A Thesis

Submitted For the Degree of

Master of Science (Engineering)

in the Faculty of Engineering

by

Kamala R

Computer Science and Automation

Indian Institute of Science

BANGALORE – 560 012

JULY 2013

TO

Friends and Family

Acknowledgements

First and foremost, I would like to thank my advisor, Prof. K. Gopinath for his valuable insights and guidance throughout my stay at IISc. I would also like to thank all my friends and fellow researchers from whose machines I have recorded the data that is important for my work. Further, I have had many stimulating conversations with Suparna, Piyush, Aarthi, Arvind, Gupta, Amar, Bharath, Raghu, Priyanka and Poorna that I will look back on with pleasure. Last, but not the least, I thank my family for standing by me during my many moments of frustration.

Research is the process of going up alleys
to see if they are blind.
– *Marston Bates*

Publications based on this Thesis

Kamala R, K. Gopinath, MIST : MIgrate the Storage Too PuCWIC 2013, Pune Celebrations of Women in Computing, April 2013

Abstract

We address the problem of migration of local storage of desktop users to remote sites. We show that it is feasible to use information about files accessed to determine clusters and hot-spots in the file system. Using the tree structure of the file system and a similarity measure across accesses, we can approximate the working sets of the data accessed by the applications running at the time. Finally, by relaxing the assumption of disconnected operation, our results indicate that our technique reduces the amount of data copied by two orders of magnitude, bringing it into the realm of the possible.

Contents

List of Tables

List of Figures

List of Algorithms

Keywords

Storage migration, Remote Access, User Access Patterns, Clustering

Notation and Abbreviations

VM: Virtual Machine

COW: Copy on Write

LAN: Local Area Network

VPN: Virtual Private Network

TWh: Tera Watt hour

TLB: Translation Lookaside Buffer

VFS: Virtual File System

MPLS: Multiprotocol Label Switching

Chapter 1

Introduction

In desktop and server environments, the storage state over a period of time can be quite substantial. There are many use cases where migration of storage state is essential. A good use case is that of synchronisation of data across multiple devices, which is becoming an ubiquitous problem. Many mobile devices provide mechanisms for complete synchronisation of the data on device with a larger backup store. The reverse may sometimes be necessary and this is where intelligent synchronisation or choosing the data to be backed up becomes important. This scenario is especially true for a single user who has multiple devices of varying capacities, each of which holds some fraction of user data.

Another good use case is the migration of local storage of virtual machines. The power consumption of desktop PCs has been found to be about 100 TWh/year and this constitutes nearly 3% of the energy consumed in the US, of which 65 TWh/year is consumed in enterprise environments according to Das et al. [9]. Since the power consumption of machines in the sleep state is 1-2 orders of magnitude lesser than in the idle state, sizeable power savings can be achieved if the machines are put to sleep while idle.

An approach that was recently put forth by Das et al. [9] is to maintain the user's desktop environment as a virtual machine, migrate it to a centralized server while idle and put the machine itself to sleep, thus maintaining an *always on* state. Live migration

involving transfer of memory and CPU state only has been extensively explored and it has been shown by Clark et al. [7] that the downtime may be as small as 60ms. Disk migration poses a bigger problem as typical hard disk sizes are now of the order of hundreds of gigabytes and cannot be migrated without incurring unacceptably slow migration times.

Our approach to the problem of local disk state migration attempts to find a middle ground between migrating the entire disk beforehand and on-demand transfer of data. The idea is to determine the state that needs to be moved from the previous accesses of the user. We base our algorithm on the assumption that a user is only working on a small set of tasks at a time. This assumption is empirically validated on a large industrial dataset. We take care to filter out the accesses corresponding to background activity such as the recording of system logs which are generally considered irrelevant to the applications run by the user. If the disks at the source and destination are not in sync, then storage migration becomes interesting.

Determining the fraction of storage state in active use has other use cases as well. One example is disconnected operation of mobile devices, which have a smaller capacity than laptops or desktop machines. Finding and migrating data corresponding to tasks that the user is working on serves to reduce the storage state, while at the same time trying to ensure minimal disruption to the user. A second use case is the manipulation of extremely large datasets or bigdata. Deciding the data to be placed in the front end cache of a large server such as an SSD drive for fast access is yet another use case.

Obtaining a good grouping of accesses hinges on selection of an appropriate similarity measure and cluster validity measure. We define an information theoretic similarity measure that derives from the tree structure of the directory paths accessed and employ agglomerative clustering technique to group accesses. Our technique works well on anonymised datasets so long as we are able to obtain the structure of the file system. An inspection of the clusters obtained from our local dataset also allows us to tag the groupings found with strong hints regarding the high level task corresponding to the group. This enables the user to take a better informed action, as required. The above is

applied to 10 days' worth of traces on user data that we locally collected as well as on the older SEER dataset to further validate our results.

Our main contributions are:-

- The selection of an appropriate similarity measure.
- Automatically discovering working set(s) of directory paths that is likely to correlate with key user tasks.

Since datasets that record file system activity of users in the personal computer setting are scarce and those which are available are not recent, we have collected fresh traces. The data collection methodology is touched upon in chapter 3. We describe the system model in detail in chapter 4. This chapter also presents the algorithm that consists of a rough heuristic grouping followed by fine-tuning, as necessary. The evaluation section compares our technique with file-caching approaches and we then discuss the storage and runtime costs of our approach. Finally, we conclude by touching upon some directions for future work.

Chapter 2

Related Work

The related work described here consists of two different sections. The first relates to *Storage Migration*. Storage migration is a cross-cutting problem and occurs in many settings. Multiple solutions have been proposed in various contexts, some of which we describe here. The second section is that of *Data Collection and File System Studies*. Since we have collected data of desktop users, a few of the techniques outlined in these papers were quite useful. Furthermore, some of the papers collected data as part of their studies which have been subsequently made public. We have used the data from these studies to make observations that support our work.

2.1 Storage migration

Local storage may be used for the availability, security, privacy and performance it provides. Some of the different solutions put forth are as follows:-

- Tait et al. [**?**] addressed the storage migration problem in the client-server model, where the aim was to intelligently pre fetch files from a server into the client cache based on a match between trees of previous accesses stored for each program that give a clue of the files that are most likely to be accessed based on the previous runs of the programs.

- The following papers describe techniques to transfer the entire contents of the disk

from source to destination. Most of them deal with migration of local storage of virtual machines. In some cases, this problem needs to addressed in order to achieve a goal such as availability.

- Bradford et al. [6], explored transfer of storage of the virtual machine during live migration across a WAN. In the bulk migration phase, the entire disk image is transferred first, followed by deltas in parallel with memory state migration of the VM until the storage state is completely synchronised. Migrating a machine with 512 MB RAM and 1 GB virtual hard disk is reported to take approximately 3700 seconds. This migration time is unacceptable for desktop users who expect quick response times. The hard disk size considered is also not representative of current user hard drives, which is of the order of 320 GB.
- Mashtizadeh et al. [15] outline three techniques for transfer of the disk storage, all of which consist of first cloning the virtual disk at the destination before employing snapshotting, dirty block tracking or I/O mirroring. I/O mirroring is found to perform best, although migration time is close to that for plain disk copy.
- Akoush et al. [4] describe a technique of maintaining the synchronisation between the source and the destination by devising an algorithm to skip over sectors that are constantly modified to only transfer those that are modified and relatively stable. This process is repeated until all modified sectors of the source have been transferred. However, the disk states must be consistent or transferred initially for this approach to work.
- Cully et al. [8] aim to provide high availability as a platform service by employing virtual machines. In order to do this, they keep the memory as well as the disk state of the active virtual machine synchronised with a backup virtual machine. Instead of lock-step synchronisation that imposes unacceptable overhead, they employ a check pointing mechanism in which no visible

machine state is lost and the machine executes speculatively. Even so, this system exhibits considerable latency especially for network intensive applications.

