International Conference on Dependable Systems and Networks (DSN-2007), Edinburgh, Scotland, June 2007.

Improving recoverability in multi-tier storage systems

Marcos K. Aguilera[*], Kimberly Keeton[*], Arif Merchant[*],
Kiran-Kumar Muniswamy-Reddy[+], Mustafa Uysal[*]
[*]HP Labs, Palo Alto, CA, USA and [+]Harvard University, Cambridge, MA, USA

Abstract

Enterprise storage systems typically contain multiple storage tiers, each having its own performance, reliability, and recoverability. The primary motivation for this multi-tier organization is cost, as storage tier costs vary considerably. In this paper, we describe a file system called TierFS that stores files at multiple storage tiers while providing high recoverability at all tiers. To achieve this goal, TierFS uses several novel techniques that leverage coupling between multiple tiers to reduce data loss, take consistent snapshots across tiers, provide continuous data protection, and improve recovery time. We evaluate TierFS with analytical models, showing that TierFS can provide better recoverability than a conventional design of similar cost.

1 Introduction

The explosive growth of digital data prompts us to rethink the way we design and use storage systems to keep enormous volumes of data at a reasonable cost. Current enterprise systems comprise multiple online storage *tiers*, each with its own features and cost: high-end tiers use expensive, highly reliable disk arrays and typically provide remote mirrors, snapshots, and daily backups; middle storage tiers may use cheaper mid-range disk arrays and might only provide snapshots and daily backups; and low-end tiers might use inexpensive disk appliances protected by weekly backups to decrease operational cost. Administrators place data on each of these tiers according to the business value and access characteristics of the data. Thus, business-critical data is placed on the highest tier, while less-important information is placed on lower tiers. However, when the administrator is constrained by limited space in the highest, most expensive tiers, data may be placed in a lower tier than is desirable.

Different storage tiers provide different levels of performance, reliability, and recoverability. Recoverability is the ability to recover data when there are problems, such as operator mistakes, disk crashes, site failures, or disasters, which are likely to occur within the lifetime of a system. Better recoverability can be achieved through remote mirroring, snapshots, and frequent backups, which all come at a cost.

In this paper, our goal is to improve the recoverability-cost trade-off of storage systems, that is, to provide better recoverability at a lower cost. We propose a multi-tier file system called *TierFS* that employs a *recoverability log* to boost the recoverability of lower tiers by using the highest tier. The recoverability log stores an extra copy of updates destined for a lower tier until the lower tier is protected by a backup. In a conventional multi-tier storage system, data stored in a lower tier typically suffers from a considerable loss of recent updates if a recovery from backup is needed (say, due to a failure), since the backups may be infrequent. The recoverability log eliminates the backup-window data loss, because recent updates can be recovered from the log copy on the highest tier. By logging updates to all tiers and keeping them around, the recoverability log can also provide *continuous data protection*, which offers users a fine-grained set of recovery points—finer than the backup frequency of each tier. Finally, the recoverability log enables TierFS to take *consistent snapshots of all tiers* (to produce a consistent backup of the file system) without blocking file system writes. TierFS also provides a new mechanism to *improve recovery time*, by applying the recovery log in the background after restoring a backup copy and by using the recovery log to facilitate coordinated recovery using both a local, fast, but less up-to-date backup copy and a relatively slow, but more up-to-date remote mirror. To the best of our knowledge, TierFS is the first system that couples multiple storage tiers to improve recoverability.

We evaluate TierFS with analytical models to determine the overheads and benefits of using the recoverability log and the parallel recovery technique. We find that TierFS can provide a system with better recoverability than a conventional system of similar cost.

The remainder of the paper is organized as follows. Section 2 provides background on storage recoverability and multi-tier storage systems. Section 3 outlines the design of TierFS and introduces several mechanisms for coupling tiers to enhance recoverability. Section 4 describes our evaluation methodology, and Section 5 presents results. We describe related work in Section 6, and conclude in Section 7.

2 Background

In this section, we first explain storage recoverability, and provide background on the data protection techniques used to provide recoverability. We then explain how these techniques are used in tiered storage systems.

2.1 Recoverability metrics and techniques

Recoverability is the ability to recover from problems, like user mistakes, disk crashes, or disasters. Recoverability is measured using two metrics: *recent data loss* and *recovery time*. *Recent data loss* measures how much recent data (measured in time) is lost when recovery is performed. For example, if a disk gets backed up every day and the disk fails, then recovery can result in loss of data of up to one day. The best value of this metric is zero, which means that all updates are recovered. *Recovery time* measures how long the recovery process takes until data is available again.

Storage systems are designed to provide recoverability of the *primary copy* of the application's evolving dataset, using techniques like backups, snapshots, continuous data protection (CDP) and remote mirroring. A storage system will typically have multiple *point-in-time copies* of the same dataset, which capture the state of the data at different points in its evolution. Techniques are often configured to create point-in-time copies with varying frequency, which is referred to as *recovery point granularity*. Snapshots and backups create discrete recovery points, while CDP creates continuous recovery points. Recovery point granularity determines the ability to restore to an earlier point-in-time copy, often referred to as *time-travel recoverability*.

Backups are used to make relatively infrequent point-in-time copies by copying the full dataset (*full backups*) or the updates since the last backup (*incremental backups*). Backups have been traditionally stored on tape; more recently, decreasing disk costs have resulted in disk-based backup systems, as well. To protect against site disasters, backup copies must be transported to an offsite vault.

Snapshots capture the state of storage at a given point in time, typically in a very space efficient way by using copy-on-write and similar techniques. Like backups, snapshots are useful to undo user mistakes and software errors. However, snapshots are less useful to recover from disk failures or disasters because they share data with the primary copy; so if the primary copy is lost, the snapshot is affected, too. Snapshots are also useful to produce consistent online backups: the system first takes a snapshot and then derives the backup from the snapshot.

