# Simplifying Automated Hoarding Methods*

Geoffrey H. Kuenning
Computer Science Dept.
Harvey Mudd College
Claremont, CA 91711
geoff@cs.hmc.edu

Wilkie Ma
Computer Science Dept.
UCLA
Los Angeles, CA 90095
wilkie@cs.ucla.edu

Peter Reiher
Computer Science Dept.
UCLA
Los Angeles, CA 90095
reiher@cs.ucla.edu

Gerald J. Popek†
Computer Science Dept.
UCLA
Los Angeles, CA 90095
popek@cs.ucla.edu

## ABSTRACT

A number of mobile computing systems have used the technique of hoarding, which allows a mobile device to store a chosen subset of known files, to give disconnected users the illusion of a complete filesystem in the presence of limited storage. We undertook an extensive and detailed simulation study of the parameters of a well-known hoarding system, SEER, in an attempt to discover the parameters that would produce the best performance. To our surprise, we discovered that the best parameter combinations were those that completely disabled SEER's complex clustering methods, reverting the system instead to a modified form of LRU hoarding. We discuss the experiments and our results, and propose designs for future systems and directions for future research.

## Categories and Subject Descriptors

D.4.3 [**Operating Systems**]: File Systems Management—*Distributed File Systems*; D.4.8 [**Operating Systems**]: Performance—*Modeling and prediction, Simulation*

## General Terms

Experimentation, Measurement, Performance

## Keywords

Hoarding, disconnected operation, clustering, LRU

*This work was partially supported a research grant from Microsoft Corporation.

†Gerald J. Popek is also affiliated with United Online.



## 1. INTRODUCTION

Hoarding [6, 7, 12] is a technique that provides the user of a mobile or handheld computer with the illusion of a virtual network, even in the presence of disconnections and network failures. Hoarding is a variant of caching, in which a relatively small device (in this case, the mobile computer's permanent memory) is made to look much larger (i.e., a distributed file system) at little cost in performance.

A hoarding system differs from most other caching systems in one important respect: the cost of failures. In an ordinary cache, the cost of a cache miss (an access to an object that is not currently stored in the cache) is relatively low, normally equal to the cost of retrieving the object from a lower-speed storage device. In a hoarding system, however, a hoard miss can be catastrophic if it occurs when the computer is disconnected from the network, because the cost of servicing the miss becomes essentially infinite. Previous studies [6] have demonstrated that a hoard miss can make a computer completely unusable.

The so-called *hoarding problem* is the task of choosing the contents of the hoard on a mobile computer, given limited persistent storage relative to the amount of data the user might access. Solving the hoarding problem requires predicting the user's future behavior, and thus the problem is insoluble in the general case.

In previous work [7], we demonstrated that in some real-world environments, the hoarding problem can be addressed sufficiently well for an automated system to be practical. The current paper extends that work by demonstrating that an efficient solution to the hoarding problem is often achievable.

## 2. RELATED WORK

Hoarding appears to have been first proposed as a component of the CODA project at CMU [6, 10, 11]. Several near-contemporaneous projects adopted a similar solution to the problem of disconnected operation. Each of these systems used different methods of controlling the hoard contents. Alonso *et al.* [1] relied primarily on user suggestions in the form of a control file. LITTLE WORK [3, 4] relied on the standard LRU algorithm from caching systems, while

CODA combined these approaches by augmenting LRU with a user-specified *hoard priority* that could be used to offset the LRU age of an object so that it was more likely to remain in the hoard. Recently, CODA has also introduced the concept of *translucent caching* [2], in which the user was given interactive control over some of the more important details of hoarding.

SPY UTILITY [12] was a more ambitious attempt to automate hoard selection. The operating system observed user behavior and attempted to select hoard contents based on the structure of the process tree and the sequence of files accessed. Like recent versions of CODA, a GUI presented the collected information to the user as suggestions and provided ways to modify the hoard contents by hand. This system has since been made commercially available by IBM.

There are also several commercial systems that attempt to address the problems of hoarding and disconnected operation. The most well-known are LapLink and Microsoft Briefcase for laptop computers, and AvantGo for handhelds. However, these systems rely entirely on the user to control hoard contents, and thus will not be discussed further here.

# 3. TUNING HOARD PERFORMANCE

Our current work was undertaken in an attempt to improve the performance of an automated hoarding system, SEER [7]. Although SEER produced very good results for a system with no user intervention [8], we were interested in further improving that performance and exploring the limits of automated hoarding systems. We also hoped to gain a better understanding of the underlying reasons for SEER's excellent behavior. The results, as will be seen, were surprising.