  - Wood et al. [21] describe consolidation of cloud storage and MPLS based VPNs of enterprises to create a VPLS which brings multiple MPLS endpoints into a LAN like network of virtual machines. Disk state migration is carried out in two stages - using asynchronous and synchronous replication. The disk state to be transferred is further reduced by employing content based redundancy techniques.

- We now discuss storage migration where the source and the destination are often not fixed and/or there are multiple copies of the data in different locations. This setting is typical of network file systems and some of the optimisations employed here are very effective. The main focus is usually user mobility and convenience.

  - Sapuntzakis et al. [19] have designed a framework for user mobility where the state of a machine is encapsulated into a data-type called a capsule which consists of the memory as well as the disk state of a machine. These capsules are implemented as virtual machines and are moved around as required. Various techniques such as memory ballooning, COW disk hierarchies, demand paging and hashing are employed to bring down the amount of state that is to be transferred so that they can be accessed at the user's location.

  - Kozuch et al. [12] superimpose virtual machine technology over distributed file system in order to improve performance. The virtual machine is copied from the server to client when it is accessed from any location and updated to the server when the client suspends it. Optimizations such as compression, transferring increments from a standard image, selectively transferring files after organising them into a tree and predictions based on context are considered.

  - Muthitacharoen et al. [17] have designed a low bandwidth network file system

using Rabin fingerprints and SHA hashes of the blocks to improve performance. Rabin fingerprints are used for finding chunk boundaries while SHA hashes are used to find data that is already present and need not be transferred. Compression is used to further reduce the data transferred.

- In some cases, transferring the entire contents of the disk is not necessary. If we can make the assumption that even a weak network connection can be maintained with the source machine, this relaxes the time constraint imposed by necessitating the movement of entire disk contents. We move only a subset of the data on disk and by choosing the data to be moved carefully, we can operate close to disconnected mode.
  - Wildani et al. [20] describe an approach for grouping block I/O traces of enterprise servers by defining a similarity measure and applying traditional clustering methods such as K-means clustering.
  - Keunning et al.[13] group accesses of users to determine the files to be cached so the users can operate in disconnected mode. Though our approach is similar, an important difference is that their approach is in the context of mobile computing, and therefore more constrained. Our approach, however, uses the directory structure of the file system into account and operates at the directory level. We also note that their approach assumes non-concurrency while our technique indifferent to the same. Since Keunning et al. do not take history into account, their clusters taken together simply represent the most recently accessed files. Our approach of persistent clustering takes history of clusters obtained in previous sessions into account and thereby exhibits better performance.
- More recently, Zhao et al. [22] have come up with a hardware based technique to track the working set in memory. The problem is detecting transition between phases which they attempt to overcome by using a monitor to detect hardware event such as a significant number of TLB or cache misses.

2.2 Data collection and File system studies

We have looked at several file system studies both for the techniques of data collection used as well as for their observations and inferences drawn from the data. The work here has been classified as desktop studies and network file system studies.

- Desktop studies
  - Meyer et al. [16] consider the problem of deduplication at the file and block level. To this end, they have conducted a study of approximately 1000 users at Microsoft Research Labs over a period of four weeks. They captured weekly snapshots of the user filesystem recording both metadata and hashed file data. Various file system properties such as file size and file age are considered and observations from the data are compared to those drawn from data of earlier studies.
  - Agrawal et al. [3] study various characteristics of file-system metadata. Of particular interest to us are the observations that there is a strong negative correlation between namespace depth and file size and that the mean file size drops be two orders of magnitude between depth 1 and 3 with a drop of 10% thereafter.
  - Douceur et al. [10] is one of the early large scale studies of desktop users. Snapshots of the file-system metadata was captured for each user and a number of properties, such as distribution of file size and directory depth were studied. Amongst other things, it was also found that there was a correlation between file extension and file size.
  - Evans et al. [11] have studied file sizes across different operating systems and claim that file sizes cannot be approximated by simple distributions or mixture models. In particular, audio and video files show peaks at specific file sizes that need to be modelled using a combination of flexible distributions. This seems to indicate that the distribution of file sizes is specific to each user and hence must be tuned with appropriate parameters.

  - The file system workloads of different types of users running both UNIX and NT was studied by Roselli et al. [18]. The paper describes the collection and analysis of the data and studies the frequency of reads and writes, the dependency of reads on file cache and the effect of write delay on write traffic.

- Network file system studies

  - Leung et al. [14] conducted a study of systems running CIFS as most of the studies on networked file systems are based on NFS data. They study various aspects of file system access and make observations on read-write ratios and sequentiality of access. It is also determined that a small number of files are responsible for majority of bytes transferred. Further, there is no clear correlation between frequently opened files and frequency of access.

  - Anderson [5] specify in detail the techniques used for capture and representation of intense network workloads. Data is represented in binary form and special analysis techniques such as approximate quantiles and data cubes are used. In particular, they emphasise the necessity of maintaining original data separately even if we choose to anonymise the same. Thorough validation of captured data is recommended to be carried out to ensure that the data correctly represents the systems being logged. Analysis of file sizes and sequentiality are carried out on captured data as an example.

Chapter 3

Data collection

Publicly available file system traces such as the Berkeley traces, SEER traces and CODA traces may not represent current desktop usage. In the more recent traces collected at Microsoft Research for their desktop studies, the file timestamps have not been preserved accurately and cannot be used for validation. Therefore, there was a need to collect fresh desktop traces.

We have conducted a measurement study in which logs from 7 users were collected over a total period of 160-240 hours. The read/write activity was recorded at different levels of abstraction both at the file system level as well as the block level. Files that were memory mapped were also recorded. We used the SystemTap tool to monitor file system as well as the disk accesses.

SystemTap allows for tracing events in a running machine without the requirement of a kernel compile. The scripts are to be written in a C-like language. These consist of events, also known as probes, and corresponding handlers. Internally, SystemTap builds a kernel module which is inserted at the time of running at all the locations where the probes have been specified. The probes may consist of kernel functions or even statements at specific locations of the kernel. This kernel module is removed once the scripts stop running.

This tool was installed on 7 machines and scripts collecting logs were run in the background for a period of one week to monitor user behaviour without unduly affecting

the performance. SystemTap wiki page [2] indicates that the performance impact of running scripts developed is not noticeable and per our experiments the CPU usage due to supplementary processes was about 2% as indicated by the *top* process.

On two of the machines we were unable to collect the details of memory mapped files as the probe functions were not compatible with the particular kernel versions that the users were running. In particular, Systemtap is extremely sensitive to the kernel version installed and this limited the number of systems from which we were able to collect logs.