Continuous data protection (CDP) creates a continuous set of recovery points, rather than just one every hour or day, to permit finer grained time-travel recovery. A simple way to provide CDP is to create a copy of a file as it is modified (e.g., using VMS file versioning or VersionFS [10]). A more space-efficient way to provide CDP is to use copy-on-write (e.g., Elephant [12] and CVFS [14]): when a file is overwritten, the new data blocks are written at a new location, but the old and new versions share all unmodified blocks.

Storage devices may also have the capability to maintain *remote mirror* copies at multiple geographic locations, typically at one or more remote sites. The geographic dispersion capability allows a storage tier to survive larger failure scopes (e.g., a site disaster). *Synchronous mirroring* ensures that both copies are always in sync, while with *asynchronous mirroring*, the remote copy may lag behind by a few seconds. Remote mirroring is provided by high-end and some mid-range disk arrays.

2.2 Tiers of storage

Enterprises keep a huge amount of online data, ranging from critical data required to run the business to emails kept according to regulations to derived or historical information for trend analysis. A variety of online storage alternatives exist for storing these vast quantities of data, ranging from high-end disk arrays with snapshot and remote mirroring capabilities, to small disk appliances used to aggregate a few disks together, which offer only limited data protection (e.g., RAID). These alternatives differ in cost by as much as an order of magnitude. As a result, enterprises use high-end disk arrays very sparingly, only to keep the most important data. They set up storage systems comprising multiple tiers of storage, and allocate their online datasets to these storage tiers based on the data's business value and access characteristics.

A sample multi-tier storage system might have three tiers. At the highest (most expensive) tier, there are high-end disk arrays with remote mirroring and frequent point-in-time snapshot capabilities. At the middle tier, there are mid-range disk arrays with less frequent point-in-time snapshots and without remote mirroring. And at the lowest tier, there are inexpensive disks managed by software or firmware RAID. Lower tiers with large capacity and/or small update rates may be backed up less frequently than higher tiers, to reduce operational costs.

Leveraging distinct capabilities of multiple storage tiers dates back to *hierarchical storage systems (HSM)*, which coupled disk systems with tape storage to transparently increase the apparent storage capacity. For example, HPSS [5] automatically moves infrequently-accessed files from a higher tier (disk) to a lower tier (tape) or another archival device, and then automatically moves data back to disk when access is requested. HPSS allows files be read or written only at the highest tier, and so if a file has been migrated to a lower tier, it needs to be migrated back before it is accessed. This restriction exists for historical reasons, because early HSM systems treated lower tiers as offline devices.

More recently, the VxFS file system introduced greater flexibility in dealing with multiple storage tiers [6]. VxFS allows users or administrators to define placement rules to

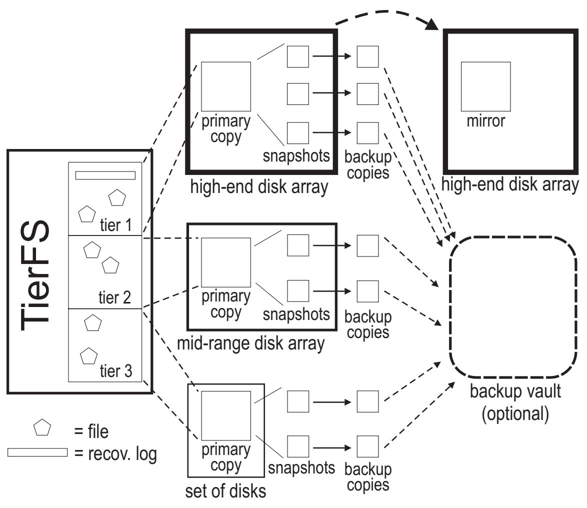


Figure 1. *TierFS uses multiple storage tiers to provide high recoverability at low relative cost. It leverages the highest tier—one with good recoverability—to boost the recoverability of lower tiers.*

indicate in which tier new files are created, and migration rules to indicate when a file in one tier should be moved to another tier. These policies can be based on a file's name, extension, creator, timestamps, and/or frequency of access.

3 TierFS design

TierFS, illustrated in Figure 1, couples multiple storage tiers to provide a high level of recoverability across all storage tiers. TierFS uses the tier with the highest recoverability and performance to store a *recoverability log*, which contains a copy of updates destined for a lower tier until the lower tier is protected by a backup. TierFS uses the recoverability log, along with the snapshot and backup facilities of the individual tiers, to provide improved recent data loss and faster recovery time for files stored in lower tiers, consistent multi-tier file system backups, and continuous data protection.

Each file or directory in TierFS has a *home tier*, where the contents of the file are stored. The home of any file can be any tier, and users are not required to know the home tier of a file to access it (i.e., there is a single namespace). TierFS can transparently migrate a file to a different tier if desired, based on user or administrator control. Similar to VxFS [6], in TierFS a file's initial home tier and any subsequent migration can be controlled by simple rules based on file attributes such as owner, extension, and modification and access time. The details of these mapping and migration rules are outside the scope of this paper.

The improved recoverability provided by TierFS enables allocation of files to tiers to be based on considerations other than recoverability, such as cost and performance. For example, a cost-driven design might choose to place no files at the highest tier, which reduces its size to what is needed by the internal logs of TierFS. Another design might use the highest tier (which is typically the fastest) to store files that are frequently used in an application's critical path to boost performance. However, these files are not necessarily the ones that need the highest recoverability.

The following sections explain several techniques that TierFS uses to provide good recoverability for lower tiers.

