Experience with an Automated Hoarding System*

Geoffrey H. Kuenning, Peter Reiher, and Gerald J. Popek†

Abstract

Hoarding is a technique in which a portable computer caches files before disconnecting from a network, so that the files will be available during the disconnection period. *Predictive hoarding* extends this approach to automatically predict the set of files that should be hoarded. We report the results of a study of the SEER predictive hoarding system, which examined both live and simulated usage. This is by far the most detailed study of hoarding that has been carried out to date. Our results show not only that predictive hoarding is feasible, but that SEER has excellent performance that significantly improves disk-space usage on mobile computers.

1 Introduction

In the past few years, mobile operation has become a major force in personal computing. Notebook, laptop, and palmtop computers are popping up everywhere: we see them on airplanes, in classrooms, in restaurants and even in taxis. The growth in portable power has been equally phenomenal. Today's average laptop machine rivals the capabilities of last year's desktop, and notebooks and palmtops are hot on its heels.

Yet one feature common on today's desktop computers has lagged far behind in the mobile arena, and that is the availability of inexpensive and ubiquitous high-bandwidth communications. Although there are many research and commercial efforts to rectify this situation by providing wireless communication [1, 2], there are also numerous roadblocks that will either slow deployment for a significant time, or remain fundamentally insurmountable. In contrast to their fixed brethren, wireless systems are expensive, slow, and frequently nonexistent, and the capacity of a wireless system cannot be increased by simply laying a new cable.

1.1 Disconnected Operation

Because networking is often unavailable or prohibitively expensive, portable computers are frequently forced to operate *disconnected*, without communications, or *weakly connected*, using a communication path that is only capable of low data rates. In this mode, the network is effectively unavailable for the sort of high-bandwidth activity that has become customary for desktop users.

While disconnected operation presents few difficulties for users who can keep all of their data on the portable machine, it presents a serious challenge for many others who have become accustomed to data sharing and instant access to large collections of networked information. The modern salesperson wishes to enter orders directly on a laptop while sitting in the customer's office. The executive flying to a board meeting wants to have the latest profit figures to include in a report. Both of these workers have grown used to having on-line information, and begin to feel frustrated and handicapped when it becomes unavailable simply because they have stepped away from their desks.

*This work was partially supported by the Defense Advanced Research Projects Agency under contract DABT63-94-C-0080.

†The authors are affiliated with the Computer Science Department, University of California, Los Angeles. Gerald Popek is also affiliated with Platinum *technology*. E-mail: geoff@fmg.cs.ucla.edu, reiher@fmg.cs.ucla.edu, popek@platinum.com

1.2 Hoarding

One popular solution to disconnected operation is *hoarding*, in which selected files are copied or *replicated* onto the portable machine's disk before it leaves the network. If the files are chosen properly, the user will never be aware that the network is no longer there, because everything he needs will be available locally.

Hoarding requires certain special support: if the user makes updates to these files, these updates must be propagated back to any other existing copies. Furthermore, if the files are shared with other users who have also made updates, the resulting *conflicts* must be resolved in some fashion, preferably an automated one.

There are a number of systems that have successfully automated all or part of the hoarding process. Some of the better-known are FICUS [3, 4], CODA [5, 6], and LITTLE WORK [7], all of which allow files to be replicated while connected and updated while disconnected.

However, replication only addresses half of the problem of disconnected operation. It is not helpful to hoard files on a mobile computer if they are the wrong files. In fact, researchers have found that mobile workers are extremely sensitive to the presence or absence of information; the lack of even a single file can force the user to switch his efforts to a lower-priority task or even to stop using the computer altogether [8]. (To get a feeling for the importance of individual files, choose a file at random from the list of those you edited today, and ask yourself how your activities would have changed had that file been unavailable.)

Because of the importance of correct file selection, early hoarding systems did not completely automate the task. In LITTLE WORK, the hoarded files are those used most recently, a choice that is both easily implemented and surprisingly effective, since many people tend to work on the same tasks for several days at a time. This is known as the LRU algorithm, a term borrowed from memory caching where it refers to the replacement algorithm: the information discarded is that which has been Least Recently Used. Although LRU often performs well, it is very sensitive to *attention shifts* [9] in which the user suddenly begins work on a new task.

CODA extends the basic LRU concept by allowing the user to specify lists of files that are considered important; these files are given an extra priority boost by applying an offset to their reference time. Although this addresses the problem of attention shifts, it does so by placing a significant extra burden on the user. As discussed in [10, p. 193] and in [11], it is exceedingly difficult for even an expert user to correctly identify the complete set of files needed for daily operation.

To address this challenge, two recent systems, SEER [11, 12] and SPY UTILITY [13] have attempted to completely automate the task of file selection. Each groups files into *projects* representing coherent user tasks and attempts to hoard only complete projects. This frees the user from the burden of precise specification, yet also deals with the problem of attention shifts by keeping a lengthy history of past behavior that can be consulted to determine project membership if the user changes tasks.

1.3 Experimental Investigation

In this paper, we introduce metrics that can be used to characterize the success of a predictive hoarding system, and present and discuss the results of usage tests based on the actual deployment of SEER. Due to space limitations, we only present an overview of the system design; the interested reader can consult [12] or [14] for more information.

Earlier investigations of predictive hoarding have presented only anecdotal data [8, 13] regarding the performance of the hoarding approach. One of the major contributions of this paper is its extensive and detailed study of the behavior of a hoarding system, using both trace-driven simulation and live deployment in a small user community.

2 Overview of Seer

Automated hoarding systems operate by observing user behavior, noting which files are being accessed, and using this information to help develop a list of what are believed to be important files. In most systems, an integrated replication mechanism then transports the chosen files onto the portable computer prior to disconnection.

SEER generally follows this paradigm, although the replication mechanism is not integrated. An *observer* process collects user accesses and feeds them to a *correlator*, which makes hoarding decisions. These decisions are then passed to a separate *replication substrate* to be executed; SEER can operate atop a number of different substrates. SEER differs from other approaches primarily in the algorithms used to make hoarding decisions.

SEER's hoarding mechanism is based on attempting to recognize *projects*, which are groups of files required to perform particular tasks. SEER estimates the membership of a project by forming one or more *clusters* of files based on its observations of user behavior.

