Autograph: Automatically Extracting Workflow File Signatures

Anna Povzner
UC Santa Cruz
apovzner@soe.ucsc.edu

Kimberly Keeton, Arif Merchant,
Charles B. Morrey III,
Mustafa Uysal
HP Labs
firstname.lastname@hp.com

Marcos K. Aguilera*
Microsoft Research
aguilera@microsoft.com

ABSTRACT

Storage management activities, such as reporting, file placement, migration and archiving, require the ability to discover files that belong to an application workflow by relying only on information from the file server. Some classes of application workflows, such as rendering an animated sequence from its graphics models or building an application from its source files, often exhibit a high degree of repeatability. We describe a system called Autograph that exploits this repeatability to discover files that belong to an application workflow. Our approach examines traces of file accesses, finds repeated and correlated accesses, and infers which files likely belong to the same workflow. Our solution targets server workflows and uses file server traces, which contain less process and file information than the local machine traces used in prior work. We show that Autograph successfully extracts workflow file signatures, even if the workflows are concurrent or share files.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval; E.2 [**Data**]: Files—*Sorting/searching*

General Terms

Algorithms, Management

Keywords

storage management, application workflow

1. INTRODUCTION

Recent trends in information generation have produced an explosive growth in the amount of digital data to be stored and managed [1]. At the same time, storage management costs have become dominant – tasks that were simple in small systems have become much more complex and time-consuming in large systems. Such tasks include:

- *File placement in tiered storage systems*: Enterprise storage systems often have multiple storage tiers with very different costs and capabilities. Choosing which files to place in each tier can be a complex task, as files related to the same application or project generally need to be placed in the same tier.
- *Understanding utilization of storage resources*: Storage administrators want to know how applications and projects use storage capacity and bandwidth. This knowledge is necessary for budgeting, billing, and making resource provisioning decisions based on business requirements.
- *Data consolidation and migration*: Administrators sometimes need to migrate files and applications from one server to another, to decommission old systems or to consolidate multiple servers into one.
- *Data archiving*: Administrators often have to archive completed projects or old applications, to preserve a historical record.

All of these tasks require the ability to identify a group of files that are used in the same way by the same applications and/or users. Files in each group tend to be reported, stored, migrated, or archived together, and so the administrator treats such groups as a unit. However, identifying these groups can be difficult, as a group may span multiple directories, a directory may contain files from multiple groups, and files may be shared among groups. With the large number of files in today's information systems, it is impractical for an administrator to manually identify groups of related files.

In this paper, we describe Autograph, a tool that automatically determines the file sets associated with application workflows. An *application workflow* is a process or group of operations contributing to an end goal. We focus on *automated* workflows, which are executed automatically after they are launched. For example, in a software development environment, a software build workflow may consist of producing a specific application from its source code by running "make," which may invoke compilers of several languages, linkers, etc.

Autograph works by capturing network traces at servers, finding repeated and correlated accesses, and inferring which files likely belong to the same workflow. We use a data mining technique called *frequent set mining* to find frequent patterns of file accesses from file traces [2]. We then cluster the frequent patterns that contain accesses to similar sets of files [3] to identify the *workflow file signatures*, the set of files associated with a workflow.

We evaluate the effectiveness of Autograph using NFS traces of document preparation and search utility workflows. Autograph successfully distinguishes workflows, even if they appear concurrently with other workflows some of the time, or if some of their files are also accessed by other workflows. The success of our approach depends on a handful of param-

*Work done while the author was at HP Labs.

Table 1: Frequency of same frame rendering (percentage of frames rendered at least a certain number of times). The sample size is 250,531 frames.

% of frames	# times rendered	% of frames	# times rendered
1	≥ 113	40	≥ 9
2	≥ 83	50	≥ 5
5	≥ 53	60	≥ 3
10	≥ 37	80	≥ 2
20	≥ 22	100	≥ 1
30	≥ 15		

eters, including a threshold for how often a set of accesses must occur before being considered "frequent." We describe rules of thumb for setting these configuration parameters, depending on workflow characteristics and the desired results.

2. PROBLEM STATEMENT

Our goal is to automatically identify the set of files associated with an *application workflow*, which is a process or orchestrated set of operations contributing to an end goal. For instance, an animation rendering workflow may execute one or more rendering applications to produce a specific shot from a set of graphics models. We assume that a workflow satisfies three properties: (a) it can occur by itself in the system, (b) it tends to reoccur over time, and (c) each recurrence involves some repetition of work. An occurrence of a workflow is called an *episode*, and the set of files associated with an application workflow is called the *workflow file signature*. Multiple workflows may execute at the same time, and their file signatures may overlap.

Our approach is predicated on the observation that workflows are highly repeatable. For example, during the production of an animated movie, the same frame may be rendered hundreds of times. Table 1 shows frame rendering statistics from a commercial animation company [4]. The table shows that 40% of frames are rendered at least 9 times, and 10% of frames are rendered at least 37 times. These statistics suggest that real-world workflows exhibit considerable repeatability.