3.1 Improving recent data loss

TierFS uses the highest tier to maintain a *recoverability log*, which contains an extra copy of updates destined for a lower tier until the lower tier is protected by a backup. This technique, similar to file system journaling, improves the recent data loss characteristics of lower tiers. When we recover a lower tier from its most recent backup (or snapshot) copy, we can replay the most recent entries in the recovery log that are not reflected in the backup, in order to prevent data loss. For this technique to work, we must ensure that log entries are garbage collected only after (1) the updates have been applied to the appropriate tier, and (2) since then, the tier has been backed up.

If protection against site disaster is desired, TierFS further delays garbage collection of the recovery log until the backup has arrived at a remote vault, which can take a long time. If the highest tier employs remote mirroring (as shown in Figure 1), TierFS can take periodic snapshots of the mirrored log at the remote site, as an optimization. We can then relax the above requirement and garbage collect the recovery log when it has been snapshotted at the remote mirror. The remote site serves as protection against disasters, in lieu of the vault. The reason for the snapshot is to keep data at the remote site after the log gets garbage collected. Only the log (not the entire highest tier) needs to be snapshotted, which is achieved by keeping the log in its own logical volume. Once the backup arrives at the vault, the log snapshot at the remote site can be deleted.

We use a *block-level* log, which refers to operations on physical blocks rather than files, to make the log *idempotent*, so that it can be replayed without destroying data. Idempotency is important because TierFS does not always have perfect control of the copying mechanisms in each tier. For example, if a disk array at a lower tier keeps an asynchronous remote mirror, the mirror may lag behind by a bounded amount of time that is unknown by TierFS. If TierFS had to recover using this mirror and the log were not idempotent, it would have to replay the log from the first entry not yet applied to the mirror, which is unknown. With an idempotent log, TierFS can replay the log conservatively, by starting at the point corresponding to the largest lag allowed by the disk array.

The benefit of the recoverability log is that the recent data loss of all tiers becomes equal to the recent data loss of the highest tier. However, there are some overheads. First, each update is written twice: once to the log and then again to the appropriate tier. TierFS partly offsets this overhead by performing both updates in parallel, where the log update is synchronous and the other update is asynchronous. Second, the log takes up space. In theory, the log can become arbitrarily large before it is garbage collected; however, our evaluations show that the log size under real workloads is reasonable.

3.2 Improving recovery time

We now turn our attention to recovery from failures. TierFS uses two techniques to improve the time to recover a tier (e.g., in the event of a disk array failure).

The first technique reduces the overhead of applying the updates in the recoverability log. Recall that recovery of a tier in TierFS comprises two steps: populating the tier with the latest available backup and applying the changes in the recoverability log that are not reflected in the backup. With TierFS, only the first step contributes to recovery time: after the first step, the tier is ready for use in TierFS (without any recent data loss), because TierFS remaps reads from that tier to the recovery log for the blocks that are in the log. Therefore, replaying the recovery log can be done online as a background task.

The second technique applies to a tier that has a remote mirror as well as local backups (e.g., the highest tier). Recovery from a remote mirror requires copying the entire remote volume into the local volume. Because the connection to the remote site is provisioned to only carry recent data updates, its bandwidth is typically small compared to that of an archival device (such as disk or tape), sometimes by orders of magnitude. Thus, transferring an entire volume from the remote site to the local site can take a long time. Recovering from a backup is faster, but the backup has less recent data than the remote mirror copy. Thus, there is a trade-off between recovery time and recent data loss. TierFS breaks this trade-off by recovering the tier from backup and then only transferring the most recent updates from the remote site. TierFS uses the recoverability log to determine the blocks that are changed but might not be reflected in the latest backup. To do so, TierFS puts markers in the recoverability log to indicate when backups are made.

3.3 Consistent full-system backups

The recoverability log can also enable consistent full-system backups of all tiers. While tiers are backed up at different rates based on cost considerations, there is also a desire to periodically perform full-system backups of all tiers, for archival purposes. The challenge is to ensure that the backup is *consistent*.

In a traditional, single-tier file system, backup consistency comes from snapshots: a snapshot preserves the state of the whole file system at a single time, which is then used to populate the backup, even as the file system is changing. In TierFS, creating consistent backups poses an additional challenge: how to synchronize snapshots between tiers when each tier might implement its own snapshots. The problem is that snapshots of different tiers could be taken at slightly different times, even if they are requested simultaneously, and a backup made from such snapshots might be inconsistent.

If the recoverability log is extended to log updates to all tiers, rather than just the lower tiers, it can help solve the consistent backup problem. Before taking snapshots, TierFS pauses the block-level writes to each tier at a point when they form a consistent file system image (e.g., there are no unattached inodes). TierFS can continue to accept file system writes without blocking, though, because they are logged in the recoverability log, even though the block-level writes to each tier are paused. Once writes are paused, TierFS takes a snapshot of each tier, without fear of inconsistency, even if the snapshots are taken at slightly different times. TierFS uses standard techniques like copy-on-write to implement snapshot capabilities for tiers that do not provide them natively.

3.4 Providing CDP

TierFS uses its recoverability log for yet another purpose: to provide continuous data protection (CDP) by using the recoverability log and appropriate prior point-in-time copies of each tier. The basic idea is to store updates to all tiers in the recoverability log, and to retain and protect them rather than garbage collecting them as soon as a backup and/or offsite copy of the data is created, as described in Section 3.1. To ensure log entries are available for CDP, we modify the garbage collection mechanism of Section 3.1 in the obvious way: a log entry is discarded only after the lower tier has been updated, and both the log and the lower tier have been backed up. In addition, if protection against site disasters is desired, both the log and lower tier updates must have arrived at the vault or have been remotely snapshotted.

TierFS's approach to CDP provides two advantages over previous approaches. First, it makes it efficient to move the voluminous amount of CDP information to backup tiers, by simply copying the sequential log. Second, because log entries are ordered by time, it is easier to manage CDP information stored online or in a backup; for example, one can delete CDP data for uninteresting periods by simply erasing the appropriate log segments (if stored online) or by getting rid of the appropriate backup archive (if stored in a backup). As a result, it becomes feasible to preserve CDP information for archival purposes.