The primary measure used for clustering is the *semantic distance* [11, 12] between individual files. Semantic distance is based on the reference sequence of an individual process, and is calculated from a set of *local* semantic distances.

Usually, the local semantic distance between references to two files X and Y is defined as zero if the reference to X was an open and X has not been closed prior to the reference to Y. Otherwise, the distance is the number of references to other files, including the reference to Y, that separate the references to X and Y. For example, in the sequence $\{A^o, B^o, A^c, B^c, C^o\}$, where the superscripts o and c represent opens and closes, respectively, the distance from A to B is 0, B to C is 1 (closes are not counted as references), and A to C is 2.

Semantic distance is defined between all pairs of files; since SEER tracks tens of thousands of files, storing the complete distance matrix would be impractical. We use two heuristics to reduce the complexity of the problem: first, we discard very large distances, and second, we track only the few files that are semantically nearest to a particular file.

In normal operation, most files are used multiple times, which generates a sequence of local semantic distances between a particular pair of files. This sequence is reduced to a single semantic distance value by incrementing each local distance (to avoid zero values) and calculating the geometric mean of the result. A clustering algorithm then generates clusters of files based on both the semantic distance information and optional hints provided by *external investigators*, which are programs that can examine individual files for hints regarding relationships (*e.g.*, C `#include` statements). The clusters are assumed to represent coherent projects that are meaningful to the user.

Clusters are chosen for hoarding based on how recently the cluster has been used; in essence one could consider SEER to be an LRU system that operates at the level of entire clusters, rather than individual files. Only complete clusters are hoarded: if there is not enough room for a large cluster, a smaller, less recently used one will be chosen instead. The rationale behind this decision is based on the fact that users normally cannot complete any work unless the entire project is available in the hoard, so there is little point in storing only part of any given project. (Since a "project" is defined by SEER as the files needed to accomplish work, this is almost a tautology. For example, a programmer performing early software development might not need to attempt compilation, so in this case SEER would not consider the compiler to be part of the project. If the machine were disconnected the first time compilation were attempted, a hoard miss would occur, but the compiler would then become part of the project in future hoarding decisions.)

A number of heuristics are also used to deal with oddities of common program behavior that would otherwise tend to make the semantic-distance calculation less effective. For example, reference histories are tracked across process invocation and termination, so that file relationships implied by process families (*e.g.*, complex compilations) can be identified. These heuristics are discussed further in [12] and [14].

Naturally, the process of observing the user's behavior and tracking file reference is not without cost. A full analysis of SEER's performance impact is given in [12]. In brief, observing the user's behavior on a 133-MHz PENTIUM adds about 35 μs to the execution path for selected system calls (the high-utilization `read` and `write` operations are not affected). The remainder of the impact is primarily due to the `correlator`, in particular it keeps a large (about 10-20 MB) database in virtual memory. (This database is not currently designed to optimize space, a modification that would not be difficult. In addition, it could easily be maintained on disk rather than in virtual memory, an option that is discussed in detail in [12].)

3 Characterizing Success

Hoarding is a variation of the caching methods commonly used in many areas of computer science. However, hoarding differs from caching in one important respect: the cost of a miss. In a traditional cache (e.g., the on-chip memory cache of a modern microprocessor), an item not found in the cache can be easily retrieved from a slower device. Thus, the cost of a cache miss is a simple performance penalty. The effectiveness of a particular caching scheme can be measured by calculating the fraction of accesses that hit or miss in the cache, applying these fractions to the performance of the two types of access, and calculating an average access time. Alternatively, cache performance is often expressed as a simple *miss ratio* that measures the percentage of accesses for which the cache is ineffective.

In hoarding, however, the cost of a miss cannot be effectively measured as a performance penalty. In the best case, a hoard miss can be serviced within a few minutes; even this relatively quick service is thousands of times slower than the time needed to access a file on the local disk. More commonly, the miss cannot be serviced for so long that the access time becomes effectively infinite. Under these conditions, the only acceptable miss ratio is zero, and comparison of nonzero miss ratios becomes meaningless for characterizing user-perceived system quality.

Because there is a need to quantify the performance of automated hoarding systems in a fashion that permits useful comparisons to be made, we have chosen two alternative measures. One is the *time to first miss*, first suggested in [8]. This simply counts the elapsed time between the beginning of a disconnection period and the occurrence of the first hoard miss in that period; it characterizes the amount of work that the user accomplishes before disconnected operation rears its ugly head and makes the lack of the network noticeable. We have adopted a slight variant, in which each hoard miss is categorized according to a user-provided severity, so that we could capture the impact of the miss on the progress of work. The severity levels are as follows:[1]

0. The computer is completely unusable as a result of the miss. No further work can be done.

1. The current task cannot proceed. Work can continue on a less-desirable task.

2. The current task will proceed, but the activity on that task will change in some way as a result of the hoard miss.

3. The hoard miss will cause little or no trouble.

Auto. The user has not reported a hoard miss, but the system has detected a missing file. Generally, these misses are so minor that the user did not even notice them. For this reason, they are not included in the total miss count.

Time to first miss is a "live" measure: it can only be meaningfully measured in actual usage, or with traces taken in a miss-free environment (which often produces behavior that is somewhat different from disconnected activity, *e.g.*, Web browsing). This makes it useful for characterizing the success of a particular system, but less so for comparing systems. Time to first miss is also very sensitive to the actual parameters under which the user operates, in particular the hoard size. If the hoard is even slightly smaller than the size of the user's required working set (defined as the set of distinct, pre-existing files used during any given disconnection), the time to first miss will be near zero under any hoarding system; conversely, if the hoard size is far larger than the working set, most hoarding systems will produce a time to first miss that approximates the total disconnection time.

Because of these problems, we developed a second measure, *miss-free hoard size*, to allow us to compare

[1] A severity level of 4 was also available to allow "hinting" that a file might be needed in the future.

different hoarding systems in a controlled scientific environment. The miss-free hoard size is defined as the minimum amount of disk space that a particular hoarding system would require to allow a complete disconnection period to take place without any misses.

In an LRU system, for example, the only hoarding criterion is recency of reference. To hoard the entire working set for a given disconnection period, for example Tuesday, the system must on Monday evening capture all files referenced more recently than some threshold. In normal operation, the threshold is chosen based on available hoard space: conceptually, it is gradually extended into the past, totaling the size of files referenced since the cutoff point, until a value is found that will cause the hoard to be completely filled. To calculate miss-free hoard size, we invert this procedure by selecting the smallest threshold that will hoard all files in the working set; the required hoard size is then the minimum hoard space that would be required to produce this threshold.