## 3.1 Characterizing Performance

One of the great difficulties in studying hoarding systems has been in finding ways to measure and compare the performance of hoarding systems. In fact, a meaningful performance metric has only recently been developed [7]. We believe that it is instructive to briefly consider the reasons for these difficulties.

The standard metric for traditional caching systems has been the miss rate: the fraction of all accesses for which the cache did not contain the requested object. This metric is simple to measure and accurately characterizes cache behavior. However, it is not appropriate for hoarding systems, because in a hoarding environment, not all misses are equal. Although some misses are relatively innocuous, the lack of a truly critical file may force the user to cease work until the miss can be serviced [6, 9], which in the case of disconnected operation may be hours or even days.

A number of alternate metrics have been proposed [6, 9], notably the elapsed time until the first hoard miss, but we have shown in earlier work that these measures also have drawbacks that make them inappropriate for characterizing hoarding systems. In particular, the time until the first hoard miss is sensitive to minor variations in user behavior patterns, and does not properly recognize the fact that some hoard misses are only a nuisance, while others are catastrophic.

The metric that we chose for the current work is one of those used in previous SEER work, namely the minimum space needed by a hoarding system to guarantee that the user will have access to all of his files during disconnection. This metric is introduced in [7], where it is referred to as the *miss-free hoard size* (MFHS in Table 1). It is determined during simulation by use the hoarding method to sort all files into priority order, and then trimming the file list after the last file that was referenced during the simulated disconnection period. The total size of all remaining files is the miss-free hoard size. If the hoarding algorithm were provided this much space, the user would be sure that no misses would have occurred during the disconnection. A smaller miss-free hoard size indicates a better hoarding method (because less space is wasted on unused files). The minimum possible miss-free hoard size is exactly equal to the size of the user's working set during the disconnection.

## 3.2 Hoarding Parameters

As described in [7], SEER is tunable, with many parameters that control details of the hoarding algorithm. The interactions between these parameters are complex and nonlinear, so that it is effectively impossible to predict the algorithm's behavior from an examination of the parameter settings. In total, there are 28 modifiable system parameters, of which 14 have an effect on hoarding. Some of the parameters are binary, others discrete integers, and still others were continuous values. However, only 8 of the 14 (all multivalued) turned out to have an effect large enough to merit detailed investigation. In this study, we investigated the behavior of 68 unique parameter combinations.

In our original studies, we chose the parameters partly by hand and partly based on limited experimentation with an earlier, less accurate measure of hoarding performance. The values we chose produced excellent results, but we did not believe that we had achieved optimal settings. The current study was undertaken specifically to explore the parameter space in an attempt to discover a better combination of parameters and to learn the limits of SEER's approach to hoarding.

At the same time, we also explored an alternative hoarding method that had been suggested by Karin Petersen [personal communication]. SEER's primary hoarding algorithm was based on a combination of agglomerative clustering, driven by a measure called *semantic distance*, with a series of *ad hoc* heuristics that served to filter out noise in the observed data. This algorithm was contrasted with simple LRU in our original work. Karin suggested that it would be interesting to investigate how LRU would perform if it were enhanced with the same noise-filtration heuristics used by the clustering algorithm.

## 3.3 Experimental Methodology

As in our previous work, we used trace-driven simulation to study the behavior of the hoarding system. The traces were collected from 7 laptop users in a software-development environment over a period of several months, and were simulated using the same framework and methodology reported in [8]. (Two of the users reported on in [8] were omitted from the current study. This omission is the cause of the discontinuities in user identification in Section 4.)

The result of simulating an individual trace was summarized as a single number, the minimum space requirement (miss-free hoard size). Since the simulated disconnection period was 24 hours, we averaged the results for all disconnections in a given trace to arrive at the numbers in Table 1. Depending primarily on the size of the trace, simulation of one user's machine with one set of parameters took between 2 and 24 hours.

Given the large size of the parameter space and the length of time needed to investigate even a single parameter combination, it was obviously not possible to search the space exhaustively. Nor did it seem reasonable to use automated exploration techniques such as hill-climbing or simulated annealing, since those methods are most appropriate when the time needed by a human to choose a new parameter combination is large compared to the time needed to evaluate the effect of that combination.

Instead, we felt it best to use human intelligence to guide the search and make optimum use of the available CPU time. For our initial investigations, we simulated several parameter combinations that had been chosen at random, to get a feel for which parts of the parameter space were likely to prove fruitful. Based on these initial results, we then explored the most promising areas more thoroughly. In each area, we first attempted to establish which parameters had the greatest effect on the results; we then tried to learn the shape of the output curve and to seek an optimal (minimum) output value.