However, using logs to provide CDP poses a difficulty: if a block has not been modified for a long time then its last log entry will be far in the past. This scenario creates a problem for garbage collection and recovery, which we call the *10-year log problem*: if a file has not been modified for 10 years and a user overwrites it, then going back just a few seconds before the overwrite requires a log that is 10 years long! This problem could be avoided by keeping an undo log in addition to the recoverability log of TierFS, but we would prefer to maintain just one log. Furthermore, undo logs have their own drawbacks: they are more expensive to maintain, as writing new entries to the undo log requires reading of the old information, and they are inefficient for restoring to points far in the past, as they require all intervening operations to be undone.

TierFS solves the 10-year log problem by using backup or other point-in-time copies of each tier, which provide a starting point from which to replay the log. Thus, to recover the file system to some time t we first use a backup or other point-in-time copy to recover the system to the latest available time t_0 before time t. We then replay the log entries from t_0 to t. In other words, the log entries between two backup copies enable CDP in that time range.

3.5 Availability and performance of tiers

Availability refers to the ability to mask failures, that is, to continue normal operation as if failures never occurred. While TierFS improves the recoverability of all tiers, each tier keeps its original availability. Thus, in the period between a failure and its recovery, some of the tiers may be available, while others are not. Because TierFS places all information needed to access a file in the same tier (including its inode and data blocks), files in available tiers will continue to be available. For other files, TierFS will block accesses until they have been recovered. This provides a form of graceful degradation, similar to the DGRAID [13] system. TierFS also places the root directory and other top-level directories at the highest tier as an additional precaution to make the top level of the name space accessible.

TierFS may improve performance of a tier when there are bursts of writes, since writes are acknowledged as soon as they are placed in the log at the highest tier. TierFS's total write bandwidth is limited by the maximum write bandwidth of the highest tier (e.g., high-end disk array bandwidth or remote mirroring interconnect bandwidth). But note that TierFS writes updates to the log sequentially, so it uses write bandwidth at the highest tier efficiently. In addition, instead of a single log, TierFS can keep several logs—one for each tier—which are stored on separate volumes and/or devices at the highest tier for greater bandwidth. As for read bandwidth, TierFS will typically read data from its home tier, unless a more up-to-date copy exists in the recoverability log. Thus, TierFS tends to preserve the read bandwidth of each tier.

Category	Metric	Description
Recoverability	recent data loss	data loss on recovery
	recovery time	duration of recovery
Operational requirements	storage bandwidth	under normal operation
	tape bandwidth	under normal operation
	remote bandwidth	under normal operation
	storage capacity	data stored after a month
	tape capacity	data on tapes after a month
	remote capacity	remote site data after a month
	vault capacity	data at vault after a month

Table 1. *Metrics for evaluation.*

4 Evaluation

We evaluate TierFS's recoverability, bandwidth requirements and capacity requirements under several workloads. We compare TierFS against two alternative approaches: a system that stores all its data on a single high-end storage tier, and a conventional multi-tier file system that does not employ our mechanisms to improve recoverability. We provide the details of our evaluation methodology (Section 4.1), the metrics and the faults we consider (Section 4.2), the alternative approaches (Section 4.3), and our experimental setup (Section 4.4).

4.1 Methodology

We evaluate the recoverability of TierFS and the alternatives we consider through analytical models. We extend the dependability models of Keeton and Merchant [8] to derive the data loss upon recovery for multi-tier systems. These extensions model the behavior and overheads of the techniques that TierFS uses to improve recoverability. We combine these models with a recovery graph representation [7] of the schedule of recovery operations to compute the recovery time under several failure scenarios and workloads, as well as normal mode capacity and bandwidth utilization. We use this model framework to evaluate the recovery time, data loss upon recovery, and capacity and bandwidth utilizations for all of the alternative approaches we consider.

4.2 Metrics and failure types

Our evaluation consists of two parts. The first part measures recoverability under different failure scenarios, and the second part measures operational requirements under normal system operation. Table 1 presents the metrics we use for our evaluation.

The recovery metrics—recent data loss and recovery time—are described in Section 2. We evaluate recovery under three different types of failures. A *storage device failure* causes complete data loss at one of the tiers. We assume that the highest tier has sufficient redundancy that it is unlikely to fail. A *site failure* causes complete data loss at all tiers. A *human error* requires a recovery from a past point in time, rather than the most recent version.

The meaning of *recent data loss* for a recovery from hu-

Parameter	Tier 1	Tier 2	Tier 3
Online Bandwidth	13 GB/s	675 MB/s	256 MB/s
Remote Mirror	Synchronous	None	None
Snapshot frequency	Every 4 hours	Daily	None
Snapshots retained	Last 12	Last 2	N/A
Backup frequency (full)	Daily	Weekly	Weekly
Backup frequency (incr)	None	Daily	None
Backups retained	Last 28 days	Last 28 days	Last 28 days
Shipment to remote vault	Weekly	Weekly	Weekly
Backups in vault	Last 52 weeks	Last 52 weeks	Last 52 weeks
CDP backups retained†	Last 28 days	Last 28 days	Last 28 days
CDP backups in vault†	Last 52 weeks	Last 52 weeks	Last 52 weeks

† for TierFS+CDP system only

Table 2. *Data protection techniques for the three storage tiers in our evaluation.*

Parameter	Harvard trace	Cello trace
Dates of trace	9-11/2001	9/2002
Capacity (GB)	450	1360
Avg. Access Rate (KB/s)	161	1028
Avg. Update Rate (KB/s)	103	799
Unique Update Rate (KB/s)	60	317
Peak:Avg. Update Ratio	1000	10
Peak:Avg. Access Ratio	1000	10

Table 3. *Summary of the workload characteristics used in our evaluation.*

System	Tier-2 failure	Tier-3 failure	Site failure
Single Tier	—	—	0
Basic multi-tier	48 hrs	192 hrs	384 hrs
TierFS	0	0	0
TierFS+CDP	0	0	0

Table 4. *Recent data loss for the Harvard workload under various failures.*

man error is the difference between the time to which the user *wants* to recover the system and the time to which recovery is possible.