We consider enterprise storage systems that comprise one or more *file servers*, and a set of hosts that access files on the file servers. We focus on managing the storage at the file servers, rather than the client machines, for several reasons. In many environments, important user files are kept only on servers that are protected by periodic snapshots, backups and virus scanning, while clients store only an operating system and a standard set of application binaries. Because client machines may vastly outnumber servers in large environments, it is easier to instrument the servers. Furthermore, in some environments, such as production animation rendering, it may be impractical to instrument the client systems, as the client resources may be fully utilized for compute-intensive tasks.

In our system, file servers are NFS servers. We note that traces of NFS operations contain less information than the local machine traces used in prior work [5, 6, 7, 8, 9]. For example, network file server traces do not contain process summary information (e.g., process id's, names or command-line arguments), process creation or termination information, or explicit file lifetime information (e.g., file opens and closes). NFS trace user ID information could aid workflow mapping, potentially reducing the interference from accesses to files from other workflows. However, the user ID is not always reliable for this purpose. For example, an HTTP server may use a common user ID for different clients. As a result, Autograph does not rely on user ID information.

Our goal is to come up with a tool that can find workflow file signatures without user intervention: users should not have to set a large number of parameters or provide hints and other manually-intensive information.

3. THE AUTOGRAPH APPROACH

In this section, we describe how Autograph deduces workflow file signatures from observed file access patterns. The overall method can be summarized in four steps, as shown in Figure 1: (1) *Trace capture*, to capture a trace of IO accesses at an NFS server; (2) *Trace preprocessing*, to extract NFS operations of interest and divide the trace into shorter access sequences; (3) *Frequent file set extraction*, to pick out frequently occurring sets of file accesses possibly representing workflow episodes; and (4) *Clustering,* to group together the frequently occurring sets of file accesses to reconstruct the workflow's file set. We now explain each of these steps in more detail.

Trace capture. We collect traces of headers of NFS requests and responses. To collect traces, we run tcpdump at the file server. We only need a small header for each NFS request and response.

Trace preprocessing. We then preprocess the trace to keep only getattr and write requests. We do not keep read requests, because clients usually cache data, and so read requests that reach the server are not representative. Instead, we use getattr requests (which verify client cache content freshness and generally precede read requests) to track read accesses to both cached and non-cached files. After this filtering, we do not differentiate between getattr and write requests. To reduce the trace length, we coalesce requests that recur within a short period (100ms) into a single file access with the time stamp of the first operation.

Next, we split the NFS access trace into shorter access sequences, *trace fragments*, so that we can apply data mining techniques to find file accesses that repeat across fragments. Ideally, the trace fragments should be long enough to capture an entire episode, but not long enough to capture too many. Unfortunately, it is hard to determine a priori the exact length of an episode, since it varies with the workflow and even across different episodes of the same workflow. We took a simple approach that splits the trace into trace fragments using non-overlapping time windows of fixed length T, where T is a tunable parameter based on an estimate of the episode length (Section 6.2).

It may seem that using overlapping windows would work better, because if one window splits an episode, another shifted window might capture it. However, even overlapping windows may fail to capture an entire episode. Instead, a final clustering step combines information from split episodes, as described in detail below. With this clustering step, we

