Validating Storage System Instrumentation

Ian F. Adams*, Mark W. Storer†, Avani Wildani*, Ethan L. Miller*, Brian A. Madden*
*University of California, Santa Cruz †NetApp

***Abstract*—There is a large body of work—such as system administration and intrusion detection—that relies upon storage system logs and snapshots. These solutions rely on accurate system records; however, little effort has been made to verify the correctness of logging instrumentation and log reliability. We present a solution, called ExDiff, that uses expectation differencing to validate storage system logs. Our solution can identify development errors such as the omission of a logging point and runtime errors such as log crashes.**

ExDiff uses metadata snapshots and activity logs to predict the expected state of the system and compares that with the system's actual state. Mismatches between the expected and actual metadata states can then be used to highlight gaps in log coverage, as well as aid in identifying specific types of missing entries. We show that ExDiff provides valuable insight to system designers, administrators and researchers by accurately identifying gaps in log coverage, providing clues useful in isolating specific types of missing log entries, and highlighting potential misunderstandings in logged action.

I. Introduction

Storage system activity logs are used in a number of diverse areas, from system administration and design to security auditing. Unfortunately, a latent problem in such approaches 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 [5, 10] and an activity log to derive the *expected* state of a system. This expected state 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. Note however, that ExDiff is *not* a logging infrastructure by design. As we discuss more in Section III, the capture of metadata snapshots and logs is separate from ExDiff.

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. Additionally, we analyze how log duration and the density of actions affect ExDiff's accuracy. We also detail how ExDiff can 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.

The remainder of the paper is organized as follows. In Section II, we discuss background and related work for ExDiff and provide further motivation. Section III describes our workflow and methodology. We describe our experimental methods and evaluation in Sections IV and V and discuss future work directions in Section VI. We conclude in Section VII.

II. Background

In order to accurately present ExDiff, we begin by detailing the terminology used in our discussions. Following that, we present several use-case examples to further motivate our work. Finally, we place ExDiff in the context of existing work.

A. Terminology

In our discussion, we borrow terminology from earlier work [4]. ExDiff works at the level of individual files or objects. Each file has *metadata* associated with it, such as the time of last modification and user ID. The set of all files that can be operated upon is the *corpus*. The corpus exists on a *system*, a combination of hardware and software.

ExDiff uses two types of captured data: activity *traces* and metadata *snapshots*. An activity trace is a log of events on the corpus. We use the terms log and trace inter-changeably. Individual entries in a log come from *actions*. Actions are atomic events on a single file, such as a read. We assume logged actions accurately reflect changes in the system. A snapshot is a view of a corpus's metadata at a single point in time.

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 called a missed action. For example, a missing action results from a developer failing to add a call to the logging system. A dropped entry 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. A gap *estimation* is a pair of timestamps predicting the start and end times of a gap. When discussing 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.

B. Example Use-Cases

Log coverage verification is critical in a number of areas. In this subsection we highlight a few key areas where ExDiff can improve the confidence of results by validating log correctness.

Intrusion Detection and Forensic Analysis: Intrusion detection systems often compare the state of the system to a known "healthy" state, with mismatches raising alarms [12]. 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: Log analysis is common in storage system management [6]. 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.

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.

C. Related Work

To the best of our knowledge, there is no existing work aimed explicitly at verifying trace coverage despite the number of utilities used to capture OS behavior, such as strace [2] and ftrace [1]. TraceFS [7] is a customizable tracing system existing in user-space that intercepts calls to the file system and systems such as Magpie [8], Stardust [19] and //TRACE [13] are designed to gain end-to-end understanding of larger systems. While these solutions are useful, none examine the coverage of the captured data.

Audit log used in transactional database systems share some similarity to ExDiff. A log can be replayed to reproduce the current state of a system, and compared to a running system for accuracy. Snodgrass *et al.* took this idea a step further by including a hashing and "notary" service to make log tampering evident, while also validating the state of a system [18]. Similarly, journaling filesystems use metadata journals to restore a filesystem to a consistent state after a crash [17]. ExDiff differs as its goal is to identify gaps in coverage, rather than validating or restoring the state of a running system. However, techniques used in ExDiff could be used to verify the correctness of a metadata journal.