We measure operational requirements through the resources utilized by each system. We consider bandwidth and capacity at each storage tier, as well as for the remote site. We also consider capacity used at the vault.

4.3 Systems

We evaluate four different systems:

- *Single-tier*: This system stores all data in a single high-end storage tier. This system is our benchmark system; its drawback is its high cost.
- *Basic multi-tier*: This system stores data in multiple tiers, but does not couple tiers together to improve recoverability.
- *TierFS*: Our TierFS system couples multiple tiers to improve recoverability. The recoverability log contains updates to all but the highest tier, and the log is garbage collected aggressively, as described in Section 3.1.
- *TierFS+CDP*: In this TierFS variant, the recoverability log contains and preserves updates to all tiers, to enable CDP, as described in Section 3.4.

4.4 Experimental setup

We use a three-tier storage system for our evaluation, as shown in Table 2. Tier 1, the highest tier in our setup, consists of two high-end disk arrays that are remote mirrors of each other and connected using four OC3 links (155 Mbps each). This tier uses remote mirroring, snapshots, local backups, and tape vaults for data protection. Tier 2 consists of mid-range disk arrays that provide snapshots, local backups, and tape vaults; but no remote mirrors for data protection. Tier 3 consists of a disk appliance and uses only infrequent backups and tape vaults. All three tiers employ RAID-protected online storage. Additionally, all three tiers share a single remote vault, which is located at the same site as tier 1's remote mirror.

We use two workgroup file server workloads for our evaluation representing research and software development workloads in an industrial research lab (Cello workload) and a university (Harvard workload [3]).The Cello trace contains I/O activity from a departmental server for a total of 254 users and the Harvard trace contains NFS activity of a Network Appliance Filer for a total of 416 users. Table 3 summarizes the characteristics of these two workloads.

The multi-tier storage system is used differently by each system we evaluate. The single-tier system places all of its data onto the highest tier. The remaining systems use all three tiers to store file system data. They place 10% of all data from a given file system on tier 1, the next 30% onto tier 2, and the remaining 60% of the data on tier 3. This allocation represents the importance and the expected use of the data within the file system.

5 Experimental results

In this section, we present our results. First we evaluate all systems for recoverability; both to recover to the most recent version of the data and to recover to a past version, i.e., time travel. We then look at the bandwidth and capacity requirements of TierFS and the other alternatives. We conclude this section with a discussion.

5.1 Recoverability

TierFS is designed to have zero recent data loss when the system is recovered to its latest version. Table 4 shows the data loss for failures of tiers 2 and 3, and a site failure for the Harvard workload; results for the Cello workload are very similar. As can be seen, Single-tier, TierFS, and TierFS+CDP have zero loss, whereas the loss for the Basic multi-tier system is considerable.

Figure 2 shows the recovery time for various systems on the y-axis, under three types of failures, represented by the x-axis. As can be seen, in both the Harvard and Cello workloads, recovery time for TierFS and TierFS+CDP is essentially the same as for a Basic multi-tier file system, because replaying the recoverability log occurs in the background. For site failures, Single-tier performs much better than all other systems, because with the other systems, recovery of tiers 2 and 3 needs to wait for the arrival of backup tapes from the vault (which takes 24 hours in our setup). Single-tier can recover from the mirror over the network.

5.2 Time travel recoverability

In this section, we explore the time travel recoverability of the various systems. Certain classes of failures, including human errors, software bugs, and virus infections, require the ability to recover the system to a point in the past. This time travel recoverability is possible in systems that support point-in-time copies and continuous data protection.

Figure 3(a) quantifies the recent data loss that the systems incur when recovering to a recovery target in the past for the Harvard workload. We observed similar results for the Cello workload. Where multiple tiers exhibit different recent data loss values, we present the maximum value.

The Single-tier system experiences different amounts of recent data loss, depending on how far in the past the recovery target is. Snapshots can recover the recent past with minimal data loss (up to four hours, the granularity of the snapshots); daily backups can recover the moderate past, with data loss of up to a day; and the weekly vault copies can recover the distant past, with up to a week of data loss. Both the Basic multi-tier system and TierFS provide a uniform recent data loss of up to a week, due to the weekly backup policy for tier 3. TierFS+CDP provides complete recoverability, with zero data loss. We note that none of the systems can recover to a point older than the retention period of the vault (one year).

Figure 3(b) quantifies the time to recover to a point in the past for the Harvard workload. We observed similar results for the Cello workload. The Single-tier system achieves very fast recovery for a snapshot, but takes several hours to restore a backup copy, and over a day to restore a vaulted copy (due to the delay to retrieve the vaulted tapes). The remaining systems (Basic multi-tier, TierFS, and TierFS+CDP) take several hours to recover recent and moderately recent targets; this recovery time is limited by the need to recover a tape backup for tier 3. As with the Single-tier system, recovery of the distant past takes over a day.

5.3 Bandwidth usage

Figure 4 shows the peak operational bandwidth of TierFS compared to the alternatives. We focus on the tier 1 bandwidth, since it is the most expensive. First, we observe that the Single-tier system has a substantially higher peak bandwidth at tier 1 than any other alternative. The Basic multi-tier system has the lowest peak bandwidth at tier 1, as we expect. The peak bandwidth of the TierFS and TierFS+CDP systems ranges between a quarter and a third of the Single-tier peak bandwidth. We note that these alternatives have similar recoverabilities. The bandwidth requirements of TierFS and TierFS+CDP are significantly higher than that of the Basic multi-tier system, due to the bandwidth requirements of maintaining the recoverability log.