For an ideal hoarding system, the miss-free hoard size is equal to the user's complete working set for the disconnection, but for real systems, it is larger than the working set due to the unnecessary hoarding of files that are not actually used. Returning to the LRU example, if a single file in Tuesday's working set was last referenced on the previous Thursday, all files referenced since then must also be hoarded on Monday evening, even if they are not members of Tuesday's working set. Thus, the miss-free hoard size is equal to the total size of all files referenced more recently than the oldest file in the working set.

In addition to these primary metrics, we also measured the number of *failed disconnections*, defined as the number of disconnections in which there was at least one hoard miss. This statistic reports only the fact that a particular disconnection interval contained a hoard miss at some point, and ignores multiple misses. We measured failed disconnections at the same severity levels used for time to first miss.

4 Simulation Results[2]

To compare SEER with alternative approaches, we ran simulations driven by traces from 9 different users. The traces were collected by instrumenting each user's portable computer, which served as his primary machine, during both connected and disconnected operation. All users were members of the same research group; the primary activity of each was software development and paper writing. Most of the traces represented from two to six months of activity; the longest trace covered about ten months.

Each trace was divided into simulated disconnection periods of one day and one week, with simulated reconnections of infinitesimal duration separating each disconnection. For each simulated disconnection period, the miss-free hoard size was calculated for both SEER's cluster-based algorithm and for a strict LRU-based algorithm. (We also conducted a preliminary investigation of a CODA-style algorithm using static hoard profiles and found that in the absence of ongoing intervention in the form of loading carefully tuned daily profiles, it performed a few percent worse than LRU.[3] We were unable to experiment with the method of SPY UTILITY due to lack of an implementation.)

We ran a total of 120 different simulations under various conditions, but report here on only 90 because one of the variables was found to be relatively unimportant. When possible, the simulations used the actual sizes of the files that the user had accessed; if a file was no longer available, we drew the file size from a random distribution based on observed file size patterns. For each of the 9 machines, we ran 5 simulations of daily disconnections and 5 of weekly periods; each simulation was executed with a different random seed. To avoid errors introduced by uncontrolled trends, the experiments were run in random order.

[2] The results given here differ somewhat from those shown in [12], due to improvements in the simulation software. However, the general character of the results is the same in both sets of simulation runs.

[3] We believe that the CODA method could perform much better with aggressive human assistance, but were unable to devote the approximately 1.5 man-months we estimate would be necessary to do this for our extensive simulations.

Machine	Periods	Size (MB) Mean	C.I.	
Aldrin	53	8	8	8
Chengho	37	9	9	9
Crockett	61	7	7	7
Erasmus	108	16	16	16
Muir	13	12	12	12
Norgay	228	15	15	16
Sacajawea	112	23	23	23
Spaulding	58	10	10	10
Yeager	138	17	16	17

Table 1: Daily working-set statistics, giving number of days simulated, mean working set sizes, and 99% confidence intervals.

Machine	Periods	Size (MB) Mean	C.I.	
Aldrin	10	17	17	17
Chengho	4	15	15	16
Crockett	12	14	14	14
Erasmus	19	43	43	44
Muir	3	8	7	8
Norgay	41	31	31	32
Sacajawea	17	52	52	53
Spaulding	10	20	20	21
Yeager	22	39	38	39

Table 2: Weekly working-set statistics, giving number of weeks simulated, mean working set sizes, and 99% confidence intervals.

4.1 Working Sets

Since no hoarding system can operate successfully in less space than the user's working set, it is important to understand this lower bound before examining other data. Table 1 shows the number of disconnection periods, and means and 99% confidence intervals for daily working sets for all simulation runs. Table 2 shows the same information for weekly working sets. (The figures given in these tables and in subsequent figures reflect only the individual user's working set, including applications; files needed by the underlying system software are not included.) Note that although weekly working sets are larger than the daily figures, they are usually less than twice as large because of the reuse of files from day to day.[4]

The reader may be tempted to conclude from the relatively small working-set size that the details of the hoarding algorithm do not really matter, given the capacity of present-day disks. We believe this conclusion incorrect, for two reasons: first, as we will show in Section 4.2, some algorithms require space approaching the total size of all files ever referenced, and second, user data has a history of approaching disk capacity (*i.e.*, most disks are nearly full most of the time), and there is no reason to believe that this trend will abate. The relatively small working sets in this study are an artifact of particular usage patterns, and should not be considered representative of the working set that would be needed by a person who made heavy use of multimedia, for example. These working sets should be considered as a standard of comparison for evaluating the various hoarding methods.

4.2 Required Hoard Sizes

To facilitate explanation of our results, we first present figures giving the results of a single simulation run for one machine, Norgay. We then present a second run at a large scale so that the data may be examined closely, followed by "small multiples" [15] of the corresponding information for six more machines. (The remaining two machines were used relatively briefly, and do not present interesting new data.)

Figure 1 shows the hoard size needed by the LRU management scheme on machine "Norgay" for simulated weekly disconnections in simulation run number 1, with the evolution of working set sizes over time included for comparison. The cumulative working set in this figure represents the total size of all files referenced since the beginning of the trace.[5] Note that the

[4] The figures for machine "Muir" violate this constraint because the warmup period, in combination with the short traces, caused the daily and weekly figures to be calculated from nearly disjoint sets.

[5] The cumulative working set occasionally drops in size be-

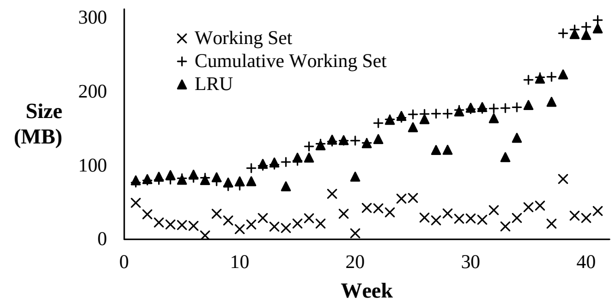

Figure 1: Weekly and cumulative working sets vs. LRU hoard sizes for machine "Norgay".