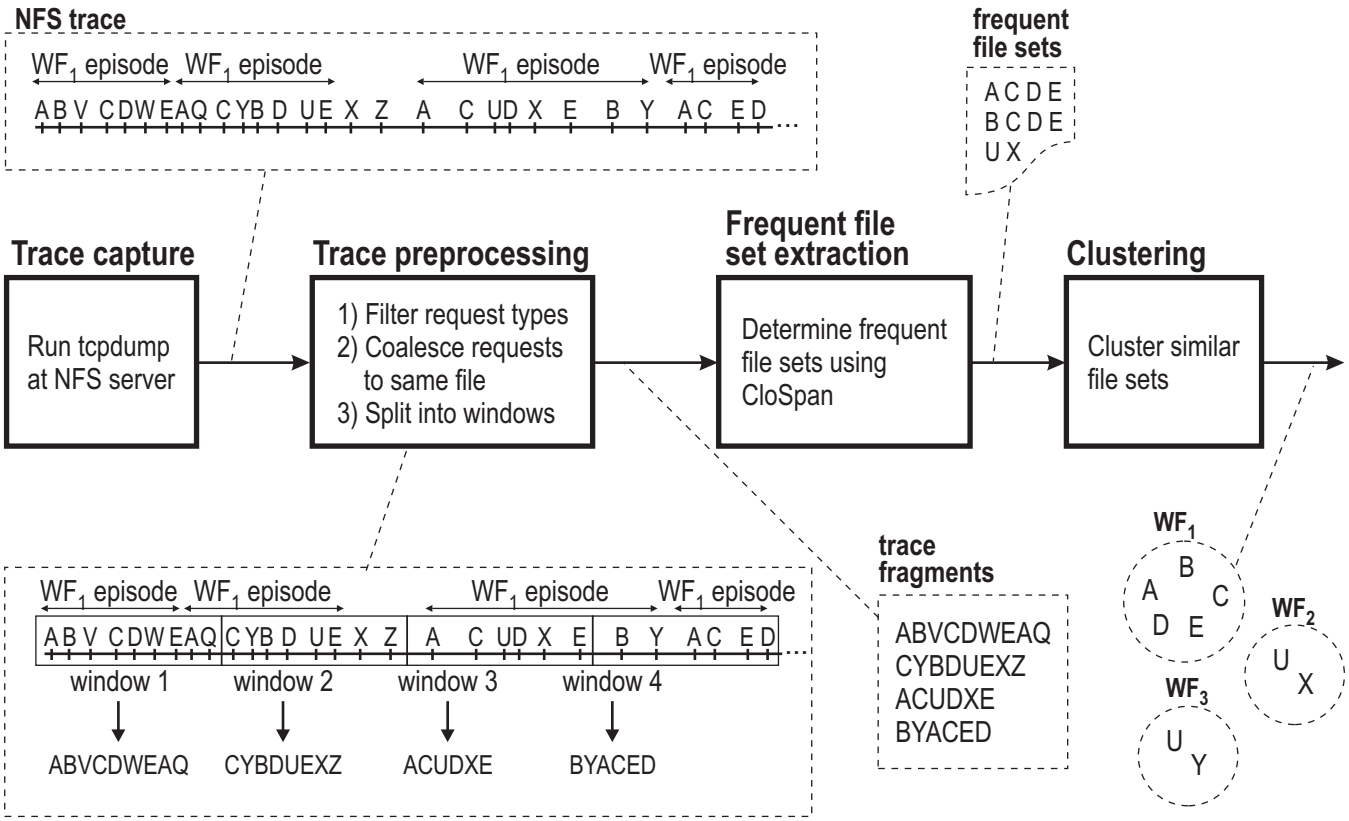

Figure 1: Autograph's approach for extracting workflow file signatures. A workflow WF_1 uses the files A, B, C, D, and E. The captured NFS trace shows accesses to files A–E and other files, which are made by workflow $WF1_1$ and other workflows. Using a threshold of 3, CloSpan discovers frequent file sets ACDE and BCDE from the trace fragments.

found that non-overlapping windows work as well as two windows that overlap at $T/2$ intervals.

Frequent file set extraction. Next, we apply a frequent set mining technique to the collection of trace fragments in order to extract file accesses that occur frequently together. A set of file accesses is frequent if it occurs in at least a specified threshold τ number of trace fragments.

Figure 1 shows an example of this process. After splitting the trace using non-overlapping windows, the set ACDE occurs in the first, third, and fourth trace fragments. Since our threshold for the example is three, ACDE is a frequent set. Smaller threshold values produce more frequent sets. Similarly, the algorithm will find very few frequent sets if the threshold is too high.

We extract only *maximal* frequent sets (i.e., frequent sets not contained in other frequent sets), because they already have all the information that subsets contain. In our example, ACDE, BCDE, CDE, and DE are frequent, but the algorithm only extracts ACDE and BCDE. If the threshold is 4, ACDE and BCDE are not frequent, so the algorithm extracts CDE instead.

Several well-known algorithms can be used for frequent pattern mining [10, 2, 11]; we used the CloSpan [2] algorithm. CloSpan is a frequent sequence mining algorithm that efficiently finds maximal frequent sequences. It takes the list of trace fragments and the threshold parameter τ, and outputs the maximal file accesses that occur τ or more times. Since we used CloSpan unmodified, we do not describe it further here.

Because we are interested in frequent sets, not sequences, we sort each trace fragment and remove duplicate accesses as we input trace fragments into CloSpan. Thus, EDCAE and ACDE become the same set ACDE. We considered omitting this sorting step, to keep ordering information, which the frequent sequence mining algorithm can exploit. However, we found that ordering information does not work well when workflow episodes are not very repeatable (Section 6.3).

Clustering. Episodes of the same workflow may not always access exactly the same files, and so the same workflow may result in different frequent file sets. In the example in Figure 1, the same workflow WF_1 produces two frequent file sets, ACDE and BCDE. We would like to combine them, to obtain the set ABCDE of all files in the workflow. A similar problem is that a window may split an episode, and so CloSpan will find different frequent file sets corresponding to different parts of the same workflow. Thus, the final step in our algorithm uses clustering to combine similar frequent file sets and then outputs the union of files in each cluster.

We employ a document clustering algorithm called Frequent Itemset-based Hierarchical Clustering (FIHC) [3], with some modification, to cluster frequent file sets together. The intu-

ition of FIHC is that each cluster corresponds to documents that share many common words, since documents under the same topic have more common words than documents under different topics. We apply FIHC by treating a frequent file set as a document, and a file as a word (item) in a document.

