Improved Analysis and Trace Validation Using Metadata Snapshots

Ian F. Adams, *UC Santa Cruz*
Ethan L. Miller, *UC Santa Cruz*
Mark W. Storer, *NetApp*
Avani Wildani, *UC Santa Cruz*
Yangwook Kang, *UC Santa Cruz*

1 Introduction

One of the most fundamental storage system research tasks is activity tracing. By understanding the behavior of a running system we can accomplish a variety of tasks ranging from debugging and system validation, to proposing techniques to improve the performance of future systems. Yet, before we can do an effective analysis of a trace, we must first understand what activities are and are *not* captured in a trace. For example, in some of our earlier works we found entire classes of activities were not captured but which we learned about them via out of band means. It turned out these activities accounted for anywhere between 50 and 90% of all data read in the storage system and would have lead to very different conclusions were we not aware of them [2]. In those cases, we were lucky in that we had contact with experts who understood both the system and the traces they generated, but this cannot be relied upon in every situation, particularly when dealing with large long-term datasets. In a worst case scenario, a multi-year trace or log may be rendered useless, or worse incorrectly analyzed as the researchers may not be aware of missing trace entries.

To help address this issue, we are exploring methods to help identify what we call the *coverage* of a trace. Coverage is a qualitative description of the activities captured, or potentially skipped, in a given trace. Our initial idea for identifying coverage is built around using filesystem metdata snapshots (a common technique in storage system analysis [3, 6]) in conjunction with file-level access logs to provide a picture of what we expect the system state to look like. We use these accesses to mutate the metadata into an expected system state. The expected state can then be compared with what it actually looks like, and the differences between them analyzed to provide clues as to what is and is not being captured in a given trace.

We next describe our methodology and early results.

2 Approach

Our prototype approach is inspired by metadata journaling and log replay for deletions [4], where updates are first stored in a journal and later applied to the full system, which we briefly described in earlier work [1]. In this approach, we treat a trace of activities like a journal to an *initial* metadata snapshot, M_{init}, using trace entries to update the relevant metadata (e.g. timestamps). This mapping process creates what we call an *expected* snapshot at time T, denoted $M_{exp}T$. The expected snapshot is what we estimate the current system state to be at T. We then utilize a second snapshot, which we call the *reality* snapshot that was taken near or (ideally) at time T, denoted $M_{rlt}T$. We then do a file by file comparison of $M_{exp}T$ and $M_{rlt}T$ to create a *metadata diff* of the system, $M_{diff}T$. This diff contains all the files that did not match up between the expected and reality snapshots and their metadata. We then analyze the diff to provide hints as to what we are missing in the logs that may be updating metadata.

The first step in the analysis is to categorize each file in the diff by the nature of its mismatch. The first type of mismatch, which we call a *type 1*, is where a file is found in the expected snapshot but is missing from the reality snapshot. The second type, which we call a *type 2*, is the reverse of the type 1, where a file exists in the reality snapshot but not the expected. A *type 3* mismatch is where a file exists in both the reality and expected snapshots, but does not match on the expected metadata entries. Within the expected snapshot, we may also have what we call a *partial* entry. A partial occurs when we

attempt to map an entry to file that we did not expect (*i.e.* the file was not in the initial snapshot) and then partially populate the metadata in the expected snapshot.

Once the files in the diff have been categorized, we analyze them based on both the observed metadata diffs and their category. As a few examples:

- Large numbers of files in the type 1 category may indicate missing delete entries.
- Files that are type 1, but otherwise have large amounts of identical metadata to files that are type 2 may indicate missing file renames or moves.
- Large number of files in the type 2 category, along with matches to partial entries, may indicate missing create entries.

Another technique we are investigating is the use of density based clustering to automatically correlate type 3 mismatches with one another. For example, if a logger fails, but the storage system continues operating, we will trigger many type 3 diffs. We can then cluster the timestamps from those files to identify if there is a particular timespan where the mismatches occurred. By calculating the start and end timestamps we can estimate the duration of a logger's malfunctioning period.

The biggest shortfall of our overall approach is its reliance on metadata that only reflects the last change to a file. This means there are many cases where missing trace entries may be masked. For example, if there are 5 consecutive actions that modify the same metadata timestamp, and the first 4 are missing, everything will still look as predicted between the expected and reality snapshots as the reality snapshot always reflects the latest activities. As we discuss next, however, the efficacy of our snapshot driven approach is highly dependent on the workload as the rate of metadata change influences the likelihood of the type of event we described above.

3 Current Status

We are currently creating a variety of synthetic workloads for testing the effectiveness of our approach under a different conditions. We are using synthetic, rather than real, workloads because we need to be able to conclusively identify ground truth for many varying workloads for our evaluation. For example, we hypothesize that 'bursty' repeat accesses may not pose a problem to identifying logger failures, but slower repeat accesses may create difficulties as they are more likely to mask prior logger failures. For further evaluation we are also going to experiment with dropping specific types of entries from traces, *e.g* file renames, or permission changes. This will help us gain a better understanding of the strengths and shortcomings of our approach.

Early results for 'slow' archival style workloads with few repeat accesses are promising. We have been able to identify logger failure intervals (discrete periods of dropped trace entries) up to 90% of the time by using the DBSCAN clustering algorithm [5].

References

[1] ADAMS, I., MADDEN, B., FRANK, J., STORER, M. W., AND MILLER, E. L. Usage behavior of a large-scale scientific archive. In *Proceedings of 2012 International Conference for High Performance Computing, Networking, Storage and Analysis* (Nov. 2012).

[2] ADAMS, I. F., STORER, M. W., AND MILLER, E. L. Analysis of workload behavior in scientific and historical long-term data repositories. *ACM Transactions on Storage 8*, 2 (2012).

[3] AGRAWAL, N., BOLOSKY, W. J., DOUCEUR, J. R., AND LORCH, J. R. A five-year study of file-system metadata. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)* (Feb. 2007), pp. 31–45.

[4] BURNS, R. C., AND LONG, D. D. E. System and method for restoring a file system from backups in the presence of deletions. United States Patent 6,938,056 B2, August 2005.

[5] ESTER, M., KRIEGEL, H.-P., SANDER, J., AND XU, X. 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).

[6] GIBSON, T., MILLER, E. L., AND LONG, D. D. E. 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)* (Anaheim, CA, Dec. 1998), CMG, pp. 976–987.