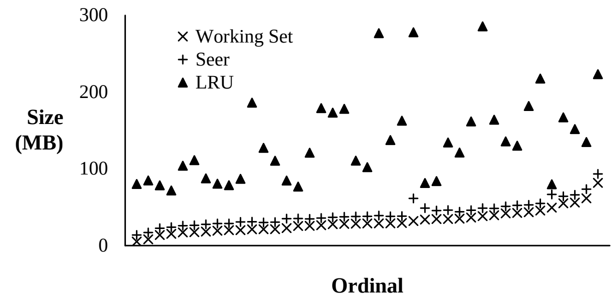

Figure 3: Weekly LRU vs. SEER working sets and hoard sizes for machine "Norgay," sorted by working-set size.

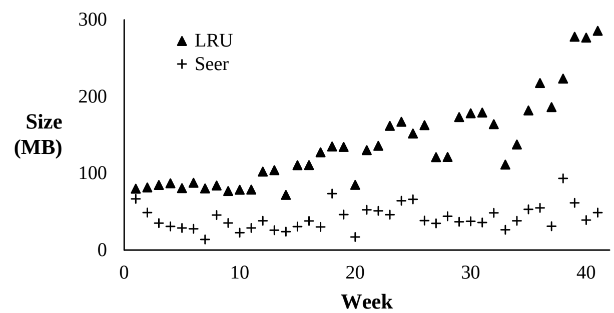

Figure 2: Weekly LRU vs. SEER hoard sizes for machine "Norgay," sorted by week number.

LRU hoard size frequently approaches the cumulative working set. Figure 2 shows the sizes needed by the clustering and LRU managers for this run; here we can easily see that SEER's clustering algorithm outperforms LRU. Finally, Figure 3 shows the required hoard sizes for both the clustering and LRU algorithms for the same simulation run, together with the weekly working-set size in megabytes. To facilitate comparison, the values in this graph have been sorted by working-set size rather than week number. Note that the clustering algorithm's hoard sizes are usually so close to the working-set size as to be indistinguishable. These results will be discussed further in Section 6.

cause it excludes files that have been deleted.

Figures 4, 5, and 6 show the same data for daily disconnections on the same machine.

The patterns in Figures 1 through 6 were repeated in all of our simulations: LRU hoard sizes are significantly larger than those needed by SEER's clustering algorithm, and the latter closely tracks the size of the working set. This holds true for both daily and weekly disconnection periods.

Figures 7, 8, and 9 show the same information as Figures 1 through 6 for six more machines, on a smaller scale.

Table 3 summarizes this information by showing means and 99% confidence intervals for required daily hoard sizes for all machines. The same table also shows the mean, confidence interval, and range for the difference between the two hoarding methods. The final column gives the percentage of days on which the LRU manager outperformed the clustering manager (difference less than or equal to zero). Table 4 presents the same information for weekly disconnections.

5 Live Results

In addition to running simulations, we instrumented nine machines and their users in actual usage of SEER. These machines were used for periods varying from 10 to 184 disconnections. As mentioned in Section 3, we measured hoard-miss statistics at severity levels

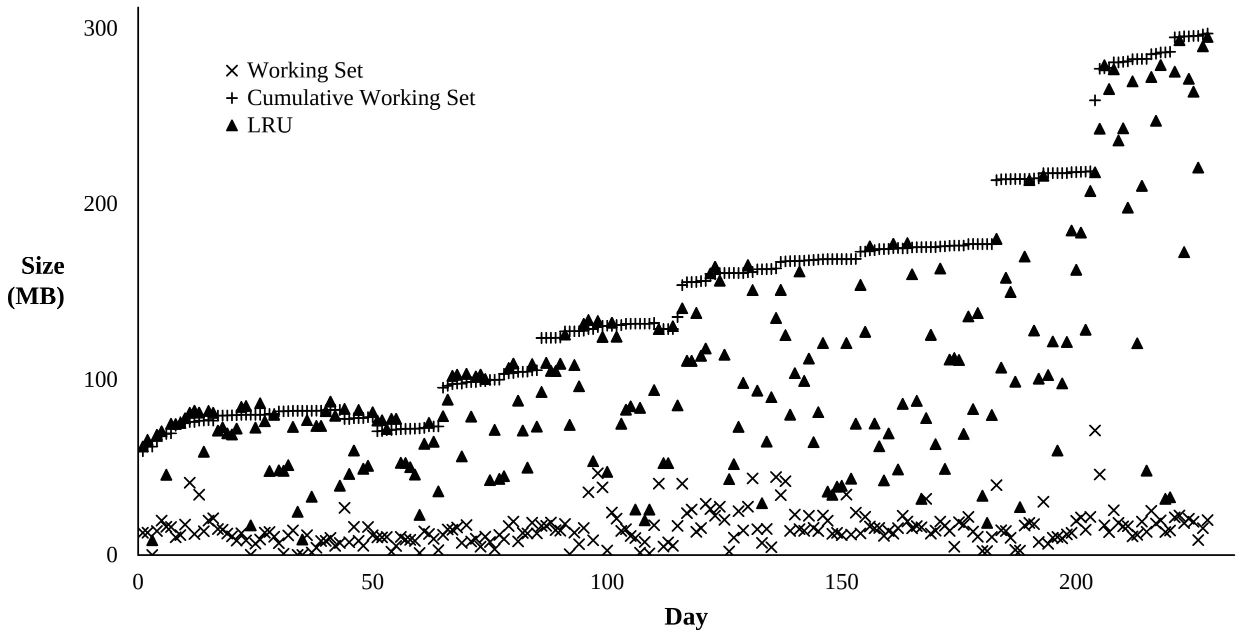

Figure 4: Daily and cumulative working sets vs. LRU hoard sizes for machine "Norgay".

	LRU			Clustering			Difference					% LRU
Machine	Mean	C.I.		Mean	C.I.		Mean	C.I.		Min	Max	Wins
Aldrin	51	50	51	12	12	12	39	38	39	-2	125	3
Chengho	36	35	36	12	12	13	23	23	23	-1	59	8
Crockett	37	36	37	8	8	9	28	28	29	-3	75	10
Erasmus	131	130	132	21	20	21	110	109	111	3	247	0
Muir	82	81	82	14	14	14	68	67	68	33	157	0
Norgay	104	102	106	26	25	26	79	77	80	-2	270	1
Sacajawea	107	105	108	34	33	34	73	71	74	-32	168	1
Spaulding	28	28	28	11	11	11	17	17	17	0	47	0
Yeager	180	178	181	20	20	21	160	159	161	6	431	0

Table 3: Daily hoard-size statistics, giving means and 99% confidence intervals for two hoarding methods, the difference between the methods, and the fraction of simulated days in which LRU outperformed SEER.