Intrusion detection systems (IDS) also rely on comparing expected states to what is observed in a system. 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 [3]. While different in detail, there are similarities in the high-level approach of using multiple data sources in concert to improve analysis. Tripwire detects modifications to a file system by periodically comparing the current state of a system to a database of file checksums [12]. 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 [14]. Hobgoblin is a language and interpreter that describes what properties a file system should have, such as permissions for a given user [15]. While ExDiff shares similarities in comparing an expected state to reality, I^{3}FS, Tripwire and Hobgoblin rely on static rules, rather than comparing state estimated from log entries.

III. 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.

There are four steps to ExDiff. The first is *input*

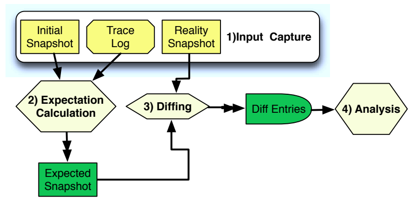

Fig. 1: An overview of ExDiff's workflow. Yellow entries are captured data from input capture, while green (diff entries and expected snapshot) are derived.

capture where an *initial* metadata snapshot is taken, followed by activity tracing, and then a *reality* snapshot is captured. The second step is *expectation calculation*, where the initial snapshot and activity log are combined to derive an expected snapshot. The third step, *diffing*, is where the expected snapshot is compared to the 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 1 illustrates ExDiff's workflow.

Input Capture: In the input capture step, the initial metadata snapshot, activity log and reality snapshot are gathered. Note that ExDiff itself does not capture the data, but relies on data produced from other sources, *e.g.* recursive `ls` and `stat`. 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* 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 captured data 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 (within some bound) mapped to activity log entries. For example, a read entry that can be mapped to a file's ATIME.

With ExDiff, we avoid requiring specific snapshot or log formats. While a rigid format may aid in automating the process, it has two major pitfalls. First, standards are notoriously difficult to be applied consistently, even when they have been in existence for many years let alone new ones being proposed to the community. Second, requiring a standardized input distances humans from the process. We *want* humans to be a part of the ExDiff process as we want to identify where a human analyzer may be misunderstanding a log in addition to identifying gaps and misses.

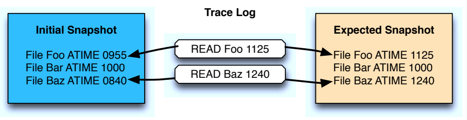

Fig. 2: Expectation Calculation. In this example, the expected state is derived by mapping file ATIME's to read activities.

While some file systems automatically provide snapshots through versioning [11, 16], 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 proof of concept, we assume snapshots are captured atomically, though we describe a method for dealing with non-atomic snapshots in future work in Section VI. Metadata snapshot capture overhead is dependent on system size, but the metadata itself is often quite small. For example, a dataset we obtained from LANL had over 112 million metadata entries, but consumed only 2 GB when compressed.

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 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 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.

When deriving the expected snapshot, *partial* entries may be created. A partial entry is created when attempting to map an activity to a file that is not known to exist based on the 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 3, ExDiff does a file by file comparison of the two snapshots, comparing each piece of metadata to each other. 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 I. A *reality drop* entry is where

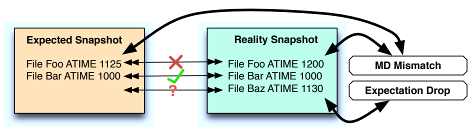

Fig. 3: Diffing. File Foo's expected ATIME does not match reality, so an MD mismatch entry is produced. Similarly, Baz is not seen in the expected snapshot, so an expectation drop entry is created. Bar exists as expected, so produces no diff entry.

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 I: Metadata diff entry types.

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 is not 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 identify 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 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 [9]. 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 discuss how varying DBSCAN parameters influences gap identification in detail in Section V, but relegate automating parameter choice to future work.