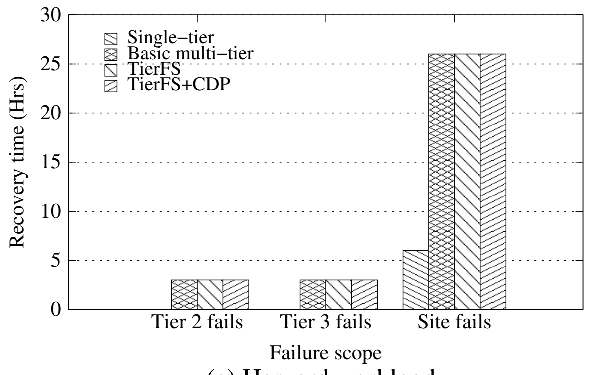
5.4 Capacity usage

We now consider the storage space used by TierFS and alternatives after one month of normal system operation, as shown in Figure 5. The Single-tier system has the highest capacity requirement in tier 1, because it stores all the data there. In addition, it requires a large number of point-in-time copies in order to preserve recoverability. The Basic multi-tier design, on the other hand, uses the lowest capacity in tier 1, but the total capacity over all tiers is similar to the capacity usage of the Single-tier system. TierFS and TierFS+CDP require tier 1 storage capacity that is intermediate between the Single-tier and the Basic multi-tier systems. The precise amount of storage required depends on the workload, but we find that it is much smaller than the Single tier system requires, in most cases. On the other hand, as seen in the capacity requirements for the Cello workload, the capacity requirements at the remote site can be considerable for TierFS+CDP, approaching that of the Single tier system. Note, however, that the TierFS+CDP system offers lower time travel data loss than the Single tier system, particularly if the desired recovery target is well in the past.

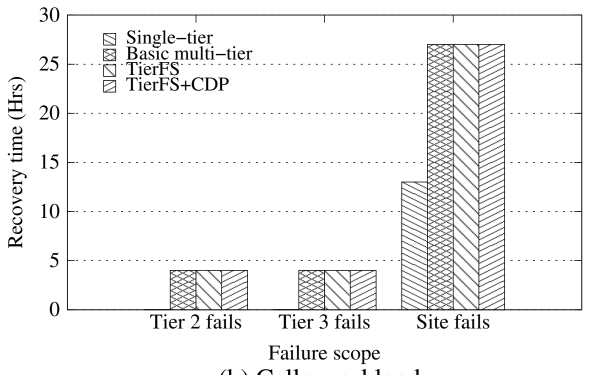
The archival capacity requirements of the four alternatives are shown in Figure 6. The Single-tier system has a high capacity requirement at the tape library because it keeps all its data in tier 1, which gets backed up frequently to maintain recoverability of the data. The other alternatives have similar (and lower) tape library requirements. The Single-tier, Basic multi-tier and TierFS systems have similar vault capacity requirements; however, the TierFS+CDP system has a slightly higher vaulting requirement since it vaults CDP data.

5.5 Discussion

We have compared four systems, the Single-tier system that stores all the data on a single, highly recoverable storage tier; the Basic multi-tier system that distributes data over disk arrays with varying levels of recoverability, without an attempt to give the data in low-recoverability tiers additional protection; TierFS, which stores data in multiple disk arrays, but adds mechanisms to protect the data stored in low-recoverability tiers; and, finally, TierFS+CDP, a system that additionally provides the ability to recover to any point in time, in order to recover from software and human errors. We find that the Single-tier system can provide high recoverability, but the cost of such a system is very high,