### *3.3.1 Parameters Investigated*

As discussed in [9], SEER has a large number of parameters, not all of which have an effect on hoarding performance. In the current study, we experimented with all parameters that had previously been shown to have an effect on hoarding, with one exception. The parameters we varied were:

**oftenPercent** Controls which files are omitted from the semantic-distance calculation due to high usage. Such files are always included in the hoard. Lower values cause more files to be considered high-usage.

**peekThreshold** Controls when a process is thought to be scanning the files in a directory, so that its accesses are not semantically meaningful. Lower values increase the probability of interpreting process behavior as directory scanning.

**ageWeightFactor** Controls the probability of a file's relationship to another being replaced by a newer one in the relation list (which is of size "keepReferences," below). Larger values increase the probability of the older relationship being kept for use in clustering. Interacts with agingThreshold and the last four parameters listed here.

**agingThreshold** Controls the probability of a file's relationship to another being replaced by a newer relationship. Larger values increase the probability of the older relationship being kept for use in clustering. Interacts with ageWeightFactor and the last four parameters listed here.

**keepReferences** Controls the number of inter-file relationships kept for each file, which in turn has a complex effect on the clustering behavior [5].

**minimumReferences** Controls both overlapping and non-overlapping clustering. If fewer than this number of inter-file references are observed, they are not considered semantically meaningful (because of the probability of noise). Smaller values cause fewer and larger clusters.

**sharedLooseThreshold** Controls overlapping clustering. If this number of references are shared between two files, their clusters are overlapped. Smaller values cause more overlaps to be found in clusters. Values greater than keepReferences will disable overlapping clusters.

**sharedTightThreshold** Controls overlapping clustering. If this number of references are shared between two files, their clusters are merged. Smaller values cause fewer and larger clusters by making it more likely that clusters will merge with each other. Values greater than keepReferences will disable cluster merging.

The first two of the above parameters control the heuristics that were used to augment clustering; the remaining six control the clustering algorithm itself.

The only parameter that we did not investigate was directoryDistanceWeight, which affects how the clustering algorithm uses directory-name relationships to adjust the clusters. This parameter was fixed at its default value of 1.0 for the early experiments and was never modified because of the results reported in Section 4.2.

### *3.3.2 Parameter Exploration*

As mentioned above, we chose to use human intelligence to guide our explorations of the parameter space. However, we still needed to select a starting point for our exploration. One obvious choice was the parameter combination used in earlier studies, but there was a clear risk that this setting represented a local maximum. To avoid that possibility, we also chose several starting points by first hand-enumerating reasonable ranges for the parameters, and then generating random combinations to guide the initial search. Some combinations are nonsensical due to interactions among parameters. Such combinations were discarded before the experiments began.

Once a random combination had been selected and simulated, we compared the results with those from other parameters. If the combination seemed to be successful, we then explored the space in its vicinity of the combination by varying one or more parameters. Generally, each dimension (parameter) was varied one step in each direction from the original setting. The step size was chosen based on the particular parameter and its sensitivity as estimated from previous tests. Due to the expense of experimentation, it was not possible to examine all points in the vicinity of the original combination; instead, only one or two parameters at a time were varied.

A search experiment was initiated by listing (by hand) the various parameter combinations to be tested. Simulations were then executed and analyzed under automatic control. Based on the results, the human experimenters would choose a new set of parameters to be investigated, and the process was repeated.

## 4. EXPERIMENTAL RESULTS

Table 1 presents a selection of our results for a number of different parameter combinations. The next-to-last column gives the miss-free hoard size observed in our simulations; the results are sorted by that value so that the best outcome is at the top of each section. Entries with "Modified LRU" for the last six parameters were run using a parameter combination that reduced SEER's clustering algorithms to simple