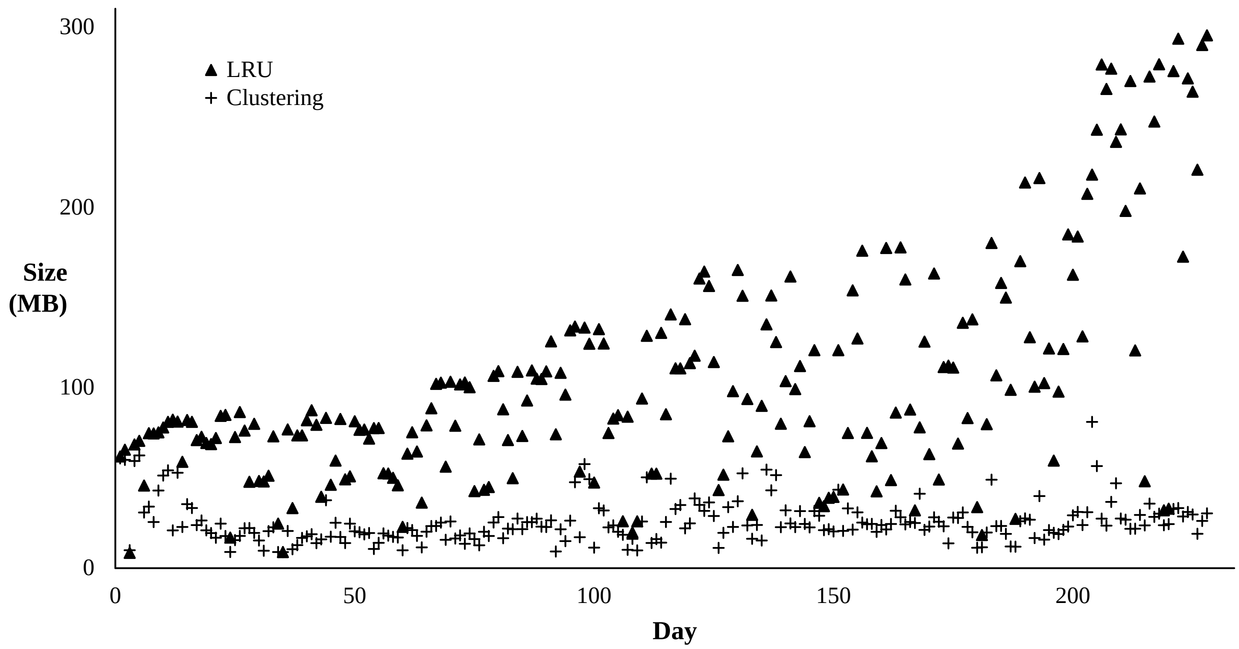

Figure 5: Daily LRU vs. SEER hoard sizes for machine "Norgay," sorted by day number.

	LRU			Clustering			Difference					% LRU
Machine	Mean	C.I.		Mean	C.I.		Mean	C.I.		Min	Max	Wins
Aldrin	86	85	87	22	21	22	64	63	65	16	118	0
Chengho	34	33	34	19	18	19	15	15	15	0	32	0
Crockett	58	57	59	16	16	16	42	41	43	7	71	0
Erasmus	163	161	165	50	50	51	113	111	115	24	197	0
Muir	95	94	96	10	10	10	85	84	86	48	130	0
Norgay	137	135	139	42	41	42	95	93	96	11	240	0
Sacajawea	123	122	124	63	62	63	61	60	61	12	156	0
Spaulding	46	45	46	21	21	22	25	24	25	15	43	0
Yeager	262	261	263	42	42	43	219	218	220	8	413	0

Table 4: Weekly hoard-size statistics, giving means and 99% confidence intervals for two hoarding methods, the difference between the methods, and the fraction of simulated weeks in which LRU outperformed SEER.

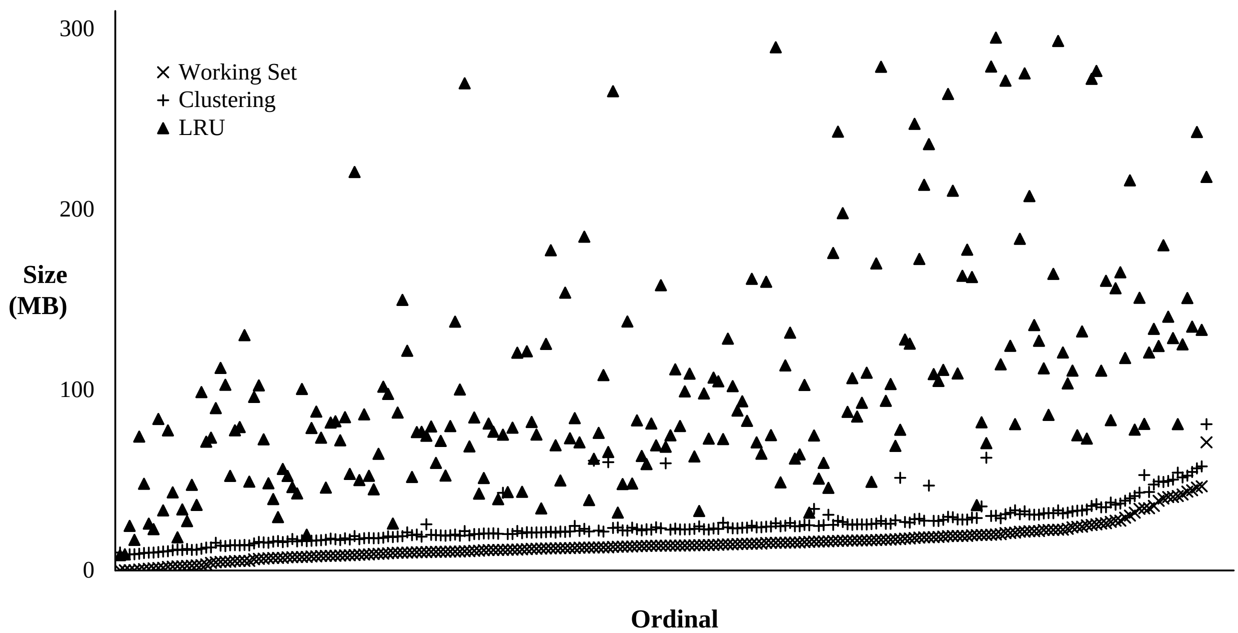

Figure 6: Daily LRU vs. SEER working sets and hoard sizes for machine "Norgay," sorted by working-set size.