FIHC contains two stages, cluster construction and hierarchical topic tree building. We modify the first stage and replace the second stage. Here, we only provide an overview of the first stage (see [3] for more details). The input to FIHC is a collection of frequent file sets. FIHC starts by finding *frequent itemsets* in the collection. A frequent itemset is a set of files contained in at least σ frequent file sets, where σ is a tunable parameter discussed in Section 6.2. We modified FIHC to find only *maximal* frequent itemsets (i.e., itemsets that are not contained in other itemsets). For each maximal frequent itemset I_j, FIHC designates an initial cluster C_i with all frequent file sets that contain I_j. This may result in clusters sharing frequent file sets. To make clusters disjoint, for each frequent file set F_j that belongs to multiple clusters, FIHC identifies the cluster that is most "similar" to F_j and removes F_j from other clusters. To define the "similarity" metric between clusters and frequent file sets, each frequent file set is represented by a *feature vector*, a vector indexed by files containing the frequency of files in the file set. Each cluster is represented by a vector containing frequencies of files in the union of all frequent file sets in the cluster. Feature vectors in FIHC only consider the set of files that belong to some maximal frequent itemset (other files are ignored). As in many clustering algorithms, frequency numbers in FIHC are weighted according to Term Frequency - Inverse Document Frequency (TF-IDF)[12]. TF-IDF assigns little weight to words that are too common across different documents. As a result, in our case, TF-IDF tends to ignore common files. This is not desired, because files may become too common for TF-IDF if they belong to less repeatable workflows that produce many frequent file sets. Therefore, we modified FIHC to use Term Frequency (TF) instead, where each component of the vector indexed by a particular file is the number of times that file appears in a frequent file set (or a cluster).

We now describe the second stage of our clustering approach, which is different from FIHC. Frequent file sets that do not contain any frequent itemset remain unclustered in the first stage. We call them the *leftover* file sets. Leftover file sets correspond to workflows that have little or no variation across episodes. For example, a workflow that always accesses the same set of files produces only one frequent file set. Therefore, we assign a cluster to each leftover file set. The result might be that a workflow results in several such clusters. A similar but more subtle problem is that the first stage of clustering may also create several clusters for the same workflow when a workflow produces different file sets containing different frequent itemsets. To address these issues, we merge similar clusters as follows.

First, we modify the feature vectors used to represent clusters. Recall that the feature vectors contain only files that appear in some frequent itemset. In this case, a leftover file set, by definition, would have a zero vector associated with it. Thus, we modify feature vectors to contain frequencies of *all* files in the trace. For each pair of clusters, we calculate inter-cluster similarity as in [3] using all items (rather than global frequent items) in scoring functions and merge the cluster pair that has the highest score. The procedure is repeated as long as there are clusters that are more similar than dissimilar.

The result of the algorithm is a set of clusters, where each cluster has a set of file sets. As the last step, we take the union of all files in each cluster. Note that while clusters of frequent file sets are always disjoint, the groups of files may not be disjoint, because the same file may appear in multiple frequent file sets.

4. EVALUATION METHODOLOGY

Our goal in evaluating our system is to determine how effective it is at differentiating the files in each workflow. To quantify the accuracy of results, we use metrics borrowed from information retrieval systems (Section 4.1). We compare Autograph with a simple existing method based on temporal locality graphs (Section 4.2).

4.1 Evaluation metrics

To evaluate accuracy of Autograph, we use the traditional information retrieval metrics of *recall* and *precision* [13]. These metrics measure how different the output of Autograph is compared to ground truth (i.e., the "right" answer). Ground truth is given by a set of workflows $\{WF_i\}$; we use WF_i to denote both the i-th workflow and its file signature. Autograph outputs a set of clusters $\{C_j\}$, where each cluster approximates the set of files of a workflow. Clusters need not be disjoint (unlike in traditional clustering) because the workflows themselves may have shared files.

Recall ($|WF_i \cap C_j|/|WF_i|$) is a measure of the completeness of retrieval, and a perfect score of 1 means that all files in WF_i are also in C_j. *Precision* ($|WF_i \cap C_j|/|C_j|$) is a measure of the exactness of retrieval, and a perfect score of 1 means that every file in C_j belongs to WF_i.

We define each "ground truth" set WF_i in two steps: (1) a human expert chooses a set V_i of files associated with the workflow, and (2) we define WF_i to be V_i intersected with the set of files that actually appear in the trace. Intuitively, the second step removes files that cannot be found by *any* trace-based algorithm. For example, there may be a "readme.txt" file that is part of the source code of a program, but which is never read by the compilation process. In Section 5 we briefly evaluate how many such files there are in real workflows.

4.2 Comparison point

We compare our system against a technique based on temporal locality graphs used in previous work [5, 7, 9]. In such graphs, nodes are files and there is an edge between two files if they are accessed close-by in time. More precisely, there is an edge between two files if and only if there is an access to both files within the same time window [9]. Weights on edges indicate the number of such accesses. We use an undirected graph to represent these relationships, though we could also use a directed graph.

The temporal locality graph by itself does not indicate the files in each workflow. To do so, we run a post-processing algorithm on the graph that drops edges below a threshold and then computes the connected components of the graph. The final output is a set of clusters, where each cluster consists of all files in a connected component of the graph.