DBSCAN's primary bottleneck is in its use of a distance matrix, with a memory and run time overhead of $O(n^2)$. With large datasets we can use DBSCAN with a sliding window approach on the input timestamps. For example, if we have a 30 day log of actions, but can only fit one day's 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. This same sort of masking can occur with other metadata as well, for example file permissions changes. We examine what influences masking in greater detail in our evaluation in Section V.

Omissions Identification: In classification of omissions, ExDiff examines the metadata diffs for clues as to the specific types of 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 omitted 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 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 captured but otherwise modify metadata) still generate diffs, which in the worst case still alerts the analyzing party that their log is missing actions. Signatures will vary depending on the underlying system and trace methodology so we describe signatures specific to our evaluation in Sections IV and V.

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 II: Metadata tracked in our simulations.

IV. 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. With real world traces we would be limited to educated guessing in how various workloads influence ExDiff.

Each file in our workload corpus has common, POSIX-like metadata, described in Table II, and is uniquely identified by its filename; in our simulations this is a numeric identifier. Generated activity logs are comprised of timestamped actions based on common, POSIX commands. We summarize these actions in Table III. Timestamps are integers, and all actions have a unique timestamp. Currently, log timestamps and metadata trace timestamps match, though we discuss how to address metadata and log time skew in Section VI.

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 selected with locality, based on the experiment. We use random picking as a control group as it is easy to understand and analyze, while picking with locality is 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 (not all, however) 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. 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 3 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 discuss 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. This is an intuitive measure that could realistically be obtained without *a priori* knowledge of the gaps, and is far from perfect. In section V 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 are covered by one or more estimates. For example, in Figure 4A, there are two gaps running from times 0 to time 2 and 4 to 5, for a total gap length

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

TABLE III: Actions we simulate and their impact on metadata as well as their drop signature.

of 3. There is one estimate covering the earlier gap entirely, and the latter gap is a *miss* as no estimate covers any portion. 2 of the 3 gaps' time units are covered by estimates, having a 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 5 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 1 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 4 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 4C, occurs when multiple estimations cover a single gap. In other words there are multiple undershooting estimates for a single gap. The latter, illustrated in Figure 4D, is when a single estimate covers multiple gaps. We also discuss 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 III, we are 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 not apparent that an entry was dropped. To examine masking, any file that is acted on during a gap is categorized as one of three types. First, an *unmasked* file, is a file where no later actions cover up evidence of a gap. The second is a *partial* mask where some, but not all, evidence of the gap was overwritten. A *total* mask is where all evidence of a prior gap has been overwritten. Masking influences both gap identification and omission analysis.

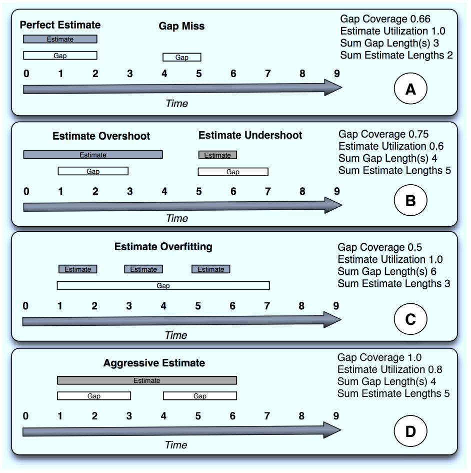

Fig. 4: These examples (A through D) illustrate our metrics and terminology. The white rectangles are gaps and the shaded rectangles are estimations. Gap coverage denotes the fraction of all gaps covered by an estimate. Estimate utilization refers to the fraction of estimates that cover a gap. Bolded terms are used to discuss types of estimates.

V. 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 discussion of omission classification and what can influence its accuracy.

A. Identifying Logger Gaps

Proof of Concept: In this experiment, we run a workload that has all actions described in Table III including creates and deletes 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 identification accuracy.

Using the action probabilities described in Table IV, this workload uses locality group sizes of 25 with action counts between 10 and 50. The initial corpus size is 100,000 files. We find that ExDiff is able to accurately identify gaps with a gap coverage consistently around 97-98% for all lengths. Estimate utilized shows a slight decrease in accuracy, and increase in variability as

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 IV: 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.

workload length increases, with a mean of 92% and standard deviation around 10% at 500k actions. This is because longer workloads have a higher likelihood of gaps being close enough together to cause an aggressive estimate. We omit the graph as all values are quite consistent, making visual comparison difficult.