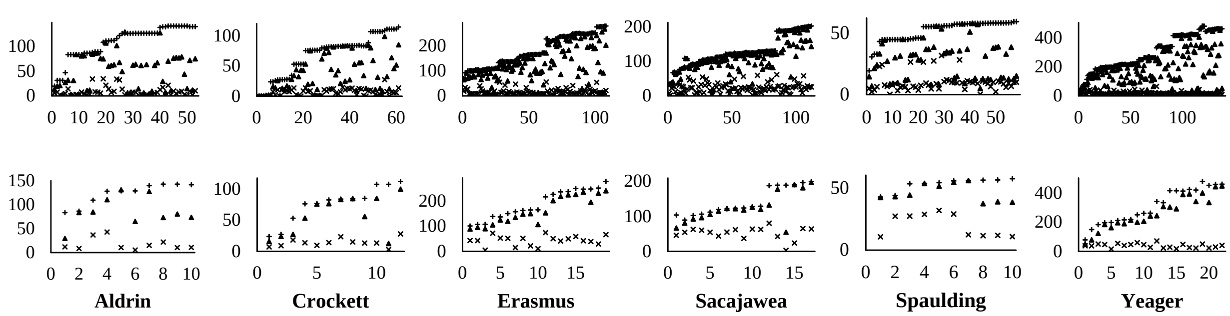

Figure 7: Daily (top) and weekly periodic and cumulative working sets vs. LRU hoard sizes.

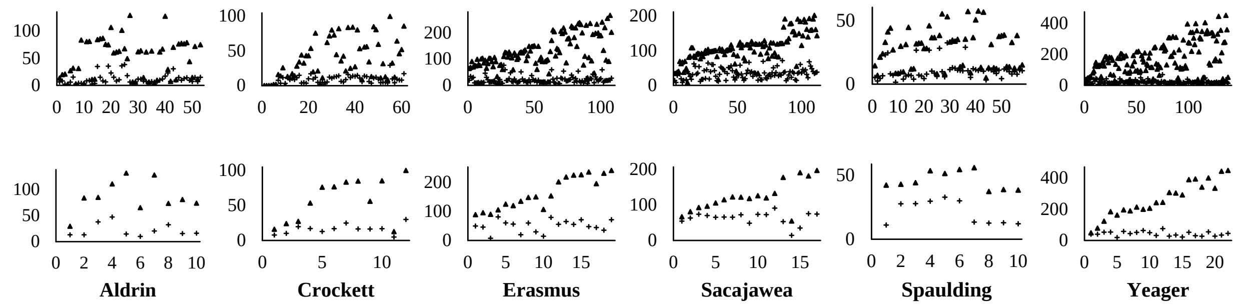

Figure 8: Daily (top) and weekly LRU vs. SEER hoard sizes, sorted by day number.

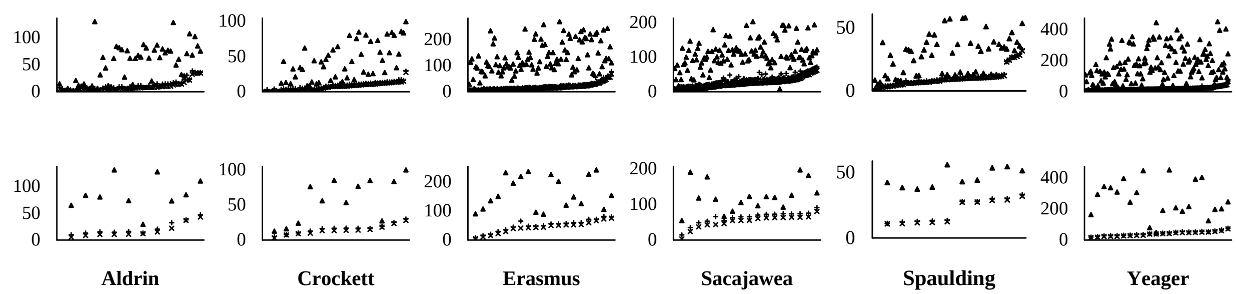

Figure 9: Daily (top) and weekly LRU vs. SEER working sets and hoard sizes, sorted by working-set size. The X axis is the ordinal, as in Figures 3 and 6.

ranging from 0 (unusable machine) to 3 (minor nuisance). All users were initially assigned a hoard size of 50 MB. One user quickly discovered that this was too small and increased the setting, and a second should have also selected a larger value; this issue is discussed further in Section 6.

5.1 Failed Disconnections

Table 5 summarizes failed disconnections at the various severity levels. As well as the number of failures, the table gives total number of disconnections, the hoard size in megabytes, and the number and percentage of failed disconnections at each level. (The total number of failed disconnections sometimes differs from the sum of the various levels because some failed disconnections suffered failures at more than one severity. The "Total" column counts each such multiple failure only once.) Note that, with one exception, the failure rates were extremely low.

5.2 Time to First Miss

Table 6 gives statistics on the time to first miss, in hours and in percentage of the total disconnection period, for failed disconnections.[6] (The number of misses at each severity level is listed in Table 5.) Table 7 gives statistics on the time to first miss, in hours and in percentage of total disconnection period, calculated for all disconnections. In this table, we

[6] Recall that automatically-detected misses are those that did not come to the attention of the user. It is instructive to consider this table and Table 7 in the absence of the "Auto" rows.