The traces were captured by running a set of three SystemTap scripts which are spawned by a shell script. The shell script polls the system periodically to check that the scripts are running and restarts them as required. Two files are maintained for check pointing purposes, which are updated every two hours and every day, respectively. These files ensure that even if the system is restarted, the amount of state lost is reduced. The shell script itself is added to the start-up applications of the user and runs in the background for the period the user logged into the system. The shell script automatically stops when it has run for the specified period of time.

The systems that the scripts were deployed on varied widely and included 8 core processors with 8GB memory, dual core processors with 2 GB memory as well as Pentium 4 processors with 1 GB of memory. However, they were all uniform in that they were running Linux.

3.0.1 Trace details

As mentioned above, for each user, traces were collected over a period of 7-10 24 hour periods, which translates to 168-240 hours of user activity logging for 7 users. A single 24 hour period could potentially comprise 3 working days. Some machines were left powered on through the periods of disuse as well. The users mostly comprised of researchers from a few different labs as well as one administrative personnel.

The traces for each day consist of

- VFS reads/writes :

  This file consists of the process id, process group id, inode number, full path name,

offset, length and type of request (Read/Write) for each read/write request that goes through the VFS layer. We have not recorded the read/write accesses that are caught by the cache.

- Block Reads/Writes :

  We record type of request, process group id, process id, device number, inode number and sector number. We also record offset and length of request.

- Files Mapped :

  For the memory mapped files, we have made of record of file name, process id, process group id, length and offset of the requested map. However, we were unable to record the inode numbers or the mode of use (read/write). The ¡process id, process group id¿ tuple can be used in combination with the other two traces to extrapolate details such as the inode number of the corresponding file accessed.

For all requests, we have recorded timestamps at the granularity of seconds.

3.0.2 Validation of trace data

As a preliminary step, we removed the logs in the traces that refer to the running of scripts for the purpose of measurement itself, i.e. self-logging information. In order to validate the trace data collected, we measured the magnitude of reads/writes at the VFS level, block level and the total size of memory map requests.

We observed that in most cases, the total size of VFS accesses are larger than than of the block accesses. We take this to reflect the effectiveness of caching for reads and that of write delay for writes. It was also noticed that that the total size of memory map requests was extremely large in terms of magnitude, running into some hundreds of gigabytes. However, on looking deeper, we found that most of the memory mapped files are library files and the number of unique files is 0.02-0.27 % of the total number of files mapped.

Finally, we checked to see that the block reads and writes reflected either VFS level requests or that from memory mapped files. We performed this check for reads and

	VFS			Block			Memory mapped	Overwrites	
	Reads	Writes	Ratio (R:W)	Reads	Writes	Ratio (R:W)		Size	Unique Files(%)
User 1	158.64	51.39	3.09	29.33	17.58	1.67	12466.21	3.55	0.37
User 2	38391.94	34.21	1121.98	84.62	22.39	3.78	4274.20	2.74	0.52
User 3	8.31	2.71	3.06	0.71	3.17	0.22	NA	0.871	0.008
User 4	539.42	206.50	2.61	8.00	52.47	0.15	6134.37	7.77	0.07
User 5	35.58	2.80	12.70	2.21	1.78	1.24	NA	0.62	0.11
User 6	123.19	83.43	1.48	29.48	47.97	0.61	6648.81	3.33	0.09
User 7	32.51	5.11	6.36	1.24	5.05	0.25	6828.72	1.50	0.18

Table 3.1: Summarizes some common metrics of the data collected for each user. The amount of overwrites is computed at the VFS level. The unique files is expressed as a percent of the total number of files overwritten. All sizes are in GB.

writes separately. For the file system requests, we found the list of common inodes between VFS requests and block level requests. The logs corresponding to these inodes were filtered out from the block level requests. Since the block level logs record only inode number and the traces for memory mapped files record only file name, we used the process id, process group id tuple for matching. Here, we are unable to distinguish between reads and writes because we do not have a record of access type.

We find that the number of unaccounted requests consists mainly of the background flush process and a small number of process id, process group id pairs that may be accounted for as the requests logged when the scripts logging other requests were being restarted.

3.0.3 Observations

From Table 3.1, we observe that in many cases the magnitude of data touched both at the VFS and the block level is extremely large. Second, the amount of overwrites and the number of files overwritten are small. The read-write ratios do not show any consistent pattern. Lastly, although this has not been represented in the table, we find that the swap writes are negligible. A significant portion of it is system data. The personal

	Reads(%)		Writes(%)		Memory mapped (%)	
	System	User	System	User	System	User
User 1	87	13	51	49	98	02
User 2	30	70	76	24	98	02
User 3	99	01	100	0	NA	NA
User 4	08	92	54	46	94	06
User 5	99	01	100	0	NA	NA
User 6	64	36	67	33	76	24
User 7	99	01	99	01	99	01

Table 3.2: Split up of VFS and memory mapped accesses at a high level. The % here indicate amount of reads or writes of the System/User data respectively. Records of browsing history are included under system data. The value 0 here indicates that the data is in the order of KB and is less than 1 MB.

	Number of file accesses(in millions)	
	Reads	Writes
User 1	9	2.8
User 2	0.5	16.25
User 3	19	22
User 4	10.66	14.78

Table 3.3: Number of read and write accesses observed for each of the four users

data accessed by some users, as reported by Table 3.2, is minimal and migration can be easily supported. It is the other users who are of interest to us as they represent the more challenging problem. Of these, only one of the four users shows uniform number of accesses temporally. Table 3.3 indicates the number of reads and writes seen by each user in this period.

Solutions based on Markov chains or tries represent some subset of file caching approach which we will substantiate as undesirable in our evaluation. A higher level approach involves the caching of relevant directories which cuts down on some of the compulsory misses. We combine grouping of related accesses with caching of directories to discover user tasks. We determine related directories by the extent of their overlap with each other and define a user task as a group of related directories that are common to a large set of accesses. The idea is to discover different tasks that are part of a session

based on past traces. The data transferred will consist of different tasks and, therefore is at the granularity of directories. The tasks so discovered may not be in one to one correspondence with the application working sets. For eg, an application may work with two different parts of the file system and this would result in the discovery of two tasks that always occur together.

Chapter 4

Methodology

4.1 Model

The set-up we have considered consists of a desktop machine running a virtual machine connected via a fast network connection such as a gigabit Ethernet to the server. Remote access to the desktop occurs through a connection having a fraction of the bandwidth such as 1Mbps. The aim is to reduce the local disk state of the desktop that is transferred to the server when the desktop is put to sleep.

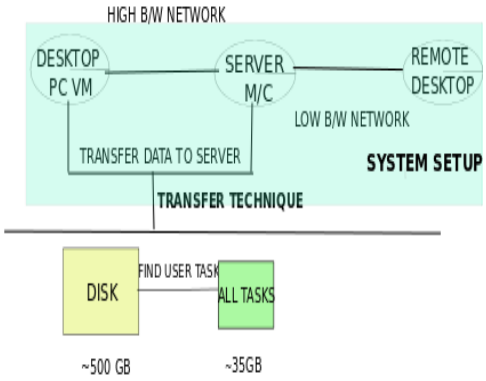


Figure 4.1: Overview of system and technique

For a typical hard disk of size 500 GB, it would take a little more than hour to transfer the contents of the entire disk, if the network bandwidth were dedicated to this transfer

and the machine were idle. Remote access to the desktop faces a similar problem but reaches the bottleneck sooner. Therefore, storage migration to the server and remote access to the desktop are essentially the same problem with different parameters.