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 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 5, there is only a small decrease in accuracy as the workload length increases. This is because 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. 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 5, 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. 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, and subsequently has significantly higher gap coverage than the read only workload as shown in the right plot of Figure 5. 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 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 because 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 spatial-temporal locality of access, in contrast to purely random accesses, influences ExDiff. Our experiments show that strong, focused locality groups in a workload have little impact in recognizing a gap occurred, 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

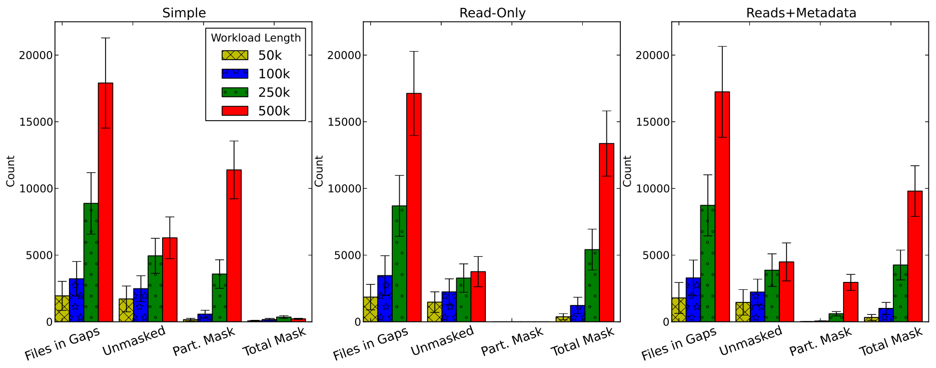

Fig. 6: Here we show how varying the number of timestamps being updated influences masking. Error bars are standard deviations.

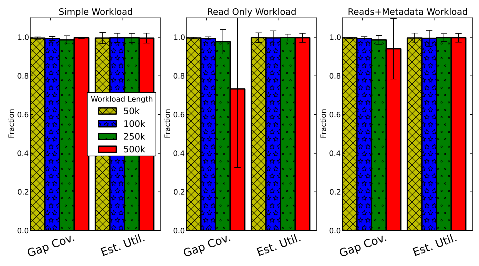

Fig. 5: A breakdown of how coverage and estimates are influenced by varying timestamp updates. Error bars are standard deviations. Note the read only workload leads to much higher variations on gap coverage as workload length increases.

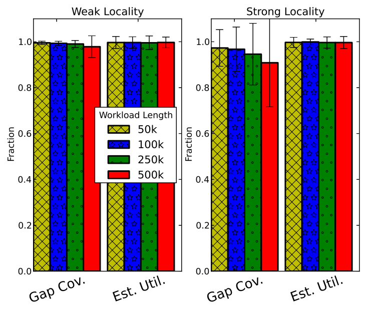

Fig. 7: Gap coverage and estimate utilization under the locality workloads. Note strong locality has much greater variation in coverage.

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 5. Interestingly, the strong locality has 25% less partially masked and 25% more unmasked files than the weak locality workload. This is because 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 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 7, we see that the strong locality test has a much greater variation in its coverage and estimate utilization than the weak locality. This is because 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. 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 8, the gap coverage is not appreciably different than the original baseline tests. However, the estimations are significantly less accurate. This is due to 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 discuss 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

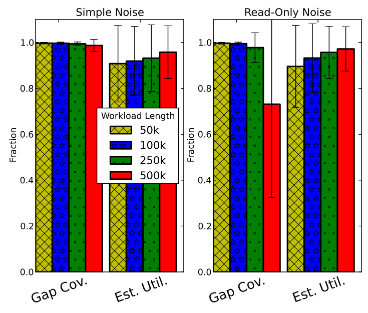

Fig. 8: Coverage and estimate utilization under the noise workloads. Note drastic increase in variability relative to the other workloads.

and estimate utilization is generally improved with lower values of *N* and *eps* in noisy environments.