Table 2: Characteristics of the workflows.

Workflow	Episodes	Files
`docprep-1`	45	38
`docprep-2`	46	32
`docprep-3`	37	97
`grep`	40	343
`grep-tex`	45	112

5. EXPERIMENTS AND RESULTS

We evaluate our Autograph implementation using macrobenchmarks to demostrate the robustness of Autograph in a realistic setting. We want to evaluate whether Autograph can successfully identify the files in each workflow, especially in scenarios where (a) the same workflow executes slightly differently each time, (b) several workflows run concurrently so that their accesses are intermingled, and (c) different workflows share some files.

In all experiments, we used two types of real workflows from our file server. Document preparation workflows comprise all the revisions made to a document and its components in its creation. Search utility workflows comprise the activity of searching for patterns in files using the "grep" tool.

We used three document preparation workflows (`docprep-1,2,3`), each corresponding to a series of edits for paper submissions to various conferences. Each workflow begins by checking out the first revision from the server. Each episode of these workflows then consists of applying the changes introduced in the subsequent revisions and building the document. Between each episode we introduce a fixed delay (20 seconds) of inactivity. Table 2 illustrates the number of episodes and the total number of files for each workflow. The workflow file set sizes used in our experiments are typical for our document repository. Because each document repository is independent, we sometimes ran more than one document preparation workflow in parallel.

We first evaluate the effectiveness of Autograph to discern repeatable workflows when episodes of different workflows overlap sometimes, but each workload accesses its own set of files. Figure 2 shows an inverse CDF of the fraction of files that appear in a certain number of trace fragments. It shows that there is some repeatability in the sets of files accessed during each episode in real workflows. The `docprep-1` workflow is more repeatable than the others, but the `docprep-2` and `docprep-3` workflows also have a majority of files appearing in several trace fragments. This behavior suggests that a low frequent set threshold value is needed for good recall results. However, workflows sometimes overlap in this experiment, so the threshold cannot be too low to be able to differentiate between the workflows.

We ran Autograph with several small frequent set threshold values (from 4 to 10). As expected, lower thresholds produced better recall values for each workflow, and too-low thresholds produced worse precision. The recall (precision) values monotonically decreased (increased) with higher threshold values, so we present the best recall and best precision results in Table 3. The best recall results were collected with the threshold set to 4 and the best precision results were collected with the threshold set to 8. The window size was set to 300 ms, which is a little less than a half of an average episode duration.

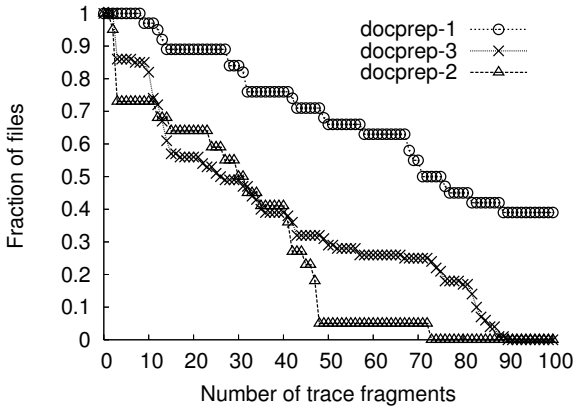

Figure 2: Document preparation workflow macrobenchmark. Fraction of files found in at least a certain number of trace fragments.

Table 3: Document preparation macrobenchmark. The best recall results were collected with the threshold set to 4 and the best precision results were collected with the threshold set to 8.

Workflow	Autograph		Temporal Locality	
	recall	precision	recall	precision
docprep-1				
best recall	100%	61%	100%	29%
best precision	97%	100%	82%	100%
docprep-2				
best recall	73%	100%	100%	17%
best precision	73%	100%	45%	100%
docprep-3				
best recall	85%	77%	99%	55%
best precision	65%	100%	1%	100%

From the behavior shown in Figure 2, we expected that with reasonably small threshold values, `docprep-1` would likely have high recall (higher than 95%), and the other workflows would have lower recalls (about 73% for `docprep-2` and 86% for `docprep-3`). The best recall results, shown in Table 3, confirm our expectations. Also, the precision in this case is low for some workflows, since we used a very small frequent set threshold. We could also achieve a perfect precision, shown in *best precision* rows, with a very slight decrease in recall for `docprep-1` and `docprep-2` and a bigger decrease for `docprep-3`.

The temporal locality results in Table 3 show a much more dramatic trade-off between recall and precision. We also found that while we could achieve a reasonable trade-off for one workflow, the results for the other workflows show a very dramatic trade-off for any parameter settings. This can be explained by the fact that the temporal locality approach captures only pairwise relationships between files indicated by weights on edges between files. Depending on workflows' behavior, connections between files in one workflow may be stronger than connections between files in other workflows. So, adjusting the weight threshold value to separate one workflow from the others may also break some weaker connections of files in the other workflows.

