UNIVERSITY OF CALIFORNIA
SANTA CRUZ

UNDERSTANDING LONG-TERM STORAGE ACCESS PATTERNS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Ian F. Adams

September 2013

The Dissertation of Ian F. Adams
is approved:

Ethan L. Miller, Chair

Mary G. Baker

Darrell D.E. Long

Mark W. Storer

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Understanding Long-term Storage Access Patterns

by

Ian F. Adams

The past two decades have seen an explosion in both the growth and roles of long-term digital archival storage. While the traditional role of tertiary storage as an archive has persisted, there are many new use-cases as well, such as public historical document archives and climate sensor data. Yet, despite this expansion, our understanding of long-term storage is out of date. We have no insights into how these new archival use-cases behave, and even our understanding of tertiary storage behavior is decades old. Without up-to-date information on their behavior we cannot validate the effectiveness of both current and future archival architectures.

To address this issue, in my thesis we explore a variety of new and old archival use cases ranging from public historical data archives to private HPC tertiary storage systems. In our investigations, we found three primary results that held true across a variety of archives. First, we found that the oft-quoted "Write-once, Read-maybe" assumption was questionable in light of unpredictable users and system generated requests, calling into question the effectiveness of architectures that assume data is cold and immutable. Second, we observed that, in contrast to enterprise storage, there was not a clear subset of files responsible for most activity, making caching ineffective from the perspective of the archive. Third, we saw that aggregate accesses were largely unpredictable, but individual users showed strong locality of access which can be leveraged to reduce the number of media accesses and improve overall system efficiency.

The latter portion of my thesis is informed by the difficulties in analyzing the various archival datasets we obtained. We found that a lack of knowledge on a dataset's *coverage*, what actions were and were not captured, caused most of our difficulties. We approached this problem by developing a method we call *expectation difference* or *ExDiff*. ExDiff uses a combination of metadata snapshots and access logs to derive an expected system state that can be compared to actual metadata. Differences between the expected state and reality provide clues as to what is and is not being captured in any given log. This coverage data can be used to improve a variety of storage system tasks ranging from trace analysis to debugging and intrusion detection.

To my mother, father, and brother.

Acknowledgments

There are many people who deserve thanks in their support of me during grad school and the creation of this thesis. Thanks to all my friends and colleagues in UC Santa Cruz's Storage Systems Research Center (SSRC). Your guys support and feedback throughout the whole process was invaluable. I also want to thank my adviser, Ethan Miller, for his patience and support through out my graduate career. Finally, I want to thank all of my family and friends for dealing with my neurotic freak outs and calming me down. It's been a great ride guys.

This research was supported by NSF grant for Long-term archival storage management CNS-0917396 (part of the American Recovery and Reinvestment Act of 2009 [Public Law 111-5]), the NSF Center for Research in Intelligent Storage (CRIS) and its industrial sponsors, and the Department of Energy's Petascale Data Storage Institute under award DE-FC02-06ER25768.

Chapter 1

Introduction

> Obsolescence never meant the end of anything, it's just the beginning.
>
> Marshall McLuhan

Recent years have seen an explosion in the number of archival use cases beyond traditional tertiary storage and backup [46, 55, 64, 73, 74]. Government regulations such as the Health Insurance Portability and Accountability Act (HIPAA) legally require data to be safely stored and retrievable for decades or longer [41]. There are also a growing number of public content archives accessible via the Internet serving a wide range of data. Theses new archives range from state run historical document repositories, like those run by Washington, Oregon and New York [59, 75, 84], to state and federal level climate sensor data stores [21, 57]. There are also now scholarly publication archives [53], personal content repositories storing data such as photos and videos [58] and even entire websites being stored for posterity at the Internet Archive [45].

In spite of this proliferation of new archives, we have little information on *how* these archives are actually used and behave in the real world. This leads to the first of two primary questions we seek to address in this thesis, *what does modern archival storage system behavior look like?* To answer this question, we examine the architecture and usage of a variety of archival storage systems. We explore the real-world usage behavior of the new class of publicly accessible archival systems as well as re-examine more traditional scientific tertiary storage systems. The latter is representative of the older, narrower view of archival storage.

As one of the primary contributions of my thesis, our study of real-world usage behaviors provides data and architectural suggestions (summarized below) to better tailor future designs to actual system usage, providing for improvements in performance, efficiency and reliability. Without studies like like those provided in this thesis we are left working under assumptions and speculation which can lead to unpredictable system behaviors.

One difficulty that plagued us throughout our analyses was the myopic view of system behavior provided in several of our traces. For example, we discovered via out-of-band communication with the Washington State Digital Archives administrators that integrity checking processes were not logged, *yet accounted for over 99% of accesses to the system*. As another example, when we studied a repository of water sensor reports, we discovered that files were being silently renamed. Given that we relied on the file name to uniquely identify files, our analysis was wildly inaccurate until we compensated for the missed renames. In effect, what we were lacking in our datasets was an understanding of the *coverage* of the trace, what activities were and were not being captured during the tracing period. An incomplete understanding of coverage can cause significant damage, ranging from making an irreplaceable multi-year dataset unusable, to silently corrupting an analysis. This leads to the second question we seek to answer in this thesis: *how can we identify the coverage of a trace?* To answer this question, we explore black-box methods (out method does not explicitly require any changes in instrumentation of a system) using metadata snapshots and file level traces to help identify what a given trace captures and misses.

Our work in answering the question of identifying log coverage has provided a new methodology for understanding storage system traces. It has application not only to research, but to a variety of other common tasks in computer science and administration ranging from intrusion detection and debugging, to performance analysis and task auditing.

1.1 Archival Workload Analyses

A lack of understanding in the usage patterns of modern archives has forced researchers and designers to rely on outdated or marginally related information such as enterprise workload studies, and even flat-out conjecture to model archival system behavior. For example, in our own peer-reviewed published work examining energy use in archival storage systems we had no more than educated guesses as to what archival workloads may look like [7]. If and when such unfounded assumptions prove false, the archival systems designed under such guesswork may have unpredictable or even pathological behavior under real world usage scenarios.

Even our understanding of the older, traditional view of archival storage as simply a tertiary storage system is out of date, despite their continued use in many HPC and enterprise environments. The most recent studies of long-term storage system behavior are of supercomputing systems from nearly two decades ago in the early 1990s [46, 55]. Since then, we have seen three orders of magnitude growth in the scale and performance of storage systems across all areas. For example, in Miller and Katz's 1993 study of the NCAR archive [55], 25 terabytes (TBs) of storage was considered massive, and it was stored primarily on tape. In the present (2012/2013), purely disk-based systems of tens to hundreds of TBs are common. Petabyte scale disk systems are becoming more common as well with the growth of cloud storage services. How this massive increase in scale and accessibility has—or has not—shifted usage patterns is largely unknown, and correspondingly our knowledge of how appropriate particular architectures and methods used in tertiary storage actually is unknown.

To address this gap in our knowledge, we examine the architecture and usages of a variety of archival storage systems. We explore the real-world usage behavior of several of the new publicly accessible archival systems as well as re-examine more traditional scientific tertiary storage systems. The study of real-world usage behaviors is important because it allows us and others to better tailor future designs to the actual system usage, allowing for improvements in performance, efficiency and reliability. For example, a good understanding of a system's workload can inform the caching technique to be used. Conversely, working under incorrect assumptions can lead to unpredictable system behaviors.

For example, consider a tape based system designed under the archetypical archival assumption of write-once, read-maybe. If this assumption holds the system will likely perform well; tapes excel at streaming writes and appends, and the low likelihood of reads reduces the impact of tape's poor random-read performance. However, if the data requires periodic updates and reads are relatively more common and random, system performance and efficiency may drop due to tape seek times. In the worst case scenario we may even observe "shoeshining" where reads and writes fall below the designed transfer rate of a drive's heads, leading to repeated seeks with many rapid rewinds and fast-forwards of a tape. This reduces the life of the drive, the life of the tape, and the maximum data density a tape may obtain [69].

Another example of undesirable system behavior that could potentially result from incorrect usage assumptions is illustrated by the Pergamum system [76]. The creators of Pergamum predicate their energy efficiency gains on the assumption—with no data to back this assumption up—that an archival workload is likely to have many writes and a low density of read accesses thus minimizing disk spin-ups. If this assumption does not hold true they may not realize their energy efficiency claims and may see excessive power-cycles on their storage devices, increasing wear and reducing the long-term reliability of the system.

To close this knowledge gap we examine the behavior of several public web accessible content archives: the Washington State Digital Archives [75] and a repository of water table reports—such as ground water levels and salinity—maintained by the California Department of Water Resources [21]. We found the variety of long-term storage use cases exhibit a correspondingly wide range of behaviors well beyond that demonstrated in older tertiary storage studies. This highlights a danger in over-generalizing the view of the archival storage space, and suggests that more studies on a variety of systems are needed.

However, even with the variability we have observed, our results have significant implications on archival system design and there were several general trends. Commonly held assumptions such as write-once, read-maybe and how “cold” archival data remains are now being called into question. Systems that explicitly rely on the write-once read-maybe assumption, such as Chronopolis [56] and MAID [25], may exhibit unpredictable behavior when data is accessed more frequently than the system was designed for. We saw many large mass accesses from integrity checking processes and external indexers such as Google, suggesting new batch interfaces are needed to address high-volume, low-sensitivity accesses to archives. Such interfaces would allow providers to easily schedule potentially disruptive accesses, while still providing low-latency access to smaller individual users. We also have noted strong per user-session content preferences, suggesting that physically grouping data by semantic content may be beneficial.

In the second portion of my thesis focusing on workload analysis, we examine scientific tertiary storage systems to bring our knowledge of their behavior up to date. It is important to re-investigate these systems as our current understanding of their use is obsolete. The most recent study (other than our own) of tertiary storage behavior was in the early 1990s [55]. In the intervening time, we have seen multiple order of magnitude increases in both the scale of HPC storage as well as the processing power of the computers themselves. Further, scientific tertiary storage represents a significantly different use case from the public content archives described above.

Our initial investigation was on data obtained from Los Alamos National Laboratory (LANL). In the LANL dataset—comprised of daily histograms of file system metadata—we found that, despite the vast increase in scale and the shift to more disk-centric systems, when compared to prior archival super-computing studies aggregate modification rates and patterns appeared similar. We also found strong temporal and namespace locality to modifications, though the coarse granularity of our LANL data limited our analysis. To provide more detailed and comprehensive results, we obtained a multi-year dataset from NCAR, which contained detailed per-file activities from January 2008 through the end of December 2010.

In our NCAR study we found several results of note, providing useful direction in understanding what are potentially beneficial or harmful design decisions with regards to archives. First, like our public archive studies, we wanted to explore the notion of write-once, read-maybe. We found that from the system perspective, write-once, read-maybe is patently false. In any long-term storage system, and NCAR was no exception, files will be migrated and potentially integrity checked, leading to inevitable reads and writes. From the user perspective, write-once, read-maybe has been weakened. Over three years, non-trivial numbers of files were deleted (13%) and updated (5%) after ingest into the archive. This tells us that relying on immutability for fundamental architectural decisions may be dangerous. In examining file and user-session locality of access we found that users tend to restrict their activities to a small subset of the namespace with most accesses in a given session occurring at the same directory depth. This suggests that using directories as a heuristic for physical grouping of data may prove useful, especially for offline media such as tape where random accesses are expensive. We also found that most repeat accessess to a file, if they occur at all, tend to occur within a relatively short period of each other, showing that basic write caching can still be beneficial.

1.2 Identifying Workload Coverage

As touched on above, a constant problem in our analyses has been understanding precisely what actions were being captured and omitted in any given trace log, *i.e.* its coverage. For example, in our study of accesses to a repository or water sensor data, we discovered *by* *accident*, that files were being silently renamed. Since our only method of identifying a file was by its name, we found our unique file count was 200% high, and correspondingly we had been inaccurately mapping actions to files that didn't actually exist. It was only with administrator expertise that this problem was caught. Without this expert knowledge we would have been completely unaware that our results, and likely our conclusions, were incorrect. As another example, in all of our workload datasets we frequently saw unexplained drops in activity. We were unable to tell if these drops were logger failures, if the underlying system was down, or even if it was a legitimate reduction in user activity. As such, these activity drops become null data and could not be used to aid in any analysis.

While our original motivation for tackling the coverage problem was in aiding researchers studying long-term traces, storage system logs have a wide variety of applications. Correspondingly, incorrect knowledge of log coverage can negatively influence many applications as well. Consider, for example, the myriad general administration tasks that make use of storage system logs such as performance tuning and troubleshooting. If a logging process is silently malfunctioning an administrator may spend time diagnosing a problem that isn't there, or conversely a problem that is present may be missed due to poor coverage information.

Our solution to the problem of identifying coverage is to use file level traces and POSIX metadata snapshots in combination with a methodology we call *expectation differencing* or *ExDiff*. The idea is to use a trace log as a delta to an initial snapshot, roughly analogous to replaying a metadata journal to bring a file system to a consistent state. Based on what is observed in the trace, the metadata snapshot is used to what derive what we *expect* the metadata state of the system to look like. Then this expected state can be compared to the *reality* (current metadata snapshot) of the system. The differences between the two can then be analyzed, providing clues as to why types of actions are being omitted, as well as aiding in identifying logger gaps, *i.e.* periods when the system being traced continues to function, but entries are being consistently dropped.

With the development of expectation differencing, we contribute to many different groups. Researchers can better understand the limitations of the data they are working with. Similarly, system administrators have an additional tool for diagnosing issues within their systems. ExDiff can even be used in the development of new systems to ensure that all the necessary areas have been properly instrumented.

In our evaluation we show that density based clustering can be used to automatically identify gaps in coverage, though its accuracy is strongly influenced by the nature of the workload. Repeat accesses to the same file or record in a short period of time can mask missing log entries. This can be mitigated by having shorter periods of time between snapshot captures, as well as having multiple distinct timestamps per file to correlate actions with. We also show that missing a particular type of entry, such as a file create or rename, creates a particular *signature* that can be used in analyzing a log's coverage.

1.3 Thesis Overview

The rest of my thesis is dedicated to providing the background and explication needed for answering our two thesis questions: *what does modern archival storage system behavior look like?* And *how can we identify the coverage of a trace?*

Chapter 2 provides background and related work helping motivate the need for our studies and tools, as well as defining terminology to provide concision and consistency in our work. In Chapter 3 we look at access and update behavior in modern publicly accessible content archives from several sources. Chapter 4 examines the evolution and behavior of tertiary storage systems. Chapter 5 provides a combined, high-level analysis of all of our workload studies as well as meta-advice lead in to our log coverage work. In Chapter 6 we describe our log validation techniques used for identifying coverage, and proof of concept evaluations. Chapter 7 explores future work directions, and in Chapter 8 we conclude.

Chapter 2

Background and Motivation

> Since I don't have any credible sources of what happened next, I'm going to go off of the next best thing: hearsay and rumor
>
> George "Maddox" Ouzounian

The purpose of this chapter is to present the necessary background, context, and related work for understanding and further motivating our investigation into long-term storage as well as our interest in examining log validation techniques. First, we begin by defining the terminology we will be using within our analyses. Second, we define what we mean by archival storage. This is necessary as it has become an increasingly overloaded and nebulous term as the field has grown. Third, we examine a variety of prior workload studies and systems to motivate the need for new studies, and highlight some of the techniques we borrow for our own studies. Finally, fourth, we look at a variety of current system tracing, monitoring, and forensics techniques to illustrate the lack of exploration in the space of log coverage and validation.

2.1 Lexicon

To begin, we establish a set of concise terminology definitions, summarized in Table 2.1, that we use throughout this thesis.

An individual element in a set of archival data is a *record* or *file*. This may be an object, bitstream, or even a literal SQL record. It is at this level that the lowest level actions may occur, such as record creation, updates, deletes and queries. Each record may have *metadata* associated with it such as timestamps, provenance, category data, and so forth. We refer to a collection of records as a *corpus*, and a copy of that corpus as an *instance*. The hardware and software used to store an instance of the corpus is the *archive*. An individual archive may store multiple corpora.

Long data lifetimes and relatively short refresh cycles found in modern hardware and software dictate that a corpus will reside on several archives over its lifetime. It is vitally important to note the distinction between the corpus and the archive itself. End users and processes will act upon the corpus, while the archive's hardware and software provide the ability with which to do so.

A *trace* or *log* (we use the terms interchangeably) is a list of timestamped activities taken on a corpus's records. We refer to each individual logged activity as an *action*. An action is a single operation on one file, such as a read or record create. The *logger* or *tracer* write actions that have been captured to a log or trace file. A *snapshot* is a static view of the state of a corpus at a single point in time. Generally this snapshot is of the file or record metadata, such as modification timestamps or file path.

We assume a logged action always accurately reflects a change in the system. Any action that is not logged is a *log omission*. There are two types of omissions: *misses* and *drops*. An action that is not logged because it was never captured is a missed action. For example, a missed action can come from a developer failing to add a call to the logging system. A dropped action is where an action that is normally logged does not produce a log entry. A contiguous period of dropped entries is referred to as a *gap*. For example, dropped entries and gaps can be the result of a crashed logging process.

We refer to the aggregate body of knowledge about a system as a *sketch*. A sketch includes trace logs, profiles, record metadata, communication with system architects and administrators and even information about the user base and interface. The importance of this aggregate view cannot be overstated. System architect and administrator insights into the system design and behavior can have drastic impact upon the interpretation of a system's workload. They have intimate knowledge of background processes and scripts that can explain oddities or omissions in system behavior not readily apparent from logs and traces. Without this knowledge, one is often left with a significantly degraded picture of what is actually occuring within a

Term	Defintion
Record or File	smallest manipulatable element, may have associated metadata
Corpus	collection of records
Archive	combination of hardware and software used to store a corpus
Accessibility	who may access records, may be public, private, or mixed
Sketch	body of knowledge, including traces, logs and personal communications
Trace/Log	a timestamped list of actions taken upon a corpus
Action	a single operation taken on a record or file
Snapshot	a static view of the state of a corpus
Log Omission	an action of interest that has not been captured
Miss	an omitted action that was not instrumented for capture
Drop	an action that is normally captured that is omitted
Gap	a contiguous period of dropped action entries

Table 2.1: Terminology overview.

system. For example, the interface presented to users can have an impact. Highly skilled users may be able to circumvent a clunky interface, thus changing the perceived workload. The same interface with relatively unskilled users, such as might be found using a public access archive, may in contrast exhibit unintentionally pathological behavior.

We also distinguish at a high level the *accessibility* of a given system, that is, who may access the records. A *public* archive has its records accessible to anyone through the Internet. A *private* archive is one that has a restricted set of users who may access the records. A private archive may still be Internet accessible, but restricted to a particular class of user, such as climate scientist or a legal team. A *mixed* access archive is one that has a mixture of public and private records.

2.2 What is Archival Storage?

There are many use cases that now fall into the realm of archival storage. Traditionally, archival storage has been considered the tertiary storage layer in the memory/storage hier-

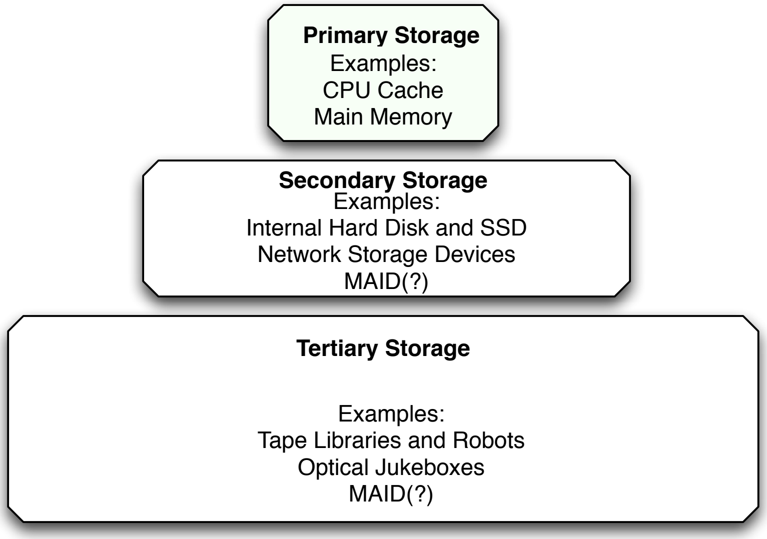


Figure 2.1: An illustration of a typical storage hierarchy in a computing system. The bottom of the hierarchy typically has the slowest, lowest cost-per-bit storage, while the top is relatively expensive, but very fast storage. We include MAID in both the secondary and tertiary layers as it blurs the distinction between the two by providing faster performance than tape with its disk based approach, but at the cost of spin-down and file migration for cold data.

archy, illustrated in Figure 2.1. Tertiary storage is typically the slowest performing and cheapest type of storage at the bottom level of the memory storage hierarchy. In large HPC systems this is often comprised of tape silos or MAID (Massive Array of Idle Disks) like arrays [25].

Yet tertiary storage is only one of a plethora of archival use cases, leading to nebulousness in the term archival storage. To one person, it may mean long-term versioned backup [64], to another it may be more akin to the library sciences view of archiving with formal ingest processes and indexing [84].

In our work, we consider archival storage to be concerned with records that are *desired* by users or organizations to have a long, potentially indefinite lifespan. This includes things such as important legacy code, historical documents, raw scientific data, simulation outputs, and personal photos, to name a few examples. We do *not*, however, consider backup data by itself to be within the realm of archival storage. This is because the purpose of backup is to provide recovery, while the data itself is of little value in a non-failure situation.

Given the large nature of what fits even within our definition, we restrict the studies in this thesis to a subset of this growing field. We examine several public archives and private tertiary stores. The former are representative of the expanded role of archival storage. Our understanding of the latter is obsolete despite their being a critical part of modern scientific systems. In our examinations of both we provide relevant, up-to-date knowledge and suggestions to guide the design and administration of current and future archival storage systems.

2.3 File System Tracing

Before we begin describing our specific workload analyses, we provide background on the primary methods used in modern file system studies.

There are two basic approaches to file system observation. The first is simply crawling the file system metadata and obtaining a static snapshot of the file system state. This can be done with simple scripts to traverse the directory structure, and calls to tools such as `stat` to obtain the relevant inode data for directories and files [10, 26, 35]. The second, and more involved, approach is to dynamically monitor file system activity. This can be done in a variety of ways, though at a high level it tends towards one of four methods. The first method is by instrumenting a running system's kernel or drivers to log specified events such as file reads and writes [17, 49, 67, 82]. Second, a system's designers may have included logging utilities that produce useful traces. It is this second method that produced most of the data we used in my thesis work. Third, tracing can be accomplished recording the file system's network traffic with a mirrored switch port and a tool such as `tcpdump` [12, 29, 39, 51]. The fourth approach is to have an application intercept calls to the storage system as they occur. This approach is what is taken in the TraceFS [15] and //TRACE systems [54].

Snapshot based approaches are easy to implement but only provide information about system state at a specific point in time. Even with multiple snapshots over time it can be difficult to accurately comprehend file system behavior. For example, while one may be able to tell a file was modified between snapshots, it is impossible to tell if there were multiple modifications in the intervening time. Further, access times are often absent and it is usually difficult, if not impossible, to accurately group access and modification behaviors at any useful granularity.

Dynamic approaches are more difficult to implement but have the advantage of showing the real time activity and behavior of a system. However, a serious challenge can arise in managing the volume and granularity of the data. Too fine a granularity can make the volume of data explode, especially in the multi-year time spans we are interested in. On the other end, too coarse a granularity can miss—or misconstrue—key activities. A real world example of this occurred in the access logs we obtained form the Washington State Digital Archives, covered in more detail in Chapter 3. We found they omitted integrity checking of records, which actually would have accounted for over 99% of activity in the system.

A dynamic trace by itself also misses the *starting* state of a system, which can be vitally important in determining the nature of a system's activities. For example, consider a trace that notes activities touching 5,000 files in a single directory. Unless we know how many files were in the system in total, we cannot tell much about the mutability and activity of the system as a whole. If there were only 5,000 files total we could state that we are seeing a very active directory. On the other hand, if there are 1,000,000 files in the directory, activities are clearly restricted to a small subset of files present in it. In short, knowing the start state of a system can drastically change the conclusions that are drawn. A dynamic trace on its own means one can only estimate the full state of a corpus at any point in time.

2.4 The Need for New Archival Studies

In this section we begin by surveying prior studies and their pitfalls and merits when applied to the archival storage field, showing that the current state of knowledge is insufficient for understanding archival behavior. We then look at several archival systems and describe how various workload behaviors may impact their claims, as we have found that often the workload used to validate the system is based on conjecture.

2.4.1 Workload Impact on Archival Systems

While we show in the next section that the currently available studies are insufficient for understanding archival behavior, we need to motivate *why* it matters that we understand archival behavior in the first place. We now look at several modern archival systems and techniques and explore how different workload assumptions can impact them.

Disk spin-down is a technique whereby the mechanical portions of the hard drive—the spindle and actuator—are powered off in order to obtain power savings. Because of their presumed low-density of reads and writes, archival storage systems are seen to be prime candidates for spin-down based power savings, realized in the Pergamum and MAID designs [25, 76].

The authors of Pergamum state they assume a write-once, read-maybe workload with a low-density of reads, while in MAID they test their system with a mass storage system workload that was already seven years old at publication. Were their workload assumptions to prove false, their power efficiency claims may fall short, and at worst, the system may exhibit pathological behavior with excessive power-cycles causing wear on the disks. Until we have up to date knowledge and understanding of a variety of archival workloads we can neither validate nor refute claims made by the authors of Pergamum and MAID, and for higher performance network systems the spin-down based approach has been called into question [22].

A more poignant example of power-efficiency claims being potentially inaccurate comes from our own earlier work in simulating a large-scale distributed archive [7]. We examined the impact of several broad data-placement policies, but our workloads were simply conjecture. As above, until we have up to date workload studies, we have no way validating or refuting our conclusions, or even the workloads we used in our simulations.

Another area frequently focused on in archival storage research is space efficiency, using techniques such as de-duplication and compression. Two example archival systems focused on this are Venti and Deep Store [64, 85]. Deep Store analyzes data for both inter and intra file compression with content addressable storage, while Venti does fixed block size de-duplication with immutable content addressable storage. While we do not study data contents, aggregate file and record access patterns can influence whether or not de-duplication and compression are worthwhile to use. Consider that de-duplication often fragments large files, potentially requiring many disk or tape seeks to read even a single file. If the rate of access to a given archive is low, this may not be of concern, but at higher access rates it can significantly degrade performance. Understanding a variety of potential archival workloads will allow current and future system designers and administrators to accurately judge for themselves what the best tradeoffs are given their resources and constraints.

2.4.2 Filesystem and Workload Studies

File system and workload studies have proven to be a boon to storage systems researchers helping guide and verify current and future storage system designs [80]. For example, the design of the Sprite log structured file system was directly influenced by the 1985 study of a

BSD file system [60, 68]. Similarly, workload studies have been of use in tuning and designing high performance storage systems [49]. Workload and file system studies are also of use in providing guidance to synthetic workload generators [36] as well as creating realistic file system states for testing [9].

It also important to have periodic studies of a variety of systems. New studies provide up-to-date information on the general structure and use of storage systems. Older studies of storage system behavior and structure are of use as well. When combined with new studies they can highlight trends and shifts in usage patterns as the hardware, software and user base evolves.

Yet, despite their obvious utility in aiding system design and management, there have been no recent studies relevant to modern digital archive behavior. In the past decade, a variety of storage systems have been examined ranging from HPC and corporate and engineering file systems, to academic, personal desktop and even peer-to-peer storage systems [10, 12, 29, 39, 51, 78]. None of the systems that were studied, however, can be considered archival in nature. For example, Leung *et al.* and Chen *et al.* studied enterprise storage usage at NetApp [24, 51] while Roselli and Anderson studied desktop and web-server workloads [66, 67]. Table 2.2 provides a summary of many of the studies of the past 30 years. Note that only studies nearing 20 years old were of arguably archival systems.

