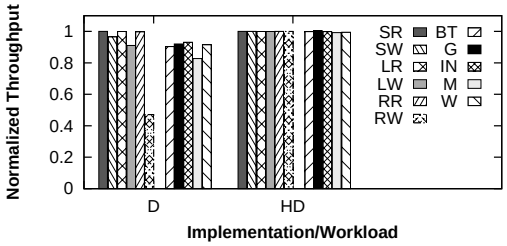
Prior to a write operation, the associated lease value must be checked to decide if the write can proceed. The cost of this check depends on several factors: the placement of lease values relative to the associated blocks on disk, or in the flash; the policy and size of the cache for lease values; and the workload. We first measured the impact of the lease lookup for different workloads, and for different lease value placements and caching policies.

We experimented systematically with different lease value placements and cache configurations, but cannot include the full set of results due to space constraints. The result show that, on the pure disk (D), one placement and cache configuration proved best across all workloads. In this configuration, the lease values are stored in equidistant extents of 1024 disk blocks (512 bytes each), followed by an extent of $1024 * 256$ associated data blocks. The placement strikes a good balance between the seek distance between lease value and data block on the one hand, and spatial locality of lease values on the other. A unified 16MB cache for both disk blocks and lease blocks proved to be the best cache configuration for the pure disk. Note that the line size of the cache is 128 blocks (512B each). The lease blocks are aligned such that a cache line is used in its entirety either for lease blocks or for data blocks.

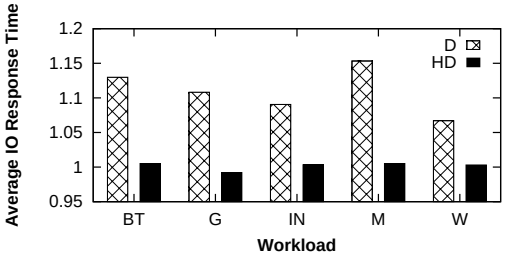
In the hybrid placement (HD), the leases are stored as a B+Tree in 64MB of flash memory. Moreover, 192KB of the 16MB DRAM are reserved for data related to leases, while the DRAM remains available as disk block cache. Of the 192KB of reserved DRAM, 64KB are used as a flash page cache, and 128KB are used to store the B+Tree node index and related data structures. This configuration proved to be best across all workloads for the hybrid disk. We use the best configurations for the pure disk and the hybrid disk in all remaining experiments.

Figure 1(a) shows the normalized throughput for the synthetic and trace workloads on the pure disk (D) and hybrid disk (HD). We make the following observations:

- The pure disk achieves performance within 82-93% of the baseline for the trace-based workloads, and within 91-100% of the baseline for the synthetic workloads other than random writes (RW).
- Random writes (RW) are the worst-case workload for storage leases on the pure disk. The throughput



(a) Normalized I/O throughput (ops/sec) for synthetic and trace workloads, for pure disk (D) and hybrid disk (HD) implementations. The results are normalized to the throughput without leases.



(b) Average per-operation I/O response time for the trace workloads using think-times. The results are normalized to the average per-operation response time without leases.

of random writes is slightly less than half of that without leases, because caching is not effective in absorbing the disk access to read the lease value, which is required to validate the write.

- The HD configuration achieves a very efficient implementation of storage leases, with a maximal overhead of less than 1%. The high access speed of the flash, together with the small DRAM cache, achieve an average lease lookup time small enough to be hidden by the access delays of the mechanical disk storage for the data.

Figure 1(b) shows the average per-operation response times for the trace-based workloads. Here, the workload traces were replayed with the recorded think times. (The response times for the synthetic workloads can be derived by inverting the throughput values shown in Figure 1(a).) The increase in response time due to storage leases on the trace based workloads is within 15% for the pure disk. With the hybrid disk, the response time is within 1% of that without leases.

The hybrid disk lease implementation with the game workload (G) has an average response time below that of the baseline by .72%! Further investigation revealed that reducing the size of the disk cache by the 192KB reserved for leases improves the response time for this workload, independent of leases. The slightly smaller cache leads to additional disk cache misses, which is expected. However, the total time required to serve the resulting slightly longer sequence of cache misses from disk is shorter, thus reducing the average response time.

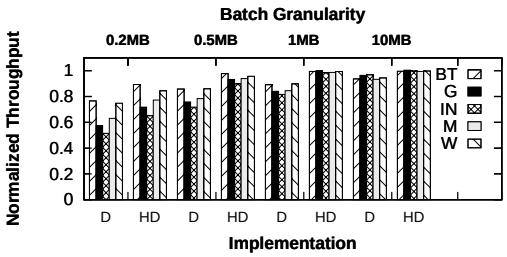
7.1.3 Lease write overhead

Next we focus on workloads where data is written with a storage lease. In this case, there are additional overheads for writing lease values and for generating a lease certificate at the end of a batch of writes with storage leases.