Table 4: Document preparation workflow and grep *.tex. The best recall results were collected with the threshold set to 2 and the best precision results were collected with the threshold set to 4.

Workflow	Autograph		Temporal Locality	
	recall	precision	recall	precision
docprep-1				
best recall	92%	100%	97%	30%
best precision	92%	100%	84%	100%
grep-tex				
best recall	100%	100%	100%	90%
best precision	100%	100%	21%	75%

Table 5: Document preparation workflow and grep. The best recall and best precision results were both collected with the threshold set to 2.

Workflow	Autograph		Temporal Locality	
	recall	precision	recall	precision
docprep-1				
best recall	80%	97%	95%	12%
best precision	65%	100%	76%	94%
grep				
best recall	100%	100%	99%	99%
best precision	99%	100%	10%	100%

Our next experiments show that Autograph works well when workflows not only run concurrently, but they also share a significant fraction of files. This experiment explores the situation where regular backups touch many files in the system while sets of those files also belong to other workflows that run in the system. The experiment contains `docprep-1` from the previous experiment and the "grep" workflow (`grep-tex`) that regularly searches *.tex files for pattern "the" in `docprep-1` directories and several other directories. Episodes of the two workflows sometimes overlap.

Table 4 shows best recall and best precision results. We observe that Autograph works quite well, and only a few files from `docprep-1` are incorrectly placed in the `grep-tex` workflow cluster. Note that this does not decrease `grep-tex` workflow precision since incorrectly placed files are shared and also belong to `grep-tex`. The temporal locality approach produces both lower recall and precision with a high trade-off between them.

Table 5 shows the results of a slightly different experiment that contains `docprep-1` and the "grep" workflow (`grep`) which searches for pattern "the" in all files in several directories. As a result, the `docprep-1` workflow files are a proper subset (10%) of the `grep` workflow. Autograph identifies `docprep-1` even when 100% of files of this workflow also belong to another workflow in the system. Autograph achieves 80% recall for `docprep-1` with very good precision (97%) and perfect recall and precision for `grep`. The temporal locality approach achieves a good recall for both workflows but at the cost of very low precision for `docprep-1`.

6. DISCUSSION

In this section, we discuss the insights that we have gained in using Autograph. These insights include an understanding of Autograph's advantages over the temporal locality approach, rules of thumb for setting Autograph's configuration parameters, and an exploration of the importance of sequence information.

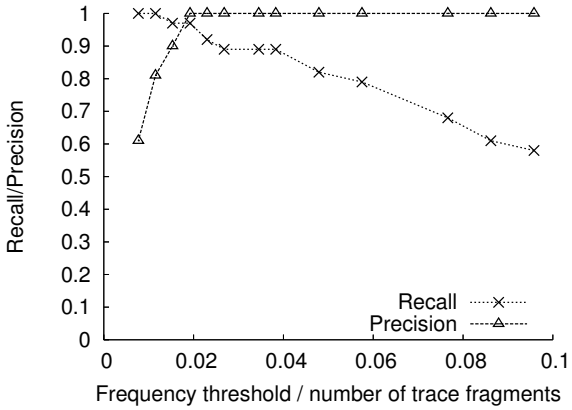

Figure 3: Recall and precision of `docprep-1`.

6.1 Advantages over temporal locality

Our evaluation has shown that Autograph works better than the temporal locality approach, which can be attributed to two main differences between them. First, the temporal locality approach keeps track of only pairwise relationships between files that are accessed together within a short time interval, while Autograph keeps track of sets of files. Second, the temporal locality approach produces disjoint clusters of files, while Autograph produces clusters of files that may not be disjoint, thus allowing the same file to appear in multiple clusters. As a result, Autograph works better in the presence of concurrent workflows and workflows that share files.

6.2 Parameter selection

Autograph requires setting three parameters: the window size T for splitting the trace into trace fragments, the threshold τ for the frequent file set extraction, and the frequent itemset threshold σ for clustering. We found that Autograph is most sensitive to the frequent file set threshold τ, which indicates in how many trace fragments we should see a file set to consider it frequent. Less-repeatable workflows need low thresholds to achieve good recall, while higher thresholds are needed to detect workflows with more overlapping episodes. We explore this tension using the results of the document preparation macrobenchmark described in Section 5. Figure 3 shows recall and precision results for `docprep-1` as a function of the threshold divided by the total number of trace fragments. Recall decreases as the threshold increases. Precision is perfect for high thresholds, and it decreases with lower thresholds, because `docprep-1` sometimes overlaps with other workflows.

The threshold value should be selected from the interval between the threshold that provides perfect (or best) recall and the lowest threshold after which precision starts decreasing (e.g. [0.01, 0.02] for this example). Because in the general case the workflow behavior is not known, the lower bound is two trace fragments (the lowest threshold value). The upper bound is derived from the probability of two workflows appearing in the same trace fragment with very high confidence level. By setting the threshold values appropriately within this interval, Autograph may be tuned to provide good recall, or good precision, based on the requirements of the usage scenario.