Most older workload studies are also not archival in nature, while additionally being obsolete given the technological progress of storage systems [17, 28, 35, 37, 49, 52, 60, 66, 73, 82, 87]. For example, Baker *et al.* and Li *et al.* looked at academic storage systems in the early 1990s, which arguably has little relation to modern archives due to the intervening time and wildly different use case. Even if any of these prior studies have similarities to archival workloads, we simply do not know given we have no "official" archival workloads with which to compare.

The last study that looked at any facet of archival storage was Dayal's 2008 examination of several HPC systems at rest [26], which included a small number of archival stores. However, the focus of his study was not specifically on archives, and further he only had access to summarized snapshots of file system metadata. He could make no statements about the behavior of the system over time. The last studies of dynamic archival workload behavior were done in the early 1990s on scientific tertiary storage systems [46, 55]. Prior to this, in

Study	Trace Year(s)	Approach	Duration	System(s)
Smith [73]	1974-75	Dynamic	12 months	Sci. Storage
Ousterhout*et al.* [60]	1985	Dynamic	3 days	Academic
Jensen and Reed [46]	1988-89	Dynamic	24 Months	Sci. Tertiary Storage
Gribble *et al.* [37]	1991,94,96-97	Both	1-44 weeks	Academic, Government
Miller and Katz [55]	1990-92	Dynamic	24 months	Sci. Tertiary Storage
Baker *et al.* [17]	1991	Dynamic	8 Days	Academic
Li *et al.* [52]	1991	Dynamic	1-14 Days	PC, Academic
Kotz and Nieuwejaar [49]	1994?	Dynamic	7 Days	HPC
Gibson and Miller [35]	1996-97	Snapshot	6 Months	Mixed Academic
Roselli and Anderson [66]	1996-97	Dynamic	1-12 months	Desktop, Web Server
Roselli *et al.* [67]	1996-97, 2000	Dynamic	1-12 months	Desktop, Web Server
Vogels [82]	1998	Both	2-4 Weeks	Engineering, Scientific
Zhou and Smith [87]	1998	Dynamic	5-45 Days	Personal Computing
Doceur and Bolosky [28]	1998	Snapshots	13 Days	Business, Engineering
Agrawal *et al.* [10]	2000-04	Snapshot	60 Months	Desktop
Ellard *et al.* [29]	2001	Dynamic	3 Months	Academic, Email
Gummadi *et al.* [39]	2002	Dynamic	8 Months	P2P File Sharing
Anderson [12]	2003-04,07	Dynamic	7-10 Months	Animation
Tanenbaum *et al.*[78]	2005	Both	4 days	Web, Academic
Leung *et al.*[51]	2007	Dynamic	4 Months	Engineering, Corporate
Dayal [26]	2008	Snapshot	1 Day	HPC Scientific Storage
Chen *et al.* [24]	2007	Dynamic	4 Months	Engineering, Corporate
Parker-Wood *et al.*[61]	2012?	Snapshot	-	Scientific

Table 2.2: Filesystem and workload studies from the past 30 years. We borrow Leung's terminology [51] to describe trace methodology. Dynamic refers to actively logging file system, while snapshot refers to a static analysis of file system meteadata. Stating "both" implies that a particular study utilized both dynamic and snapshot techniques. Question marks denote the data capture periods were ambiguous. Note there are no archival or tertiary systems studied in 18+ years.

the early 1980s Smith studied the file system of the Stanford Linear Accelerator in the context of optimizing file migration algorithms and exploring the basic patterns of tertiary storage behavior [73, 74].

Given the radical growth in scale and performance of both their attached compute systems and the storage systems themselves, it would be naive to blindly assume that modern tertiary storage behavior has remained unchanged. One of the primary goals of the thesis is to bring our knowledge of such tertiary storage systems, a traditional archival use case, back up to date.

Even with new knowledge of tertiary systems, there are many new archival use cases which we remain completely ignorant about. Tertiary storage systems are merely one facet of the modern archival storage space. There are growing numbers of other long-term data repositories, such as publicly accessible scientific and historical archives [11, 21, 57, 59, 75, 84], medical patient information [41], and document archives kept to satisfy legal requirements [71]. To the best of our knowledge, there have been *no* studies on the usage patterns of these relatively new archival systems. In this thesis we examine several of these new types of storage systems in order to guide future research and design.

While the earlier studies we have explored do not directly help us understand the modern archival storage space, they still have applicability in providing useful metrics and methodologies for understanding storage system behavior. File inter-reference intervals and data lifetimes provide us with insight into how often, and how long, files are actively utilized which can guide caching and migration policies [17, 55, 67]. Understanding file type popularity can help guide how files should be physically grouped and cached as well [51]. Looking at file sizes and how sparse the file is can guide how file system data and metadata should be organized and optimized [26]. We use metrics inspired by these and some of our own design in our studies, which we provide greater detail on in Chapters 3 and 4 where we examine public content archive and tertiary storage behaviors respectively.

2.5 Log Validation

As described in the introduction, one of the biggest difficulties in our trace analysis has been in understanding a trace's coverage, especially in the context of archival traces where years worth of data may be needed to accurately understand an archive's behavior. Due to this long time scale, seemingly trivial actions such as renaming a file, if omitted or improperly logged, can make accurate analysis of system behavior difficult or impossible. Such mistakes can cause months or even years of trace data to be made useless, or worse, if left unfound, silently corrupt analyses leading to flawed system designs, poor administration and or unpredictable system behavior. Because of this, we see a pressing need for researchers to be able to both identify the coverage of a trace.

Our approach called *expectation differencing*, or *ExDiff*, covered in detail in Chapter 6, is to make use of both dynamic traces and snapshots in concert to help identify a trace's coverage. The basic idea is to use a trace as a delta, to modify the state of a snapshot to create an estimate of the system's metadata. We can then compare this estimate with what the system metadata actually looks like, with the differences between the two providing clues as to what actions may be missing from a given trace.

In the rest of this section, we provide several use case examples to further motivate the need for ExDiff, and then cover related works in the field to show that there are no existing solutions to the log coverage problem.

2.5.1 Example Use Cases

Intrusion Detection and Forensic Analysis: Intrusion detection systems often compare the state of the system to a known "healthy" state, with mismatches raising alarms [48]. In addition to providing log validation, ExDiff can assist in detecting alterations in either the activity log or system metadata. By requiring an intruder to alter multiple data sources, the difficulty of silent intrusions increases dramatically. Similarly, the field of forensic analysis depends heavily on the ability to recreate activity accurately and detect when users have attempted to cover their tracks

System Management: Logs analysis is common in storage system management [14]. ExDiff can identify when and where logs may be suspect, leading to improved analysis accuracy. For example, ExDiff can help eliminate false positives where a system may have been running correctly, but is dropping log entries. It can also identify scenarios where problems exist in the logger itself, as opposed to the system being logged. Recognizing that a log may be inaccurate is important, because if entries are being dropped, then attempts at tuning may become pathological as the underlying system does not have an accurate view of its own activity.

Systems Research: Many traces and snapshots are available to researchers. Unfortunately, it is often impossible to speak with the original source administrators and architects to understand their coverage. This is particularly challenging for research into archival storage systems where years of data may be required, along with changing log formats with little or no documentation. Using a combination of snapshots and trace logs, researchers can use ExDiff to derive an understanding of the log's coverage without the need for expertise from the original system developers. ExDiff can also validate trace replays, by comparing the expected end state after a trace replay with the actual replay result.

2.5.2 Related Works

To the best of our knowledge, there are no existing systems aimed explicitly at verifying trace coverage despite the number of utilities used to capture operating system behavior, such as blktrace [16] strace [3], dtrace [1], and ftrace [2]. TraceFS is a customizable tracing system existing in user-space that intercepts all calls to the file system [15]. Systems such as Magpie [18], Stardust [79] and //TRACE [54] are designed to gain end-to-end understanding of larger systems. While all of these solutions are useful, none examine the coverage of the captured data.

DataSeries is a format with associated utilities designed around storing large amounts of structured serial data, of which traces generally are [13]. While DataSeries offers excellent performance and low overhead, its goal is efficiency and consistency, not identifying coverage of the logs stored therein.

Our work shares similarity to log replay in transactional database systems. A detailed log can be replayed from a set start-state to reproduce the current state of a system [20]. Snodgrass *et al.* took this idea a step further and included a hashing and "notary" service to make the log tamper evident and more useful for validating the state of a system. Similarly, journaling filesystems use metadata journals to restore a filesystem to a consistent state after a crash [72]. While our high-level approach share similarities to some applications of audit logs (recreating an expected system state and comparing it to reality), our underlying goal and approaches are different. We aim to find out what is missing from a log by comparing what the log says is occurring in a system to the observed reality, as opposed to assuming perfect knowledge.

SherLog improves log analysis by correlating it with source code to help narrow possible execution paths [86]. Similarly, Jiang *et al.* look at improving log analysis by combining multiple log messages and events together [47]. In both of these cases, however, the focus is on improving the use of the logs, not confirming what they do and do not capture.

Our work also shares some similarities with intrusion detection systems (IDS). The Tripwire system detects modifications to a filesystem by periodically comparing the current state of a system to a database that details what the system *should* look like (analagous to our estimated state mentioned above), and raises an alarm if the system fails one or more checks [48]. Hobgoblin is a language and interpreter that describes what properties a file system and its files should have, such as permissions for a given user [65]. I^{3}FS is a file system built around a Tripwire like integrity checking system, but checks integrity on the fly, rather than at administratively defined times [62]. Our work shares similarities to the above in that our technique is based around comparing system states to a ground truth, but I^{3}FS, Tripwire and Hobgoblin rely on what are effectively static rules, rather than an estimation based on observed activity. Abad *et al.*'s work on log correlation for intrusion detection uses multiple logs in concert to identify anomalies that may not be apparent from a single log [6]. While different in detail, their work shares a similar high-level approach of using multiple data sources in concert to improve their analysis. Lane and Brodley examine patterns of user actions compared to a learned user profile [50]. Their approach may be useful in extending our work for better automated detection fo dropped and missed entries.

Forensics tools such as SleuthKit [5] and Log2timeline [38] make use of file metadata and logs for a variety of legal and technical purposes. Log2Timeline in particular uses timestamps from both logs and file metadata to create a timeline of predicted activities. ExDiff also makes use of timestamps in metadata and trace logs, but is focused on understanding the specifics of a particular log's coverage, rather than trying to put together a cohesive picture of a storage system.

With S4, Strunk *et al.* describe a system with tamper evident logs and versioning for the purposes of intrusion detection and recovery [77]. The S4 approach provides for the recreation of any system state in the past, but is purposed for easy recovery rather than analyzing log-coverage. Their logging techniques and versioning however may be useful for more proactive incarnations of our log validation approach.

2.6 Chapter Summary

In this chapter, we have provided detailed background and motivation for my thesis work. We have shown that the currently available workload and file system studies are insufficient to describe the current state of archival storage due to their obsolescence and or unrelated use cases. We showed that this lack of knowledge can negatively influence current and future archival storage designs. We also showed that there is a serious lack of work in validating the coverage of traces. This lack of work on log validation influences not only our ability to accurately analyze storage system behaviors, but many other areas as well, including system administration and security.

Chapter 3

Public Content Archive Behavior

> We are the only species on the planet, so far as we know, to have invented a communal memory stored neither in our genes nor in our brains. The warehouse of this memory is called the library.
>
> Carl Sagan

If anything illustrates the expanding nature of archival storage, it is the explosion of long-term data now being served, and even ingested, via the web. There are many state run digital archives that have appeared in the past five years[11, 59, 75, 84] storing a variety of historical documents such as marriage records and census data. Academic paper archives [53], as well as publicly accessible scientific and sensor data [21, 57], are becoming common place as well. Even general web content, including personal websites and blogs, are now being crawled and stored at the Internet Archive in the hopes of being useful in the future [45].

Despite this rapid growth, the behavior of such archives is poorly understood; to the best of our knowledge, there have been *no* studies of public content repositories. Given that storage systems should be optimized for the common cases, this is a precarious situation. We as systems designers are left trying to design archival architectures based on marginally related workloads, and intuitive assumptions that may or may not hold true. In particular, we are frequently left relying on the oft-quoted write-once, read-maybe description of archival storage.

The contribution of my thesis work in this chapter is in completing one of the first studies ever done on file-level access behavior to publicly accessible archives. In this study, we focus on examining the types of access locality, file popularity distributions, and assessing the validity of write-once, read-maybe in content archives. We found that for the content archives we studied, there was little in the way of popular “hot” content, severely limiting the effectiveness of read caching. Individual user sessions showed strong content preferences—*e.g.* looking exclusively through marriage records—which provided the relevant metadata is available, is a useful heuristic for physically grouping and prefetching data. We also found several instances where write-once, read-maybe did not hold, limiting this assumption as a useful rule of thumb in archival system design.

3.1 Public Content Dataset Descriptions

The first public content source we examine, illustrative of the shift the Internet and lowering storage costs and have brought, is a public repository of digitized historical documents, the Washington State Digital Archives [75]. The second source we examine is a publicly accessible repository of water table reports—such as ground water levels and salinity—from the California Department of Water resources [21]. This source is particularly interesting as it illustrates yet another new direction in the long-term data space. Small, per-department specialized content repositories. Understanding the use of these small corpora is important, as many may be stored on a single physical archive where the aggregate behavior may be more important than the individual behavior of a particular corpus.

Washington State Digital Archives The publicly accessible Washington State Digital Archives serve a collection of digitized, historical artifacts—such as census information, military records, photographs, and land records—stored in an SQL database at the Washington State Digital Archives. The database is the primary way of retrieving data, accessed through a web interface. All records are maintained on disk, with tape copies maintained for emergency backup purposes. According to the administrators there was no read caching, and this is corroborated by results where we saw occasional re-retrievals of the same record within a few seconds.

We refer to the data and metadata stored in this archive historical as refer to this as the *historical* corpus. At the time of capture, it contained approximately 90 million records, 28 million of which are accessible via their web interface; the rest must be accessed on-site, but are otherwise unrestricted in access. Records occasionally move between web accessible and on-site access based on content or explicit request. In this study, we focus on the web viewable

Field Name	Example	Null
Record ID	123555	No
Date 1	10-10-1910	Yes
Date 2		Yes
Type	Marriage Record	No
Ingest date	11-12-2008 12:25:06	No
Modify date	9-4-2009 12:52:00	Yes
Num. of objects	0	No

Table 3.1: Historical record metadata. A yes in the Null column indicates the value may be null. Number of objects is the number of digital objects associated with a record, possibly zero. The two date fields are used to hold record specific dates, such as birth and death times.

records, since this is the only portion of the corpus covered in the provided user access logs.

We obtained two logs for this corpus, spanning September 27, 2007 to June 17, 2010. The first log details per-record metadata, described in Table 3.1. The second is a user access log that records accesses to individual records. Each record is linked to zero or more digital objects—such as photographs and documents—but each digital object is only associated with one record. The trace does not note whether the digital objects linked to that record were retrieved. Further, while the access log provides information to group accesses from the same session, we cannot link different sessions to specific individuals or hosts.

It should be noted that these logs only reflect user retrieval of records within the corpus database and do not reflect access to any other content, *e.g.* HTML pages. Additionally the logs do not track the activity from data migration or integrity checking processes. As we describe further in Sections 3.3 and 3.4, these administrative processes actually make up the dominant fraction of accesses.

California Department of Water Resources The second corpus in our study, the *water* corpus, is a relatively small set of water table reports consisting of 57,000 records. The files were accessible through a simple web interface and directly downloaded from the server. We suspect there was no front-end caching of files due to our periodic observation of files being logged as re-downloaded within a few seconds of a prior download. Regardless, any potential flash traffic does not appear to have been filtered out due to caching, and as such we assume our

Field name	Example
Site	A00268
Site type	Surface Water
Parameter	Flow
Period of record	1997
File name	GW_DEPTH_POINT_DATA
File size	13050
File type	Plot

Table 3.2: Water record metadata, and representative values. Unique records are identified using a (Site, Period of Record, File Name) tuple.

data is representative of unfiltered user activities.

We have two traces for the water corpus. The first is a set of update logs from approximately weekly and quarterly batch scripts. Each update log notes the records written to, the date, and record metadata, summarized in Table 3.2. The second is a set of access traces consisting of a per-user access log, where each entry contains the IP address that retrieved the record, as well as the site, period of record, and record retrieved. As with the historical corpus, the logs do not reflect accesses to general web content, only downloads of the reports themselves. Similarly, if there are any internal indexing or integrity processes running, they are not reflected in the logs.

In the update trace, we identify a unique record using a tuple of site name, period of record, and file name. Complicating this, however, was an intermittent change in file naming conventions that made it difficult to map old names to new, introducing the danger of overcounting files and mapping updates to incorrect file names. To address this, we only count updates to files that map to names in existence on the last day of the update log. Though this discards approximately 50% of the 1.7 million updates, it ensures both a correct file count and an accurate lower-bound on file update activity. More updates may have been required to keep the relevant files up to date, but no fewer.

3.2 Public Content Archive Modifications

In this section we focus on data modification rates, where we found results that call in to question the frequent assumption that archives are “write-once.” Long-term content corpora are actually quite dynamic. We observe that 50% of records in the water corpus received five or more updates, often stemming from automatic data management processes. Similarly, 75% of the records in the historical corpus saw at least one update during the trace.

To begin with, we examine data updates within the publicly accessible water corpus. One complication to note is that we can only deduce record creation in the water sketch by noting a record’s first appearance in an update log. In the analysis, we associate each record with a list of updates generated from the update logs. As described in Section 3.1, we filter the updates such that only updates mapped to files present on the last day were included in the analysis. Though this introduces the danger of under-counting updates, it ensures that the results remain conservative and removes potentially misleading update counts caused by record renaming.

Examining the logs in the water sketch reveals a surprisingly high number of updates caused by corpus management: automatic policy rules frequently overwrote generated reports, whether or not they had actually received updated data. Two scripts in particular generated a large volume of data updates. The first ran approximately weekly, and modified any report that had data updated within 30 days. The second ran on an irregular, but roughly quarterly schedule, and overwrote all reports in the corpus regardless of the last update they received.

To identify the source of updates, we break our analysis into three stages. The first contains all the updates seen by the corpus. To isolate the results of the weekly script, the second set only considers updates that occur to a file after 30 days have passed. We call this the *30-day filter*. The last set takes the results of the 30-day filter, and removes all mass updates that touch over 10,000 records. We call this the *quarter filter*. Using this approach, we can identify a lower bound on the number of necessary updates; more may have been required to keep the relevant reports up to date, but no fewer.

The results, shown in Figure 3.1, demonstrate behavior that deviates dramatically from the write-once assumption of traditional tertiary storage. When no filters are applied, only 40% of the records receive 5 or fewer updates, and those that receive 20 or fewer updates still only account for around 65% of all records. Applying the 30-day filter, 50% of the corpus still

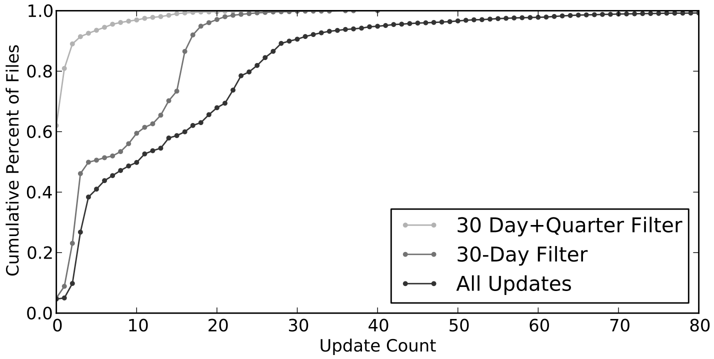


Figure 3.1: CDF of records, showing the number of updates per record over the duration of the water sketch's update trace.

receives 5 or more updates. A shift appears, however, if the quarterly updates are filtered out, as 60% of the records receive zero updates. This is still far from the write-once scenario; 10% of the records receive 3 or more updates, and 20% receive one or more. Many of updates are for complete "Period of Record" reports (records) that are running summaries of all prior data for a site.

The historical sketch contained each record's *last* update times, but otherwise did not make note of updates to individual records. This meant that while we could identify if a record had been updated, we could not tell if it had received multiple updates. Despite this, we found that over 75% of records received at least a single modification to either their metadata or associated object. Like the activity we saw on water corpus, this is in stark contrast to the oft-quoted write-once assumption.

There is a caveat to note in our analysis of content mutability, however. Even though we have literally years of data, this may only be a small fraction of the corpora lifetimes. The mutability we have observed, while it flies in the face of the write-once assumption, may be focused on relatively new records in a given archive. A record may very well "stabilize" after several years.

When we examine the inter-arrival time of updates within the water corpus, the time between any two consecutive updates to a record, illustrated in Figure 3.2, there are surprisingly large numbers of records with long inter-update periods. 35% of the approximately 900,000 observed updates occurred after a period of over 64 days. When the 30-day and quarterly filters

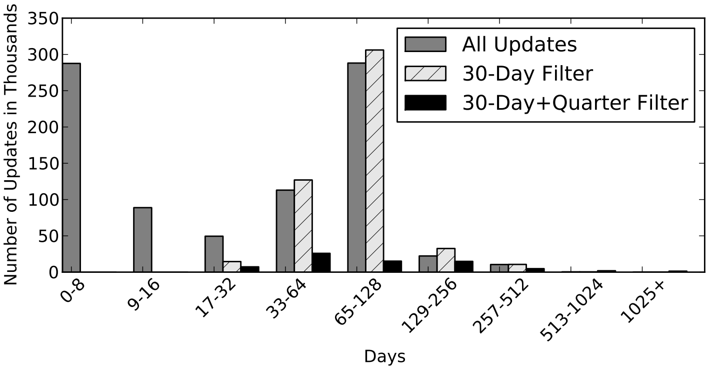


Figure 3.2: Histogram of inter-update arrival times for all records in the water corpus.

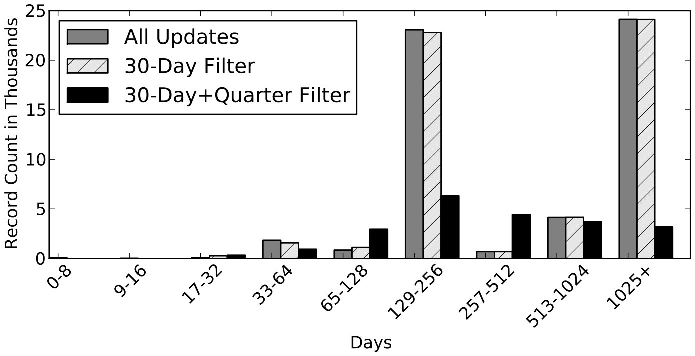


Figure 3.3: Histogram of time between a record's creation, and its last update within the water corpus. Note records that receive no updates are not counted.

are applied, 70% and 50%, respectively, of updates occur with an inter-arrival time of more than 64 days, though the total volume of updates drops significantly.

To further investigate update behavior within the water corpus, we examine the range of time over which records were receiving updates. Figure 3.3 shows a histogram of the time between a record's creation and its last known update. There are, however, two important points to note with this histogram. First, it does not include records that were never updated after creation as they would not contribute to the update count. Second, the record's ingest time relative to the start and end of the trace period impacts the update range we observe. For example, a record ingested two days before the end of trace would have at most a two day range.

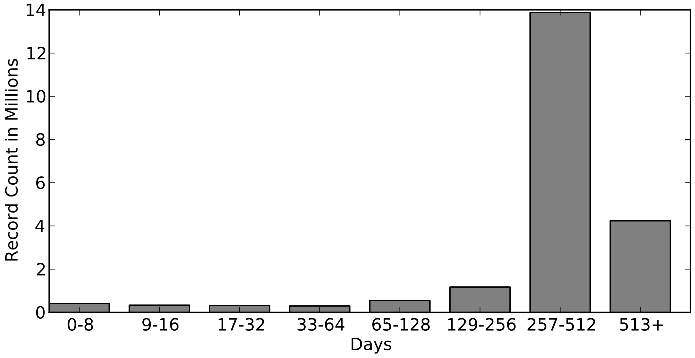


Figure 3.4: Histogram for the historical sketch, showing the range of time between a record's ingest date and its last modification time. Note that records not updated are not counted.

Nonetheless, this is still a valid method of demonstrating that records continue to be modified long after their ingest time. Using this approach, we observe that over 50% of records that receive updates do so over a range of over 256 days. When we apply the 30 day and quarter filters the proportions remain roughly the same; approximately 50% of records that received updates were modified over 256 days after record creation. However, the *number* of records receiving updates drops due to the filtering.

In the historical sketch we do not have the same level of access granularity as in the water sketch; rather, we can only see a record's last modification time. This time reflects the most recent time that any of a record's fields or associated digital objects were modified. This level of detail is still sufficient to show that of the approximately 28 million publicly accessible records, over 21 million had a non-null modification date, meaning that approximately 75% of the corpus content was updated at least once. This is significantly more than that shown in both Agrawal's desktop filesystem study [10], where over 80% of files remained unwritten each year for over five years; and the modern tertiary storage behavior illustrated in Figure 4.2, where approximately 80% of the corpus remained unmodified.

Update time ranges were also similar between the water and historical sketches. When looking at the time between a historical record's ingest and its last recorded modification time, shown in Figure 3.4, we note that 85% of the modification times show a difference of 256 or more days from a record's creation.

The surprising amount of update activity we see across both the water and historical corpora is made possible—and easy—by the use of cheap random access media. The use of tape or optical media in the face of so many modifications would be problematic, as they require significant extra hardware to maintain high access rates. Additionally, the long access times of such media are a barrier to frequent modification of data. The relative ease of updating modern media may have subtle, but important, implications however. For example, repeated mass updates, like those seen in water corpus, can make identifying the source and nature of an update difficult. While relatively innocuous in the water corpus, repeated, potentially unnecessary, modifications of data can have profound implications in other situations, *e.g.* legal rulings or scientific findings based on retrieved data.

3.3 Public Content Archive Accesses

In both the water and historical sketches, record modifications appear as session-less, system-generated operations. In contrast, record accesses are associated with a distinct session. In the historical sketch accesses were grouped into sessions with the provided data, while in the water sketch we grouped accesses based on access intervals and IP address of the requesting party. Specifically, we used a sliding window approach where if a retrieval occurred within 10 minutes of a prior access from the same IP address, it was grouped into the same session. Our analysis in this section looks at both aggregate and session-oriented access behavior.

3.3.1 Aggregate System Behaviors

Within the water and historical sketches, we find that accesses are dominated by a few, often machine generated, large-scale retrievals, such as a Google crawl or integrity checking process. In the historical sketch, we observe approximately 5.88 million distinct accesses between September 27, 2007 and June 16, 2010. The accesses are across 1.05 million user sessions, accessing 2.28 million distinct records. From discussions with the repository administrators, we also know that *all* records are integrity checked monthly. Though only 8% of the 28 million web viewable records were accessed by users over three years, 100% of the records were read via the integrity checking process each month. If integrity checking is considered to be equivalent to record retrieval, then less than 1% of reads come from end-users. Even assuming that files are checked for integrity at a much lower rate of once per year, still only 10% of read traffic would come from end users. This finding has significant implications on archive design. Effective, low-latency, end-user retrievals are critical to the perception of a useful system, but only make up a small fraction of the actual workload. On the other hand, administrative processes, which make up the bulk of accesses and are critical to the integrity of the system, are typically less latency-sensitive. As we describe further in this section as well as in Section 3.4, a separate batch interface for bulk accesses could be beneficial to future systems.

In the water sketch, there are roughly 98,000 distinct retrievals between August 28, 2007 and July 1, 2010. By artificially grouping accesses originating from the same IP address that arrive within 10 minutes of one another, we identify approximately 8,500 user sessions. We chose 10 minutes as the threshold based on our observation that the number of sessions created by this grouping method taper off after approximately 10 minutes. We exclude approximately 1,200 retrievals that had a null value for their files, accounting for approximately 1% of all retrieval requests.