Machine	Disconn's	Hoard Size	Failures 0	1	2	3	Total	Auto	Percentage 0	1	2	3	Total	Auto
Aldrin	38	50	0	0	0	0	0	2	0	0	0	0	0	5.3
Chengho	10	50	0	0	0	0	0	0	0	0	0	0	0	0
Crockett	75	50	0	0	0	0	0	1	0	0	0	0	0	1.3
Erasmus	90	50	0	0	0	0	0	5	0	0	0	0	0	5.6
Muir	25	50	0	0	0	0	0	1	0	0	0	0	0	4
Norgay	184	50	0	3	6	11	15	2	0	1.6	3.3	6.0	8.2	1.1
Sacajawea	107	98	0	0	0	0	0	3	0	0	0	0	0	2.8
Spaulding	75	50	0	0	0	0	0	0	0	0	0	0	0	0
Yeager	116	50	0	1	0	0	1	5	0	0.9	0	0	0.9	4.3

Table 5: Summary of failed disconnections, giving number of disconnections observed, hoard size used, and counts and percentages at various severities.

Machine	Sev.	Hours Mean	S. D.	Min	Max	Percentage Mean	S. D.	Min	Max
Aldrin	Auto	1.8	2.3	0.21	3.4	43.5	59.7	1.3	85.7
Crockett	Auto	1.6	—	1.6	1.6	48.8	—	48.8	48.8
Erasmus	Auto	0.9	0.5	≈ 0	1.3	42.7	43.5	8.5	98.6
Muir	Auto	11.0	—	11.0	11.0	90.8	—	90.8	90.8
Norgay	1	10.6	16.3	≈ 0	29.4	49.9	49.6	0.2	99.3
	2	6.6	9.1	≈ 0	21.5	50.1	30.8	0.2	94.8
	3	3.4	4.9	0.1	12.9	49.9	35.5	0.5	95.5
	Auto	20.4	28.4	0.3	40.5	54.0	57.1	13.6	94.4
Sacajawea	Auto	0.5	0.3	0.2	0.8	37.4	24.1	21.4	65.1
Yeager	1	1.0	—	1.0	1.0	20.8	—	20.8	20.8
	Auto	0.9	0.6	0.1	1.8	32.2	24.0	5.7	60.2

Table 6: Times to first miss for failed disconnections, giving severity and mean, standard deviation, and range of time to first miss in hours and as percentages of total disconnection time.

considered a disconnection interval with no misses to have a time to first miss equal to the disconnection interval.[7] To save space, we only report those table entries for which there was at least one miss observed, except for machines that suffered no misses at all. The values in these tables show that even when there was a failure, users were generally able to work for a significant time before experiencing any effect due to being disconnected.

[7]The "minimum" and "maximum" columns primarily represent the minimum and maximum disconnection times, since failures were so rare.

6 Discussion

Figures 1, 4, and 7 demonstrate the surprising ineffectiveness of LRU-style hoarding algorithms, which frequently need a hoard size equal to the cumulative working set. This is due to two phenomena: first, attention shifts require access to large numbers of "old"

Machine	Sev.	Hours Mean	Hours S. D.	Hours Min	Hours Max	Percentage Mean	Percentage S. D.	Percentage Min	Percentage Max
Aldrin	Auto	10.7	15.9	0.2	71.9	97.0	16.1	1.3	100.0
Chengho	—	43.2	127.2	0.4	404.9	100.0	—	100.0	100.0
Crockett	Auto	9.9	40.9	0.3	348.2	99.3	5.9	48.8	100.0
Erasmus	Auto	2.8	4.2	0.0	26.5	96.8	16.1	8.5	100.0
Muir	Auto	1.8	2.4	0.3	11.0	99.6	1.8	90.8	100.0
Norgay	1	9.3	16.3	0.0	90.6	99.2	8.2	0.2	100.0
	2	9.1	16.2	0.0	90.6	98.4	10.2	0.2	100.0
	3	9.0	16.3	0.1	90.6	97.0	14.5	0.5	100.0
	4	8.8	16.0	0.1	90.6	97.3	14.6	1.7	100.0
	Auto	9.3	16.3	0.3	90.6	99.5	6.4	13.6	100.0
Sacajawea	Auto	8.0	38.3	0.2	390.6	98.2	10.9	21.4	100.0
Spaulding	—	10.2	41.1	0.3	348.2	100.0	—	100.0	100.0
Yeager	1	2.3	4.3	0.3	27.7	99.3	7.4	20.8	100.0
	Auto	2.2	4.2	0.1	27.7	97.1	14.5	5.7	100.0

Table 7: Times to first miss across all disconnections, giving severity and mean, standard deviation, and range of time to first miss in hours and as percentages of total disconnection time.

files, and second, even in the absence of attention shifts, users often refer to one or two files that have been unused for a long time. An LRU algorithm can only be successful if it has a hoard size large enough to reach back to these unused files.

Figures 2, 5, and 8 show that SEER's clustering method significantly outperforms simple LRU. Although the hoard sizes for LRU show a general upward trend with time, the clustering algorithm requires a hoard size that is essentially independent of time. This shows that the clustering algorithm was able to intelligently group these old files with their associated projects so that they would be available when needed.

Perhaps the most dramatic evidence of the success of the clustering algorithm appears in Figures 3, 6, and 9. The striking aspect of these graphs is that the clustering algorithm achieves a required hoard size that is so close to the working set that it is almost indistinguishable: only occasionally do the "+" signs appear sufficiently distant from the "X" marks to be individually identified, and even then they are extremely close to the lower bound marked by the working set. By contrast, the LRU algorithm is all over the map; at times it performs very well, while at others it needs a hoard space that is many times the size of the working set. (Recall that in these graphs, the X axis is an arbitrary index, and does *not* represent the passage of time.)

The daily and weekly hoard-size summaries given in Tables 3 and 4 bear this observation out. SEER's clustering algorithm consistently outperforms LRU; even in those few cases where LRU is better, the margin of "victory" is negligible. In Table 3, LRU outperformed SEER's algorithm rarely and by only a small margin, with the exception of one machine (Sacajawea), where LRU was significantly better in one single disconnection.

Table 5 tells much the same story for live usage. Only one user suffered a significant number of hoard misses; this was probably because his hoard size was set too close to his working set (note in Figure 1 that the working sets for this machine often exceeded the configured hoard size of 50 MB). This user has since increased his hoard size and has not since experienced significant numbers of misses. In view of the under-configured hoard size, we believe that an 8% rate of failed disconnections, most of which took place at minor severity levels, is extremely good performance. All other users suffered negligible failure rates, even when measured by the more conservative automated algorithm.