For example, data archival requires high recall, but allows lower precision. If files are missing from the archive, it may prove to be unusable; however, it is acceptable to include some extra files. Since recall is more important in this scenario, we suggest setting the threshold to lower values within the threshold interval. Note, however, that Autograph's recall is only as high as the repeatability of the workflows; thus, it is only appropriate to use Autograph to archive highly repeatable workflows.

Reporting of storage resource utilization has moderate precision and recall requirements, with the requirements in both dimensions being higher if the goal is to track utilization for individual workflows than if the goal is to track aggregated utilization across multiple workflows. Autograph should be useful for reporting for a wide range of different workflow behaviors. Since recall and precision are similarly important, the threshold value should be set in the middle of the threshold interval.

File placement in tiered storage systems and data consolidation and migration are performance optimizations that are more tolerant of wrong answers. If files are omitted from the correct group, they can be migrated to the appropriate location, albeit at a slight performance penalty. Similarly, if extra files are included, they can ultimately be returned to the appropriate location. This usage scenario is tolerant to any threshold settings within the interval.

Since window size defines the size of trace fragments, it affects the probability of two workflows appearing in the same trace fragment. The window size should be small for lower probabilities – the lower bound found by our sensitivity analysis is 35-40% of the expected episode duration of the longest workflow (not shown).

Finally, we found that Autograph is not very sensitive to the frequent itemset threshold σ used by clustering (not shown). We found that 5%-10% (of the number of total trace fragments) works well for most cases, and a lower threshold is needed if all workflows in the trace share same files. In this case, higher threshold values would find only shared files frequent and group all workflows together into one cluster based on these files. For this reason, we used 0.1% threshold for the experiments with `docprep-1` and "grep" workflows, where the trace contained only two workflows that shared a large fraction of files.

6.3 Importance of sequence information

The current Autograph design operates on sets of files. We also explored an approach that extracts sequences of file accesses that appear frequently across episodes. We expected that information about the order of file accesses would provide better results for highly concurrent workflows and workflows that share many files that are accessed in different orders.

We first examined the sequence extraction approach with the same final step of clustering sets of files that belong to frequent sequences. However, this approach did not perform better than Autograph, because the order information was lost in the final step. Thus, we modified this approach to cluster frequent sequences using sequence similarity based on edit distance between sequences. We found that this modified, ordered approach also did not outperform Autograph's unordered approach. To be able to handle cases such as concurrent workflows better, the clustering of sequences should be very strict with regards to sequence similarity and cluster only very similar sequences. However, such strict clustering does not work well when workflow episodes are not very repeatable. So, to be able to cluster frequent sequences from not very repeatable episodes, we need to be more loose about clustering similar sequences. This tension makes our current approach best suited for extracting workflows.

7. RELATED WORK

Previous work addressed similar problems as ours using different techniques, or different problems using similar or related techniques. We now explain these in detail.

Similar problem, different technique. Coda is a file system that supports disconnected operation, that is, operation without access to the network [14]. Before disconnecting from the file server, the system must decide what files it needs to copy locally, so that they will be available offline. This is similar to determining the files in the workflows that will execute off-line. To choose such files, Coda relies on a user-specified database that contains project files and system files that the user and/or system administrator choose. Creating and maintaining this database can be an arduous and error-prone task.

SEER [7] is a system for automatically choosing files to copy for offline usage, with little user involvement. SEER defines a *lifetime semantic distance* based on the number of intervening opens between file accesses; files that are close together (using this distance) are clustered into projects, and the system chooses to copy the currently-active projects (projects whose files have been accessed recently). To compute the semantic distance, SEER uses knowledge of when files are opened and closed and by which process. SEER is similar to our system in that determining files in a project is similar to determining files in a workflow. Our system does not use information about when files are opened and closed, and by which process; this information is not available at NFS servers.

Related technique, different problem. C-Miner [15] is an algorithm that uses the CloSpan frequent sequence mining algorithm to discover correlations of access to data blocks in a storage system. This information is useful for predicting which block will be accessed next, given which blocks have been accessed recently, for the purpose of prefetching, caching, and disk scheduling. In contrast, our system uses frequent sequence mining to find files related by a workflow.

Griffioen and Appleton [5] also consider the problem of which files to prefetch in a file system. Their technique tries to estimate which file B has the highest probability to be opened (accessed) after a file A is opened. Then, if A is opened, the system automatically prefetches B. To estimate the probability, they create a graph where vertices are files and there is an edge from A to B if A is opened before B and at most k files were open in the meantime (where k is a system parameter). Weights on edges indicate how many times this happened. Kroeger and Long [6] address the same problem, but use a more powerful technique called Prediction by Partial Match, in which the latest k characters (a character is a file open event) are used to predict the next character.

Connections [9] is a file system search tool that uses context information to improve search quality, where context refers to what files are accessed together. To establish context, Connections traces file system calls and creates a graph where vertices are files and edges indicate files accessed at nearby times. This graph is used to augment content-based search.