We find that approximately 70,500 of the 98,000 total accesses in the water sketch originated from Google, and 27,000 from other users. Since there were 57,000 records in the last quarterly update, and non-Google users made 27,000 requests, we observe that no more than 50% of the archive's contents could be retrieved by non-Google users. On the other hand, Google likely requested nearly all of the reports given the methodical nature of their crawls, though it cannot be conclusively stated given the file renaming issues we noted earlier.

One peculiar behavior we notice in both the water and historical sketches are significant numbers of user-sessions re-retrieving the same record in the same session, often within a few seconds. Communication with system administrators and architects yielded no explanation for this odd behavior. We note that these re-retrievals accounted for 3% of the retrievals in the water-sensor log, and nearly 35%(!) (2.04 million) of the record retrievals for the historical archive. These re-retrievals have a noticeable impact on the results and we account for them explicitly in our experiments and analysis.

From the daily access counts in the historical sketch shown in Figure 3.5(a), we find that the number of accesses on any given day is relatively stable, and exhibits a slow growth trend. We also note a number of moderate spikes, and one large spike around day 900.

A microanalysis of the large day 900 spike finds that it is comprised almost entirely of sessions that only retrieved a single record, and that the records retrieved were predominantly (over 90%) photographs. Further, the upper quartile of distinct records retrieved in the spike received 5 or more accesses, as opposed to the usual one or two on most prior days examined. Consultation with the system and corpus administrators yielded no clear explanation for behavior seen in the spike—their only theory was that someone managed to link their entire web viewable photo set, and they admit that such an explanation is only a guess at best. Further, we confirmed that external indexers, such as Google, only have access to around 6,000 records in the historical corpus, ruling out another possible explanation. Though we could draw no conclusions from this spike, it is worth noting as it as an example of anomalous behaviors that are intuitively more likely to be seen in a long-term corpus served over many years.

To explore the potential effectiveness of caching on daily traffic and spike mitigation, we ran the daily access count analysis with two different sizes of a simple LRU caching filter: 0.01% and 0.1% of the total number of retrievals, corresponding to 500 and 5,000 records. When we include re-retrievals during the same session in the count, even a small cache is shown to be moderately effective at absorbing accesses, with overall hit ratios of 37% and 38% for a 500 and 5,000 record cache, respectively. When we remove the re-retrievals the cache effectiveness plummets, exhibiting an overall hit ratio of less than 7% for even the 5,000 record LRU cache.

Interestingly, as Figure 3.5(c) shows, the cache is effective at reducing the magnitude of several of the access spikes. Even the small cache absorbed nearly 50% of the traffic during the day 900 spike. Thus, while their overall impact is low, read caches in long-term content stores may be useful for handling flash traffic and record re-retrievals.

Next, we examine daily access counts and cache effectiveness for the water repository, illustrated in Figure 3.6. One of the first things to note is an extended access spike, approximately between days 700 and 750. Using a reverse IP look up we confirmed this was Google slowly crawling the repository contents. In total, Google accounted for over 70% of *all* record retrievals. For the subsequent analysis of the water accesses, we filtered the large, external index crawl from the dataset. Note that while other user sessions did occasionally exhibit bot-like behavior (fast inter-retrieval times, and mass retrievals) we could not conclusively identify them as such, and left them in the trace.

In the water sketch, as with the historical sketch, we see a moderate number of re-retrievals within user sessions, and examine the impact of caching with and without these re-

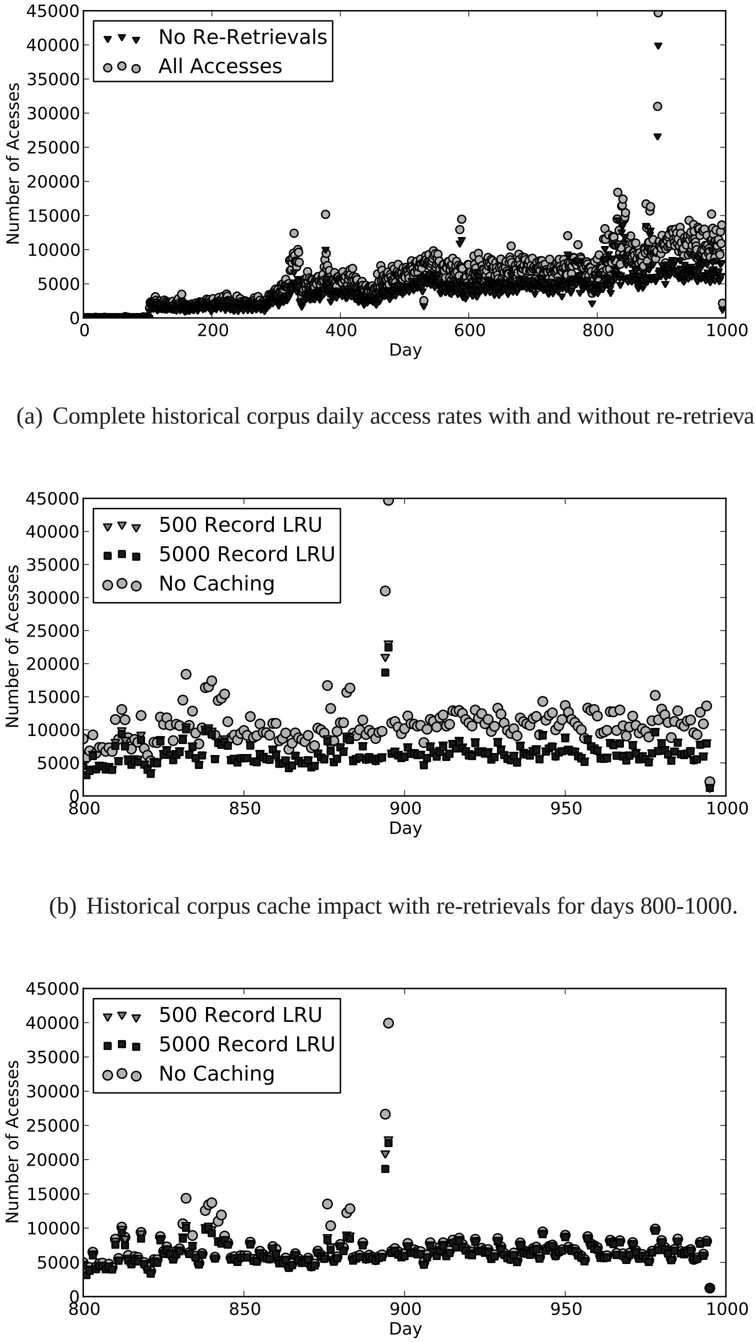
retrievals, shown in Figure 3.7. As with the previous observations, with re-retrievals included, there is low to moderate cache effectiveness with hit rates of 12% for a cache size of 10 records, and 17% for 100 records. When session level re-retrievals are eliminated the hit ratios drop to 2% and 8% respectively. In the water sketch, however, caching remains largely ineffective even on days with significantly increased traffic, as figure 3.7(b) illustrates.

3.3.2 Per-Session Behaviors

One of the first results we noticed in our session-based analysis was the strong traffic skew; only a few sessions account for the majority of traffic. Figure 3.8 illustrates the number of retrievals per session with and without re-retrievals. Interestingly, for both the historical and water sketches, over 50% of sessions only retrieve a single record. Further, we observe that the distribution quickly flattens out, with approximately 90% of sessions retrieving 15 or fewer records. Since many sessions are coming from humans interacting via a web interface, the time between user retrievals is relatively long, often seconds to minutes.

While 50% of sessions—with re-retrievals—only retrieve a single record, those sessions in the historical sketch account for fewer than 10% of the total retrievals, and fewer than 5% for the water sketch, as shown in Figure 3.9. The vast majority of data was accessed from larger sessions. In the historical sketch, 40% of all accesses come from sessions of more than 20 retrievals, and nearly 80% in the water sketch are made during similarly large sessions. In the water sketch, this skew is due to a Google index crawl of the corpus that occurred over several large sessions, each retrieving hundreds to thousands of records. The prevalence of these large mass retrievals, much like the large integrity checking, suggests the utility of a batch interface, as which we describe more in Section 3.4.

Next, we look at content popularity, independent of sessions, to see if we can identify a subset of records or content types that account for a disproportionate fraction of accesses. Figure 3.10 shows that all of the distributions exhibit a long tail, with the exception of the types-based popularity for the historical sketch. For example, the sites—*i.e.* water well location—in the water corpus exhibit the second strongest popularity affinity with 20% of sites accounting for 60% of accesses, but the next 20% of sites only account for another 20% of accesses. At file level granularity, this trend becomes even more pronounced. Note, however, that the file naming issues within the water corpus may mask some amount of file popularity. The content



(a) Complete historical corpus daily access rates with and without re-retrievals

(b) Historical corpus cache impact with re-retrievals for days 800-1000.

(c) Historical corpus cache impact without re-retrievals for days 800-1000.

Figure 3.5: Daily access counts to the historical corpus with and without re-retrievals and the associated LRU cache impacts. If a retrieval was absorbed by a cache hit it was not counted.

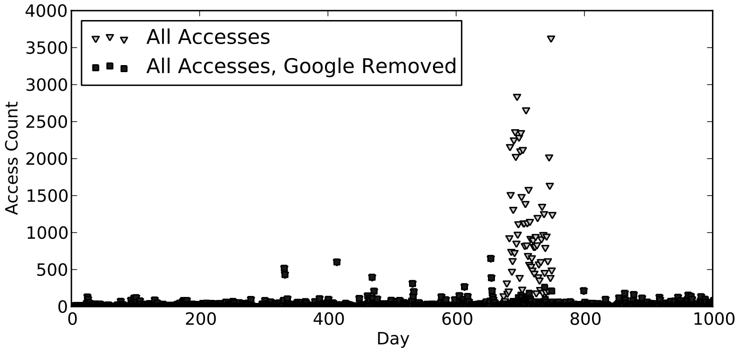


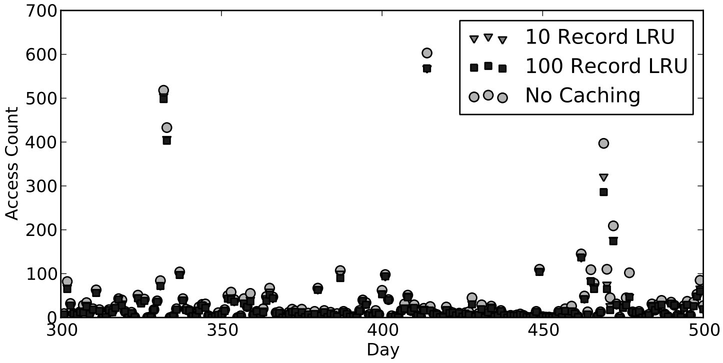
Figure 3.6: Daily access counts to the water corpus with and without the retrievals by Google

popularity distribution corroborates our early findings showing and explaining why LRU read-caching appears to be largely ineffective; while certain categories of data are more popular, individual records do not appear to be particularly more popular.

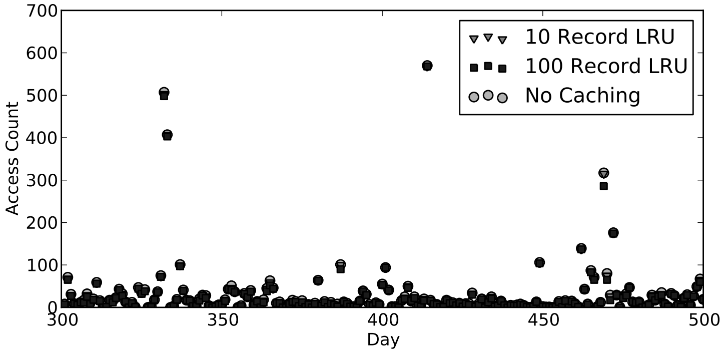
We next examine access locality within sessions to see if individual user sessions tend to access a single or few types of content. In our analysis, we first remove re-retrievals from all sessions and then exclude sessions in which only a single record is retrieved. Additionally, we eliminate records from the historical sketch that are found to be missing metadata (less than 0.5% of retrievals), as well as those with the category “Restricted Type.” These records were originally public and subsequently moved into the private archive, so we cannot determine their category. The restricted type retrievals account for 12% of accesses after removing re-retrievals and excluding singleton sessions.

Figure 3.11 shows that, across both the historical and water sketches, individual user sessions tend to retrieve strongly related content. We observed that nearly 50% of sessions in the historical sketch retrieve three or fewer record types, and similarly 50% of sessions in the water sketch retrieve data pertaining to only a single site. When we include the year, 25% of sessions in the water sketch retrieve records within a single site-year combination, but still exhibit strong per-session content locality.

Across sessions, however, a wide variety of content and individual records are retrieved. This is evident in the poor cache performance, as shown earlier in Figures 3.5 and 3.7. The strong individual session locality does suggest that grouping data based on content along



(a) Water corpus daily access counts with re-retrievals, days 300-500.



(b) Water corpus daily access counts without re-retrievals, days 300-500.

Figure 3.7: Daily access counts for the water corpus with and without re-retrievals, and associated LRU cache impacts of size 10 and 100 records. If a retrieval was absorbed by a cache hit it was not counted. Google accesses have been filtered out. Note cache impact is nearly eliminated when re-retrievals are removed.

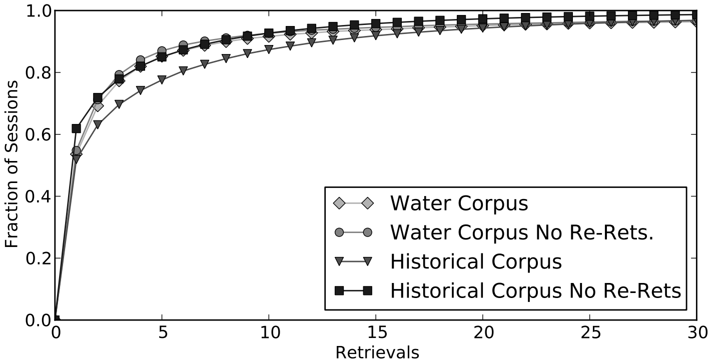


Figure 3.8: CDF of accesses per session showing the number of records retrieved per session in the water and historical corpus, with and without per session re-retrievals. The plot is truncated at 30 accesses, as the few large sessions would distort the plot.

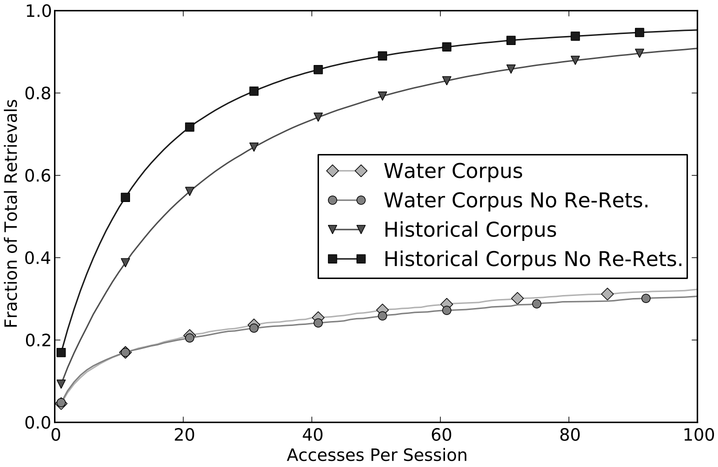


Figure 3.9: CDF showing what fraction of total retrievals were contributed per session size with and without re-retrievals for the water and historical corpora. Note, truncated x-axis is truncated at 100 to maintain readability as retrievals to the water corpus dominated by a few large sessions; sessions with over 300 retrievals account for 60% of the total retrievals.

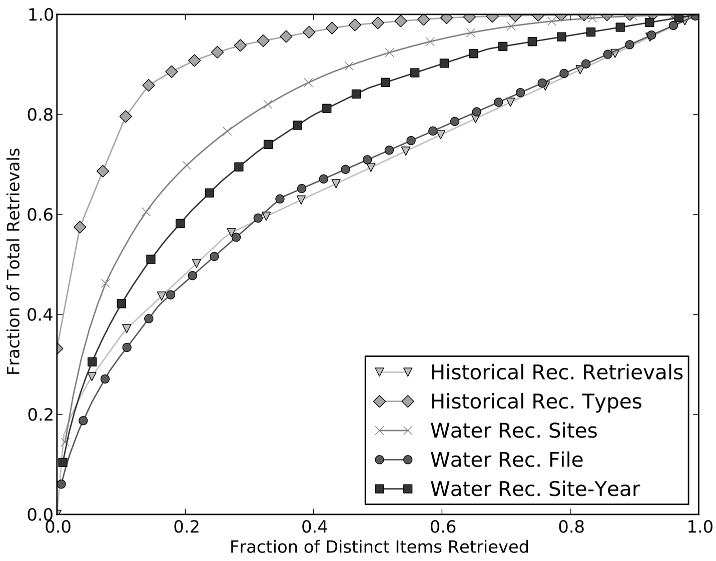


Figure 3.10: CDF of record popularity by individual record, and distinct content type. Note that x-axis values are ordered by popularity, that is, the most popular items are plotted first. With the exception of the historical corpus record types all CDFs are subsampled.

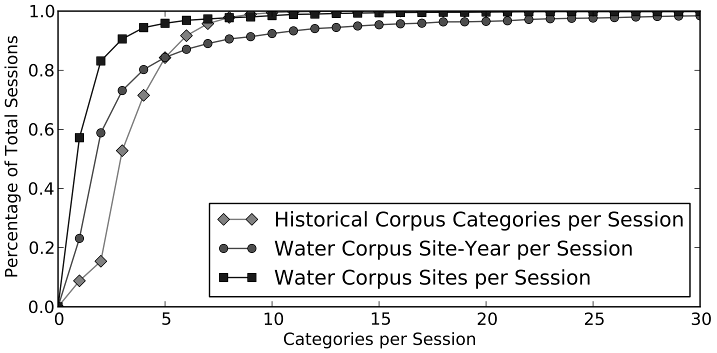


Figure 3.11: CDF showing the number of different content categories retrieved per session. In the historical corpus, records have an associated category. In the water corpus, we examine how many distinct sites as well as site-year combinations are accessed per user session. The plot is truncated at 30 types to avoid a distortion.

with pre-fetching may be effective [83], provided the content type has a sufficiently small number of records. For example, this would likely be more effective for the water corpus where any given site-year combination rarely has more than 15 or so records at a few tens of kilobytes apiece, than for the historical corpus where there may be many millions of records connected by an associated category. In contrast, the lack of individually popular records we noted earlier may impact systems that aim to conserve energy by duplicating or migrating commonly used data [25, 63]. This is because while strong statements can be made about individual session behavior, aggregate system wide activity is largely unpredictable in regards to the popularity of records.

3.4 Lessons Learned and Implications

In this section, we cover overall lessons learned from our work in examining public content archives.

Data modifications were frequent, widespread, and over a much longer duration than expected suggesting that file immutability should be an enforceable policy independent of the media type. This flexibility is even handy for explicitly immutable data–such as compliance stores–as such datasets often have a specific expiration date, after which the owners would like the data to be immediately deleted.

With respect to reads, we found that from the system perspective both the water and historical corpora were quite active. While user requested reads were relatively rare, data management tasks, indexing requests, and the inevitable migration of long-term data, make the “read-maybe” pattern patently false; *all* content is eventually read, and it is often read *en masse*. This could severely impact the effectiveness of system designs that rely on low read-rates. For example, systems that rely on spun down disks for power savings may overestimate the cost savings they can deliver [63, 76] unless accesses can be tightly controlled and scheduled.

Further, the results suggest a potential danger in optimizing for the wrong operations. In the water trace, the vast majority of total accesses were from a few large-scale requests—such as Google crawls—with the remainder originating from user accesses that often only retrieve a single record. We argue, however, that these small numbers of user requests are latency sensitive, and critical to the users’ perception of effective, long-term storage.

Within these user sessions, we found that there were a few favored content types, and the same data was often requested multiple times within a single session. Across sessions, however, it was much more difficult to identify popular content. Thus, aside from assisting with the re-retrieval problem, strategies that rely on migrating popular data may be ineffective at best, and harmful at worst [63, 88]. In a disk spin-down scenario, such movement could incur additional energy penalties for little or no benefit. Depending on the scale, this may also make the use of tape-based architectures, and those based upon immutable media types significantly less efficient.

It is important, however, not to downplay the importance of bulk accesses, since they dominate an archive's workload. Recall that we observed integrity checking accounting for 99% of read accesses in the analysis of the historical sketch. The disparity in large and small access properties suggests that current archive interfaces are insufficient. Since the data suggests that these large-scale accesses are often latency-insensitive administrative processes, we suggest the use of an asynchronous batch interface for large requests as a complement to the traditional single record interface. The benefit to the system is that such a request would provide full *a priori* knowledge of the records in the requests, allowing the archive to optimize its resource scheduling most effectively.

Such an interface would allow a client to specify the set of records desired, a schedule of when it needs the requests fulfilled by, and a means to alert the client when the request is complete. For writing to a corpus, this could be useful for data management functions: We observed that public content archives provide anonymous read access, but writes came from the system itself. In the case of external indexing services, such an interface could help shift the large-scale requests from appearing parasitic at an energy cost and workload spike standpoint, to a more symbiotic relationship; the indexing service receives the data and provides search capabilities, and the archive can efficiently provide the means for users to retrieve it. To prevent pathological use of the traditional, single-record access interface, archives could utilize strategies such as throttling or retrieval caps.

3.5 Chapter Summary

In this chapter we explored the long-term workload behavior of two publicly accessible content archives, representative of the expanded nature of modern archival storage. We obtained data from the Washington State Digital Archives and a repository of water table reports maintained by the California Department of Water Resources.

From our analysis, our contributions take the form of three broad conclusions. First, in both the water and historical sketches, we found activity that directly contradicts the write-once, read-maybe assumption. We found that records are actually updated, often multiple times over many months, and when integrity checking and indexing processes are accounted for, entire corpora are read. Second, we found that large, mass accesses account for most of the activity to a given corpus. For example, batch updates to records in the water corpus. This suggests there may be merit in a separate batch interface for large scale, latency-insensitive accesses to an archive and a separate high-priority interface for smaller, latency sensitive user operations. Third, we noted that individual user-sessions tend to have strong content locality, while aggregate behavior shows that any individual records popularity is limited. Because of this, it may be useful to physically group data based on semantic content, though popular data concentration and caching may be of limited effectiveness.

Chapter 4

Scientific Tertiary Storage System Behavior

> The greater our knowledge increases the more our ignorance unfolds.
>
> John F. Kennedy

Despite the fact that tertiary storage systems continue to be used in large scale scientific and enterprise systems as long term archives, it has been 20 years since the last detailed study of such a system [55]. In that time, the scale of storage systems has drastically grown. Even relatively small organizations have disk based storage that exceeds the total capacity of the system studied at NCAR in 1993, which at the time was considered large-scale with approximately 26 TB of capacity. By contrast, a system we examined from Los Alamos National Labs (LANL) had over 1.3 *petabytes* of data and was expected to grow to over 2 PB by early 2011. Further, this archive is only one of several at LANL, and is considered a small archive at that. This is to say nothing of the myriad other hardware and software innovations that have permeated the super-computing community. Understanding modern behavior can help us validate if older architectures are still optimal, and if not provide guidelines for future systems and optimizations.

In this chapter we begin bringing our knowledge of tertiary storage systems up to date. Our contributions in this chapter take the form of observations on tertiary storage activities and architectural suggestions. First, much like the public content archives most activity is automated in nature. Because of this, we see merit in the use of asynchronous batch processing, particularly for large-scale administrative accesses. Second, we see strong locality of access, where files tend to be accessed in groups in small portions of the hierarchical namespace. This suggests that grouping files based on directory may be useful, particular for tape and other offline media where accesses to many locations can be very expensive. Third, we saw that there was no clear subset of files responsible for most activity in the archive, much like the public content archives. This means that read caching is largely ineffective, at least from the perspective of the archive. Fourth, we saw fewer data updates relative to the public content archives, but still enough that the idea of a write-once archive is called into question for the scientific use cases. We call this out as write-once is often used as an assumption when designing archival architectures.

In the first section, we provide an overview of our two datasets, coming from Los Alamos National Laboratory (LANL) and the National Center for Atmospheric Research (NCAR). Following the dataset descriptions, we explore coarse-grained results we obtained from analysis from LANL, followed by our analysis of the NCAR storage system. Our analyses of NCAR and LANL are discretized due to the radically differing nature of the datasets making low-level comparisons difficult. We conclude this chapter with an overall summary of our tertiary storage system studies.

4.1 Dataset Descriptions

LANL: The system under consideration at LANL is one of their super-computing archives. We refer to the archival data it stores as the *general scientific* corpus, and the archive itself closely resembles the structure and intent of the classical view of long-term storage as tertiary storage. The corpus, at the end of the time our dataset covers, contains approximately 60 million files, totaling 1.3 PB spread across disk and tape. The typical use case occurs when a user is allocated compute time; he or she is provided a top-level directory in the archive for storing his or her data.

The raw data we obtained on the LANL system is in the form of 13 months—from May 2009 to June 2010—of daily histogram reports collected from a daily crawl of the system's inode metadata by FSstats [26]. One of the daily reports covers the entire file system; the second covers each top-level directory corresponding roughly to summaries of individual projects. Table 4.1 describes the histograms we used. Note that atime (access time) tracking was explicitly disabled in the file system, so we could not effectively analyze retrieval patterns.

NCAR: The National Center for Atmospheric Research (NCAR) is dedicated to me-

Histogram type	Description
Reported size	File length returned by `stat`
Allocated space	Number of bytes actually allocated
mtime	File modification times
mtime (KB)	File modification times, grouped by file size
Overhead	Difference between reported size and allocated space

Table 4.1: FSstats histogram reports collected over the general scientific repository. One set of histograms covers the entire archive, and the other set is run once for each individual top-level directory, corresponding roughly to specific projects.

teorological and climate research and its associated impacts. Our analysis is focused on the mass storage system (MSS), a tape-based storage archive used for storing a variety of datasets for months to years.

The MSS consists of tape libraries with a disk cache "in front" to serve as a write and read cache for files under 10 GB as illustrated in Figure 4.1. For files under 10 GB, clients write directly to the disk cache; the files are later migrated to tape via an automated migration process. Files over 10 GB are written directly to tape. For reads, the first read of a file goes directly from tape to the user. If another read of the *same* file occurs within 24 hours, it is then cached on disk, assuming its size is under 10 GB. Because of these policies, the cache is primarily utilized as a write-buffer, which in turn generates large amounts of automated migration traffic as data is moved to tape. The dataflow is illustrated in Figure 4.1.

Users interact with the MSS through a FIFO queue. Files are written to any currently mounted media that has sufficient space. Reads are queued to a specific tape or disk volume, allowing for multiple reads from a single tape mount. Given the FIFO nature of request handling we see a relatively "pure" ordering of user-requests. Thus, our analysis reflects user behaviors with less noise than an analysis of a system that aggressively re-orders and groups requests.

As an example of MSS operation, consider a user wanting to archive a 1 GB, file `myData.dat`. Initially, `myData.dat` is created on the disk cache. As space is needed on the disk cache, the `myData.dat` file will be copied to the first tape that has sufficient space and subsequently removed from the disk cache. If `myData.dat` is later read, the initial read will go directly from tape to the user, bypassing the cache. If the file is read a second time within

Date	Hardware
Jan 2008	5 StorageTex Powderhorn Silos 40 TB Disk Cache 70 STK 9940B Drives
Jun 2008	5 StorageTex Powderhorn Silos 100 TB Disk Cache 70 STK 9940B Drives
Jan 2009	5 StorageTex Powderhorn Silos † 2 SL8500 Tape Libraries 100 TB Disk Cache 70 STK 9940B Drives † 70 STK T10000B drives
Mar 2010	2 SL8500 Tape Libraries 100 TB Disk Cache 70 STK T10000B drives

Table 4.2: Evolution of MSS hardware. January 2008 is the start state of the system. With the addition of the SL8500 libraries, the corpus was migrated from the Powderhorn libraries in preparation for their decommissioning. † denotes hardware in the process of decommissioning.

24 hours, `myData.dat` will be cached on disk.