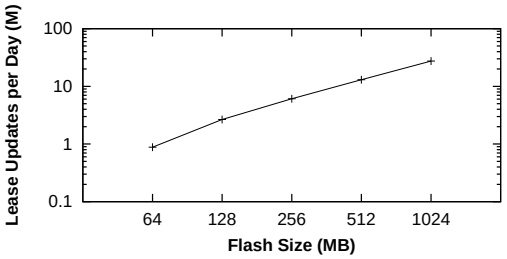
In the experiment, we run the different trace workloads (replayed without think time) concurrently with a mock backup workload, which writes 1GB of data sequentially in batches of increasing size. A lease certificate is requested for each batch. As discussed in Section 5.4, we assume that the disk controller has a low-end crypto co-processor. In our simulation, we assume a coprocessor like the SafeNet SafeXcel-1741 [18], which can hash data at 325MB/s, and generate an RSA signature in 8.4ms. We added support to DiskSim to keep track of the crypto co-processor utilization and block if a new request is posted while it is still busy with a previous request.

In addition to the cryptographic overhead, the updated lease blocks must be written to disk or flash when the end of a batch is requested. This requires flushing all modified data and lease blocks associated with that batch from the cache.

Figure 1(c) shows the throughput of the combined trace and mock backup workload for different batch sizes, normalized to the throughput of the same workload on a disk without lease support. Decreasing the batch size increases the frequency at which updated leases must be flushed to disk/flash and a signature generated. The results show that decreasing the batch size below 10MB for the disk-based, and below 512KB



(c) Normalized throughput (I/O ops/sec) for combined trace and mock backup workload with lease, with different batch sizes shown on the x axis in bytes. Crypto processor with 325 MB/s hash throughput, RSA latency 8.4ms.



(d) Maximum daily lease update rate, in millions, for different sizes of the flash log and a minimal lifetime of 10 years.

to 1MB for the hybrid placement, reduces the performance noticeably. The performance increases for larger batch sizes are insignificant. In backup applications, it is easy to achieve the batch sizes necessary to achieve the full disk bandwidth. It should be noted that our results are conservative, because (i) we assume a low-end crypto chip (more powerful co-processors can generate an RSA signature in tens of microseconds), and (ii) we replayed the trace workloads without think times.

7.2 Flash lifetime

The flash memory used in the hybrid disk can endure only a limited number of erase/program cycles. We need to make sure the flash does not wear out before the expected lifetime of the magnetic disk; to be conservative, we assume that the flash must last at least 10 years. The lifetime is influenced by the rate of lease updates (more updates require more flash writes), the number of extents, and the size of the flash log (smaller logs wrap around faster and lead to higher utilization, which in turn reduces cleaning efficiency and requires even more flash writes).

Figure 1(d) shows the maximal permissible average daily rate of lease updates, as a function of the size of the flash log, given a minimal target lifetime of 10 years and an assumed average extent size of 256KB. With a flash log size of 64MB (as used in our simulations), the average number of lease updates can be as high as 880,940 per day to ensure a minimal lifetime of 10 years. Larger flash logs increase the rate of updates roughly proportionally. To put this number in perspective, it would allow indefinite continuous writes of new leased extents with an average size of 6MB or more into the disk at full sequential bandwidth. Of course, in practice writes are not all sequential, not all writes commit new leases, and the disk is not continuously utilized with writes.

7.3 Performance on modern disks

Because DiskSim supports only SCSI disks and we did not have access to a validated model of a recent SCSI disk, our evaluation has been limited to a somewhat obsolete disk. In this section, we consider the performance of storage leases on modern disks, through analysis and extrapolation.

First, we consider a 4TB hybrid disk. To achieve similarly low overhead as reported above, we would require 2GB of NAND flash memory and 5MB of DRAM for leases and the associated metadata. This is not a problem—even the much smaller current 750GB hybrid disks already have up to 8GB of flash and 32MB of DRAM. Because the expected number of flash reads is still small (<3 reads/lookup) and each flash read takes about 0.025 ms, the lease lookups would still be insignificant compared to the disk latencies. Updating lease information in the B+Tree still requires a single page write. However, writing a complete snapshot of the node index would approach 2MB, which has high latency. However, it is possible to write only snapshot diffs to the flash, and visit the chain of delta snapshots during recovery.

Second, we considered solid state drives (SSD). While a design of storage leases for SSDs remains future work, we believe that a very efficient implementation is possible by integrating lease support with the flash

translation layer (FTL). For instance, lease values could be stored in the extra 64 bytes provided by NAND flash memory, which is normally used for error correcting code and other FLT metadata [21].

8 Conclusion and future work

We introduced storage leases, a simple yet powerful primitive to protect the integrity of on-line data. Storage leases can be implemented in the firmware of commodity storage devices, bringing enterprise-grade integrity protection to consumer storage. Leases ensure the integrity of stored data independent of bugs or vulnerabilities outside the device hardware and software. Moreover, enabled devices can issue signed statements about data committed under a lease, allowing end-to-end data integrity checks. A simulation-based evaluation indicates that leases can be implemented with near zero performance overhead, using a small amount of flash memory (e.g., 2GB flash for a 4TB disk).