We use a clustering based technique to find working sets of directories corresponding to user tasks. Figure 4.1 provides a high level view of our way of tackling this problem. We quote numbers of a single user for illustrative purposes only. Discovering tasks reduces the disk state to be transferred from 500 GB, which represents total disk size to 35 GB. Although the initial cost of transfer can be high, tasks are typically repeated which amortizes the cost of transfer. For example, a single working set of directories was discovered to be 7.5 GB in size. The same set of directories was then repeatedly accessed across the week.

4.2 Algorithm for Discovering Tasks

Our approach is based on the premise that user accesses are not completely random. More specifically, if a file belonging to a specific directory is accessed, it is likely that another file from the same or neighbouring directory will be accessed and that the likelihood of files in a directory being accessed is proportional to the number of accesses previously observed. The above observation leads to the problem of finding hot spots in the file system i.e. we attempt to find clusters of directories by using just the structure of the file system.

We rely on the assumption that a typical user accesses files in only a small set of directories at any given point of time to make our technique practical. In this aspect, the usage of desktops is significantly different from that of servers in which multiple parts of the file system may be active at any point of time depending on the diversity of content and the client base. The above assumption is empirically validated using the data that was collected at Microsoft Research in 2009 as part of the work for [16].

The idea is to discover different tasks that are part of a session and the set of directories corresponding to each task based on past traces. The data transferred will consist

of different tasks and since there is a large degree of overlap in the directory paths, we try and find groups of directory paths, each of which is common across a large set of accesses. The algorithm assumes by default that certain system directories can be filtered out. We initially considered having the user specify directories of interest but discarded this idea as the set of system directories which, if accessed, can be safely ignored form a much more standard list of defaults.

4.2.1 Prune

The prune phase takes the past traces as input and tries to come up with the set of directories that are most representative of what the user is currently working on. The number of directory paths accessed is of interest to us and we want to make the list as compact as possible. First, based on the directory paths, a prefix tree is constructed. Since we are only interested in frequently accessed items, lower thresholds are defined to discard prefixes that correspond to sporadic file reads/writes, for which there is no necessity to transfer the whole directory. Essentially, directory paths that constitute less than, say, 1% of accesses are ignored.

4.2.2 Group

After the list of frequently used directory paths has been generated, the next step is to cluster them based on the degree of overlap between different paths. We formalize the notion of similarity using a information theoretic similarity measure. Agglomerative clustering starts with each data point being in its own cluster, with each successive step merging the clusters to which the data points closest to each other belong. To terminate the aggregation of clusters, it is necessary to specify a threshold-value or a cut-off point. Clusters whose closest data points are further apart than this threshold value will not be merged.

The distance measure is derived from the similarity measure defined, and the distance value is computed between each pair of paths. After pruning of the prefix tree, the

Algorithm 1 Discover Tasks

Stage 1: Prune

Remove from consideration paths that are rarely accessed in the traces

Input: Past file system traces

Based on the directory path, filter out accesses that may be safely ignored

Build a prefix tree

$Lower\ Bound \leftarrow 0.01 * Number\ of\ Directory\ Accesses$

Remove all paths whose accesses less than Lower Bound

Output: Full directory paths which satisfy the above criteria

Stage 2: Employ a heuristic grouping method

Cluster the directory paths to represent different tasks based on the degree of overlap of incoming paths

Input: Output from stage 1

$maxdep \leftarrow Maximum\ depth\ of\ a\ directory\ path$

$depth \leftarrow 0$

while $depth \leq maxdep$ **do**

for Each cluster and directory path **do**

Check directory at level depth for mismatch.

Increment mis-match count by 1, if applicable

Compute distance measure

Split directory path and sub-directories into new cluster

based on value of distance measure

end for

$depth \leftarrow depth + 1$

end while

Output: Rough estimate of number of clusters

Stage 3: Determining the clusters

Fine-tune Clustering

Input: Estimate of number of clusters

Compute distance values between each pair of paths

Determine the distance value corresponding to rough estimate

$Threshold\ Values \leftarrow Set\ of\ distance\ values\ in\ reduced\ search\ space$

for Threshold values in search space **do**

Place each directory path in its own cluster(agglomerative)

Cluster paths

Compute point biserial value, to measure cluster validity

Update clusters based on validity measure

end for

Output: All tasks identified from traces seen thus far

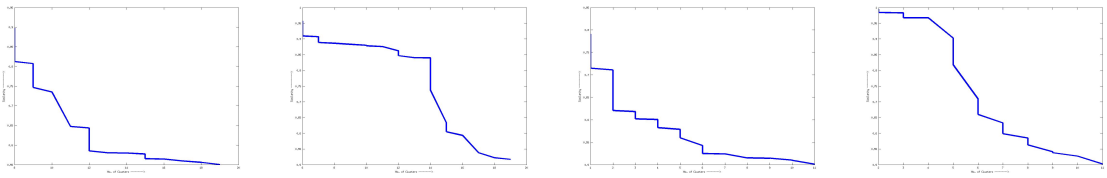


Figure 4.2: This figure plots the distances v/s the number of clusters. The first two graphs represent accesses made by the laptops in the SEER system corresponding to two different days. The next two graphs are representative of accesses made on two different days by an user in our dataset.

number of paths under consideration is small, thus enabling us to maintain the distance between all pairs of paths at low cost. To find the threshold for clustering, we plotted a graph of distance versus number of clusters for two different datasets, the SEER dataset as well as for the data that we collected. Figure 4.2 is representation of the graphs seen in the two datasets.

As can be seen from Figure 4.2, setting a single hard threshold for clustering will not work well and an alternate solution needs to be found. In order to reduce the search space of possible threshold values and get a rough estimate of the number of clusters, we employ a heuristic method. We define the distance of a path from any other path in a cluster as:-

$$Distance\ Heuristic \leftarrow \frac{\#\ of\ Directories\ that\ do\ not\ match}{Total\ length\ of\ directory\ path}$$

To start with, we assume that all paths are in a single cluster. We look at the top level directory first, followed by the second level directory and so on splitting to form new clusters as necessary. Starting with a rough estimate of the number of clusters obtained from the heuristic grouping, we now search for an local optimum value. The information theoretic similarity measure and corresponding distance are defined as follows:-

$$Similarity \leftarrow \frac{2*log(P(Common))}{log(P(Path1))+log(P(Path2))}$$

$where\ Common \leftarrow Overlap\ between\ Path1\ and\ Path2$

$$Distance \leftarrow \frac{1}{1+Similarity}$$

To select a threshold value in the reduced search space, we need a measure of cluster validity. For this purpose, we use the point biserial value [1], which takes into account the intra-cluster distance, inter-cluster distance and standard deviation of the data. We compute the point serial value for the clusters determined using the threshold values in the reduced search space and select the value that performs the best empirically. Once the clusters for the appropriate time period have been determined, we can simply move the clusters that are of a size appropriate for network transfer automatically and prompt the user for guidance as regards to the remaining clusters.