Purges (permanent deletions) of files from the system are based upon a file's retention policy, or done by explicit request from a user. Each file has a retention policy describing the number of days it should be retained in the archive, *e. g.* 360 days. If a file passes its retention period without extension, it is considered to be in the *trash*. After 30 days in the trash, the file is permanently deleted from the system and its removal is logged as an action. Note that the default retention period and trash time are system parameters set by NCAR staff, though actual retention periods can be modified by users.

The hardware evolved significantly during the logged period, as summarized in Table 4.2. As we show further in the following sections, the data migrations associated with this technology change are not noted in our logs. No further hardware changes occurred from March 2010 until the end of our dataset.

The data corpus stored on the MSS, which we refer to as the *NCAR corpus*, is comprised of approximately 80% simulation output, 15% observational data for validation and seed-

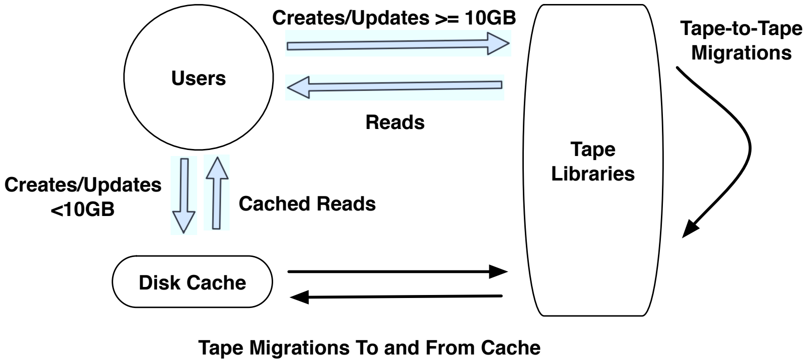


Figure 4.1: This figure provides an overview of the NCAR MSS. Note that all files under 10 GB that are created or written to are cached first on the disk cache, while reads are only cached if a file is read twice within 24 hours. Files greater than 10 GB in size are written directly to tape, and all reads initially come from tape if they are not already present on the disk cache.

ing of simulations, and 5% system backups. At the beginning of our dataset in January 2008, the corpus was known to hold approximately 4 PB of data; at the end of our dataset, it held approximately 11.7 PB of data in 69 million files. However, approximately 3.3 PB were duplicate bytes: extra copies of files for reliability purposes. A typical long-term use case of data stored in the archive is storing simulation output and retrieving it two to five years later for re-analysis and validation.

4.2 LANL Analysis

We beging by comparing the LANL storage system to prior archives, before moving onto details on modification and update behavior.

4.2.1 LANL Storage Hardware and Scale Evolution

One of the first things we note in our examination of the LANL archive is that it has a significantly higher fraction of disk, relative to tape, for its storage compared to prior studies. Summarized in Table 4.3, the scientific corpus from LANL contained about 1.3 PB at the end the report period, and is hosted on 1000 TB of tape, and 285 TB of hard drives. Note however that the LANL corpus was continuing to grow at the time of our analysis, and was estimated to grow beyond 2 PB in 2011. While the LANL administrators have designed the tape library to

	Disk (TB)	Tape (TB)	Total (TB)
1993	0.1	26.2	26.3
2010	300	1000	1300
Ratio	1:3000	1:38.2	1:49.4
CAGR	60.2%	23.9%	25.8%

Table 4.3: Tertiary storage comparison between the NCAR system in the 1993 Miller study [55], and the current LANL system, showing the ratio between the 1993 value and the 2010 value, and compound annual growth rate (CAGR).

expand to partially accommodate this growth, overall system growth will still be proportionally dominated by disk. Compared to the scientific corpus of the NCAR study [55], the total corpus exhibited a compound annual growth rate (CAGR) of 25.8%. However, most of the growth in capacity occurred in hard drive storage. Compared to the earlier study, the data shows a hard drive CAGR of 60.2%, though one must take this number with a grain of salt given the varying nature and relative scales of the systems in question. Note that we restricted the hard drive comparison to a holistic high level view, as the NCAR archive did not use commodity hard drives, relying instead on proprietary storage modules. Interestingly, that hardware was fairly old when it was studied in 1993; the IBM 3380 systems [44] in the NCAR archive were introduced in 1980, with the final revisions released in 1987.

Similarly, the 1992 NCAR archive used IBM 3480 tapes, with a capacity of 200 MB per tape. This format was introduced in 1984, making it nine years old at the time of the study; by 1992 IBM was producing the fourth generation 3490E IDRC tapes, with a capacity improvement of 12 times that of the 3480. By comparison, the 2010 LANL archive uses the relatively recent LTO-4 format tapes, with 1 TB of capacity, 5000 times the storage of the IBM 3480 tapes. Even with the potentially exaggerated gap in tape capacity, we still see a CAGR of 23.9%, which lags slightly behind the total storage CAGR of 25.8%. The impact of this shift towards more disk-centric—and in the near future potentially SSD—tertiary storage on file usage and migration patterns is of keen interest. In the NCAR system we describe later in this chapter, the disks are primarily used as a file cache, but with continuing increases in disk density as well as new media such as phase-change memory and high-density flash becoming available we may see moves towards disk becoming the primary storage medium in these larger archives.

4.2.2 LANL Storage Data Modifications

We next look towards the mutability of the scientific corpus, examining file updates and locality. We found that despite despite rapid growth in the fraction of storage provided by hard drives, a media that is much easier to update and access than tape, traditional tertiary corpora continue to be fairly static; 60% of the LANL content we observed was not modified in nearly a year.

Figure 4.2 illustrates the general scientific corpus's aggregate update behavior with a heatmap. The y-axis corresponds to histogram bucket ranges, and the x-axis the day of the trace. The heat-scale on the right maps shade to the total fraction of archive contents. Thus, a corpus that exhibited a high degree of content modification across many files would be warmest along the base of the y-axis; many records would have a recent modification time.

When records are ingested into the archive, they tend to be ingested in batches, and they maintain their existing modification time, explaining why the temperature warms in areas other than the histogram bucket for 0–2 days. At the start of the trace, the archive ingested a batch of files with recent modification times. In Figure 4.2 this appears as a warm area near trace day 0, for the histogram bucket with files 2–4 days old. As the trace proceeds, those records remain static, and age steadily. This is seen on the heat map by the high temperature region of the histogram moving from the 2–4 day bucket to the 64–128 day bucket as the trace proceeds from day 0 to day 100. Other ingests follow the same behavior, as seen near days 60, 100, 150 and finally 310.

Despite the growing use of hard drives, the results show that aggregate modification behavior in traditional tertiary storage appears much the same as it was at NCAR over 15 years ago. That study showed that 65% of files referenced in the 24 month trace were only written to a single time, and over 20% were read but never written to. Similarly, at the end of the dataset's duration, despite only having a 13-month trace, we see that approximately 60% of corpus records had modification dates more than 256 days in the past. The concentration of heat towards the upper end of the heatmap, rather than periodic and concentrated *lower* MTIME values which one might expect with consistent periodic ingests, is due to two factors. First, the archive is continuously growing. If data is remaining unmodified, most of it will continue to age and the relative fraction each new ingest makes up of the total archive corpus will be continuously decreasing. Second, the histogram buckets logarithmically scale. Each bucket

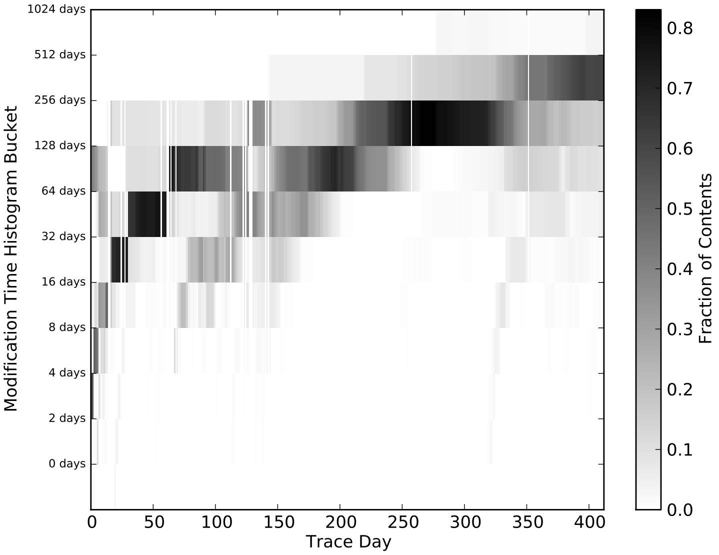


Figure 4.2: Heatmap of the general scientific corpus's daily record modification histograms over 400 days. The color indicates the fraction of the total archive contents, x-axis the day in the trace, and y-axis the modification time histogram bucket. For example, on day 275 of the trace, 80% of archive contents received their most recent modification 128–256 days ago. Note y-axis is log scaled (to match the histogram report), and we truncate it after 1024 days, as most contents are below that age.

covers twice the time range of the prior bucket, so higher values necessarily cover more time, and thus will cover a greater fraction of older files.

Unfortunately, given the coarse granularity of our data we can't glean individual file behavior. Given this and the relatively short time period our dataset covers, we are leery of making strong suggestions towards system design based solely on our investigation of the LANL data. That said, this sort of behavior suggests that older techniques and architectures for file migration and caching may still be relevant in modern tertiary storage systems, and this observation is reinforced in our study of the NCAR system, described next in Section 4.3.

To examine update locality in the LANL sketch, we looked at the individual top-level directory histograms as opposed to those showing aggregate system behavior. It is important to keep in mind throughout this dicussion, however, the coarse granularity of the data we are working with comprised of histograms with logarithmically scaling in bucket sizes and relatively few populated project directories.

Before any of the analysis was done, we filtered out directories that never contained any files, which was apparent based on histograms with 0 file counts and sizes. This accounted for approximately 500 of the 600 total project directories on the last day of summaries. We consider a directory to be updated if at any time after the initial ingest—the first day a non-zero file count is seen—the number of files present within the directory changes, or a we observe change in the mean modification time of the files resident in the directory. We chose to focus on the mean modification time as it was the simplest method given the very coarse granularity of the data we had. We had no observations of individual file behaviors.

Figure 4.3 shows a CDF illustrating the number of updates a typical directory receives. It is important to note however that this *may* be potentially misleading as we only see activity at a day level granularity; a directory that receives updates everyday but is only 17 days old, will look to be average, when it may very well continue to be active for much longer. Regardless, we still see that 80% of directories receive 25 or fewer updates. This lines up with the conventional wisdom that a relatively small fraction of a system receives a disproportionate amount of activity. Note this fact is only a part of the picture however; we cannot easily tell the *magnitude* of the update(s). For example, a directory receiving a single, small file update every day is extremely difficult distinguish from a directory with a very large file being modified everyday, or similarly one with many files.

Figure 4.4 shows the amount of time between a directory's first population with files and the last noted update. The times are relative to the first appearance of files in the directory. However, despite this we still see nearly 40% of directories having some sort of activity over 128 or more days. While we have evidence that there is activity over long periods of time, we have no notion of the magnitude of the activity. For example, even though we might be able to see that 15,000 files are now marked as being in a bucket of mtimes between 0 and 2 days prior to the summary, we can tell nothing about the nature of the modifications. The files may have been entirely replaced with new data or simply touched with a single bit flipped somewhere.

Figures 4.5 and 4.6 illustrate the distribution of inter-modification times to individual project directories—recall the day level granularity of our histograms however. As we can see, most updates occur within a few days of each other and quickly drop off. This follows behavior similar to that shown in the 1993 NCAR study [55]. The same issues still apply from above; we have not come up with an effective way to gauge the magnitude of the changes in a consistent automated fashion for more than a single directory with the coarse nature of our data.

Regardless, the preliminary evidence we have discovered here does suggest that modfication patterns are still similar to that of older studies such as the 1993 NCAR study. We found moderate to strong temporal and namespace correlations to updates. While we must keep in mind the coarse granularity of this dataset, these results suggest that physically grouping data on user directories could yield performance and efficiency benefits, and this is corroborated by our NCAR MSS study as shown later in the chapter.

4.2.3 LANL File Size and Space Distributions

Here we describe the distribution of file sizes and allocated space across the LANL sketch, again with comparisons to the 1992 NCAR study.

Moving to the file level, we next examined file sizes within the corpus. Figure 4.7 shows a CDF of file sizes calculated from the last day's histogram in the dataset. Nearly 50% of the data written in the NCAR study consisted of files between 10 and 100 MB; in contrast, we found that 40% of the total reported usage in the LANL corpus consisted of files between 1 and 2 GB.

Interestingly, however, when comparing the reported file sizes to the amount of storage space actually allocated to files, note that 60% of allocated space is consumed by files

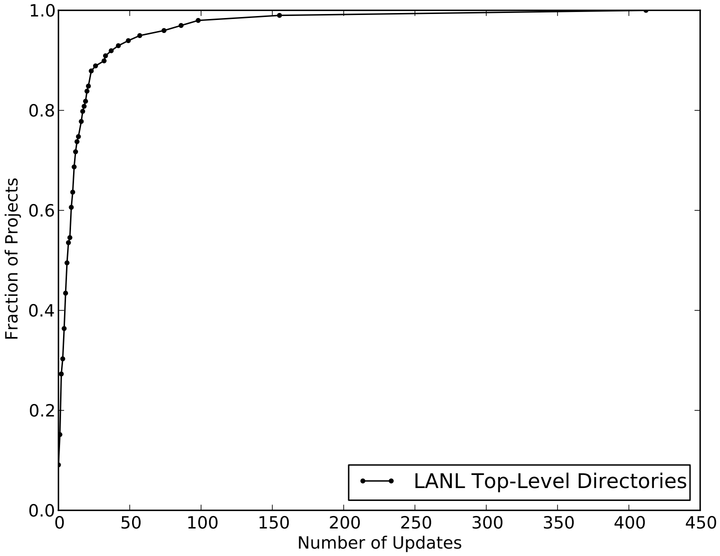


Figure 4.3: CDF of the number of updates per individual project directory.

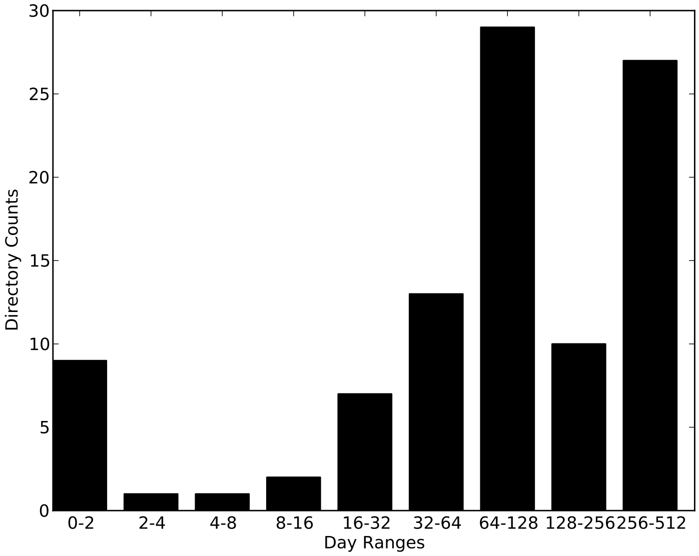


Figure 4.4: A histogram showing the amount of time between the the first file being noted in a directory and the last seen update. Those that are never updated are not counted.

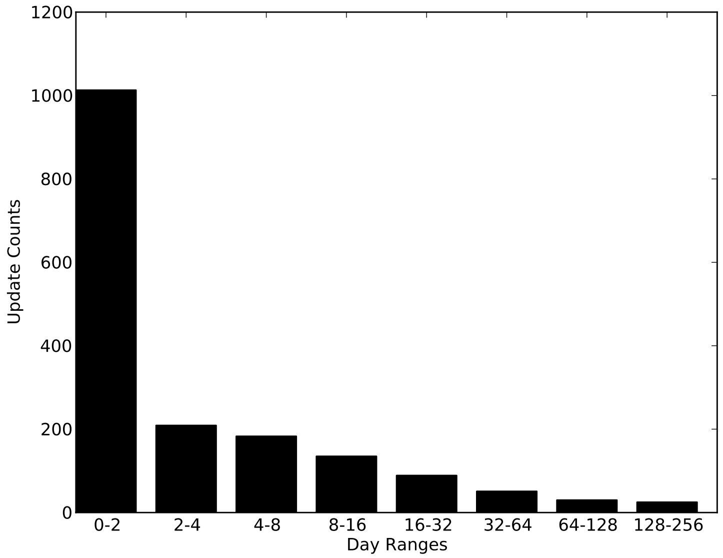


Figure 4.5: A histogram of the update inter-arrival times for individual project directories. Note the day level granularity of our data, so we cannot identify inter-arrival times finer than one day.

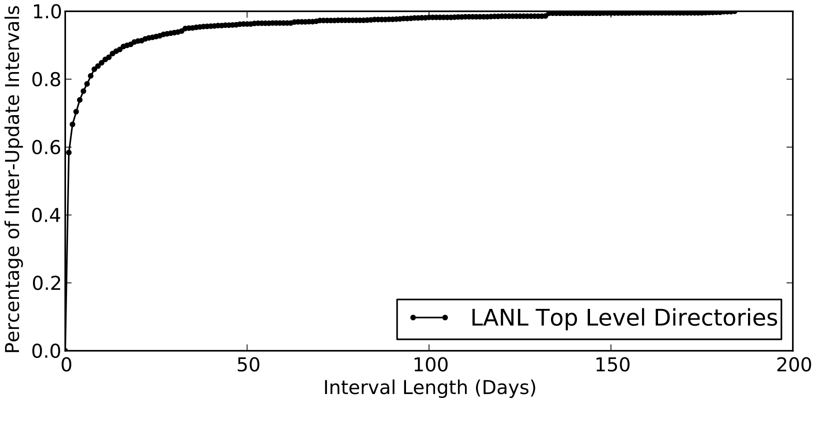


Figure 4.6: A CDF of update intervals for top level directories in the general scientific corpus. The update count is a count of the number of times we observed a change in some value for a directory. For example, 25 updates means we observed differences on 25 distinct days.

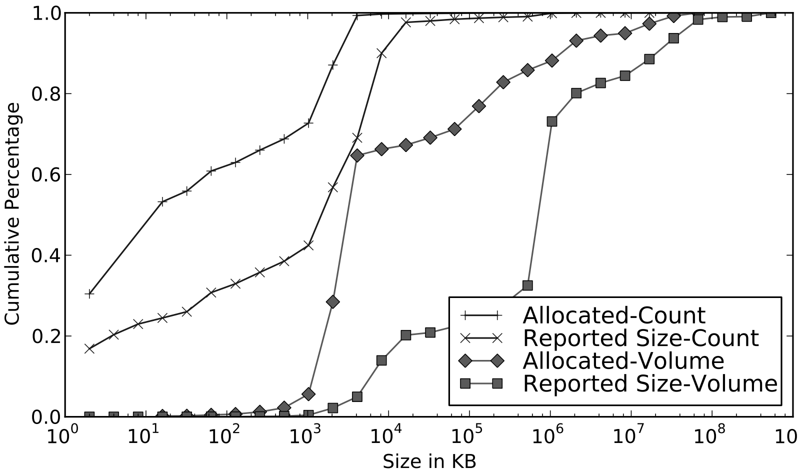


Figure 4.7: CDF of reported file sizes and allocated space at the end of the general scientific trace. Volume refers to the aggregate amount of storage consumed (in KB), while count refers to the number of files. Sparse files have larger reported file size than allocated space, causing the difference between the curves.

between 2 and 8 MB in size. Thus, while the bulk of storage is consumed by files that are considerably larger than the previous study, those files tend to be sparse; over a petabyte is accounted for when looking at file sizes, but only around 100 TB is actually allocated. This behavior may be partially attributable to scientific super-computing's use of shared checkpoint files [19].

Figure 4.8 shows a CDF breaking down the fraction of space that each directory contributes to the total archive. We observe that a relatively small fraction of directories contribute the dominant fraction of space. The same holds true for raw file counts as well, as shown in Figure 4.9. More data is needed to see if this is an artifact of the specific system at LANL or is a general trend across scientific tertiary storage archives.

4.3 NCAR Analysis

While the LANL sketch provided illuminating, if preliminary, results, its very coarse granularity limits our analysis. To address this, we have obtained a new detailed dataset covering three years of file migration activity from NCAR. In this section we begin by providing an

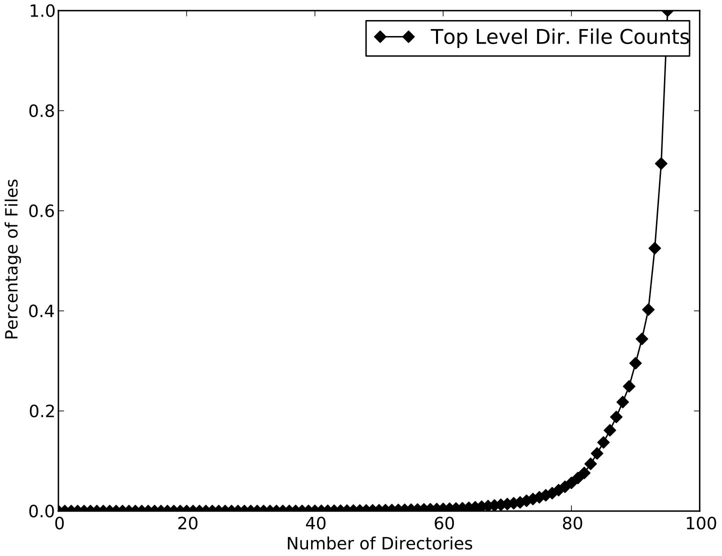


Figure 4.8: CDF showing the proportion of the total file count each populated directory contributes to the total number of files. Note this is the count on the last day of summaries, July 9 2010.

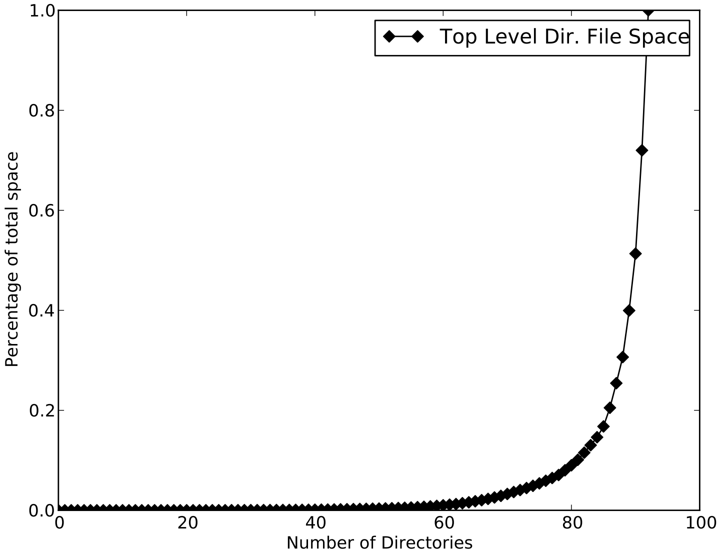


Figure 4.9: CDF showing the proportion of the space each populated directory contributes to the total. Note this is the count on the last day of summaries, July 9 2010.

overview of our methodologies, before diving into micro-analyses of several facets of the MSS activities.

4.3.1 Methodology Overview

For the purpose of our analysis we group actions into three broad categories. The first, which we call *user activities*, consist of actions that create, read or update a file's data. These actions almost always come from end-users of the archive. The second is *migration activities*: automated moves of data to and from the disk cache, as well as a small number of tape-to-tape migrations and integrity checks of data stored on tape. The third category of actions are *purges*, the permanent deletion of files within the archive. We first describe the actions associated with user and migration activities. We do not go into greater detail on purges since there is only a single action, also called a purge, associated with that category.

There are three actions that we group under the category of user activities with which we are concerned: creates, reads, and writes. A create is the ingestion of a file and all of its associated data into the MSS. A write is an *update* to a file already in existence in the archive. When we refer to user activities, we use the terms write and update interchangeably. A read is simply a read of a file's data.

There are two types of actions in the category of migration activities. The first is a *migration read*, which is simply a read of a file in preparation for writing the file elsewhere in the system. The second is a *migration write*, which is a write of the file data read from a preceding migration read. With the exception of a small fraction of aborted or mis-logged activities, every migration write has a corresponding migration read. Thus, most user activities (those that are to the disk cache) have at *least* two associated migration actions that will follow it at some point. As we show in Section 4.3.2.1, these migration activities are actually responsible for most of the data movement in the system.

Throughout our workload study when we calculate the number of bytes involved in any action or group of actions, we are summing the file sizes. As we lack data describing how much data is actually moved or written, we use the file size to approximate an upper bound.

Activity Type	Count	% of Total
All Actions	189,364,952	100%
User Acts. Total	65,980,770	34%
User Reads	22,272,374	12%
User Writes (Updates)	5,797,550	3%
User Creates	37,910,846	20%
Migrate Acts. Total	111,895,464	59%
Migrate Read	55,952,916	30%
Migrate Writes	55,942,548	29%
Purges Total	11,488,718	6%

Table 4.4: Summary of actions observed during the trace period. Percentage of total is the fraction of all actions counted. Migrate and user totals are the total number of actions within those categories. Subtotal percentages are approximate to ensure a sum of 100%.

4.3.2 NCAR MSS Analysis

We now provide a top down analysis of activities in the NCAR MSS, starting with aggregate system wide observations before moving on to user and file level analyses.

4.3.2.1 NCAR Aggregate Observations

From January 1, 2008 to December 31, 2010 we identified approximately 50.5 million unique files in our logs. From our sketch, we know the total corpus contained approximately 69 million files at the end of 2010; however our logs only account for files acted upon during the 3 years of observation. We only know of the total corpus file count from out-of-band communication with administrators. As we describe in Section 3.4, these communications were invaluable in aiding our understanding and observations.

Figure 4.10 provides a high-level overview of what the workload looks like over the course of a week. We see strong diurnal patterns linked to the workday [32], and note that there is significantly more activity due to file migrations than user actions, which we investigate more later in this section.

Figure 4.11 is a cumulative distribution function (CDF) of file sizes observed over the three years of data. *Count* refers to the fraction of files of a given size, while *volume* refers to

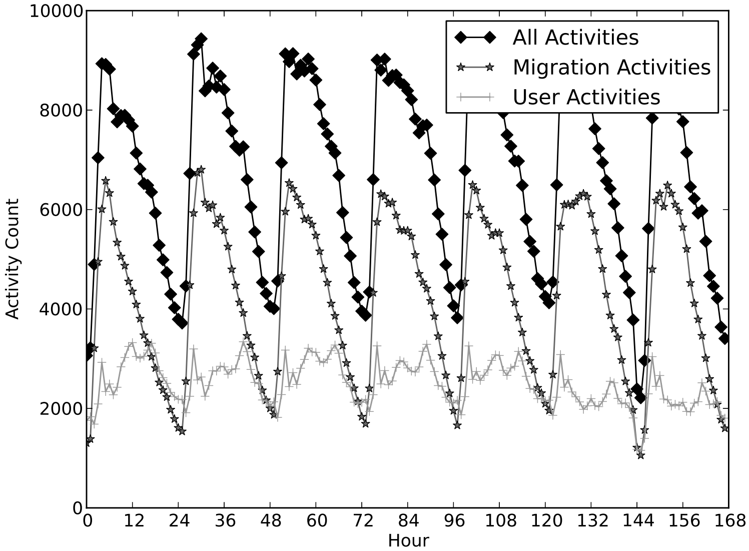


Figure 4.10: Averaged hourly activity rates for actions, exclusive of purges. The "all activities" line is the sum of the user and migration activities for that hour. Hour 0 corresponds to Monday at midnight. Around hour 144 was the weekly maintenance cycle, where all activities went to near zero.