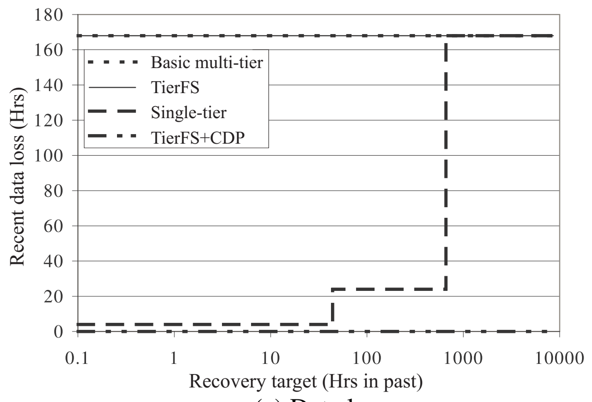


(a) Harvard workload

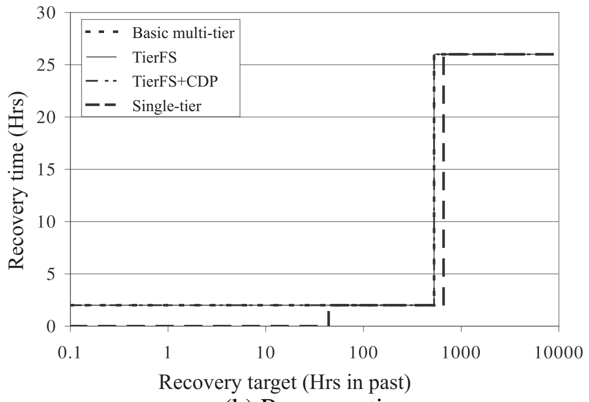


(b) Cello workload

Figure 2. *Time to recover to the latest version.*

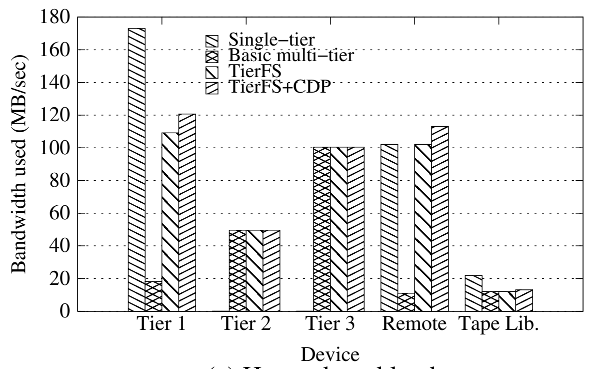


(a) Data loss

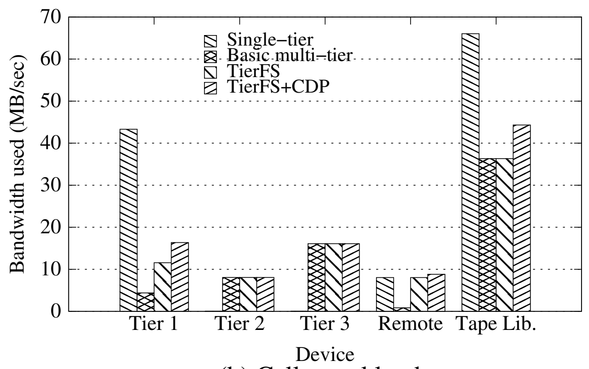


(b) Recovery time

Figure 3. *Data loss and recovery time to travel to past version for the Harvard workload. Results for the Cello workload are similar.*

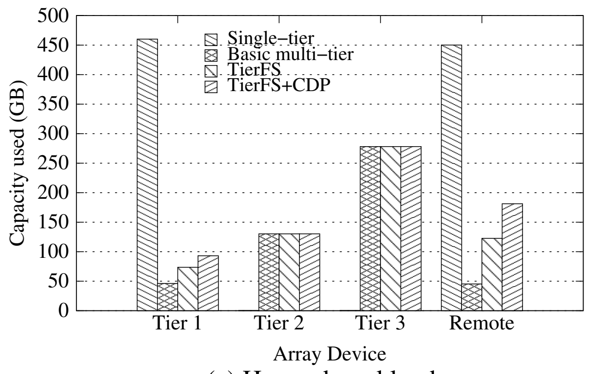


(a) Harvard workload

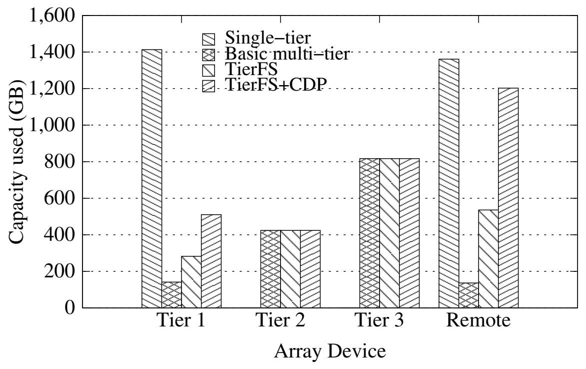


(b) Cello workload

Figure 4. *Peak bandwidth used under normal system operation.*

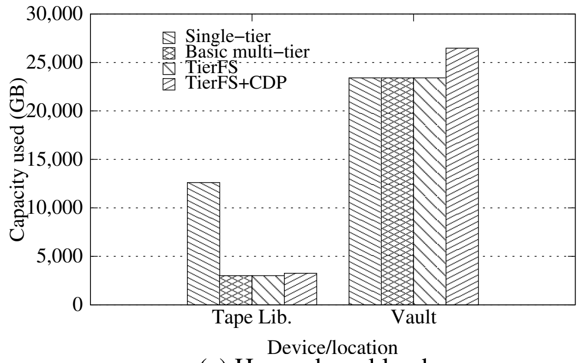


(a) Harvard workload

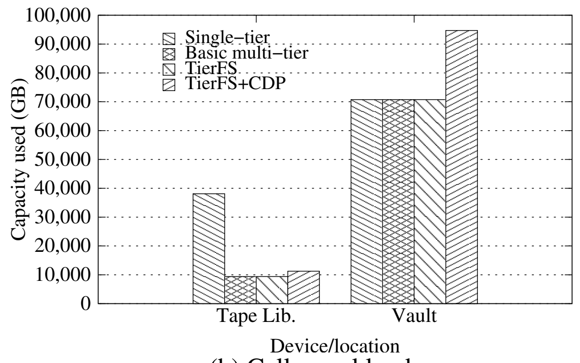


(b) Cello workload

Figure 5. *Capacity usage after one month of normal system operation.*



(a) Harvard workload



(b) Cello workload

Figure 6. *Archival capacity usage after one year of normal system operation.*

because it stores all its data on an expensive medium. The Basic multi-tier system is much less expensive, but it is subject to substantial data loss in case of failures in the lower tiers. TierFS enables us to store data in multiple tiers, and enables recoverability comparable to the Single- tier system, but requires a fraction of the capacity that Single-tier uses on the highest storage tier. The only significant downside of TierFS compared to the Single-tier system is that, in case of a site failure, the recovery time of TierFS is substantially higher because of the time to recover data from a vault. TierFS+CDP goes a step further in recoverability, by allowing recovery to arbitrary points in time, again with a much lower cost than a Single-tier system.

We also evaluated scenarios where the bandwidth and capacity requirements are increased by a factor of ten for both workloads. We observed the same trends across the systems in recovery behavior for the various failure scenarios, and in bandwidth and capacity usage.

6 Related work

The VxFS file system places files on multiple tiers and uses user-defined rules to place and migrate files [6]. Unlike TierFS, VxFS provides no mechanisms to improve recoverability of storage tiers.