With respect to the time until the first miss, Table 6 shows that even when there was a failure, most users were able to work for a significant period before the failure occurred. Although the minimum time observed was well under an hour (and in three cases was almost immediate), it is clear that it was far more common to work for at least an hour before trouble arose. (It is interesting to note that two of the three near-zero times occurred on Norgay, the machine with the under-configured hoard, and the third, on Erasmus, was automatically detected and probably did not bother the user.)

Finally, Table 7 recasts the information of Table 6 by including in its statistics all disconnections, even those with no failures. Here, it is clear that a user's average experience was one of getting large amounts of work done, not of suffering from hoard misses. In fact, when the values in this table are compared to statistics on average disconnection times (given in [12]), the numbers are almost indistinguishable, which demonstrates that automated predictive hoarding can make disconnected operation a viable option even for the unsophisticated user.

7 Conclusions

This research demonstrates, via both simulation and actual use, that automated hoarding of files for portable computers is feasible and that it has significant benefits over other methods like LRU hoarding. We have shown that our method of hoarding files by clustering performs better than LRU in the vast majority of cases, and that the amount of disk space required by our hoards is between two and ten times lower than that required by LRU hoarding. We have also demonstrated that SEER's clustering method usually produces hoards that are almost the same size as the actual minimum working set required to avoid hoard misses over reasonable periods of disconnections.

The small number of hoard misses experienced by the users of our system suggests that not only are the SEER hoard sizes nearly the same as the working set hoard sizes, but that the contents must be nearly the same. If they were not, since all files in the working set are used (by definition), our similarly sized hoards would experience far more misses. Nine users experienced 35 misses in 720 disconnection periods, and 19 of those misses were not noticed at all by the users. Of the remaining 16 hoard misses, 10 seriously inconvenienced the user, and most of those happened to a single user whose working set size was uncomfortably close to his hoard size.

While SEER has only been applied to one particular working environment so far, that environment is realistic and representative of a significant group of users. More experimentation and simulation would be required to expand the results to a wider variety of user types, but these results do show that SEER's benefits are not limited to behavior dependent on the idiosyncrasies of an individual user. The nine users tested worked in very different ways on a variety of tasks.

In general, our results also suggest that predictive methods are feasible for controlling important resources in modern operating systems performing today's tasks. The basic method employed by SEER may have application far beyond automated hoarding of files for portable computers. We are currently investigating the use of prediction in areas such as network routing, disk performance, and multiprocessor scheduling.

Other researchers have concluded that user intervention is required to significantly improve on the hoarding decisions made by the LRU algorithm. Our research indicates that a fully automated hoarding system can produce hoard sizes almost the same as the ideal working set size algorithm, which relies on information available only after the fact. Given the difficulty that even sophisticated system developers have in predicting their file needs by hand, this demonstration of the sufficiency of automated methods was required to enable use of hoarding systems by more ordinary users, and perhaps the most important result of our work is the fact that our system enables even the novice user to work disconnected in an effective fashion.

8 Acknowledgments

We would like to express our thanks to the members of UCLA's File Mobility Group, whose willingness to try an unproven hoarding system made this study possible. We would also like to acknowledge the contributions of several of the anonymous referees, whose helpful suggestions have significantly improved the readability of this paper.

References

[1] Gregory J. Pottie. System design choices in personal communications. *IEEE Personal Communications Magazine*, 2(5):50–67, October 1995.

[2] Otto Spaniol, Andreas Fasbender, Simon Hoff, Josef Kaltwasser, and Jürgen Kassubek. Impacts of mobility on telecommunication and data communication networks. *IEEE Personal Communications Magazine*, 2(5):20–33, October 1995.

[3] Richard G. Guy, John S. Heidemann, Wai Mak, Thomas W. Page, Jr., Gerald J. Popek, and Dieter Rothmeier. Implementation of the Ficus replicated file system. In *USENIX Conference Proceedings*, pages 63–71. University of California, Los Angeles, USENIX, June 1990.

[4] John S. Heidemann, Thomas W. Page, Jr., Richard G. Guy, and Gerald J. Popek. Primarily disconnected operation: Experiences with Ficus. In *Proceedings of the Second Workshop on Management of Replicated Data*, pages 2–5. University of California, Los Angeles, IEEE, November 1992.

[5] James J. Kistler and Mahadev Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, 1992.

[6] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, April 1990.

[7] L. B. Huston and Peter Honeyman. Disconnected operation for AFS. In *Proceedings of the USENIX Symposium on Mobile and Location-Independent Computing*, pages 1–10. USENIX, 1993.

[8] Mahadev Satyanarayanan, James J. Kistler, Lily B. Mummert, Maria R. Ebling, Puneet Kumar, and Qi Lu. Experience with disconnected operation in a mobile computing environment. In *Proceedings of the USENIX Symposium on Mobile and Location-Independent Computing*, pages 11–28, Cambridge, MA, August 1993. USENIX.

[9] Geoffrey H. Kuenning, Gerald J. Popek, and Peter Reiher. An analysis of trace data for predictive file caching in mobile computing. In *USENIX Conference Proceedings*, pages 291–306. USENIX, June 1994.

[10] James Jay Kistler. *Disconnected Operation in a Distributed File System.* Ph.D. dissertation, Carnegie-Mellon University, May 1993.

[11] Geoffrey H. Kuenning. Design of the SEER predictive caching system. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994.

[12] Geoffrey Houston Kuenning. *Seer: Predictive File Hoarding for Disconnected Mobile Operation.* PhD thesis, University of California, Los Angeles, Los Angeles, CA, May 1997. Also available as UCLA CSD Technical Report UCLA-CSD-970015.

[13] Carl D. Tait, Hui Lei, Swarup Acharya, and Henry Chang. Intelligent file hoarding for mobile computers. In *Proceedings of MobiCom '95: The First International Conference on Mobile Computing and Networking*, pages 119–125, Berkeley, CA, November 1995.

[14] Geoffrey H. Kuenning and Gerald J. Popek. Automated hoarding for mobile computers. In *Proceedings of the 16th Symposium on Operating Systems Principles*, St. Malo, France, October 1997. ACM.

[15] Edward R. Tufte. *Envisioning Information.* Graphic Press, Cheshire, Connecticut, 1990.