In future work, we intend to generalize storage leases to provide additional interesting capabilities. For instance, by supporting read restrictions, one can obtain confidentiality guarantees like time capsules (confidential until a chosen date). By supporting read and write restrictions that begin in the future also, one can obtain additionally data expiration (confidential beyond a chosen date), or snapshots (integrity beyond a chosen date).

References

[1] Self-Encrypting Hard Disk Drives in the Data Center. Tech. Rep. TP583, Seagate Inc., (2007).

[2] Agrawal, D., Ganesan, D., Sitaraman, R., Diao, Y., and Singh, S. Lazy-Adaptive Tree: An Optimized Index Structure For Flash Devices. *Proc. of the VLDB Endowment* (2009).

[3] Bairavasundaram, L. N., Sundararaman, S., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. Tolerating File-System Mistakes with EnvyFS. In *Proc. of USENIX ATC* (2009).

[4] Bucy, J. S., Schindler, J., Schlosser, S. W., Ganger, G. R., and Contributors. The DiskSim Simulation Environment Version 4.0 Reference Manual. Tech. Rep. CMU-PDL-08-101, CMU PDL, (2008).

[5] Chervenak, A., Vellanki, V., and Kurmas, Z. Protecting File Systems: A Survey of Backup Techniques. In *Joint NASA and IEEE Mass Storage Conference* (1998).

[6] Cox, L. P., Murray, C. D., and Noble, B. D. Pastiche: Making Backup Cheap and Easy. *SIGOPS Operating Systems Review 36*, SI (2002).

[7] e-TimeStamp: A Web-Based Security Service for Data Authentication. `http://www.digistamp.com`.

[8] dtrace. `http://hub.opensolaris.org/bin/view/Community+Group+dtrace/`.

[9] Garret, R. A Time Machine Time Bomb. `http://rondam.blogspot.com/2009/09/time-machine-time-bomb.html`.

[10] Hayakawa, M. WORM Storage on Magnetic Disks Using SnapLock Compliance and SnapLock Enterprise. Tech. Rep. TR-3263, Network Appliance, (2007).

[11] Hitz, D., Lau, J., and Malcolm, M. File System Design for an NFS File Server Appliance. In *Proc. of the USENIX Winter Technical Conference* (1994).

[12] Kincaid, J. T-Mobile Sidekick Disaster: Danger's Servers Crashed, And They Don't Have A Backup. `http://goo.gl/nPByo`.

[13] McCoy, K. *VMS File System Internals*. Digital Press, Newton, MA, USA, (1990).

[14] Patterson, D. A., Gibson, G., and Katz, R. H. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proc. of SIGMOD* (1988).

[15] Post, A. *Autonomous storage management for low-end computing environments.* PhD thesis, Rice University, 2010.

[16] Quinlan, S., and Dorward, S. Venti: A New Approach to Archival Data Storage. In *Proc. of USENIX FAST* (2002).

[17] Riedel, E., Faloutsos, C., Gibson, G., and Nagle, D. Active Disks for Large-Scale Data Processing. *IEEE Computer 34*, 6 (2001).

[18] SafeXcel-1741. `http://www2.safenet-inc.com/Library/EMB/SafeNet_Product_Brief_SafeXcel_1741.pdf`.

[19] Santry, D. S., Feeley, M. J., Hutchinson, N. C., Veitch, A. C., Carton, R. W., and Ofir, J. Deciding When to Forget in the Elephant File System. In *Proc. of SOSP* (1999).

[20] Seagate®Momentus XT®. `http://www.seagate.com/docs/pdf/datasheet/disc/ds_momentus_xt.pdf`.

[21] Micron SLC NAND Flash Charactreistics. `http://download.micron.com/pdf/datasheets/flash/nand/2gb_nand_m29b.pdf`.

[22] Strunk, J. D., Goodson, G. R., Scheinholtz, M. L., Soules, C. A. N., and Ganger, G. R. Self-Securing Storage: Protecting Data in Compromised Systems. In *Proc. of OSDI* (2000).

[23] Time Machine. `http://www.apple.com/support/timemachine/`.

[24] Tran, D. N., Chiang, F., and Li, J. Friendstore: Cooperative Online Backup Using Trusted Nodes. In *Proc. of SocialNet* (2008).

[25] Vrable, M., Savage, S., and Voelker, G. M. Cumulus: Filesystem Backup to the Cloud. In *Proc. of USENIX FAST* (2009).

[26] What Is Volume Shadow Copy Service. `http://technet.microsoft.com/en-us/library/cc757854(WS.10).aspx`.

[27] Windows Backup and Restore. `http://windows.microsoft.com/en-US/windows7/products/features/backup-and-restore`.

[28] Solaris ZFS. `http://www.oracle.com/us/products/servers-storage/solaris/034779.pdf`.