There are many file systems that use logs, for purposes other than improving recoverability. The log-structured file system of Rosenblum and Ousterhout [11] uses sequential logs to avoid scattered writes to disk, hence improving write performance. Journaling file systems record file system changes in a log, called the journal, before applying them to the file system structures, to speed up file system recovery after a crash. Upon recovery, replaying the journal obviates the need for a full file system scan for inconsistencies. In contrast, TierFS uses the log for improving recoverability and providing continuous data protection for a less recoverable tier. Unlike a log-structured file system, which stores updates in the log as their final destination, with TierFS updates must eventually be sent to the appropriate storage tier. Thus, the recoverability log in TierFS is more similar to a write-ahead log.

Versioning file systems allow users to recover older versions of files (e.g., VMS, AFS [9], Elephant [12], and WAFL [4]). None of these file systems span multiple tiers, and many of them do not provide continuous data protection, either because they only snapshot at specific points in time (WAFL) or upon request (AFS), or because they do not track metadata operations such as renames (VMS). The only file systems that provide continuous versioning, as far as we know, are Elephant [12], CVFS [14], VersionFS [10], and the Wayback file system [2]. Of these, Elephant, CVFS, and VersionFS use specialized data structures to keep track of the common and distinct information of each version, rather than using logs to keep old versions of data. Thus, the old version's data is scattered, which makes it inefficient to transfer to archives or other tiers; data transfer, however, is not a goal of these systems. In contrast, TierFS uses a log to store the update history, which can be read sequentially and transferred efficiently. The Wayback file system [2] also uses logs, to store versioning information. TierFS is different from Wayback in three aspects: (1) Wayback uses undo logs, not redo logs; so, to create a log entry, Wayback needs to read a file's old data, which imposes extra overhead, (2) Wayback uses a separate log for each file, and logs are not stored sequentially one after the other, (3) in Wayback, log entries are permanent, without any proposed mechanisms for garbage collection, whereas in TierFS, old log entries are optionally archived and eventually removed.

Continuous data protection is also available in products from many companies [1]. These products operate either at the block level (typically block devices on a storage area network), file level (integrated with a file system), or application level (e.g., integrated with an email server or a database system). The idea is to tap into the stream of updates, and then store those updates to allow subsequent recovery of arbitrary points in time within a protection window. In this respect, TierFS is similar to file-level CDP products. The difference is that TierFS makes use of point-in-time copies to improve the efficiency and flexibility of CDP. For example, existing products can only provide CDP within a recent time window, typically a few days, and there is no way to archive CDP information.

AutoRAID [15] provides block-storage comprising two tiers: the first one uses data mirroring for high performance, and the second uses RAID-5 for space efficiency. The system automatically migrates frequently-written blocks to the higher tier and less frequently written blocks to the lower tier to optimize performance. Unlike TierFS, AutoRAID does not provide mechanisms to bridge the reliability levels of its backing storage tiers, and couples tiers only for performance and capacity trade-offs.

7 Conclusions

We present the design and evaluation of TierFS, a file system that takes advantage of different storage tiers in an enterprise system. TierFS maintains a recoverability log at the highest tier to increase the recoverability of lower tiers. The result is a multi-tier system that reduces recent data loss to that of a single-tier system that uses only the highest tier, while using significantly fewer resources at the highest tier. TierFS uses the recoverability log in conjunction with snapshots and backups at the tiers to improve recovery time. The recoverability log also enables consistent multi-tier backups and CDP. Our results show that coupling the tiers can yield substantial benefits over a conventional multi-tier system.

Acknowledgments. We thank Xiaozhou Li and the anonymous reviewers for many useful comments.

References

[1] CDP buyers guide, 2005. Available at http://www.snia.org/tech_activities/dmf/docs/CDP_Buyers_Guide_20050822.pdf.

[2] B. Cornell, P. A. Dinda, and F. E. Bustamante. Wayback: a user-level versioning file system for Linux. In *Proceedings of Usenix Annual Technical Conference, FREENIX Track*, pages 19–28, June 2004.

[3] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer. Passive NFS tracing of email and research workloads. In *Proceedings of the Conference on File and Storage Technologies*, pages 203–16, Mar. 2003.

[4] D. Hitz, J. Lau, and M. Malcolm. File system design for an NFS file server appliance. In *Proceedings of the USENIX Winter Technical Conference*, pages 235–246, Jan. 1994.

[5] HPSS user's guide, 1992. http://www.hpss-collaboration.org.

[6] G. Karche, M. Mamidi, and P. Massiglia. Using dynamic storage tiering, Apr. 2006. Available at http://www.symantec.com/enterprise/yellowbooks/index.jsp.

[7] K. Keeton, D. Beyer, E. Brau, A. Merchant, C. Santos, and A. Zhang. On the road to recovery: restoring data after disasters. In *Proceedings of the European Systems Conference*, pages 235–48, Apr. 2006.

[8] K. Keeton and A. Merchant. A framework for evaluating storage system dependability. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 877–886, June 2004.

[9] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, Feb. 1992.

[10] K.-K. Muniswamy-Reddy, C. P. Wright, A. Himmer, and E. Zadok. A versatile and user-oriented versioning file system. In *Proceedings of the Conference on File and Storage Technologies*, pages 115–128, Mar. 2004.

[11] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52, Feb. 1992.

[12] D. S. Santry, M. J. Feeley, N. C. Hutchinson, A. C. Veitch, R. W. Carton, and J. Ofir. Deciding when to forget in the Elephant file system. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 110–123, Dec. 1999.

[13] M. Sivathanu, V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Improving storage system availability with D-GRAID. *ACM Transactions on Storage*, 1(2):133–170, May 2005.

[14] C. A. N. Soules, G. R. Goodson, J. D. Strunk, and G. Ganger. Metadata efficiency in versioning file systems. In *Proceedings of the Conference on File and Storage Technologies*, pages 43–58, Mar. 2003.

[15] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems*, 14(1):108–136, Feb. 1996.