Provenance systems [16] aim to track the lineage of files, so that users can know which files influenced the creation or modification of a given file, or to determine which files are influenced by a given file. This information is obtained by tracing system calls to determine when a process reads a first file and later creates or writes to a second file, indicating that the first file possibly influenced the second file. Doing so requires information that is not available at file servers, such as the process that invokes a system call.

8. CONCLUSIONS

Automatically identifying and grouping files that are related to the same workflow is important for improving the usefulness and efficiency of management tasks like reporting, file placement, migration, and archiving. We presented Autograph, a system tool that automatically extracts workflow file signatures by examining traces of file accesses, finding repeated and correlated accesses, and inferring files that likely belong to the same workflow. Autograph leverages several techniques from the data mining community, including frequent set mining and clustering using frequent itemsets. Our approach relies only on network file system traces, meaning that it can be deployed at file servers, which may be easier than client-based tracing in large or production environments.

We evaluated Autograph using NFS traces of real workflows and found that workflows are repeatable, in that they access a similar set of files across different episodes. Leveraging this repeatability, Autograph successfully distinguishes between multiple concurrent workflows and between workflows that access overlapping sets of files. It outperforms an alternative approach based on temporal locality graphs, because it tracks more than just pairwise relationships, and because it permits files to be included in multiple clusters, if appropriate. Autograph can be tuned to provide high recall or high precision, to meet the needs of the usage scenario. We describe rules of thumb for setting Autograph's configuration parameters to achieve these results.

A limitation of our approach is that it can only extract file signatures containing files that have been accessed. It does not capture files that are not accessed or not accessed "frequently enough," such as documentation files in a compilation workflow. Capturing these files requires augmenting our technique with other techniques, such as looking for unaccessed files that reside in the same directory as those files that are accessed.

9. REFERENCES

[1] P. Lyman and H. R. Varian, "How much information." 2003. Retrieved from http://www.sims.berkeley.edu/how-much-info-2003 on September 17, 2007.

[2] X. Yan, J. Han, and R. Afshar, "CloSpan: mining closed sequential patterns in large datasets," in *Proc. SIAM Intl. Conf. on Data Mining (SDM)*, (San Francisco, CA), pp. 166–177, May 2003.

[3] B. C. M. Fung, K. Wang, and M. Ester, "Hierarchical document clustering using frequent itemsets," in *Proc. SIAM Intl. Conf. on Data Mining (SDM)*, (San Francisco, CA), pp. 59–70, May 2003.

[4] Y. Zhou, T. Kelly, J. Wiener, and E. Anderson, "An extended evaluation of two-phase scheduling methods for animation rendering," in *Proc. 11th Workshop on Job Scheduling Strategies for ParallelProcessing (JSSPP)*, (Cambridge, MA), pp. 123–145, June 2005.

[5] J. Griffioen and R. Appleton, "Reducing file system latency using a predictive approach," in *Proc. USENIX Summer Technical Conference*, (Boston, MA), pp. 197–207, June 1994.

[6] T. M. Kroeger and D. D. E. Long, "Predicting file-system actions from prior events," in *Proc. USENIX Annual Technical Conference*, (San Diego, CA), pp. 319–328, Jan. 1996.

[7] G. H. Kuenning and G. J. Popek, "Automated hoarding for mobile computers," in *Proc. 16th ACM Symposium on Operating Systems Principles (SOSP)*, (St. Malo, France), pp. 264–275, Oct. 1997.

[8] S. Shah, C. A. N. Soules, G. R. Ganger, and B. D. Noble, "Using provenance to aid in personal file search," in *Proc. USENIX Annual Technical Conference*, (Santa Clara, CA), pp. 171–184, June 2007.

[9] C. A. N. Soules and G. R. Ganger, "Connections: using context to enhance file search," in *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, (Brighton, UK), pp. 119–132, Oct. 2005.

[10] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," in *Proc. 1993 ACM SIGMOD Intl. Conf. on Management of Data*, (Washington, D.C.), pp. 207–216, May 1993.

[11] M. Zaki and C. Hsiao, "Charm: an efficient algorithm for closed association rule mining," in *Proc. 2nd SIAM Intl. Conf. on Data Mining (SDM)*, (Arlington, VA), pp. 457–473, Apr. 2002.

[12] C. J. Van Rijsbergen, *Information Retrieval, 2nd edition*. London: Butterworths, 1979.

[13] B. Larsen and C. Aone, "Fast and effective text mining using linear-time document clustering," in *Proc. of Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, (San Diego, CA), pp. 16–22, Aug. 1999.

[14] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," in *Proc. 13th ACM Symposium on Operating Systems Principles (SOSP)*, (Pacific Grove, CA), pp. 213–225, Oct. 1991.

[15] Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou, "C-Miner: Mining block correlations in storage systems," in *Proc. Conference on File and Storage Technologies (FAST)*, (San Francisco, CA), pp. 173–186, Mar. 2004.

[16] K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware storage systems," in *Proc. USENIX Annual Technical Conference*, (Boston, MA), pp. 43–56, June 2006.