Chapter 5

Evaluation

Our solution technique is based on assumptions which we validate using data collected by the authors of [16] as part of their study in 2009. This dataset contains weekly scans (static snapshots) of approximately 1000 users of different job functions at Microsoft Research, recording file system metadata, file metadata as well as hashed file data over a period of 4 weeks.

5.1 Empirical Observations

Figure 5.1 plots the cumulative percentage of users against the modified size of the disk (expressed as a percentage of data on disk volume). As can be seen from the figure 5.1, files accessed by users is a small fraction of data present in the disk. This leads to:-

Observation 1: Up to 90% of the users access less than 20% of the data in their disk and most access significantly lesser percentages.

From the above, we can surmise that the disk state essential to the running of user applications is much smaller as compared to the data on disk. The feasibility of our approach is strengthened by the above observation.

One pre-dominant solution put forth to the storage migration problem is to cache

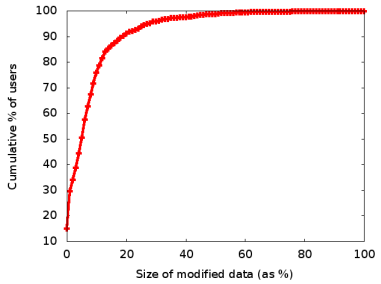


Figure 5.1: Cumulative percentage of users is plotted against the size of data modified, expressed as a percentage of the size of data on disk. For eg., the x-axis value (p%) corresponding to 40% on the y-axis indicates that 40% of users have a modified size less than p%

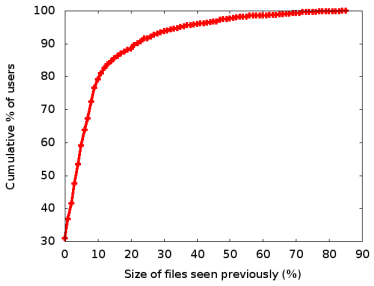


Figure 5.2: Cumulative percentage of users is plotted against the mean size of overlap between time slots, expressed as a percentage of size of modified data. For eg., if 'file1' is common across time slots 1,2 and 3 for a particular user, we calculate overlap between slots 1 and 2, and 2 and 3 respectively, and then average the overlap percentage.

some subset of files seen previously based on criteria such as time of access, frequency of access, etc. Caching of all files seen previously is the best that can be done in terms of file caching based on previous file accesses. We used data that was collected as part of the study conducted by Meyer et al. [16] as well as our local dataset to determine the performance when all previously seen files are cached.

Figure 5.2 plots the cumulative percentage of users against the size of common files with a previous time slot (expressed as a percentage of modified size in that time slot). The time slot considered here is a week. We find that the extent of overlap is quite small.

	Reads and Writes(%)	Writes only(%)
User 1	78	71
User 2	62	77
User 3	69	53
User 4	81	97

Table 5.1: Percentage of compulsory misses seen in our traces even if we cache all files accessed so far.

For our local dataset, we computed the miss ratio of file caching for a time slot of 24 hours. As can be seen from Table 5.1, the average miss rate for each user is quite high for file caching. Therefore, we make the following observation:-

Observation 2: File accesses are not very good predictors of future accesses.

This observation contradicts Keunning et al. [13] who define a hoard as a subset of files seen and report the discovery of a hoard size for files which is less than 50 MB. They claim that there are almost no hoard misses during operation. Per our data, as can be seen from Table 5.1, on certain days there is a miss rate of upto 90% even when all files are cached, as a result of compulsory misses.

We explore the ramifications when the data on the source machine and the destination machine are synchronised and when they are not. The latter case is the more interesting one and almost all of our experiments are connected with it.

5.2 Lower Bound on data to be transferred

The first scenario we consider is that of the data on server and user's desktop data being nearly synchronised. This is the simplest case and in order to estimate the data to be transferred, we look to the block writes of each user that we have recorded. The number of modified blocks represent a lower bound on the data to be transferred between the source and destination.

To quantify the amount of data written at the block level, we monitored the block dirtying rate at 10 minute intervals. From Table 5.2, it is clear that the data that needs to be transferred is quite small and is further reduced if we ignore block writes to system files such as those recording system logs. Therefore, when the data on the source machine is mostly synchronised with the destination, small block transfers at regular intervals in the background is the best solution.

	Block Writes	
	Total	Actual Writes (System Writes omitted)
User 1	1398	786
User 2	1758	89
User 3	2789	175
User 4	1687	55

Table 5.2: Cumulative number of block writes seen by 90% of the 10 minute time intervals when all writes are considered and after removing block writes corresponding to background processes and to system files. Each write is usually 4096 bytes in size.

5.3 Task Based Transfer

We now turn to the case where the data at source is not synchronised with the target machine. In this case, we work at the file system level instead of at the block level. An important reason for our choice is that predicting future accesses at the block level is much harder than at the file system level. Furthermore, predicting which parts of the file system may be accessed in future allows us to provide more meaningful feedback to the user on the current state of system operation.

Our evaluation is performed for sessions of lengths starting from 4 hours and progressing to 24 hours in steps of 4 hours. Our dataset spans a period of 10 days and by staggering sessions by half an hour, we obtain about 500 sessions for each session length, hence ensuring that all our results are statistically significant.

Table 5.3 represents the number of unique file paths and directory paths considered for each of the four users whose accesses were collected. It can be observed that the number of directory paths accessed are one order of magnitude lesser than the number

of unique file paths accessed. This is further corroborated by similar observations from the SEER dataset.

	Number of unique file paths	Number of unique directory paths
User 1	5109	341
User 2	388830	23506
User 3	477	46
User 4	74297	5888