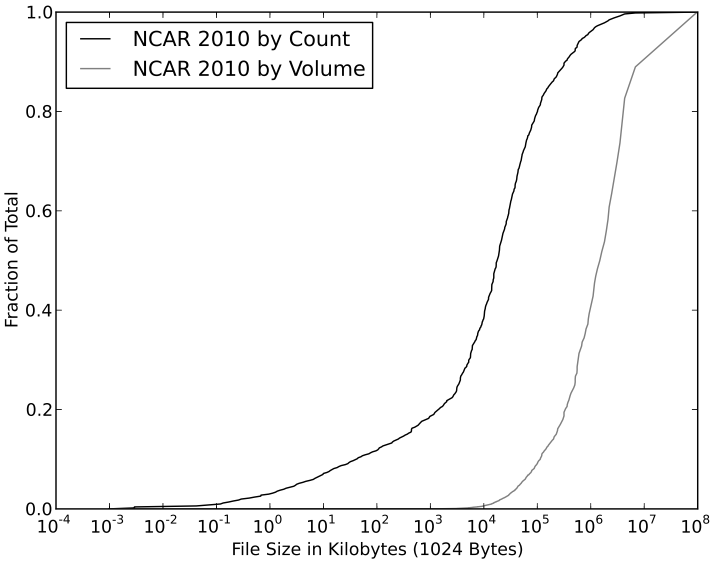


Figure 4.11: CDF of file sizes by the fraction of files of a given size, and the file sizes responsible for a given fraction of space.

the amount of space taken up by files of a given size [32]. We found that 80% of observed files are smaller than 100 MB, but most space is taken by files larger than 100 MB. The typical file size in the NCAR MSS is larger than that noted in Dayal's study of HPC systems at rest [26], though Dayal notes that there is significant variance in average file size from system to system.

In our first set of observations, we examine the fraction of actions devoted to file migration. This is of interest as "maintenance" and other supporting actions are easy candidates for optimization—they are predictable and often latency-insensitive. In examining these activities we found that automated file migrations make up the majority of data movement in the MSS.

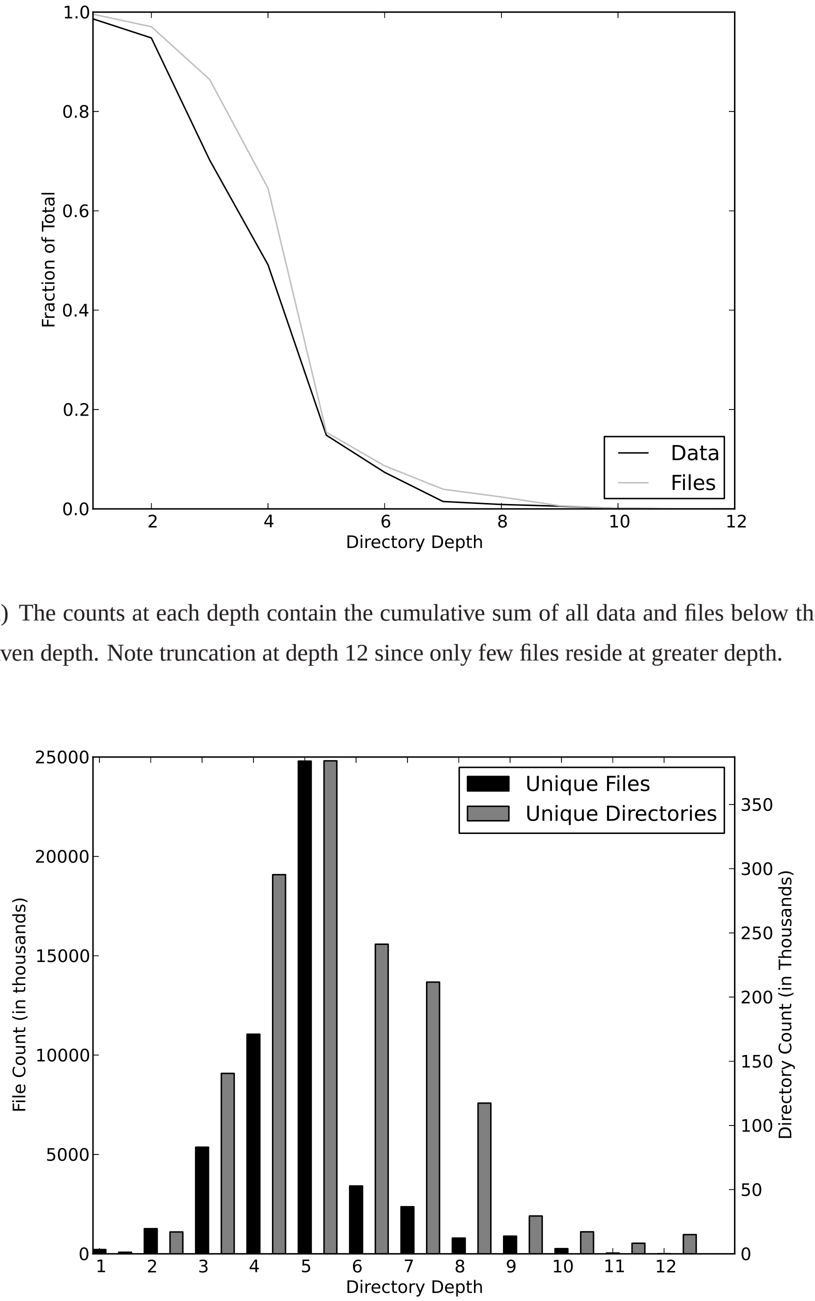
In Table 4.4, we show the number of user activities and migrate activities by count. Activities due to migration significantly outnumber those from user-sourced reads, creates and writes. This observation is consistent with conclusions drawn from both other archival and enterprise systems in which the dominant fraction of activities are automated "supporting" operations such as integrity checking and metadata manipulations [8, 24, 82]. The reason for this large number of automated actions in the MSS is the use of the disk cache as a staging area file for creations and caching file reads. If the data is not already on tape, it must be copied onto a tape prior to removal from the cache.

Over the period covered by our sketch, the archive was migrated to a new generation of equipment, ultimately resulting in the entire corpus being both read and written. However, our logs did not note activities from this data migration. This provides us with two key insights. First, the oft-quoted “Write-Once, Read-Maybe” assumption within archives is false from a system maintenance perspective. When we account for these inevitable technology driven data migrations, data is inevitably both read and written. There is still a grain of truth to “Write-Once, Read-Maybe” from the user perspective, but as we describe later, this assumption is not unequivocally true there either. Second, the lack of evidence of these migrations in our logs highlights the importance of communicating with system administrators and architects to understand the limitations of any data being provided. Without their input we would have been ignorant of the migrations from one set of hardware to the next.

We next examine the distribution of files and data across the namespace because it can offer hints as to how a system should physically group and organize its data. Figure 4.12 shows two views of the distribution of files at different depths of the namespace. As we explore in greater detail next, most files and data are concentrated around the same relative directory depth, though there is wide variation in the number of files and data in any given directory.

Figure 4.12(a) shows a breakdown of the number of files and bytes contained recursively at each directory as a fraction of all the data and files observed. For example, the root of the directory tree, depth 0, would contain 100% of the files and data since it includes everything beneath it. We include files that were purged when calculating the distribution of files and bytes because many files were only observed upon deletion.

In contrast, Figure 4.12(b) shows the amount of files and data at a *particular* level; approximately 90% of data and individual files are contained at depths three through five, inclusive. Although most files are at depths three through five, there is significant variation in the typical number of files per directory. The median at depth five is 11 files per directory, but the mean is 70 and the standard deviation is 607. Since the vast majority of directories contain a modest number of files and bytes, physically grouping whole directories on individual physical media is viable, and as we show in our analysis of user behaviors, may yield performance and efficiency benefits.



(a) The counts at each depth contain the cumulative sum of all data and files below the given depth. Note truncation at depth 12 since only few files reside at greater depth.

(b) Number of unique files and directories identified at a given depth. Note separate axes are scaled so that the columns for unique files and directories at depth 5 are the same height.

Figure 4.12: Two views of the distribution of files and bytes by directory depth. For example, `/user/foo/` would be at depth 2. The distribution is across 50.5 million files in 1.4 million directories.

4.3.2.2 NCAR User Behaviors

The raw data we obtained from NCAR did not have user actions grouped into sessions, so to approximate them we artificially grouped activities from individual users into temporally-based sessions. To do this, we used a sliding window of 15 minutes. Any actions (exclusive of purges since they were only logged once per week) that were within 15 minutes of the previous activity for a user were grouped into a session. Any actions that occurred after a 15 minute idle period were put into a new session. The 15 minute idle period was chosen by examining the number of sessions created as the window length grew. Selecting too small of an idle period resulted in many single action sessions, while using an idle period longer than 15 minutes yielded sessions with very few additional actions. Using this method we identified approximately 640,000 unique sessions.

We examine the distribution of activities on a per-user and per-session basis as it allows us to understand typical user-behavior, which in turn is often the behavior for which a system should be optimized. In our examinations we found that most actions come from a relatively small subset of users and sessions. Further, sessions are comprised of only one type of action.

We find that a relatively small fraction of users are responsible for most actions in the archive, particularly in regards to writes. 20% of the users were responsible for nearly 90% of the logged actions and 90% of the data volume accessed. At the session level, we see a similar distribution of data volume and activities. 10% of the sessions are responsible for nearly 90% of the logged activities and 90% of the observed data movement.

In Figure 4.13 we show a breakdown of the number of user activities occurring during sessions; the majority of sessions have fewer than 100 actions. Figure 4.14 shows the sum of file sizes seen in a given session, providing an upper bound on the amount of data that could be manipulated or transferred during the session. Most sessions (over 90%) act on less than 100 GB of data.

We find that sessions never mix user action types. Rather, a given session is comprised of just creates, just reads, or just writes. This makes sense on an intuitive level as it takes time after reading a file to analyze the data and then write results. Moreover, the archive is not a “scratch” space meant for interactive jobs where mixtures of reads and writes are common.

The primary implication we draw here is that there is likely a benefit to having a

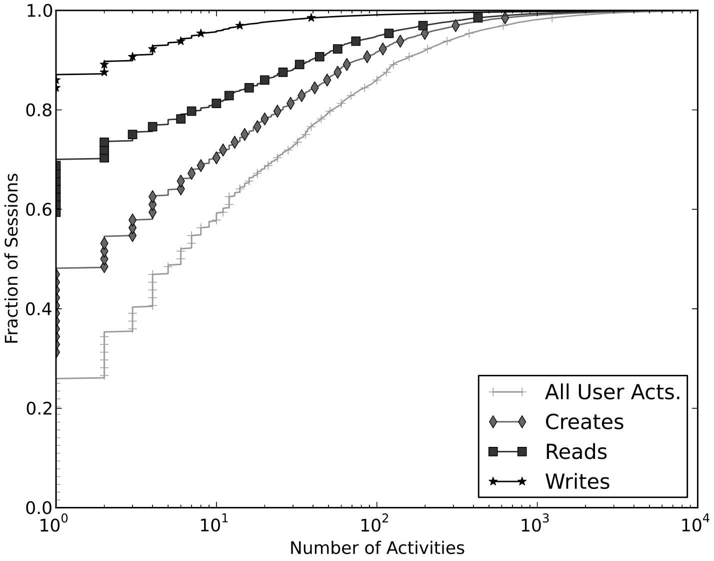


Figure 4.13: Breakdown of user activities per session. Note that each line has a unique number of sessions, thus changing the relative fractions at each level, with all user actions having the most sessions.

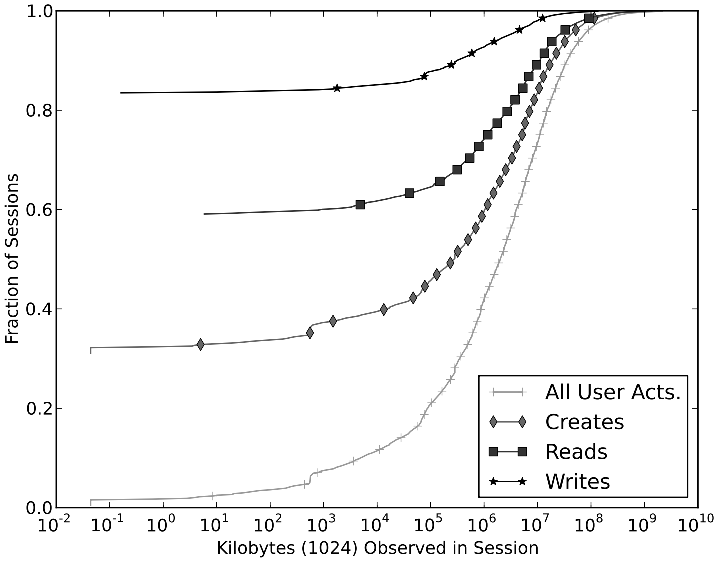


Figure 4.14: CDF illustrating the amount of data on which sessions act, broken down by user activity type.

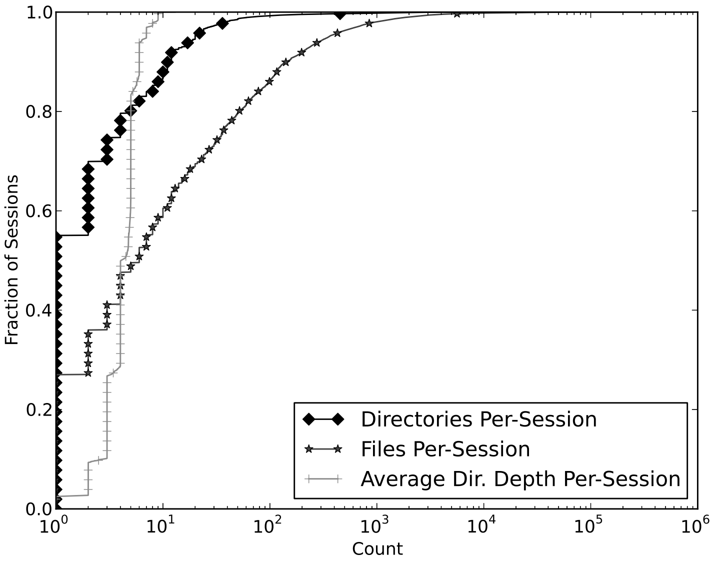


Figure 4.15: CDF showing the number of files, directories, and directory depth that sessions act at.

priority-driven, asynchronous batch interface for very large accesses. Presumably, larger accesses are less latency-sensitive than smaller accesses, providing greater flexibility than hard-coded policies, and allowing large accesses to be intelligently ordered and arranged around smaller, potentially latency-sensitive accesses. Working around latency-sensitive accesses is especially important for systems that may have limited concurrency or drive spin-up policies.

We next look at how session actions are distributed through the namespace hierarchy. This can provide hints as to how archival systems should physically group data to reduce seek times and media activations. Here we find that most sessions stay within relatively few directories.

In Figure 4.15 we show the typical directory depth, number of directories, and number of files touched during sessions. The number of directories accessed is typically nearly an order of magnitude less than the number of files, meaning that sessions access multiple files within each directory. The average directory depth per-session is typically (80% of sessions) less than 5, and the directory depth average almost always remains a whole number, suggesting that users tend to access files grouped *at the same depth*. Taken together with earlier observations, this finding suggests that it may be useful to physically group data based on user and directory depth. This technique could yield improved performance, minimizing seeks, and providing for easy pre-fetching and streaming reads and writes for tape and low-power disk-based systems. Physically grouping data by user was similarly done by the RASH portion of NASA's MSS-II in the late 1980s [40, 81] where data was physically grouped by user. Modern systems, such as HPSS [4], also provide tools for the automatic grouping of files to be managed as a single administrative unit. For tape based systems as well as archives built from spun-down disks [25, 76], this can be a boon because it would reduce the need for multiple media activations and mounts.

Observation:

We examine the total activity of users to see how many were effectively idle during the trace period. Depending on how the idle users' data is utilized, it may provide a heuristic for purging data. We find that many users did not have any operations beyond deletes associated with them.

We identified 1400 users that had user and migration activities associated with them—this is what we focus our session level analyses on. However, we found an additional 200 unique users that were only associated with deleted files. That is, the only activities associated with those users were purge actions. While we do not examine the temporal distribution of individual user activities, this suggests some users and their data—see observations on file lifetimes—may be transient members of the system.

4.3.2.3 NCAR File Level Behaviors

In this section we go deeper and look at the behavior of individual files within the system.

We examine file lifetimes for two reasons. First, archives are often considered to be immutable datastores, and as we show, this is not the case. Second, identifying how often, and when, files are deleted can help guide the organization and policies of a system. We find that around 15% of files are deleted within a year of creation.

In our analysis of file lifetimes, we only consider files for which we observed a create during the logged period. Of the 50 million observed files, we saw 36 million unique file creations, and 11 million deletes. We then correlated approximately 5 million deletes to files with observed creates. We focus our analysis on these files with observed creates as we can

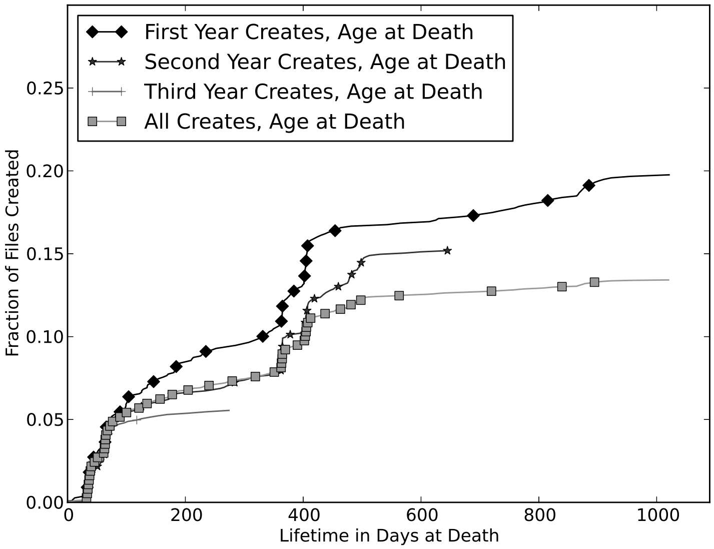


Figure 4.16: CDF of the distribution of the lifetime of files created during the trace. "*nth* year creates" refers only to files created during the *nth* year of the trace, while "all creates" refers to *all* files created during the trace period. For example, the "all creates" line ends at around 13.5%, meaning 86.5% of files that were created had not been deleted by the end of the trace.

precisely determine their lifetimes, illustrated in Figure 4.16.

We make two primary observations in this area. First, it appears that approximately 80% of the files created in the first year are in existence at the end of trace, suggesting that the majority of the files have long lifetimes. Second, of those files that *are* deleted, they are most likely to be deleted from the system within one year of creation. While this strong temporal spike is due in part to file retention policies—360 days is one of the most common—it actually makes for a stronger basis in estimating user intentions as they are forced to explicitly decide which files will remain in the system. This is further reinforced by the fact that users are charged for data stored in the archive.

These file lifetime results are interesting for several reasons. First, archives are clearly not immutable data stores—a significant number of files are deleted. Second, while the notion of files being deleted relatively quickly, or not at all, is not new [17, 51, 67], the deletes on the archive occur after months, as opposed to within seconds of creation on enterprise and personal storage. Because of these behaviors it may be useful to add another logical "probationary" level to the storage hierarchy: a place where files that are likely to be deleted are stored before entering a more "permanent" state in the system.

We examine the distribution of actions upon files because it can strongly impact caching policies. When examining file activities, we do not include files that are only acted on by a purge. Thus, we study around 43 million files. We find that most files receive few actions. Exclusive of migration activities, 65% are only acted upon once in three years.

Our observation is that, as a whole, actions are evenly distributed across files, and that most files are the target of relatively few actions, as shown in Figure 4.17, which illustrates the distribution of actions across files. The line is relatively flat, indicating that actions are evenly distributed across files; a vanishingly small fraction (less than 0.1%) receive hundreds or thousands of activities. This is further illustrated in Figure 4.18, where we show the distribution of files by activity count. Across all user activities, 70% of the files we observed only received a single action, usually the initial create of the file.

Our suggestion, based on this observation, is one that NCAR already implements: use the disk cache primarily to absorb creates/writes. Individual file activities are sufficiently spread out and rare that read caching in general would be largely ineffective. The small number of files that are very active, however, could easily be serviced on a disk cache.

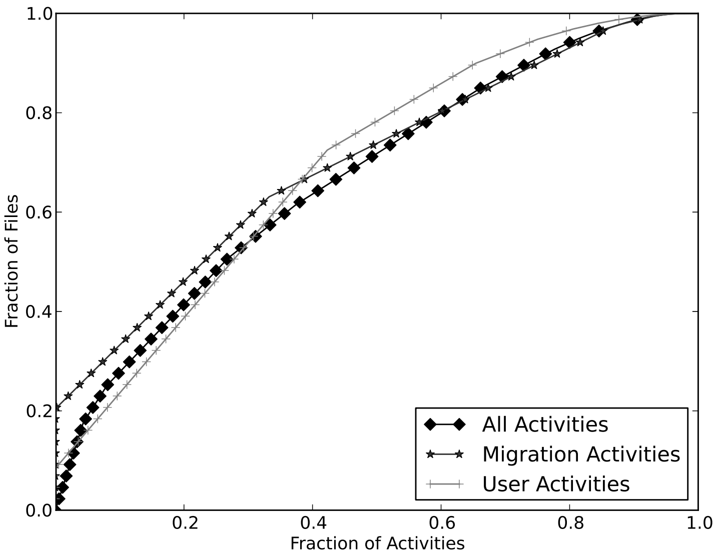


Figure 4.17: CDF showing the fraction of files responsible for a given fraction of total activities. Note the user and migration plots do not start at 0 files since not all files had only user activities or only migration activities.

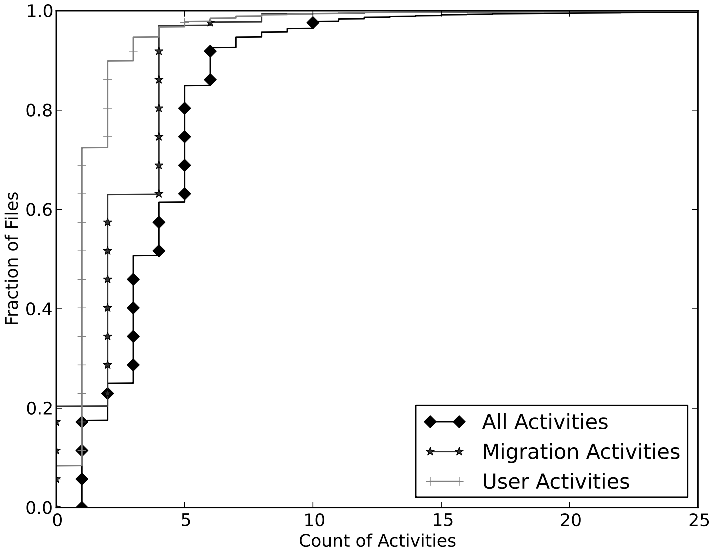


Figure 4.18: CDF of the fraction of files receiving particular activities from users (reads,writes, creates) and migration processes. We truncate the count at 25 because only a very small fraction of files receive hundreds or thousands of activities.

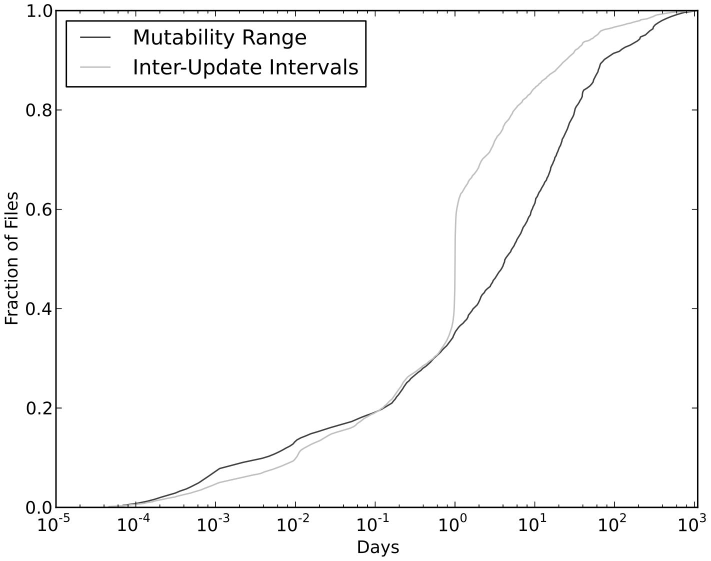


Figure 4.19: CDF showing the inter-update interval of files that receive more than a single mutation (more than a single create OR single write), as well as the time range observed between the first and last mutation of a file. These files account for approximately 2.5 million of the observed 50 million files. The x-axis is on a log scale.

We examine the mutability of files since a common assumption in archival design is the notion of “Write-Once” files. This can influence caching policies and how data is organized within the system. We see that approximately 5% of files had updates to their data with most updates occurring within one day of another. A small subset of files receive updates over long periods of time (longer than 100 days).

Before we move on to our observation, we describe what we consider to be an update, or *mutation* to a file. If a particular file has been created multiple times, all creates after the initial are counted as mutation to file state. A second create to a file overwrites the original data; it may be identical, or entirely different. Any write to a file is treated as an update to its data.

In Figure 4.19, we show the interval of time between successive updates to the same file. A file is not accounted for if it does not have at least two creates and or writes. Thus, we observe inter-references for 2.5 million files, around 5% of the unique files observed during the trace. Most updates (roughly 65%) occur within one day of another update. After this, we see a marked increase in the interval of time between updates. The next 20% occur within ten days

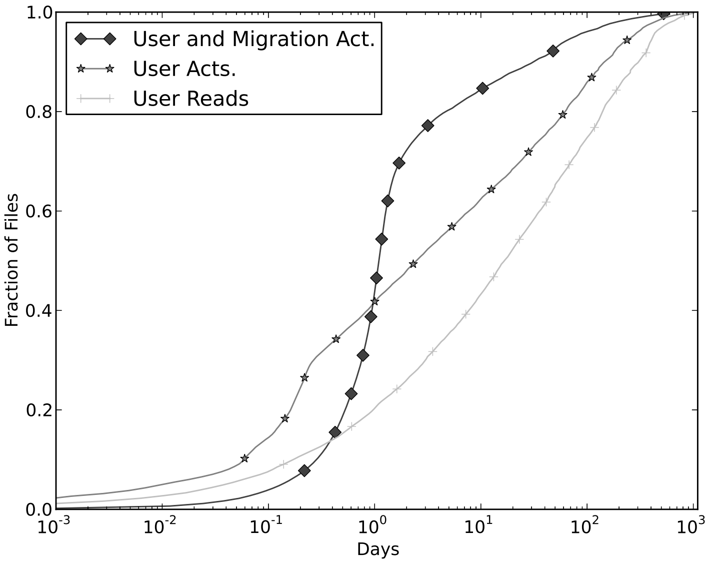


Figure 4.20: CDF of the inter-reference interval files under several different filters. “user and migration act’’ includes all user and migration activities, while “user acts.” includes reads, writes and creates, and “user reads” includes just reads. Note that 100% only accounts for files touched by the respective activities, and files only referenced once are not counted because we are unable to calculate an interval for them.

of each other, and the subsequent remainder occur between 10 and 1000 days.

Figure 4.19 also shows the range of time between a file’s initial creation (or first observed write) and its last update or overwrite, which we refer to as the *mutability range*. 40% of the files for which we calculate a mutability range have ranges greater than ten days.

To further explore a file’s mutability we count the number of potential updates a file could receive, only counting files that have at least one observed update. This could be either a create action overwriting an existing file or a write action updating a file’s data. Over 75% received a single update, and over 95% of files receive fewer than ten updates. The remaining files receive anywhere from 10–100 updates or more. Given the relatively short inter-update time for file data, caching should be able to absorb most updates before writing back to the archive.

We examine inter-reference intervals for two reasons: to explore the impact that file migration has on a file’s inter-reference period, and to measure the frequency of accesses to a file. Both factors may impact caching and data placement policies. When calculating the inter-reference interval including migration activities, we treated each migration (read and write, possibly to multiple copies of the same logical file) as a single reference, preventing a misleadingly high inter-reference count for files, particularly those with multiple copies.