We take the same parameters 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 9 is that smaller *eps* and *N* values tend 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 tend to increase how likely 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 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 because 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.

B. *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 discussion on operating in a limited knowledge environment. In both cases, we leave a quantitative evaluation to future work, and focus on discussing issues in identifying omitted entries.

Perfect Knowledge: The same set of actions and signatures described in Table III are used in our discussion. 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 make signatures ambiguous. A 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 an expectation drop which superficially overlaps 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 name may indicate what file it actually is/it was renamed to. When this match is found, it can 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

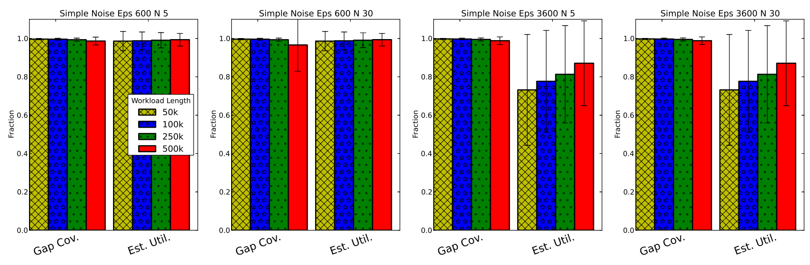

Fig. 9: Here we show how varying the parameters to DBSCAN influenced gap coverage and estimate utilization under the simple noise workload. Note smaller parameter values tend to produce better results.

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.

VI. Future Work

We need 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 greater insight in ExDiff's behavior.

We also need 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.

Actions may continue to update metadata state as the snapshot is being captured, so we need to investigate ways to handle non-atomic snapshots. In this situation, a diff entry may be produced even if the log is accurate. One approach is to add an additional timestamp to each file's metadata when it is captured. In expectation calculation, 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.

VII. Conclusion

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 discussed how ExDiff can provide insight into specific actions that have been omitted from a log, and in certain situations may be able to reconstruct missing log entries.

References

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

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

[3] Abad, C. *et al.* Log correlation for intrustion detection: A proof of concept. In *ACSAC 2003*.

[4] Adams, I. F. *et al.* Analysis of workload behavior in scientific and historical long-term data repositories. *ACM TOS (8)*, 2 2012.

[5] Agrawal, N. *et al.* A five-year study of file-system metadata. In *FAST 2007*.

[6] Anderson, E. *et al.* Hippodrome: running circles around storage administration. In *FAST 2002*.

[7] Aranya, A. *et al.* Tracefs: A file system to trace them all. In *FAST 2004*.

[8] Barham, P. *et al.* Using Magpie for request extraction and workload modeling. In *OSDI 2004*.

[9] Ester, M. *et al.* A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDDM 1996*.

[10] Gibson, T. *et al.* Long-term file activity and inter-reference patterns. In *CMG 1998*.

[11] Hitz, D. *et al.* File system design for an NFS file server appliance. In *Winter USENIX 1994*.

[12] Kim, G. H., and Spafford, E. H. The design and implementation of Tripwire: A file system integrity checker. In *Computer and Communications Security 1994*.

[13] Mesnier, M. P. *et al.* //TRACE: Parallel trace replay with approximate causal events. In *FAST 2007*.

[14] Patil, S.*et al.* I^{3}FS: An in-kernel integrity checker and intrusion detection file system. In *LISA 2004*.

[15] Rich, K., and Leadley, S. Hobgoblin: A file and directory auditor. In *LISA 1991*.

[16] Santry, D. S. *et al.* Deciding when to forget in the Elephant file system. In *SOSP 1999*.

[17] Seltzer, M. *et al.* Journaling versus soft updates: Asynchronous meta-data protection in file systems. In *USENIX 2000*.

[18] Snodgrass, R. T. *et al.* Tamper detection in audit logs. In *VLDB 2004* (2004).

[19] Thereska, E. *et al.* Stardust: Tracking activity in a distributed storage system. In *SIGMETRICS 2006*.