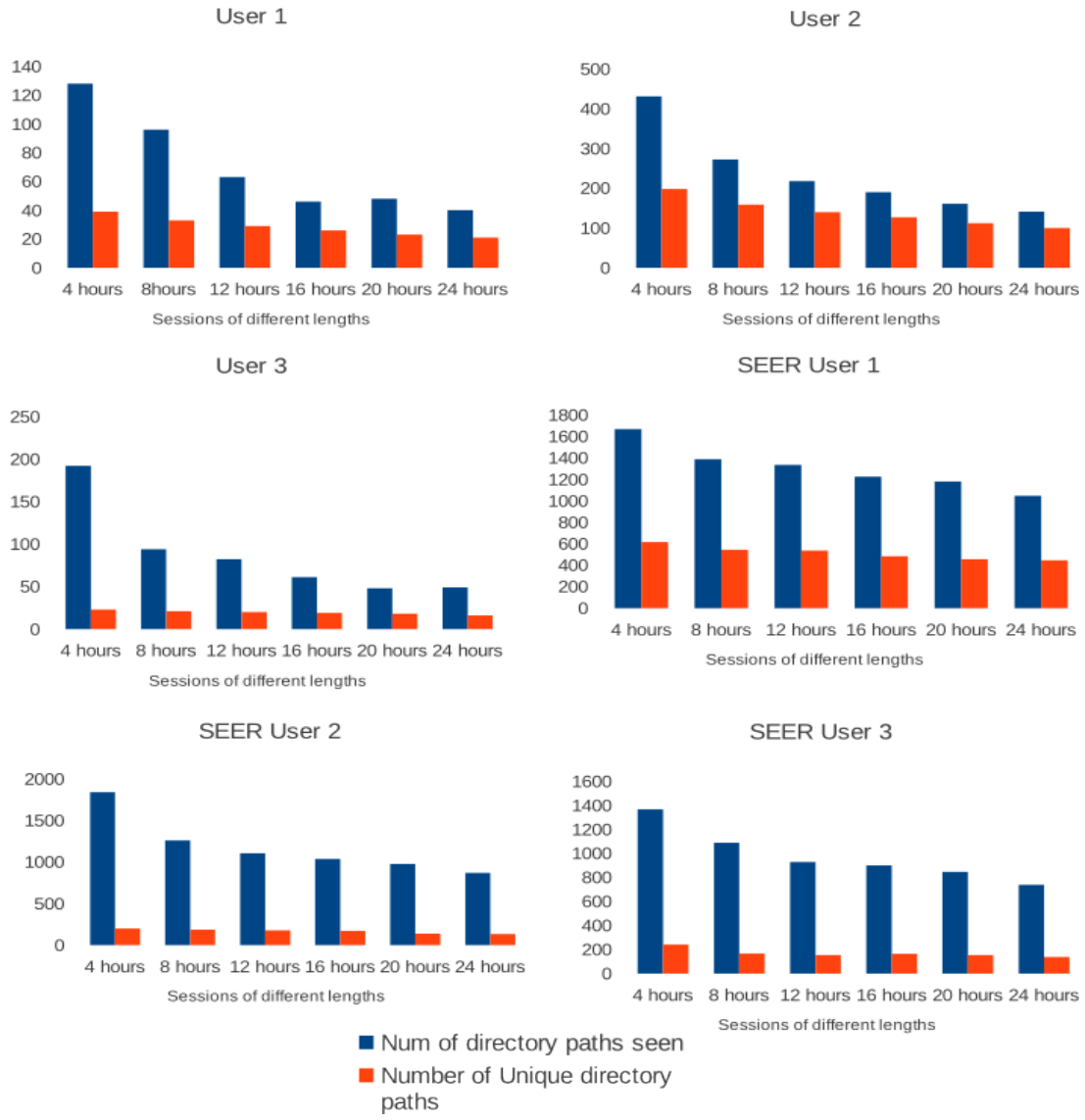
Table 5.3: The number of unique file paths is contrasted with the number of unique directory paths seen. It is observed that the number of unique directory paths is one order of magnitude lesser than the number of unique file paths. The user data collected as part of the SEER dataset show a similar trend.

Next, for each session length we compared the number of directory paths accessed across all sessions with the number of unique directory paths. This is repeated for each user and each session length, both for our locally collected data as well as for the SEER dataset. As can be observed from the graphs below, the number of unique directory paths accessed is a fraction of the total number of paths accessed confirming that paths are repeated across different sessions. This in turn reduces the cost of transferring directories as it is now amortized across the different sessions in which it is accessed.

Table 5.4 represents the total number of tasks discovered over ten days as well as the average number of tasks involved in per day. As indicated by the values of average and total number of tasks, there is an overlap of tasks across multiple days making it reasonable to transfer directories based on their membership in a particular task.

Based on the groupings obtained, we made the following observations:-

- The number of paths accessed as well as pairs with non-zero similarity determine the quality of clusters obtained. If either of these is too small, stopping at the hierarchical clustering stage is a good option.



- We notice that sometimes the estimation of the rough number of clusters can stand by itself, i.e on computing the cluster validity measure, we find that the data is clearly delineated. In other words, there is no necessity for fine-tuning.

We have manually inspected the data and compared the results with clusters obtained by applying our approach. We find that there is a close match between the automatically generated clusters and those that are found by inspection. Manual inspection allows for strong hints to be provided regarding the user tasks discovered. While we may not know the application running, discerning the high level intent of an user is possible. As an

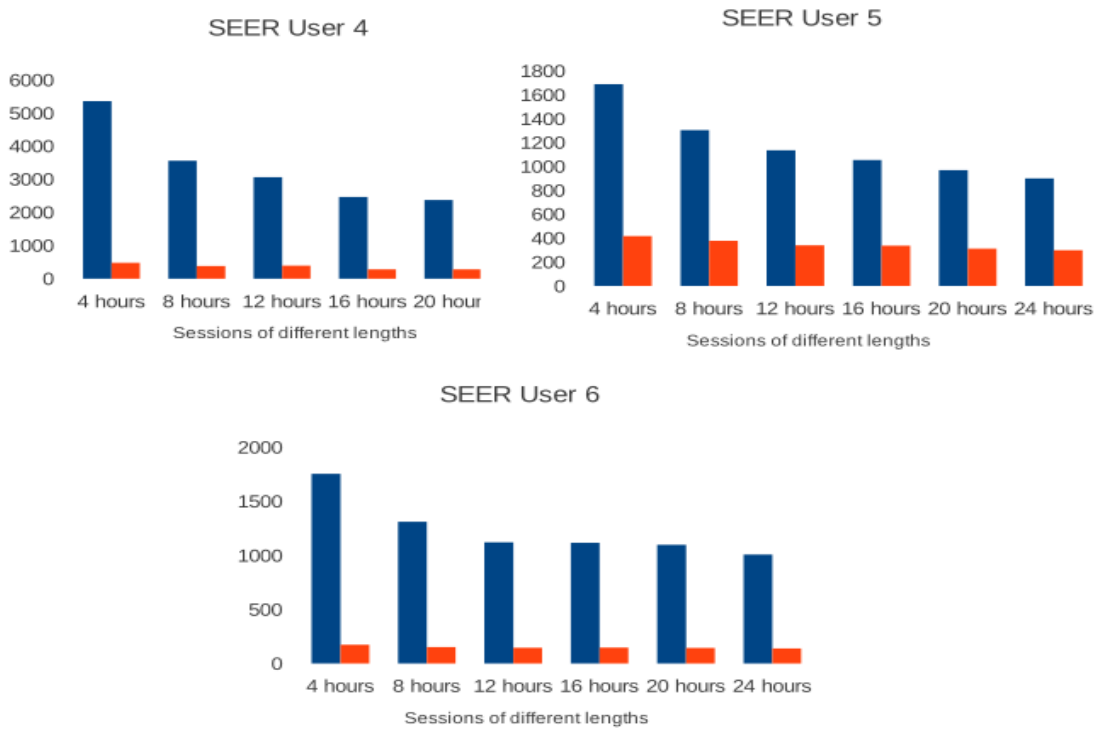


Figure 5.3: A comparison of total number of directories accessed versus number of unique directories accessed.

example, we can determine that the user is working on or browsing through the Linux source code, but are unable to know which application is accessing the data.

5.4 Miss Ratio Comparison

As mentioned before, the most widely used policy in file caching is to maintain a subset of the files seen thus far. We take this strategy one step further and consider the situation when all files seen are cached. This is the best possible strategy if we base our predictions solely on past accesses. The only misses will be compulsory misses to files that have not been seen before.