When we account for automated file migrations in a file's inter-reference interval, file inter-reference intervals appear superficially shorter. Considering only user actions causes files to exhibit longer inter-reference intervals, albeit with fewer files accounted for.

Figure 4.20 shows the inter-reference intervals for files under a variety of filters. Note that in order to be counted a file must have had at least two actions under the relevant filter. When we include migration activities, the general inter-reference interval appears superficially shorter, due to daily migrations of files off of the disk cache. However, when migration activities are filtered out, we see a larger fraction of files under consideration with longer intervals, though this includes *fewer* total files. In essence, automated migration processes can skew how the workload is perceived. Consider, for example, how much the full migration due to the tape library upgrade would have skewed the overall inter-reference period.

The second observation we make is that when we are only concerned with reads, the files that do see repeat reads (approximately 2.2 million) can see very extended periods of time between them. This lines up with the anecdotes we were given about a typical use case: retrieving stored data at long intervals to validate old experiments and seed new ones.

4.4 Combined Analysis and Implications

Our experiences in analyzing activity on the general scientific (LANL) and NCAR corpora lead to suggestions for long-term tertiary storage system designers.

Bring back the batch interface. We found that most accesses occur from a subset of users and sessions, and are often very predictable, *e.g.* data migration processes and large user-sourced scripted jobs, similar to the large-scale access behaviors we saw even in the public content archives. With these behaviors, there may be significant benefit gained from an asynchronous batch interface that allows a scheduler to intelligently group and place writes while scheduling around more latency sensitive processes. Because end-users are often less tolerant of latency than something like an integrity checking process, the integrity checking can easily be given a lower-priority and run in batch mode in the background, improving user quality-of-service while still providing for administrative and maintenance tasks.

User and namespace grouping may aid archives with offline media. We believe that physically grouping data based on user and namespace heuristics may prove fruitful. We noticed that in the NCAR corpus files and bytes tended to be concentrated around the same levels and that user sessions tended to act within the same directory level. Similarly, we saw temporal and namespace locality in the general scientific LANL corpus. In addition, in the NCAR corpus few files were shared across many users, so grouping by user, like that done in the MSS-II System [40, 81] may be a simple but effective approach to physically grouping data on media, and modern tertiary stores such as HPSS provide tools to largely automate such grouping [4]. This grouping is of prime importance for archives with offline media, such as spun-down disks or unmounted tapes, since they incur high seek penalties and startup costs. This suggestion ties in with the previous suggestion for a batch interface: by delaying groups of writes and batching them, we can better group them based on their user and relationship to the directory hierarchy. If rich metadata are available, more intelligent file grouping and placement techniques can yield significant improvements in access times as demonstrated by Chen *et al.* [23] and their work on multi-dimensional grid data on tertiary storage systems. Other types of grouping may also be relevant such as *filecules*, groups of co-accessed files that are dynamically identified, suggested by Doraimani and Lamnitchi [27].

Write-Once, Read-Maybe doesn't always hold. An interesting conclusion we have come to is that from *user* perspective, the old assumption of “Write-Once, Read-Maybe” is not unequivocally true. As far as “Write-Once” is concerned, while most files (95%) were not updated, a non-trivial fraction were eventually deleted from the archive. A smaller (5%), but also non-trivial fraction received one or more updates to their data. These updates came either through explicit updates to the data, or complete overwrites.

The assumption of “Read-Maybe” appears to at least superficially hold. However, while three years is an enormous length of time compared to most prior studies, we have only seen a fraction of the potential behaviors if the data is intended to survive in perpetuity. We hypothesize that continual data growth rates may superficially mask the real fraction of an archive likely to be read. Consider that we know via communication with administrators that data is often revisited up to *five* years later, yet our logs only covered *three* years of activity. However, when we consider the silent migration of data to new technologies, “Read-Maybe” conclusively becomes “Read-Eventually”. As long as the file otherwise survives, it will inevitably be read as it moves to new media.

All told, it is dangerous to rely on any rigid assumptions about the expected read behavior or mutability of files in a system, especially in light of potentially unpredictable impacts of hard policies on user behavior. For example, Holloway found users often attempted to subvert file retention and migration policies through the use of scripts to update file metadata [43].

4.5 Chapter Summary

In this chapter we re-examined the behavior of scientific tertiary storage systems. In our study of a LANL archive we found modification behavior that is similar to that of studies from 20 years ago. We also found that while file sizes have grown some, the general trend that most data in the system is composed of larger files, while most files are relatively small has continued. However, we could not do a detailed access behavior analysis given the coarse granularity of the data we obtained from LANL.

To address this, we have analyzed three years of logs detailing activities on the NCAR mass storage system (MSS). We found that much like public content archives, most activity comes from automated behaviors. We also discovered that while a small fraction of users are responsible for the majority of activity, this activity is widely spread throughout the corpus. On a per-user level however, individual users appear to show strong locality, leading us to suggest that physical grouping techniques based on namespace and content can lead to performance and efficiency improvements.

Chapter 5

Combined Workload Discussion

> I wanna make a jigsaw puzzle that's 40,000 pieces. And when you finish it, it says 'go outside.'
>
> Demetry Martin

In this chapter we provide a high-level analysis of all of our workload studies in aggregate. We explore the common and divergent characteristics across both the public content and scientific tertiary archives. We also cover a variety of lessons learned in workload analysis and provide advice and best practices for improving future workload studies, which in turn lays the ground work for our last chapter that looks towards validating trace-logs.

5.1 Workload Characteristics

One of the most surprising results to come out of our workload analyses is how similar the per-file/record popularity is across disparate corpora, illustrated in Figure 5.1. The water corpus was 4.5 GB of data, the historical around 10 TB, and the NCAR over 10PB. Despite these enormous differences in scale and intended use case, the relative popularity of records and files accessed (recall that we only calculate based on the files actually accessed, not those never touched during the trace) is very similar. This suggests that across the board, any sort of long-term hot-cold grouping or read caching is not likely to be effective.

Another commonality we saw across the various corpora was the prevalence of large, automated accesses to the system. These came from a variety of sources, ranging from external

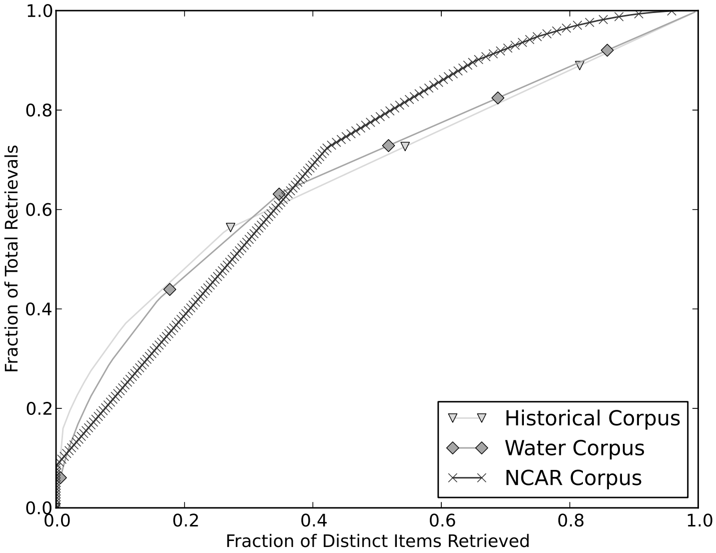


Figure 5.1: CDF of file/record popularity as a fraction of all accesses, broken up by dataset.

indexing service such as Google, to internal administrative processes such as integrity checking and media migrations. Given how common and heavy-weight these activities are, we reiterate our suggestion for a separate asynchronous administrative interface to handle these operations. Such an interface can simplify administration by removing the need for many separate static policies for common tasks such as indexing and migration, and further schedule these latency-insensitive tasks around the much more latency sensitive end-user.

However, despite all of the observed corpora having some form of large-automated access, the underlying causes and magnitudes were not universal. For example, in the water corpus Google was responsible for 70% of all retrievals, but this was entirely absent in the NCAR corpus as the system had no interface to through the Web. Similarly, the historical corpus had an aggressive integrity checking policy (all data and metadata was checksummed once per month) that was absent in both the NCAR and water corpora.

In all of the corpora, the notion of write-once, read-maybe was plainly false from the system perspective. Any corpus that is migrated across hardware (a common activity given the long lifetime of archival data) will inevitably be read and written. There are also integrity checking and indexing processes that do wholesale reads of archival data. From the user perspective write-once, read-maybe still has a kernel of truth to it, but is weak. We found that exclusive of system sourced reads, most archives only had 5-20% of the their files/records read. The number of records or files being updated was widely variable, with less than 5% of files in the NCAR corpus receiving updates, up to 100% of files in the water corpus. All told, this illustrates the danger of over-reliance on rules of thumb, as well as overgeneralization of any one data point.

The relatively high rates of file updates we observed also highlighted the unpredictability of humans in any system. In the NCAR corpus we found that many of the updates came from a single user who may have been using the long-term archive for backups (the data was ambiguous). In the water corpus, all reports in the archive were overwritten once a quarter due to a mix of policies that made it possible for data updates to be otherwise missed, and it was simpler for them to simply re-run all scripts once-per quarter to ensure all data was up to date. A knee-jerk reaction to these sorts of activities would be to implement a hard policy, but as Holloway showed, policies that users find onerous will often be circumvented with potentially pathological impacts on the system [43].

While not covered in this thesis, a relevant side investigation we did was on how a scientific archive evolves. NCAR has been dedicated to the same type of research in the years since Miller and Katz's 1993 study, providing a unique opportunity to do an apples-to-apples comparison to the 1993 study. We identified, with far less noise than would otherwise be possible due to NCAR's narrow mission scope, evolutionary trends over several decades of hardware and software evolution. We found several results of note, including an inversion of read-write rations (the system changed from write heavy from read heavy), and an increase in the latency to first byte for tape-based systems. Details are provided in Frank *et al.*'s study [33].

Object storage devices [31], where data is manipulated as objects rather than individual blocks on a hard drive, are of growing interest in the storage community. Our architectural suggestions should work equally well in object stores. The same basic grouping techniques will work with object stores, if not better, as object stores divorce the underlying details of the storage device from the actual use of the device itself. For example, using hierarchical namespace clues (directory and file paths) to group data becomes simpler in an object store as the dictated 'path' is completely independent of the underlying structure of the stored data. In contrast, traditional hierarchical file systems rely on the directory structure to physically lay out and locate data on devices.

One of the original sub-objectives of our study of archival storage systems was to develop a taxonomy of archival storage characteristics and workloads. While we have found a variety common and divergent characteristics across archives, as mentioned above, we are leery of over-generalizing our results. Given that most of the systems we studied had radically different use cases, it would be premature to concretely define characteristics of any particular category of archival use.

5.2 Lessons in Logging and Tracing

In our work, we have run into many challenges in interpreting and understanding the data we have obtained. We learned several lessons that can aid in future tracing and analysis that informed our log-validation work described in the next chapter.

Communication is key. If there is a single lesson we have learned in our experiences, it is the importance of communication with system administrators and architects. Time and again we relied on them to understand the system architecture and artifacts of the workload as well as clarify what we did and did not observe in the logs. For example, in the NCAR corpus, without their input we would have been ignorant of file migration due to hardware upgrades. As another example, this time in the water corpus, without administrator input we would have been unaware of the silent file renames, drastically changing our results throughout. Without the shared knowledge provided by system experts we would have had a significantly degraded and even potentially even inaccurate understanding of the archive.

A system start state is invaluable. A good "complete" analysis of a system depends on having *both* snapshots and an activity trace; just one or the other is often insufficient to answer many questions. For example, in this the NCAR dataset we lacked a view of the start state of the system that a snapshot could have provided. Because of this we were unable to answer questions such as what fraction of files were left entirely untouched or what fraction of the namespace does a given user occupy. However, having access to only snapshots without a dynamic trace can be equally difficult because it limits the ability to understand the source, timespan, and magnitude of activities occurring in a system.

Don't add semantic meaning to field values. One specific issue we ran into with the NCAR sketch was understanding semantic meanings encoded in field values. For example, we observed situations where the base directory of a path was subtly and silently renamed, *e.g.* `/USER/Foo/bar` to `/uSER/Foo/Bar/` to denote file being in the "trash". This caused our unique file counts to be off by 10% and our directory counts to be off by nearly 20%. Luckily we were able to communicate with an administrator to identify and understand the nature of these silent renames, but this cannot be relied upon for all datasets. As such, we stress to any group that may provide data to others to avoid using field values to encode non-intuitive information unless they are carefully documented.

Be consistent, or be visible. We encountered several subtle format changes in logged activities across our datasets, including changes as trivial as a field moving one character over or swapping field locations. At best, they cause annoying parser errors and force data reprocessing with a band-aid fix. At worst, they can silently corrupt results and be extremely difficult to detect. Our suggestion is to ensure that format changes will break a parser, provide an external method for keeping the logs in a consistent format, or document the changes.

Tag or explain activity drop-offs. Sudden reductions in activity rates are difficult to deal with in an analysis since it is often difficult, if not impossible, to determine the true cause of the reduction, *e.g.* it could be due to an actual reduction in the number of activities, a crash, or even a holiday. Without understanding what caused a dip in activity, we have to treat it as null data and discard it, lest we misconstrue what is occurring. Our proposed approach is to have the logger take a more proactive approach and explicitly note when the process that generates the log entries fails or otherwise times out. Another simple fix is to have the logger note any time it starts in the logs it keeps. This can help identify times when the logger is simply inactive, versus ones that are a legitimate reduction in activity rates.

Identify the coverage of a dataset. A question we are always asking about any trace or log we have obtained from outside sources is "what *aren't* we seeing in this trace?". Put another way, we don't know what the *coverage* of a dataset is. For example, we only knew about the data migration from one set of hardware to another via out-of-band communications with system administrators. Our proposed solution is to take a snapshot of a system's state immediately prior to the start of logging/tracing, then after completion of the trace take an additional snapshot. At this point, use the first snapshot and the trace as a delta to create a third *expected* snapshot. We can then compare this expected snapshot with the one taken after the trace and identify where they differ, and help verify what the trace is and is not covering. This idea provides the basis of ExDiff, described in the next chapter.

Chapter 6

Log Validation Using Metadata Snapshots

> The logic of validation allows us to move between the two limits of dogmatism and skepticism.
>
> Paul Ricoeur

Our original direction was informed by our problems with long-term storage system logs comprised of data gathered over multiple years, however, storage system activity logs are used in a number of diverse areas. These areas range from system administration and design to security auditing. A latent problem in all of these applications is identifying the *coverage* of the collected logs; it is critical to know which events have been captured and which have been omitted. With no form of validation or understanding of a log's coverage, it is easy to form incorrect conclusions from log analysis.

For example, consider a system that silently drops entries due to a logging-buffer overflow or logging-process crash. A security or performance audit of the system may mistakenly conclude that the system is behaving correctly, as no warning messages have been logged. Similarly, unless the developers who instrumented the system are present, it can be difficult to identify precisely which activities are and are not being captured. The latter can be a particularly vexing problem for debugging a system, as well as for anyone trying to analyze a system from captured traces.

To address this issue, we have developed a methodology we call *ExDiff*, that uses *expectation differencing* to determine when the true state of a system diverges from the expected state. ExDiff uses an initial file or object-level metadata snapshot [10, 34], and an activity log to derive the *expected* state of a system. This expected state of the system is compared to a second metadata snapshot capturing the system's current *reality*. ExDiff uses the resulting set of differences between the expected and real state to identify logging omissions from crashed logging processes or unrecorded system activities.

Using ExDiff, we can accomplish three key validation tasks. First, ExDiff can identify both when and for how long a logger may have dropped entries. Second, it can highlight when there may be activity that has *not* been captured. Third, it can aid in identifying the specific actions, such as a file creation that may have been dropped from a log or not captured in the first place. Providing the ability to accomplish these taks is our primary contribution in this chapter.

Using a variety of simulated workloads and snapshots, we demonstrate ExDiff's ability to retroactively identify periods where log entries are being dropped but the underlying system is still functioning. We show ExDiff can accurately identify gaps in logs with as many as 500,000 actions, and given the relatively low rates of activity we've observed in our prior studies, this should work very well within archival storage settings. Additionally, we analyze how log duration and the density of actions affect ExDiff's accuracy. We also detail how ExDiff can be used to identify specific types of dropped entries, such as an entry noting a file permission change, and how the same issues that influence accuracy in recognizing gaps in log coverage can impact the ability to identify missing entries.

Note that in our work we use the terms log and trace inter-changeably, and that a gap *estimation* (recall a gap is a contiguous period of dropped entries) is a pair of timestamps predicting the start and end times of a gap. When describing gaps, it is important to distinguish between wall-clock time and the number of dropped entries; a gap in wall-clock time may involve any number of actions.

6.1 ExDiff Design

ExDiff operates at the level of file or object metadata. We do not consider the raw data, although ExDiff could be extended to incorporate data capture by capturing content hashes. Note that while the overall methodology is agnostic to the underlying trace and snapshot capture methods, the data that comes from these captures will be specific to individual systems.

The ExDiff process (explained in greater detail in the following paragraph) could be used for a variety of applications and systems well beyond our original intent of aiding trace

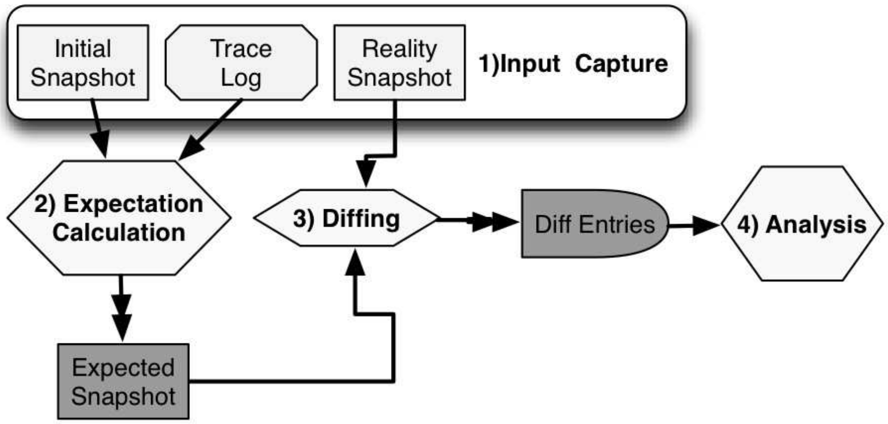


Figure 6.1: An overview of ExDiff's workflow. Yellow entries are captured data from input capture, while green (diff entries and expected snapshot) are derived.

analysis. One potential application is as part of a periodic log verification process. Whenever an ExDiff run finds unexpected modifications to the system based on logged actions it can alert an administrator of the issue. Another use is in verifying that the necessary areas of a system have been instrumented. Consider a cloud service that charges per action. Assuming each billable action modifies some metadata state, ExDiff can be used to verify that all relevant actions are being properly accounted and charged for.

There are four steps to ExDiff. The first is *input capture*. In this step an *initial* metadata snapshot is taken, followed by activity tracing, and finally a new *reality* snapshot is captured. The second step is *expectation calculation*, where we combine the initial snapshot and the activity log to derive an expected snapshot. The third step, *diffing*, is where we take our expected snapshot and compare it to a *reality* snapshot, generating a list of differences. The fourth and final step is *analysis*, where we utilize the list of differences, the activity trace, and snapshots to analyze log coverage. Figure 6.1 illustrates ExDiff's workflow.

Input Capture: In the input capture step, the initial metadata snapshot, activity log and reality snapshot are gathered. The initial snapshot is a picture of the metadata state of the system immediately prior to a trace log of actions. The reality snapshot captures the state of the system at the end of a tracing period. While both snapshots represent the ground truth of a system's state (we assume the file system metadata is correct), we refer to them as the initial and reality snapshots to keep them notationally distinct. The log must capture actions *between*

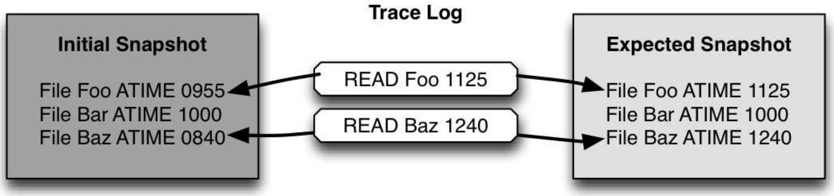


Figure 6.2: Expectation Calculation. In this example, the expected state is derived by mapping file ATIME's to read activities.

the initial and reality snapshots. Note that more than two snapshots can be captured, but ExDiff only calculates coverage between pairs of snapshots.

For ExDiff to function, the underlying system needs two characteristics. First, one or more action entries in the log should reflect changes to the system's underlying metadata. For example, if a snapshot captures a file's permissions and a logged action notes changes to that file's permissions, we can then use that entry to predict the new state of that file's permission metadata. Second, in order to estimate gaps, we require one or more metadata timestamps that can be accurately mapped to activity log entries. For example, a read entry that can be mapped to a files ATIME.

While some file systems provide snapshots through versioning [42, 70], capturing snapshots is not always atomic, such as a recursive `ls` and `stat` to capture metadata. Actions may continue to mutate a system's metadata state as a snapshot is being captured, which in turn influences the diffing and analysis steps in difficult to predict ways. In our work, we assume snapshots are captured atomically, and relegate addressing non-atomic snapshots to future work.

Expectation Calculation: ExDiff uses the activity log to update the state of the initial snapshot and create the expected snapshot, a prediction of the system's metadata state. As illustrated in Figure 6.2, this process is straightforward: an action in a log may update one or more parts of a file's metadata. For example, in many file systems, a data modification will update the change time (CTIME), the modification time (MTIME) and the file size metadata. How, and which, actions should be mapped is specific to the snapshots and actions being captured. Though this is a human driven, and potentially error prone, errors in mapping can be caught in the diffing step and highlight misunderstandings of a trace's coverage and the semantics of its actions, which is useful in its own right.

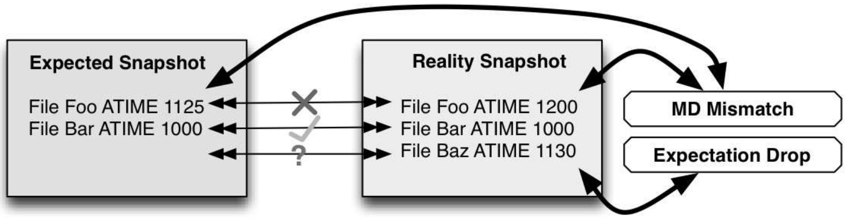


Figure 6.3: Diffing. File Foo's expected ATIME does not match reality, so an MD mismatch entry is produced. Similarly, Baz is not seen in the expected snapshot, so an expectation drop entry is created. Bar exists as expected, so produces no diff entry.

When deriving the expected snapshot, *partial* entries may be created. A partial entry is created when there is an attempt to map an activity to file that is not known to exist based on the current log and expected state, so ExDiff populates as much metadata as possible for that particular file.

Diffing: After the expected snapshot is created, we compare it to the reality snapshot and collect the differences between the two for analysis. As shown in Figure 6.3, ExDiff does a file by file comparison of the two snapshots, comparing each piece of metadata to each other, with a runtime of $O(N)$ as it is just a comparison of each file to its counterpart. Any time there is a mismatch, either from a file being missing in one of the snapshots, or a piece of metadata not matching as expected, we pull the files out and create a *metadata diff entry*. Each diff entry also tracks which metadata came from the expected snapshot, and which came from the reality snapshot.

Each diff entry is categorized as one of three types, summarized in Table 6.1. A *reality drop* entry is where a file is found in the expected snapshot but is missing from the reality snapshot. An *expectation drop* entry, is the reverse of the reality entry; a file exists in the reality snapshot but was not found in the expected. A *metadata* or *MD mismatch*, is where a file exists in both the reality and expected snapshots, but one or more metadata fields do not match.

Analysis: ExDiff now diverges into two distinct types of analysis. The first is identifying gaps in log coverage. The second focuses on classifying log omissions to provide clues as to which specific actions were omitted from the log.

Gap Identification: Given the requirement that at least some logged actions update timestamps, ExDiff can leverage mismatches between expected and reality timestamps to iden-

Diff Type	**Description**
Reality drop	File is in expected, but not reality snapshot
Expectation drop	File is in reality, but not expected snapshot
MD mismatch	File exists as predicted, but metadata does not match

Table 6.1: Metadata diff entry types.

tify gaps. When an action that updates a timestamp is dropped from the log, it will lead to an MD mismatch as the expected and reality snapshots will not match on one or more timestamps. These mismatched timestamps can be used to identify likely log gaps. Consider the example shown in Figure 6.3. The metadata mismatch notes that file Foo has an access time of 1200, while the expected entry gave Foo an access time of 1125. This tells ExDiff that an action that occurred at 1200 was missed or dropped.

When identifying gaps, ExDiff pulls out all of the diff entries that come from the reality snapshot which have mismatched timestamps. A density based clustering algorithm is then run to group timestamps together based on their temporal distance; large numbers of similar timestamps from diff entries are indicative of a gap. After clustering, the earliest and latest timestamps from each cluster are presented as a gap estimate.

We use DBSCAN (density based spatial clustering of applications with noise) for creating gap estimates [30]. We chose DBSCAN due to its simplicity, the fact that it does not require detailed knowledge of the underlying data distribution, and its ability to deal with noise data points. Its ability to automatically handle noise is relevant because we have run into situations where loggers periodically drop random individual entries in addition to full coverage gaps.

DBSCAN has two parameters, N (neighborhood size) and eps (shape parameter). Clusters are produced when a datapoint has at least N other datapoints within a distance of eps. Any datapoint that is within eps of an already identified cluster is merged into that cluster, and all others are discarded as noise. As with most clustering techniques, DBSCAN's parameter choice can influence its accuracy. We explore how varying DBSCAN parameters influences gap identification in detail in Section 6.3, but relegate automating parameter choice to future work. Figure 6.4 provides a simple example.

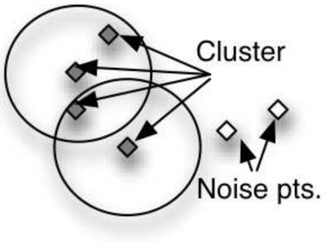


Figure 6.4: An example of DBSCAN clustering. The circles represent the EPS value, n is 3. Blue points are a cluster and the unshaded point is noise.

DBSCAN's primary bottleneck is in its use of a distance matrix, with a memory and run time overhead overhead of $O(n^2)$. With large datasets we can use DBSCAN in conjunction with a sliding window approach on the input timestamps. For example, if we have a 30 day log of actions, but can only fit a days worth of diffs into memory at a time, we can use a 24 hour window, moving it 12 hours at a time. The windowing will not impact estimate accuracy unless a gap is longer than the window.

It is possible, however, to *mask* omissions, and subsequently gaps. Consider a file with one timestamp that is updated every time it is read or acted upon. If the last action to a file is dropped, it will not be reflected in the expected snapshot and will show up as an MD mismatch entry. However, if another action occurs after the dropped one, the expected and reality comparison will not trigger any diffs, as the expected and reality timestamps will match by both reflecting the result of the second action. We examine what influences masking in greater detail in our evaluation in Section 6.3.

Omissions Identification: In classification of omissions, ExDiff examines the metadata diffs for clues as to the specific types of underlying actions that were omitted (either a drop or a miss) in the log. The key to classifying an omission is recognizing its *drop signature*; the set of diffs and mismatched metadata produced by a particular missed or dropped action. Signatures are specific to both the snapshot metadata and the actions explicitly captured in the trace, and thus may vary from system to system. If one already has detailed knowledge of the expected operations within the system (whether or not they are captured in a given trace), and how they are expected to modify metadata, signature identification is relatively straightforward. However, even if the details of the underlying system are not perfectly understood, there are many common operations that will leave predictable signatures when missed, such as a delete which always leaves a reality diff entry. Missed actions (those that are not capture but otherwise modify metadata) still generate diffs, which in the worst case still alerts the analyzing party that their log is missing actions. Given that signatures will vary depending on the underlying system and trace methodology, we describe signatures specific to our evaluation in Sections 6.2 and 6.3.

There are a few promising methods we have found for choosing eps. One method, is to simply “eyeball” a graph of activity rates over time, and pick based on the length of the suspected logger failure(s). An alternative is to maintain running averages of logged actions, and when there is significant deviation from said average, identify how long the trace remains beneath that average to inform the choice of eps. Picking N is dependent on the amount of noise and the estimated gap sizes. If we already know the log is reliable and does not randomly drop entries, any small value (*e.g.* 10) should work reasonably well.

6.2 Experimental Design

Workload and Snapshot Generation: We chose to generate synthetic workloads and metadata for ExDiff’s validation. We took this route for two reasons. First, we require a variety of snapshots and workloads with verifiable ground truth in order to check the accuracy of our methods. With real world traces we ourselves would not know their coverage, weakening our evaluation. Second, we need the ability to fine tune the workload to examine how various actions impact ExDiff’s accuracy.

Each file in our workload corpus has common, POSIX-like metadata, described in Table 6.2, and is uniquely identified by its filename; in our simulations this is a simple numeric identifier. Generated activity logs are comprised of timestamped actions based on common, POSIX commands. We summarize these actions in Table 6.3. Timestamps are integers, and all actions have a unique timestamp. Currently, log timestamps and metadata trace timestamps match. We relegate metadata and log time skew to future versions of ExDiff.

Activity log entries consist of two elements: an action, and a file to perform the action upon. Actions are randomly picked based on experiment-specific parameters. Files are either picked randomly or they are selected with locality, based on the experiment. We use the random picking as a control group as it is easy to understand and analyze, while picking with locality is

Field Name	**Description**
BTIME	File birth (creation) time
ATIME	Last read time
MTIME	Data modification time
CTIME	Metadata change time
UID	User ID
GID	Group ID
Permissions	Text string denoting permissions
Name	Unique numeric identifier for the file
Size	File size in bytes

Table 6.2: Metadata tracked in our simulations.

Action	**Description**	**Metadata Impact Notes**	**Drop Signature**
CREATE	Creates new file in corpus	Times initialized to create time	Exp. drop
READ	Read of a file	Updates ATIME	MD mis: ATIME mismatch
MODIFY	Update of file data	Changes M/CTIME and size	MD mis: M/CTIME, size mismatch
DELETE	Removes a file from the corpus	-	Rlty. drop
CHMOD	Updates a file's perm	Changes CTIME and perm	MD mis: CTIME, perm mismatch
CHOWN	Change user ID of a file	Changes CTIME and UID	MD mis: CTIME, UID mismatch
CHGRP	Change user ID of a file	Changes CTIME and GID	MD mis: CTIME, GID mismatch
RENAME	Change file name	Changes numeric ID	Rlt and Exp drop simultaneously

Table 6.3: Actions we simulate and their impact on metadata as well as their drop signature. Perm refers to file permissions. M/CTIME refers to both modification time and metadata change time.

representative of real workloads; people often work on specific subsets of a storage system for varying amounts of time.

We simulate locality of access by dividing the corpus into locality groups of a fixed size. When generating the workload, a locality group is picked, and a tunable number of actions, which we call the locality action count occur within that locality group; each action is applied to a file selected at random from the locality group. This process is repeated until the action count is reached, and then another locality group is picked.

The workloads we generate act on either a fixed or dynamic corpus, depending on the needs of the experiment. In a fixed corpus, all files are present prior to the trace, and no files will be created or deleted. In a dynamic corpus, files can be created and deleted during the course of the trace.

Common Experimental Parameters: Unless otherwise specified, we use a fixed corpus size of 100,000 files. We chose to do most of our experiments without creating or deleting files as they added book-keeping overhead to the experiments without meaningfully influencing results specific to validating our log failure identification method. Omitted `CREATE` actions make ExDiff's job easier as it triggers an unmaskable expectation drop entry. `DELETE` actions provide no information either way about logger gaps using our method, and when dropped will simply show up as a reality drop entry.

To examine how the number of actions between the initial and reality snapshots influences ExDiff's accuracy, every experiment is run with workload lengths ranging from 50 to 500-thousand actions, respectively. We refer to these as 50k through 500k workloads. This models how increasing the duration between snapshots might impact ExDiff, and further, are well in excess of what we saw for daily activity rates under the various corpora we observed, making our results conservative when examined in the context of archival storage workloads. For each workload length, we generate 10 *base* workloads that each have 10 sets of randomly generated gaps for a total of 100 runs, with results averaged across all runs. In each workload, an action is generated every 1 to 10 time units, with the type of action selected based upon the experiment specific parameters. Each base workload also has an initial snapshot and reality snapshot associated with it.

For each action there is a 1 in 15,000 (.00006%) chance of a gap occurring. This means a 50k length workload averages three gaps per run, while a 500k length workload averages 32 gaps. Each generated gap drops between 100 and 1000 entries. We chose this rapid rate of gap generation for two reasons. First, the number of gaps has little impact on ExDiff's ability to identify gaps, rather as we cover later, it is the number of actions and masking that have influence. Second, this allows us to stress test the cluster based approach as larger numbers of gaps increase the likelihood of estimations erroneously grouping distinct gaps.

For most experiments, we use fixed DBSCAN parameters, with an *N* value of 10 and an *eps* value of 1800 time units. The *N* value is set low to encourage aggressive clustering as a worst case scenario. The *eps* value was chosen as a simple visual inspection of the logs showed periods of no actions typically between 1000 and 10000 time units, so this is an intuitive measure that could realistically be obtained without *a priori* knowledge of the gaps, and is far from perfect. In section 6.3 we examine how varying the DBSCAN parameters influences accuracy.

Metrics: Recall that a gap is a contiguous period of dropped entries, and an estimate is the predicted start and end of that gap. We use two metrics to evaluate ExDiff's ability to identify a gap: *gap coverage* and *estimate utilization*. Gap coverage is the fraction of all gap durations that have been covered by one or more estimates. For example, in Figure 6.5A, there are two gaps running from times zero to time four and four to five, for a total gap length of three. There is one estimate covering the earlier gap entirely, and the latter gap is a *miss* as no estimate covers any portion. two of the three gaps' time units are covered by estimates, having total gap coverage of 0.66.

Estimate utilization is the fraction of all estimates combined that cover gap durations. For example, an estimate of length five that completely covers a gap of length 3 would have an estimate utilization of 0.6. We call this an *estimate overshoot* as the estimate is too long. An estimate of length one that only covers a part of a longer gap would still have a utilization of 1.0, but the coverage for that individual gap would be below 1.0. We call this second case an *estimate undershoot* as the estimated time is shorter than the actual gap. Figure 6.5 B illustrate these concepts.

To provide greater granularity in our examination of estimates, we also look at how *overfit* or *aggressive* they are. The former, illustrated in Figure 6.5C, occurs when multiple estimations are used in covering a single gap. In other words there are multiple undershooting estimates for a single gap. The latter, illustrated in Figure 6.5D, is when a single estimate covers multiple gaps. We also describe whether or not a gap has been *hit* or *missed*. A gap hit occurs when an estimate covers any portion of a gap, while a gap miss is one that is not covered by any estimates.

In all cases, high values for both gap coverage and estimate utilization are desired, as this means that gaps and their duration are identified with high levels of accuracy, with little or no under or overshot estimates. A high gap coverage value with a low estimate utilization value suggests large numbers of overshot or agressive clusters. A low gap coverage with high estimate utilization suggest gap misses and undershot estimates.

As mentioned in Section 6.1, we are also concerned with masking, where later actions remove evidence of prior gaps. For example, a dropped ATIME update would be evident as a MD mismatch, but if a later ATIME update that was not dropped overwrites that files ATIME, it is no longer apparent that an entry was dropped. To examine masking, any file that is acted on during gap is categorized as one of three types. The first, an *unmasked* file, is a file where no later actions cover up evidence of any gap. The second is a *partial* mask where some, but not all, of the evidence of the gap was overwritten. A *total* mask is where all evidence of a prior gap has been overwritten. As covered later on in our evaluation, masking influences both gap identification and missed/dropped entry analysis.

6.3 Evaluation

Our evaluation is broken up into two sections. First, we quantitatively demonstrate ExDiff's ability to automatically identify gaps in log coverage, and explore what can influence its accuracy. Second, we provide a qualitative exploration of omission classification and what can influence its accuracy.

6.3.1 Identifying Logger Gaps

Proof of Concept: In this experiment, we run a workload that has all actions described in Table 6.3 with the goal of demonstrating ExDiff's ability to identify gaps. We demonstrate that ExDiff can accurately identify log gaps and their duration with high estimate utilization and gap coverage values. This initial experiment uses a dynamic corpus as locality type accesses. Later experiments utilize micro-analysis to explore the bounds of ExDiff's gap

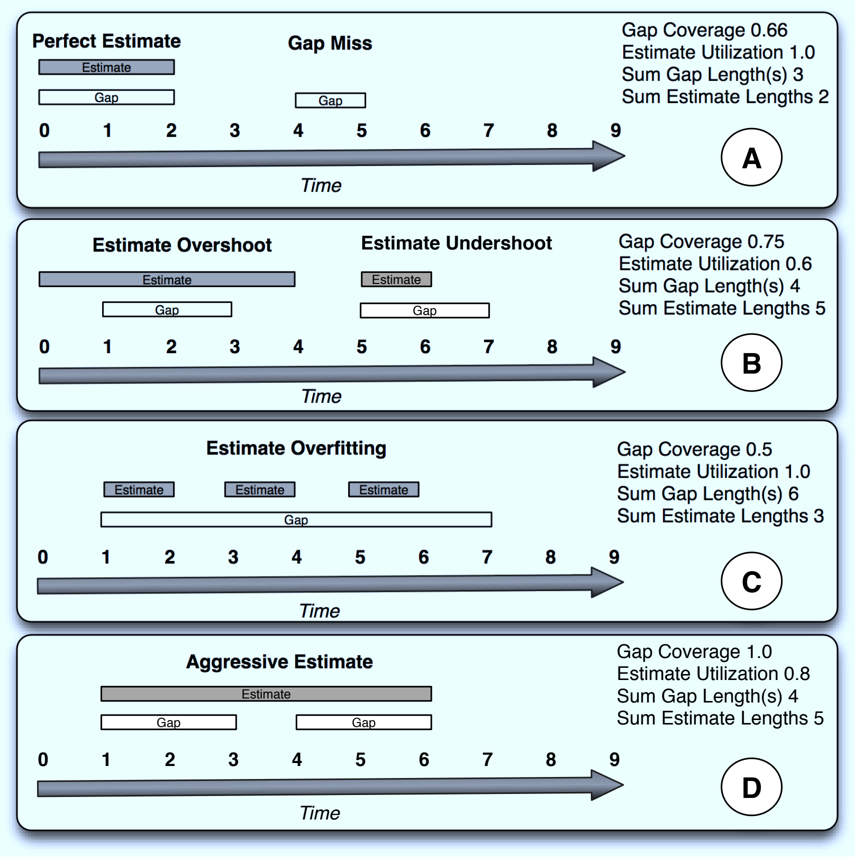


Figure 6.5: These examples (A through D) illustrate our metrics and terminology. The white rectangles are gaps while the shaded rectangles are estimations. Gap coverage denotes the fraction of all gaps covered by an estimate. Similarly, estimate utilization refers to the fraction of estimates that cover a gap. Bolded words are the terms we use to describe various types of estimates.

Exp. Name	Create	Delete	Rename	Read	Data Update	Metadata Update
Simple	0	0	0	34	33	33
Reads+Meta	0	0	0	95	0	5
Read-Only	0	0	0	95	0	5
POC	5	2	3	45	35	10

Table 6.4: Base workload parameters. All experiments are small variations on these parameters, with the variations described in the body text. Each number represents the percent chance of that event being chosen when generating an action. Meta update refers to the chance of picking a UID, GID or permissions change action. POC refers to the proof of concept workload.

identification accuracy.

Using the action probabilities described in Table 6.4, this workload uses locality group sizes of 25 with action counts between 10 and 50. The initial corpus size is 100,000 files. We observe that ExDiff is able to accurately identify gaps with a gap coverage consistently around 97-98% for all lengths, illustrated in Figure 6.6. Estimate utilized shows a slight decrease in accuracy, and increase in variability as the workload length increases, with a mean of 92% and standard deviation around 10% at 500k actions. This is due to the fact that the longer workloads have a higher likelihood of gaps being close enough together to cause an aggressive estimate.

Corroborating this, we find that less than 15% of the 50k runs had aggressive estimates, and never more than one, while over 25% of the 500k runs had aggressive estimates, maxing out at 4. With the parameters we used, we saw no over-fit estimates for any workload length, and surprisingly we only entirely missed gaps in less than 5% of the 500k length runs, and missed zero gaps for any of the shorter workloads. This further demonstrates ExDiff's accuracy in gap identification.

Varying Timestamp Updates: In this set of experiments, we explore how changing the number of distinct timestamps influences ExDiff's ability to produce accurate gap estimates. First, we find that masking has a strong effect on accuracy, and longer durations between snapshots increase masking likelihoods. Second, larger numbers of timestamps can markedly improve ExDiff's accuracy by reducing total masking.

The first experiment uses the *simple* workload. In this workload, we have a fixed size corpus of 100,000 files, and the workload picks each file to act on uniformly at random. Each

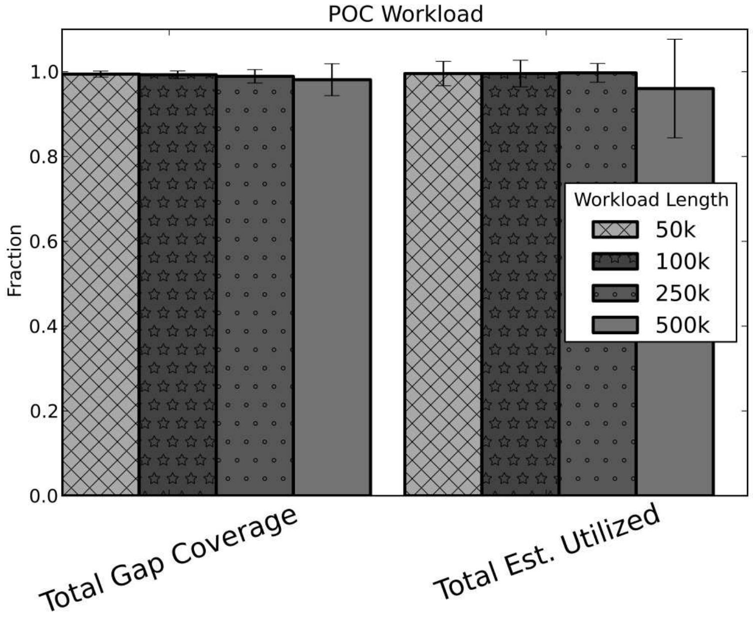


Figure 6.6: A breakdown of how gap and estimation accuracy under the POC workload. Note high gap identification accuracy across all workload lengths, as well as high estimate used values. Error bars are standard deviations.

action has an equal chance of being a data modification, a read, or a metadata update such as a permissions or GID/PID change.

As shown in the leftmost plot of Figure 6.7, there is only a small decrease in accuracy as the workload length increases. This is due to the fact that we are uniformly picking both files and actions, resulting in equal odds of modifying the three timestamps in the file's metadata (CTIME, MTIME, and ATIME). This makes it difficult for arbitrary gaps in coverage to be totally masked by later actions overwriting timestamps, corroborated by the very low amount of total masking illustrated in Figure 6.8. We did see a small, but consistent amount of aggressive estimates for the longer workloads. The 500k workload saw 50% of runs with at least one aggressive estimate, 75% saw over two and maxed out at five aggressive estimates per run. However, the number markedly decreased with the shorter workloads, with the 50k workload only showing 5% of runs with one or two aggressive estimates. We observed no overfit estimates under this workload.

The next experiment used the *read only* workload. In the read only workload, all actions are reads, subsequently ATIME is the only timestamp updated. As we show in the center plot of Figure 6.7, this has a strong impact on the consistency and accuracy of our method as the workload increases in size, because total masking becomes much more prevalent as shown in the center plot of Figure 6.8. This is due to only a single timestamp being used and updated relatively more frequently.

The final experiment looks purely at timestamps under the *reads+metadata* workload to examine how even a small chance of a second timestamp being updated can influence gap estimates. In this workload, each action has a 95% chance of being a read and subsequent ATIME update, while the other 5% may be a metadata action that updates CTIME. Interestingly, even this relatively low chance of CTIME change has a significant impact on masking relative to the read only workload as shown in the center plot of Figure 6.8, and subsequently has significantly higher gap coverage than the read only workload as shown in the right plot of Figure 6.7. Note that there is a significant increase in coverage variability with a decrease in mean coverage at the 500k length. While there is less masking than the read-only workload, there is still quite a bit of total masking occurring. As in prior tests, we saw no overfitting estimates, and the number of aggressive estimates decreased with workload size.

One thing to note across all the timestamp varying experiments is that unless two gaps

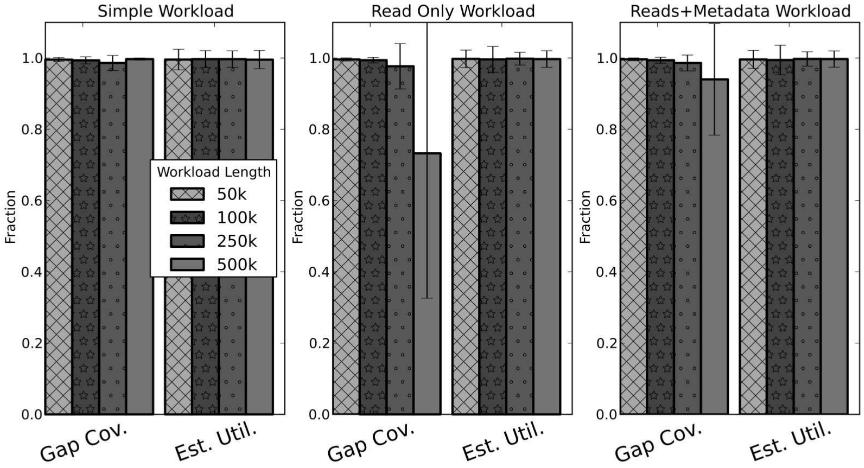


Figure 6.7: A breakdown of how coverage and estimates are influenced by varying timestamp updates. Error bars are standard deviations. Note the read only workload leads to much higher variations on gap coverage as workload length increases.

were covered by an aggressive estimation, we never observed any estimate overshoot a gap and completely subsume it. At most they perfectly matched the end points of a gap. This is due to the fact that in our simulations we have perfect knowledge, and the logger is either perfectly functioning, or not at all, eliminating the possibility of extra diff entries causing false positives or estimate overshooting.

Locality Influence: In our next set of experiments, we examine how adding simple locality of access influences ExDiff. We illustrate how strong, focused locality groups in a workload have little impact in recognizing that there was a failure, but make accurately identifying duration more difficult with wider variation in gap coverage.

For these experiments, we use a locality group size of 25. For the *weak locality* workload, we use an action count between 10 and 50. In the *strong locality* workload, the action count is picked between 100 and 200. Both workloads are otherwise identical to the simple workload, where metadata update, data update, and read actions are all equally likely.

In the weak locality workload we observe the same trends and amount of masking as the simple workload shown in Figure 6.7. Interestingly, the strong locality has 25% less partially masked and 25% more unmasked files than the weak locality workload. This is due to the fact that fewer total files were accessed in the strong locality test as more activities were done per locality group. However, this means that within each locality group there was a higher

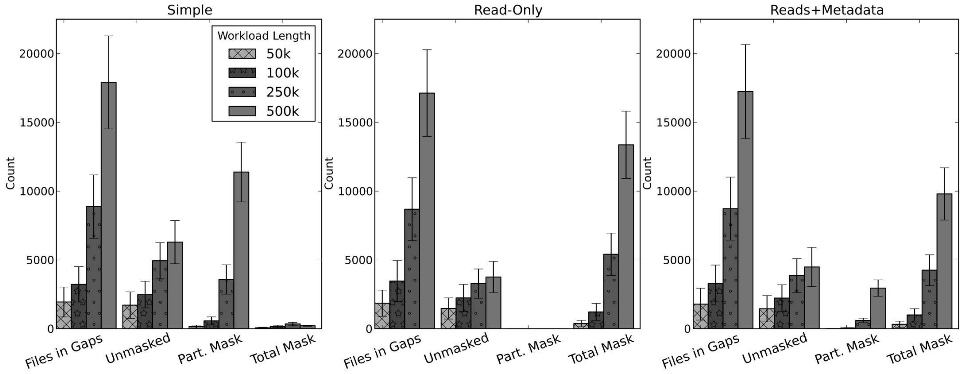


Figure 6.8: Here we show how varying the number of timestamps being updated influences masking. Error bars are standard deviations.

probability of actions temporally near one another causing masking *within* a gap which our metric does not measure.

When examining gap coverage, shown in Figure 6.10, we see that the strong locality test has a much greater variation in its coverage and estimate utilization than the weak locality. This is due to the fact that with stronger locality, we see masking *within* a gap as described above, which in turn leads to significant amounts of estimate undershooting. This is reinforced when we see that in both the strong and weak locality tests over 95% of gaps were a part of at least one estimate, but coverage noticeably decreases with workload length, as shown in Figure 6.9. We also looked at the locality workloads under a reads+metadata access pattern and found it followed the same trends as shown in the initial reads+metadata workload.

Adding Noise: To explore how noise influences ExDiff, we take the simple and read only workloads and add noise in the form of randomly dropped entries in addition to the full gaps. We show that small amounts of noise do not seem to have a large impact on gap hit and miss rates, but can have a very strong influence on estimates as noise makes aggressive and overshooting estimates more common.

All entries in the workload for these experiments have a 1 in 1500 chance of being dropped. This can influence ExDiff's accuracy as there are now points that may be erroneously considered a part of a gap, thus changing the length of an estimation.

As we show in Figure 6.11, the gap coverage is not appreciably different than the original baseline tests. However, the estimations are significantly less accurate. This is due to

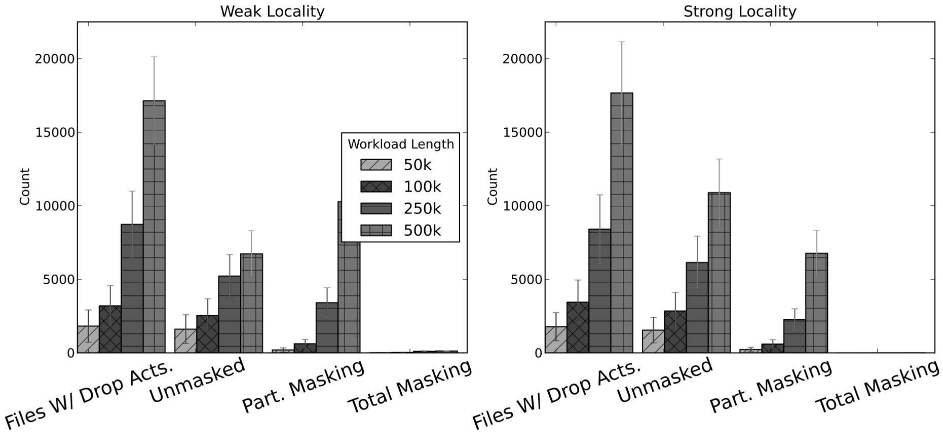


Figure 6.9: File masking in the locality workloads.

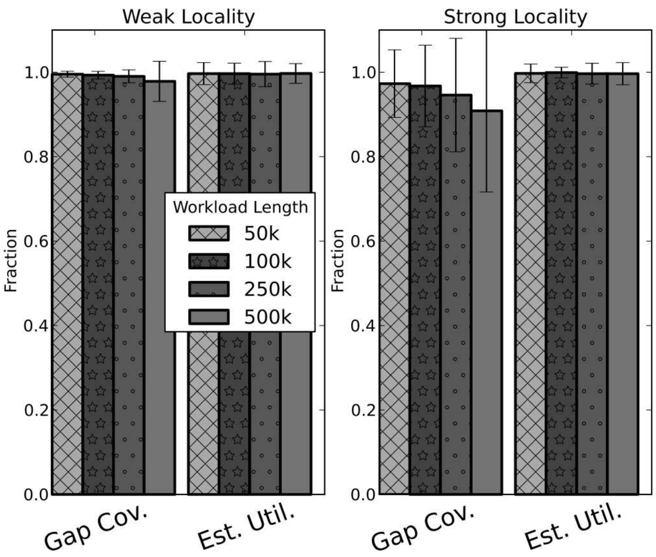


Figure 6.10: Gap coverage and estimate utilization under the locality workloads. Note strong locality has much greater variation in coverage.

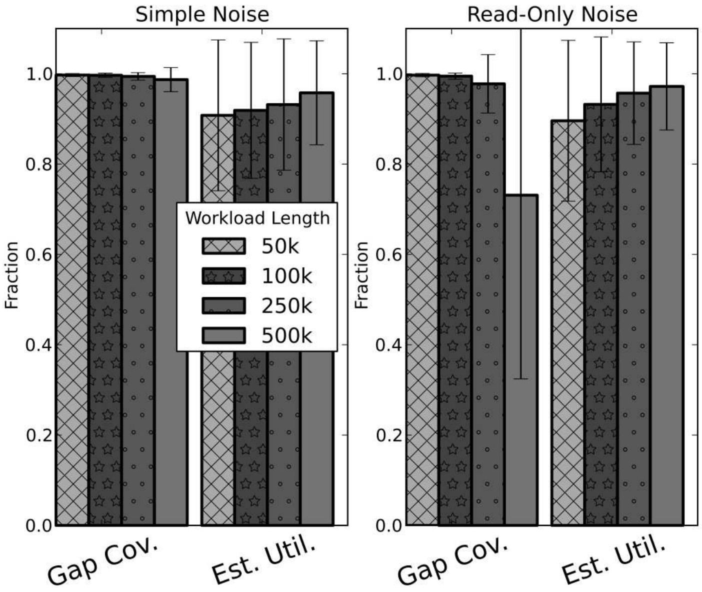


Figure 6.11: Coverage and estimate utilization under the noise workloads. Note drastic increase in variability relative to the other workloads.

the fact that the noise makes it very easy for a gap estimation to overestimate the duration. This can be mitigated by tuning the DBSCAN parameters, as we show in the next section. In terms of gap hits and misses, as well as the incidence of aggressive and over-fit clusters, there were no significant differences from the simple, read-only and reads+metadata tests. We omit the masking charts as they are not noticeably different from the timestamp varying tests.

Varying DBSCAN Parameters: In this set of experiments, we examine how sensitive our results are to varying DBSCAN parameters. We see that gap coverage and estimate utilization is generally improved with lower values of *N* and *eps* in noisy environments.

We take the same parameters as we used for our tests with noise, as these are a worst case scenario in terms of difficulty for estimations; they are likely to lead to significant overshoots estimations in failure durations. We omit the graphs from the read-only noise workload as their trends were similar.

The general trend we notice, illustrated in Figure 6.12 is that having both a smaller *eps* and smaller *N* value tends to increase estimation accuracy for our workloads, keeping in mind we are micro-benchmarking. We see fewer aggressive clusters, and contrary to our expectations, little impact on the number of over-fit estimates. Higher *N* values do tend to increase how likely

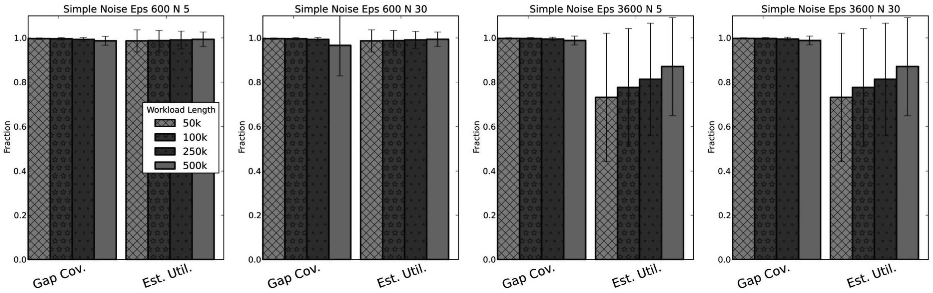


Figure 6.12: Here we show how varying the parameters to DBSCAN influenced gap coverage and estimate utilization under the simple noise workload. Note smaller parameter values tend to produce better results.

we are to miss a small gap that has many masked entries, however. Higher *eps* values tend to cause more aggressive estimations with estimates merging distinct gaps. It is important to note though, that these trends will not be universal across all workloads. For example, we found an outlier case in the read-only noise workload where increasing the value for *N* actually led to a small increase in overfitting. This was due to the fact that the noise was causing the density of the perceived gap to be inconsistent.