Clustering, in which groupings are discovered based on accesses of the previous session only, essentially implements caching all frequently used directories to better performance. As can be seen from Figure 5.4, it does significantly worse than file caching as higher level

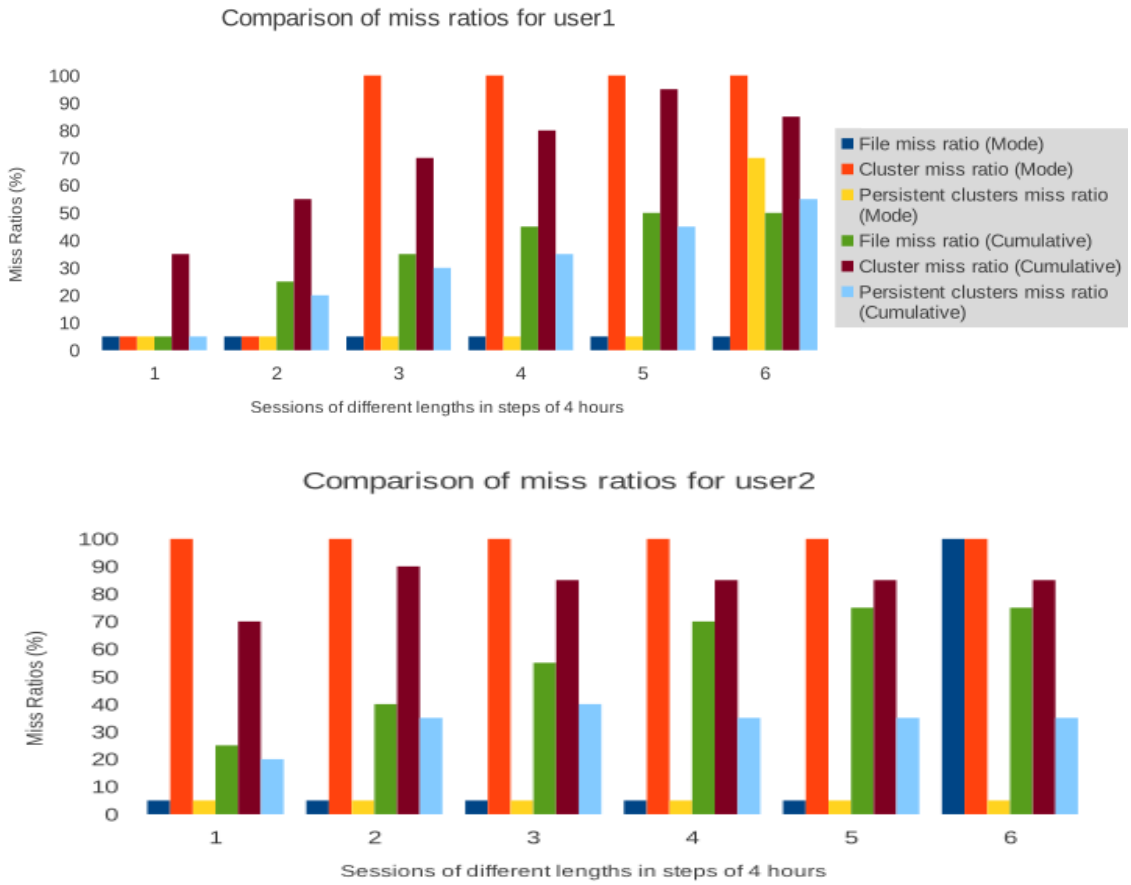
	Using Reads & Writes				Using Writes only			
	# of Tasks	Avg	Actual # of tasks	Actual avg	# of Tasks	Avg	Actual # of tasks	Actual avg
User 1	16	2.5	14	2.1	15	2.7	12	2.4
User 2	31	5.7	26	4.5	18	3	13	2.8
User 3	6	1.6	6	1.3	4	1	4	1
User 4	9	4	7	3.8	11	3	8	2.5

Table 5.4: Number of unique tasks found for each user and the average number of tasks involved in per day. Multiple tasks occur on a single day and there are several days on which the same tasks recur, hence pushing up the average in some cases. The fields specifying the actual number of tasks and actual average been obtained by manual interpretation of the data.

directories that are considered to have too large a footprint are omitted from caching. Persistent clustering, which maintains all clusters seen thus far, outperforms file caching in many instances and performs comparably in others. A more detailed comparison can be seen in Figure 5.4.

To strengthen our results further, we applied our technique to the SEER dataset. Although the data is not recent, it captures file accesses made by nine different users who were using personal computers. This scenario matches our setting exactly and therefore, we judged this to be the best suited dataset for further validation of our technique. As mentioned before, since our technique uses only the structure of the file system, the anonymisation of the actual directory names makes no difference.

Furthermore, since human interpretation of the clusters is not possible, choice of cluster validity measure plays an important role for anonymised data. However, on inspecting the clusters reported by our technique, we note that high spatial locality



within each cluster is maintained and that the paths which are deeper are separated from shallower paths effectively. This is due to the fact that shallower paths are much larger in size that deeper paths and we would like to keep the two separate, where possible.

5.5 Storage and Runtime Cost

The storage cost incurred by the construction of the prefix tree is proportional to the number of unique paths amongst the accesses, denoted by N. The full pathname of a file is an unique path. We first determine the number of accesses to each file and then build the prefix tree. The time taken to build this tree is bounded above by O($N*max-depth$). Since $max-depth$ can be considered to be a constant, the complexity is effectively O(N).

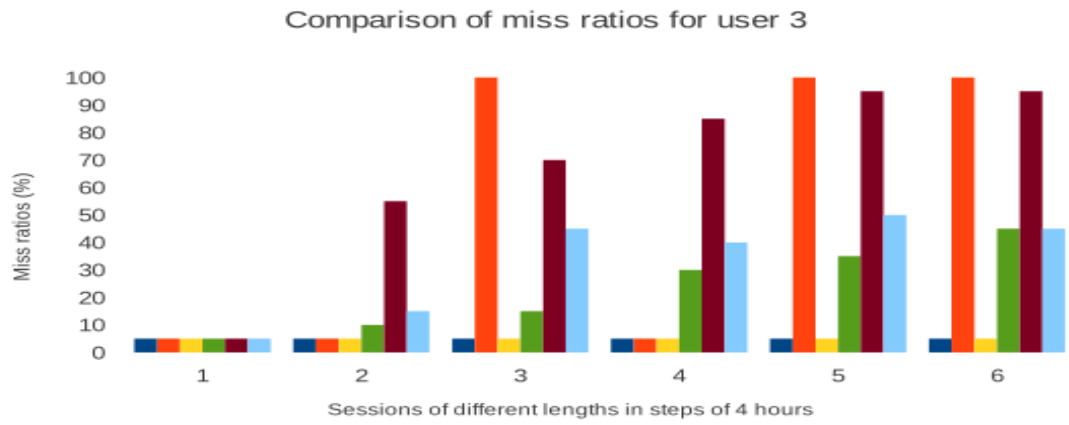


Figure 5.4: A comparison between the miss ratios for file caching (caches all files seen thus far), clusters based on previous session accesses and persistent clusters (all previously seen clusters are cached) for two different users. The x-axis corresponds to sessions of varying time intervals starting from 4 hours and continuing to 24 hours in steps of 4. The comparison is charted for each of the session lengths. The first three columns represent the most frequently seen miss ratio range amongst all sessions for each of the three approaches, i.e. if we have 10 sessions and a miss ratio in the range 0-5 % is most frequent, 5% is plotted in the bar chart. The next three columns report the cumulative miss ratio of the three approaches which means that for a majority of the sessions, the miss ratio is less than the corresponding value in the bar chart for that particular approach.