Based on our observations, smaller parameters to DBSCAN tend to improve ExDiffs estimate accuracy, with the caveat that they be adjusted for the rate of activity and potential masking in a given log. Second, despite having a noticeable impact on gap coverage, even widely varying parameters rarely miss a gap. Third, while these trends in general hold, gaps may superficially have multiple timestamp clusters of varying density, leading to overfitting. Visualization may help in some of these cases, as it is often readily apparent to human observers when multiple overfit clusters are in reality a single large cluster.

6.3.2 Omission Classification

We also explore how diff entries can help identify the type of entries that may be missing or dropped from a log. We begin by assuming we have perfect knowledge of all possible activities in the system, and follow up with a examination of operating in a limited knowledge environment. In both cases, we leave a quantitative evaluation to future work, and focus on exploring issues in identifying omitted entries.

Perfect Knowledge: The same set of actions and signatures described in Table 6.3

are used in our examinations. Note that we use signatures from our workloads for illustrative purposes, and that signatures may vary from system to system.

A perfect knowledge scenario is likely when logging is included as a first class entity in a system. Even with perfect knowledge, periodic validation of the logs and metadata is useful in many scenarios, such as intrusion detection and debugging.

Basic signature detection is a straightforward comparison of known signatures versus observed diff entries. For example, a missing group ID change entry leads to an MD mismatch entry, with the expected and reality snapshots not matching on both the timestamp and the GID fields. Similarly, a missing data update action would cause an MD mismatch entry with mismatches on the MTIME, CTIME, and possibly the file size. In short, in many cases a perfect signature match can point to a specific omitted action.

Masking persists as an issue in the perfect knowledge case, in addition to masking whole prior actions it can cause a signature to be less clear. Consider a dropped GID change entry, followed by dropped dropped UID change entry. While the diffs may note the mismatch between CTIME, GID and UID, ExDiff loses information regarding the time of the GID change entry.

Actions that change file identifiers (renames) can also make signatures ambiguous. A file rename causes both expectation and reality diff entries, as the missed rename means in the expected a file will exist that doesn't match to a file in the reality snapshot, and vice versa. Similarly, a dropped delete leads to a reality drop entry, and a dropped create causes a expectation drop which superficially overlap with the signature from the rename.

The ambiguities caused by a rename can be addressed by comparing reality diff entries to all files in the reality snapshot. A match on a large fraction of fields other than filename may indicate what file it actually is/it was renamed to. When this match is found, it can also be used to discard expectation drop entries that map to the same file, as they are now known and can be removed to prevent them being classified as another action.

Partial Knowledge: Partial knowledge is less straightforward to work with as omitted actions may be modifying metadata. Thus, the signatures of such actions may be unknown. Despite this, ExDiff has the ability to provide significant benefit. Unexpected diffs are at the very least indicative that something is omitted from the log. Further, actions that change identifiers, add or remove files—such as creates, deletes, and renames—will likely be obvious given the blatant mismatch between the expected and reality snapshots.

6.4 Chapter Summary

We have explored the problem of identifying log coverage; what is and is not being captured in a given trace. To address this issue, we have developed ExDiff, a methodology that uses a combination of activity logs and metadata snapshots to validate log coverage. We show that ExDiff can identify where a logger may have dropped entries while the underlying system is functioning. ExDiff's accuracy is strongly influenced by the number of actions that occur between snapshots, as well as the number of timestamps that are available to work with. We also showed how ExDiff can provide insight into the specific actions that have been omitted from a log, and in certain situations may even be able to reconstruct entire missing log entries.

Chapter 7

Future Directions

> Life is divided into three terms - that which was, which is, and which will be. Let us learn from the past to profit by the present, and from the present, to live better in the future.
>
> William Wordsworth

In this chapter, we explore future directions for both archival workload analysis and our work on log validation and coverage identification.

7.1 Workload Analyses

Though our work into understanding long-term archival workloads has been extensive, there are many remaining questions.

We see a pressing need to better understand metadata accesses and updates as it is our belief that the metadata will be more frequently accessed and updated than the data itself. Our reasoning behind this is based on a few observations and predictions. First, metadata search will likely the primary method of accessing data in many archives as it is unlikely that users will know precisely where the data they are interested in is stored. Second, based on anecdotal evidence, we know that metadata is often updated after data ingest, including things like the time of last data modification and semantic tags. Third, the distinction between metadata and data is becoming increasingly blurry, particularly in observational science data. For example, with astronomy data, the actual data may be a few pixels on a flat background, while the metadata is of prime interest, with information such as time, atmospheric conditions and so forth. With its use to end users and its importance to search, metadata is arguably just as important as the data itself. Unfortunately, none of the datasets we obtained clearly distinguished between metadata and data access or update, so we were unable to make any observations or suggestions on its use.

As touched on in section 2.1, we also see a need to understand how interfaces and the user base influence the workload. A search-based interface will likely have different patterns of access than one that requires manual exploration of an archive. Similarly, skilled users may create pathological access patterns when working with an interface they find onerous.

It would be useful to explicitly examine multi-tenant and cloud based systems, rather than estimate aggregate behavior based on individual corpora activities. While the results we have presented have helped to expand our understanding of long-term corpus behavior, a number of trends motivate the need to understand the aggregate behavior of multiple corpora hosted by a single archive. First, the growth of cloud storage marks a shift towards centralized data centers. Second, increasingly digital workflows have spurred the proliferation of small and mid-sized corpora. This leads to potential problems in optimization, as superficially similar corpora could be hosted on the same archive, despite the fact that they may benefit from different configurations. For example, while both the historical and water corpora showed strong content locality within user sessions, their record granularity is vastly different; a single record type in the water corpus may only contain 20 or 30 records, while in the historical corpus one record type may have millions of records. An optimal retrieval and data layout technique for one may be pathological for another.

One of the shortfalls of our own studies was in being overly focused on aggregate long-term behaviors. It may be worthwhile to return to our datasets to examine usage behaviors on a daily or weekly basis to see if there are trends that can be leveraged for future system designs. In particular, while we saw that in over long timescales files and records did not see much repeat access, it is possible that in shorter timescales there would be a more skewed distribution of accesses.

In addition to examining activities at shorter timescales, it would be useful to examine longer timescales as well in regards to digital archives. We examined one to four years in each our analyses, but what if the corpora is in existence for 100 years? Will the be any novel patterns? And if so, will they be worth changing system designs for?

Our work here has only covered a small number of the new archival use cases out there. There is also a need to study other systems such as legal compliance and personal data archives. We suspect that these systems may have colder data, particularly compliance archives where modification may be prohibited. However, until they are studied we cannot say for certain.

Even in the class of archives we have studied, there will always be a need for periodic studies of any system's behavior. There is a feedback cycle whereby the last generation of studies influence the next generation of systems, which in turn necessitate further study. This is to say nothing of the evolution of hardware, software, and human interests and their influence how data is retrieved and manipulated.

7.2 Log Validation

Although our work on identifying log coverage has proven fruitful, it is merely the first step towards developing a mature set of techniques.

We would like to acquire verified, real-world workloads in order to extensively understand the bounds of ExDiff as logs scale up to millions of actions. Further, such workloads would likely exhibit non-uniform file modifications; most "real-world" systems have a subset of popular files that may change over time, providing us a greater insight in ExDiff's behavior.

We would also like to explore cases where log entries may have inaccurate timestamps and semantic information. For example, the clock used in generating log entries may not perfectly match the timestamps used in the metadata. ExDiff may be able to handle this case by allowing for some "wiggle" room when creating diff entries. An expected and reality snapshot's timestamps that nearly match may be considered close enough to be considered a match.

As actions may continue to update the metadata state as the snapshot is being captured, we would like to investigate methods to handle non-atomic snapshots. Such workloads can lead to situations where a diff entry may be produced even if the log is otherwise accurate. One approach is adding an additional timestamp to each file as it is being captured. When we reach the mapping step we can prevent actions made after metadata capture from updating our expected state, thus preventing accidental diff entries from being produced.

Quantitative investigation of signature detection is also needed. We wish to explore how different workloads and levels of information can change our ability to accurately recognize what types of actions are missing from the log. We could also examine reconstructing log entries; certain signatures provide enough information to re-create missing log entries.

Chapter 8

Conclusion

> I like to think of my behavior in the sixties as a "learning experience." Then again, I like to think of anything stupid I've done as a "learning experience." It makes me feel less stupid.
>
> P. J. O'Rourke

In this work we have asked questions pertaining to long-term storage system behaviors. First, *what does modern archival storage system behavior look like?* Our understanding of modern system behavior is out of date. There are many new archival use cases—personal, scientific, medical, etc.—with unknown usage behavior. The same can be said of the traditional view of archival storage as tertiary storage systems; the last studies were done nearly two decades ago. To address this shortfall in our knowledge, we completed two primary studies. The first examining the behavior of web accessible archives. The second explored scientific tertiary storage systems. The observations and suggestions (summarized below) from each make up two of the primary contributions of my thesis.

Our first study, covered in Chapter 3, focused on the relatively new use case of publicly accessible content archives, serving historical and scientific data sets. We found several results of note. First and foremost, we found that the write-once, read-maybe assumption is severely weakened in many use cases. Data may be updated, and administrative processes such as indexers and integrity checkers often read data *en-masse*, suggesting the use of a separate batch interface for large-scale accesses. We also found that user-sessions show strong content locality *within* sessions, but any individual record or file is at most modestly more popular than another. This implies that intelligent grouping and pre-fetching based on content type may be useful for improving performance and efficiency, but naive caching and data migration may be of limited utility on modern archival storage systems.

Our second study, described in Chapter 4, is on scientific tertiary storage systems, representative of the older, more traditional view of archival storage. We examined file system snapshot summaries from one of LANL's super-computing archives and found that at a high level, aggregate storage system behavior appears similar to that of tertiary storage systems from two decades earlier. The LANL sketch, however, was of a coarse granularity and limited the depth of our analysis. To tackle this shortcoming we obtained a more detailed data set from NCAR covering three years of activity from 2008 through the end of 2010. In this dataset, we found that in tertiary storage the idea of write-once, read-maybe is also weakened. From the system perspective, data will inevitably be read and written, due to large administrative tasks like media migration. Because of these large tasks, an asynchronous batch interface much like that we proposed for the public content archives may be useful as well. From the user-perspective, we see strong per-user spatial and temporal locality in access, with modest rates of data updates/overwrites after ingest into the system. Based on this observation we have two key suggestions for archival architectures. First, even naive grouping based on user and directory may provide benefit for tape and other offline media based systems by reducing the number of media accesses. Second, even from the user perspective, write-once, read-maybe is an unreliable rule of thumb.

Difficulties in interpreting our datasets led us to realize there is a lack of work in identifying the coverage of a storage system trace, that is, understanding what actions are and are not being captured in a given trace. Without knowing a trace's coverage, at best one is left with a degraded understanding of a system's behavior and at worst entirely incorrect conclusions may be drawn from an analysis. This leads to the second question we ask in this thesis, *how can we identify the coverage of a trace?*

To address this, we have designed expectation difference, or ExDiff, a methodology for identifying a trace's coverage using a combination of file-level metadata snapshots and trace logs. ExDiff uses a trace and an initial snapshot of a system's state to derive what we *expect* the system to look like. This expected state can then be compared to reality, and mismatches between the two provide clues as to what is and is not being captured in a given trace. With this methodology, we have provided a way for researchers, administrators, and engineers to validate what is and is not being captured in a given storage system. Researchers can use ExDiff to better understand the limitations of their datasets and administrators can use it to diagnose problems and eliminate potential false positives in their troubleshooting, or even aid in intrusion detection analyses. As an example of use in intrusion detection, consider ExDiff noting that metadata did not match as expected, illustrating that logs may have been tampered with.

Using simulation and analysis, we demonstrated ExDiff's ability to accurately identify when a logger may have been dropping entries, as well as how ExDiff can be used to even recreate dropped or missed log entries. In total, our work has provided up-to-date knowledge on archival storage behavior to help guide and validate future archival designs, and we have provided a new methodology for identifying what is and is not being captured in a given trace to improve the accuracy of analyses based on storage system traces.

Bibliography

[1] dtrace. http://docs.oracle.com/cd/E19205-01/820-4221/.

[2] ftrace. http://linux.die.net/man/1/ftrace.

[3] strace. http://linux.die.net/man/1/strace.

[4] HPSS installation guide, release 7.3 (revision 1). www.hpss-collaboration.org/documents/hpss731/install_guide.pdf, May 2010.

[5] The sleuth kit. http://www.sleuthkit.org/, 2013.

[6] Cristina Abad, Jed Taylor, Cigdem Sengul, William Yurick, Yuanyuan Zhou, and Ken Rowe. Log correlation for intrustion detection: A proof of concept. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC)*, 2003.

[7] Ian Adams, Ethan L. Miller, and Mark W. Storer. Examining energy use in heterogeneous archival storage systems. In *Proceedings of the 18th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '10)*, pages 297–306, August 2010.

[8] Ian F. Adams, Mark W. Storer, and Ethan L. Miller. Analysis of workload behavior in scientific and historical long-term data repositories. *ACM Transactions on Storage*, 8(2), 2012.

[9] Nitin Agrawal, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Generating realistic *impressions* for file-system benchmarking. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST)*, pages 125–138, February 2009.

[10] Nitin Agrawal, William J. Bolosky, John R. Douceur, and Jacob R. Lorch. A five-year study of file-system metadata. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)*, pages 31–45, February 2007.

[11] Alaska's digital archives. `vilda.alaska.edu`, 2010.

[12] Eric Anderson. Capture, conversion, and analysis of an intense NFS workload. In *7th USENIX Conference on File and Storage Technologies*, 2009.

[13] Eric Anderson, Martin Arlitt, Charles B. Morrey III, and Alistair Veitch. Dataseries: An efficient, flexible data format for structured serial data. *ACM SIGOPS Operating System Review*, 43, 2009.

[14] Eric Anderson, Michael Hobbs, Kimberly Keeton, Susan Spence, Mustafa Uysal, and Alistair Veitch. Hippodrome: running circles around storage administration. In *Proceedings of the Conference on File and Storage Technologies (FAST)*, Monterey, CA, January 2002.

[15] Akshat Aranya, Charles P. Wright, and Erez Zadok. Tracefs: A file system to trace them all. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST '04)*, pages 129–145, San Francisco, CA, April 2004. USENIX.

[16] Jens Axboe, Alan D. Brunelle, and Nathan Scott. blktrace. `http://linux.die.net/man/8/blktrace`, 2006.

[17] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout. Measurements of a distributed file system. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP '91)*, pages 198–212, October 1991.

[18] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using Magpie for request extraction and workload modelling. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 259–272, San Francisco, CA, December 2004.

[19] John Bent, Garth Gibson, Gary Grider, Ben McClelland, Paul Nowoczynski, James Nunez, Milo Polte, and Meghan Wingate. PLFS: a checkpoint filesystem for parallel applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Portland, OR, USA, November 2009.

[20] Nino Bilic. Transaction log replay from offline backup. `http://technet.microsoft.com/en-us/library/aa996189%28v=exchg.65%29.aspx`, March 2006.

[21] California Department of Water Resources water reports. `http://www.water.ca.gov/waterdatalibrary/docs/Hydstra/index.cfm`.

[22] Enrique V. Carrera, Eduardo Pinheiro, and Ricardo Bianchini. Conserving disk energy in network servers. In *17th Annual International Conference on Supercomputing, ICS 2003*, 2003.

[23] L.T. Chen, R. Drach, M. Keating, S. Louis, D. Rotem, and A. Shoshani. Efficient organization and access of multi-dimensional datasets on tertiary storage systems. In *Information Systems*, volume 20, pages 155–183. Elsevier, 1995.

[24] Yanpei Chen, Kiran Srinivasan, Garth Goodson, and Randy Katz. Design implications for enterprise storage systems via multi-dimensional trace analysis. In *Proceedings of the 23rd Symposium on Operating Systems Principles (SOSP'11)*, pages 43–56, Cascais, Portugal, 2011.

[25] Dennis Colarelli and Dirk Grunwald. Massive arrays of idle disks for storage archives. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing (SC '02)*, November 2002.

[26] Shobhit Dayal. Characterizing HEC Storage Systems at Rest. Cmu-pdl-08-109, Carnegie Mellon University, 2008.

[27] Shyamala Doraimani and Adriana Lamnitchi. File grouping for scientific data management: Lessons from experimenting with real traces. In *International Symposium on High Performance Distributed Computing (HPDC)*, 2008.

[28] John R. Douceur and William J. Bolosky. A large scale study of file-system contents. In *Proceedings of 1999 ACM SIGMETRICS*, 1999.

[29] Daniel Ellard, Jonathan Ledlie, Pia Malkani, and Margo Seltzer. Passive nfs tracing of email and research workloads. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)*, pages 203–216, San Francisco, CA, March 2003. USENIX.

[30] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, 1996.

[31] Michael Factor, Kalman Meth, Dalit Naor, Ohad Rodeh, and Julian Satran. Object storage: the future building block for storage systems. In *In Proceeding of Local to Global Data Interoperability - Challenges and Technologies, 2005*, 2005.

[32] Joel Frank, Ethan L. Miller, Ian Adams, and Daniel Rosenthal. Evolutionary trends in a supercomputing tertiary storage environment. In *Proceedings of the 20th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '12)*, August 2012.

[33] Joel C. Frank, Ethan L. Miller, Ian F. Adams, and Daniel C. Rosenthal. Evolutionary trends in a supercomputing tertiary storage environment. In *Proceedings of the 20th IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2012.

[34] Timothy Gibson, Ethan L. Miller, and Darrell D. E. Long. Long-term file activity and inter-reference patterns. In *Proceedings of the 24th International Conference for the Resource Management and Performance and Performance Evaluation of Enterprise Computing Systems (CMG98)*, pages 976–987, Anaheim, CA, December 1998. CMG.

[35] Timothy J. Gibson and Ethan L. Miller. Long-term file activity patterns in a UNIX workstation environment. In *Proceedings of the 6th Goddard Conference on Mass Storage Systems and Technologies / 15th IEEE Symposium on Mass Storage Systems*, pages 355–372, College Park, MD, March 1998.

[36] Maria E. Gomez and Vicente Santonja. A new approach in the modeling and generation of synthetic disk workload. In *Proceedings of the 8th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '00)*, 2000.

[37] Steven Gribble, Gurmeet Singh Manku, Eric Brewer, Timothy J. Gibson, and Ethan L. Miller. Self-similarity in file systems: Measurement and applications. In *Proceedings of*

the 1998 SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pages 141–150, Madison, WI, June 1998.

[38] Kristinn Gudjonsson. log2timeline. `http://www.log2timeline.net/`, 2012.

[39] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, Bolton Landing, NY, October 2003.

[40] Robert L. Henderson and Alan Poston. MSS-II and RASH: A mainframe UNIX based mass storage system with a rapid access storage hierarchy file management system. In *Proceedings of the Winter 1989 USENIX Technical Conference*, pages 65–84, 1989.

[41] Health Information Portability and Accountability Act, October 1996.

[42] Dave Hitz, James Lau, and Michael Malcom. File system design for an NFS file server appliance. In *Proceedings of the Winter 1994 USENIX Technical Conference*, pages 235–246, San Francisco, CA, January 1994.

[43] Alexandra Holloway. The purge threat: Scientists' thoughts on usability in peta-scale. In *Proceedings of the 6th Petascale Data Storage Workshop (PDSW '11)*, November 2011.

[44] IBM 3380 direct access storage device. `http://www-03.ibm.com/ibm/history/exhibits/storage/storage_3380e.html`.

[45] Eliot Jaffe and Scott Kirkpatrick. Architecture of the Internet Archive. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, May 2009.

[46] David W. Jensen and Daniel A. Reed. File archive activity in a supercomputing environment. In *Proceedings of the 7th International Conference on Supercomputing (Super Computing 1993)*, pages 387–396, Tokyo, Japan, July 1993.

[47] Weihang Jiang, Chongfeng Hu, Shankar pasupathy, Arkady Kanevsky, Zhenmin Li, and Yuanyuan Zhou. Understanding customer problem troubleshootinbg from storage system logs. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies*, 2009.

[48] Gene H. Kim and Eugene H. Spafford. The design and implementation of Tripwire: A file system integrity checker. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 18–29, 1994.

[49] David F. Kotz and Nils Nieuwejaar. File-system workload on a scientific multiprocessor. *IEEE Parallel and Distributed Technology*, 3(1):51–60, 1995.

[50] Terran Lane and Carla E. Brodley. Sequence matching and learning in anomaly detection. In *Workshop on AI Approaches to Fraud Detection and Risk Management*, 1997.

[51] Andrew W. Leung, Shankar Pasupathy, Garth Goodson, and Ethan L. Miller. Measurement and analysis of large-scale network file system workloads. In *Proceedings of the 2008 USENIX Annual Technical Conference*, June 2008.

[52] Kester Li, Roger Kumpf, Paul Horton, and Thomas Anderson. A quantitative analysis of disk drive power management in portable computers. In *Proceedings of the Winter 1994 USENIX Technical Conference*, pages 279–291, San Francisco, CA, January 1994.

[53] Cornell University Library. arXiv.org e-Print archive . `http://arxiv.org/`, November 2010.

[54] Michael P. Mesnier, Matthew Wachs, Raja R. Sambasivan, Julio Lopez, James Hendricks, Gregory R. Ganger, and David O'Hallaron. //TRACE: Parallel trace replay with approximate causal events. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)*, pages 153–167, February 2007.

[55] Ethan Miller and Randy Katz. An analysis of file migration in a Unix supercomputing environment. In *Proceedings of the Winter 1993 USENIX Technical Conference*, pages 421–433, January 1993.

[56] David Minor and Don Sutton. Chronopolis digital preservation network, 2010.

[57] National Climatic Data Center. `http://www.ncdc.noaa.gov/oa/ncdc.html`, 2010.

[58] Chronicles of Life. Chronicles of life: Save your memories forever. `http://www.chronicleoflife.com/`.

[59] Oregon. The oregon state digital archives. `http://www.digitalarchives.state.or.us/`, November 2010.

[60] John K. Ousterhout, Hervé Da Costa, David Harrison, John A. Kunze, Mike Kupfer, and James G. Thompson. A trace-driven analysis of the Unix 4.2 BSD file system. In *Proceedings of the 10th ACM Symposium on Operating Systems Principles (SOSP '85)*, pages 15–24, December 1985.

[61] Aleatha Parker-Wood, Brian A. Madden, Michael McThrow, Ian F. Adams, Avani Wildani, and Darrell D. E. Long. Examining extended and scientific metadata for scalable index designs. In *6th Annual International Systems and Storage Conference (SYSTOR 2013)*, 2013.

[62] Swapnil Patil, Anand Kashyap, Gopalan Sivathanu, and Erez Zadok. I^{3}FS: An in-kernel integrity checker and intrusion detection file system. In *Proceedings of the 18th USENIX Large Installation System Administration Conference (LISA 2004)*, 2004.

[63] Eduardo Pinheiro and Ricardo Bianchini. Energy conservation techniques for disk array-based servers. In *Proceedings of the 18th International Conference on Supercomputing*, June 2004.

[64] Sean Quinlan and Sean Dorward. Venti: A new approach to archival storage. In *Proceedings of the Conference on File and Storage Technologies (FAST)*, pages 89–101, Monterey, California, USA, 2002. USENIX.

[65] Kenneth Rich and Scott Leadley. hobgoblin: A file and directory auditor. In *5th Large Installation System Administration Conference (LISA V)*, 1991.

[66] D. Roselli and T. E. Anderson. Characteristics of file system workloads. Research report, University of California, Berkeley, June 1996.

[67] Drew Roselli, Jay Lorch, and Tom Anderson. A comparison of file system workloads. In *Proceedings of the 2000 USENIX Annual Technical Conference*, pages 41–54, San Diego, CA, June 2000. USENIX Association.

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

[69] Margaret Rouse. Shoeshining or backhitching. `http://searchdatabackup.techtarget.com/definition/shoeshining-or-backhitching`, July 2012.

[70] Douglas S. Santry, Michael J. Feeley, Norman C. Hutchinson, Alistair C. Veitch, Ross W. Carton, and Jacob Ofir. Deciding when to forget in the Elephant file system. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, pages 110–123, December 1999.

[71] Sarbanes-Oxley act 2002. `www.soxlaw.com`.

[72] Margo Seltzer, Greg Ganger, M. Kirk McKusick, Keith Smith, Craig Soules, and Christopher Stein. Journaling versus soft updates: Asynchronous meta-data protection in file systems. In *Proceedings of the 2000 USENIX Annual Technical Conference*, pages 18–23, June 2000.

[73] Alan Jay Smith. Analysis of long term file reference patterns for application to file migration algorithms. *IEEE Transactions on Software Engineering*, 7(4):403–417, July 1981.

[74] Alan Jay Smith. Long term file migration: Development and evaluation of algorithms. *Communications of the ACM*, 24(8):521–532, August 1981.

[75] Washington State. Washington State Digital Archives. `http://www.digitalarchives.wa.gov/`, 2010.

[76] Mark W. Storer, Kevin M. Greenan, Ethan L. Miller, and Kaladhar Voruganti. Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST)*, February 2008.

[77] John D. Strunk, Garth R. Goodson, Michael L. Scheinholtz, Craig A. N. Soules, and Gregory R. Ganger. Self-securing storage: Protecting data in compromised systems. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 165–180, October 2000.

[78] Andrew S. Tanenbaum, Jorri N. Herder, and Herbert Bos. File size distribution on UNIX systems: then and now, 2006.

[79] Eno Thereska, Brandon Salmon, John Strunk, Matthew Wachs, Michael Abd-El-Malek, Julio Lopez, and Gregory R. Granger. Stardust: Tracking activity in a distributed storage system. In *Proceedings of the 2006 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Saint-Malo, France, June 2006.

[80] Avishay Traeger, Erez Zadok, Nikolai Joukov, and Charles P. Wright. A nine year study of file system and storage benchmarking. *ACM Transactions on Storage*, 4(2), May 2008.

[81] David Tweten. Hiding Mass Storage Under Unix: NASA's MSS-II Architecture. In *Proceedings of Mass Storage Systems, Crisis in Mass Storage*, 1990.

[82] W. Vogels. File system usage in Windows NT 4.0. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, pages 93–109, December 1999.

[83] Avani Wildani and Ethan L. Miller. Semantic data placement for power management in archival storage. In *Proceedings of the 5th Petascale Data Storage Workshop (PDSW '10)*, November 2010.

[84] New York. New York State Digital Archives. `http://www.archives.nysed.gov/aindex.shtml`, 2010.

[85] Lawrence L. You, Kristal T. Pollack, and Darrell D. E. Long. Deep Store: An archival storage system architecture. In *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)*, Tokyo, Japan, April 2005.

[86] Ding Yuan, Haohui Mai, Weiwei Xiong, Lin Tan, and Yuanyuan Zhou. Sherlog: Error diagnosis by connecting clues from run-time logs. In *Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2010.

[87] Min Zhou and Alan Jay Smith. Analysis of personal computer workloads. Technical Report UCB/CSD-991038, UC Berkeley, 1999.

[88] Qingbo Zhu, Zhifeng Chen, Lin Tan, Yuanyuan Zhou, Kimberly Keeton, and John Wilkes. Hibernator: Helping disk arrays sleep through the winter. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*, Brighton, UK, October 2005. ACM.