The clustering of directories occurs rapidly after the accesses have been processed. The pruning of the prefix tree causes the inserted paths associated with the files, which were in the thousands, to be reduced drastically. Thus, the number of directories returned by our approach has been found to be less than 50.

Similarity and distance measures are computed between every pair of paths. This has a complexity of O(N^2). Further, the hierarchical clustering as well as the agglomerative clustering each have a linear time complexity. To be precise, in order to find the optimal threshold value empirically, it is necessary to run the agglomerative clustering for a few different threshold values. This makes the complexity of finding clusters, O($N * num -$

attempts), where $num - atttempts$ is the number of times agglomerative clustering is run.

The search space of the threshold value is reduced by obtaining a rough estimate of the number of clusters by hierarchical clustering and a binary search is employed in the reduced search space. This limits the number of times agglomerative clustering is to be run, thereby preserving the linear bound. Since the value of N is small, clustering is efficient even though the overall complexity is O(N^2).

Chapter 6

Conclusions

We have demonstrated the feasibility of finding and migrating working sets of directories as an effective solution for storage migration of virtual machines. The ideas are intuitive and make use of the fact that most users actively use only a fraction of their disk contents. We have validated the groupings both by manual inspection and by employing a cluster validity measure for the local dataset. We have further strengthened our argument by successfully applying our approach to anonymized dataset and have therefore established that our technique can be employed without infringing on privacy.

Although on an average persistent clustering outperforms file caching consistently, there may be times when file caching performs exceptionally well. The next step would be to come up with a hybrid approach of file caching and clustering. The challenge here will be to predict which approach to employ for a particular timeslot beforehand to achieve the best performance.

It would also be interesting to explore similarity between pairs of files within a directory to determine groups of files that are related to each other. These groups may contain files that are related to those accessed previously but which have not been accessed themselves. As well as comparing the performance of the above with the file caching, the memory footprint of this approach can be contrasted with that of file caching. In other settings, such as network file servers, more involved techniques based on a similar idea can be explored for feasibility.

References

[1] Point biserial co-efficient wiki. http://en.wikipedia.org/wiki/Point-biserial-correlation-coefficient.

[2] Systemtap wiki page link to webinar. http://sourceware.org/systemtap/wiki/RH2010Webinar.

[3] Agrawal, N., Bolosky, W. J., Douceur, J. R., and Lorch, J. R. A five-year study of file-system metadata. *ACM Transactions on Storage 3* (October 2007).

[4] Akoush, S., Sohan, R., Rice, A., and Hopper, A. Activity based sector synchronisation: Efficient transfer of disk-state for wan live migration. In *Proceedings of the 2011 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (2011), MASCOTS '11.

[5] Anderson, E. Capture, conversion, and analysis of an intense nfs workload. In *Proccedings of the 7th conference on File and storage technologies* (Berkeley, CA, USA, 2009), USENIX Association, pp. 139–152.

[6] Bradford, R., Kotsovinos, E., Feldmann, A., and Schiöberg, H. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the 3rd international conference on Virtual Execution Environments* (New York, NY, USA, 2007), VEE '07, ACM, pp. 169–179.

[7] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. Live migration of virtual machines. In *Proceedings*

of the 2nd USENIX symposium on Networked Systems Design and Implementation (Berkeley, CA, USA, 2005), NSDI'05, USENIX Association, pp. 273–286.

[8] Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., and Warfield, A. Remus: high availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2008), NSDI'08, USENIX Association, pp. 161–174.

[9] Das, T., Padala, P., Padmanabhan, V. N., Ramjee, R., and Shin, K. G. Litegreen: saving energy in networked desktops using virtualization. In *USENIX 2010 Annual Technical conference on Annual Technical conference* (Berkeley, CA, USA, 2010), USENIXATC'10, USENIX Association, pp. 3–3.

[10] Douceur, J. R., and Bolosky, W. J. A large-scale study of file-system contents. In *Proceedings of the 1999 ACM SIGMETRICS International conference on Measurement and modeling of computer systems* (New York, NY, USA, 1999), SIGMETRICS '99, ACM, pp. 59–70.

[11] Evans, K. M., and Kuenning, G. H. A study of irregularities in file-size distributions. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems* (2002), SPECTS 02.

[12] Kozuch, M., and Satyanarayanan, M. Internet suspend/resume. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications* (Washington, DC, USA, 2002), WMCSA '02, IEEE Computer Society, pp. 40–.

[13] Kuenning, G. H., and Popek, G. J. Automated hoarding for mobile computers. In *Proceedings of the sixteenth ACM symposium on Operating systems principles* (New York, NY, USA, 1997), SOSP '97, ACM, pp. 264–275.

[14] Leung, A. W., Pasupathy, S., Goodson, G., and Miller, E. L. Measurement and analysis of large-scale network file system workloads. In *USENIX 2008*

Annual Technical Conference on Annual Technical Conference (Berkeley, CA, USA, 2008), USENIX Association, pp. 213–226.

[15] Mashtizadeh, A., Celebi, E., Garfinkel, T., and Cai, M. The design and evolution of live storage migration in vmware esx. In *USENIX 2011 Annual Technical conference on Annual Technical conference* (Berkeley, CA, USA, 2011), USENIXATC'11, USENIX Association.

[16] Meyer, D. T., and Bolosky, W. J. A study of practical deduplication. In *Proceedings of the 9th USENIX conference on File and stroage technologies* (Berkeley, CA, USA, 2011), FAST'11, USENIX Association, pp. 1–1.

[17] Muthitacharoen, A., Chen, B., and Mazières, D. A low-bandwidth network file system. In *Proceedings of the eighteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2001), SOSP '01, ACM, pp. 174–187.

[18] Roselli, D., Lorch, J. R., and Anderson, T. E. A comparison of file system workloads. In *USENIX 2000 Annual Technical conference on Annual Technical Conference* (Berkeley, CA, USA, 2000), USENIX Association, pp. 4–4.

[19] Sapuntzakis, C. P., Chandra, R., Pfaff, B., Chow, J., Lam, M. S., and Rosenblum, M. Optimizing the migration of virtual computers. *SIGOPS Operating Systems Review 36* (December 2002), 377–390.

[20] Wildani, A., Miller, E. L., and Ward, L. Efficiently identifying working sets in block i/o streams. In *Proceedings of the 4th Annual International Conference on Systems and Storage* (New York, NY, USA, 2011), SYSTOR '11, ACM, pp. 5:1–5:12.

[21] Wood, T., Ramakrishnan, K. K., Shenoy, P., and van der Merwe, J. Cloudnet: Dynamic pooling of cloud resources by live wan migration of virtual machines. In *Proceedings of the 7th international conference on Virtual execution environments* (New York, NY, USA, 2011), VEE '11, ACM, pp. 121–132.

[22] Zhao, W., Jin, X., Wnag, Z., Wang, X., Luo, Y., and Li, X. Low cost working set size tracking. In *USENIX 2011 Annual Technical conference on Annual Technical Conference* (Berkeley, CA, USA, 2011), USENIX Association.