| User | Often Pct | Peek Thresh | AWF | Aging Thresh | Keep Refs | Min Refs | Loose Thresh | Tight Thresh | MFHS (KB) | Ref. No. |
|---|---|---|---|---|---|---|---|---|---|---|
| A | 3 | 11 | —— Modified LRU —— | | | | | | 24580.113 | *1* |
| | 2 | 11 | —— Modified LRU —— | | | | | | 24581.849 | *2* |
| | 3 | 11 | 1.0 | 10000 | 26 | 9 | 24 | 25 | 24583.302 | *3* |
| | 3 | 11 | 1.0 | 10000 | 30 | 9 | 28 | 29 | 24583.302 | *4* |
| | 4 | 11 | 1.0 | 94201 | 25 | 9 | 23 | 24 | 24584.642 | *5* |
| | 3 | 11 | 0.5 | 94201 | 25 | 9 | 23 | 24 | 24591.170 | *6* |
| | 3 | 11 | 2.0 | 94201 | 25 | 9 | 23 | 24 | 24591.170 | *7* |
| | 5 | 11 | 1.0 | 10000 | 25 | 7 | 23 | 24 | 24596.528 | *8* |
| | 3 | 11 | 0.0 | 94201 | 25 | 4 | 23 | 24 | 24626.340 | *9* |
| | 1 | 11 | —— Modified LRU —— | | | | | | 24627.226 | *10* |
| | 3 | 11 | 1.0 | 5000 | 3 | 9 | 1 | 2 | 24753.453 | *11* |
| | 3 | 11 | 1.0 | 10000 | 3 | 9 | 1 | 2 | 24755.340 | *12* |
| | 3 | 35 | —— Modified LRU —— | | | | | | 25399.000 | *13* |
| | 3 | 50 | 0.0 | 94201 | 8 | 4 | 1 | 5 | 27600.189 | *14* |
| | 3 | 75 | 0.0 | 94201 | 8 | 4 | 1 | 5 | 27600.189 | *15* |
| | 2 | 58 | 0.0 | 7503 | 11 | 5 | 2 | 6 | 28425.585 | *16* |
| | **1** | **40** | **1.2** | **25000** | **20** | **3** | **12** | **14** | **28829.774** | *17* |
| | 1 | 44 | 1.8 | 13054 | 21 | 4 | 2 | 9 | 32885.453 | *18* |
| | 3 | 11 | 0.0 | 94201 | 25 | 9 | 1 | 2 | 41397.000 | *19* |
| | 3 | 11 | 0.0 | 94201 | 20 | 4 | 18 | 19 | 166005.870 | *20* |
| | 99 | 40 | 1.2 | 25000 | 40 | 0 | 0 | 0 | 545851.510 | *21* |
| B | 3 | 11 | —— Modified LRU —— | | | | | | 24266.297 | *22* |
| | 3 | 11 | 0.0 | 94201 | 20 | 4 | 18 | 19 | 24272.865 | *23* |
| | 2 | 58 | 0.0 | 7503 | 11 | 5 | 2 | 6 | 24420.000 | *24* |
| | **1** | **40** | **1.2** | **25000** | **20** | **3** | **12** | **14** | **24662.216** | *25* |
| | 1 | 44 | 1.8 | 13054 | 21 | 4 | 2 | 9 | 24801.297 | *26* |
| D | 3 | 11 | —— Modified LRU —— | | | | | | 33774.519 | *27* |
| | 3 | 11 | 0.0 | 94201 | 20 | 4 | 18 | 19 | 33885.648 | *28* |
| | 2 | 58 | 0.0 | 7503 | 11 | 5 | 2 | 6 | 34994.907 | *29* |
| | **1** | **40** | **1.2** | **25000** | **20** | **3** | **12** | **14** | **35733.324** | *30* |
| | 1 | 44 | 1.8 | 13054 | 21 | 4 | 2 | 9 | 39875.167 | *31* |
| E | 2 | 58 | 0.0 | 7503 | 11 | 5 | 2 | 6 | 21523.562 | *32* |
| | **1** | **40** | **1.2** | **25000** | **20** | **3** | **12** | **14** | **21643.812** | *33* |
| | 3 | 11 | —— Modified LRU —— | | | | | | 21699.875 | *34* |
| | 3 | 11 | 0.0 | 94201 | 20 | 4 | 18 | 19 | 21704.438 | *35* |
| | 1 | 44 | 1.8 | 13054 | 21 | 4 | 2 | 9 | 21911.188 | *36* |

**Table 1: Results of exploring various parameter combinations. "Often Pct" is the threshold for identifying often-referenced files; "Peek Thresh" is the threshold for detecting directory scans; "AWF" represents the age weighting factor; "Aging Thresh" is the aging threshold; "Loose Thresh" and "Tight Thresh" represent the shared loose and shared tight thresholds; and "MFHS" is the miss-free hoard size (minimum space requirement) observed. All of the parameters are described in Section *3.3.1*. "Ref. No." is a cross-reference number used to aid discussions in the text. Measurements taken with the default parameter settings are shown in boldface. User F was not measured with the default settings; users C and G were not included in the experiments. Lines with "Modified LRU" for the last six parameters represent settings in which modified LRU hoarding (using additional heuristics) was enabled. *(Table continues on following page.)***

| User | Often Pct | Peek Thresh | AWF | Aging Thresh | Keep Refs | Min Refs | Loose Thresh | Tight Thresh | MFHS (KB) | Ref. No. |
|---|---|---|---|---|---|---|---|---|---|---|
| F | 3 | 11 | | —— Modified LRU —— | | | | | 60533.465 | *37* |
| | 3 | 11 | 0.0 | 94201 | 20 | 4 | 18 | 19 | 60961.211 | *38* |
| | 2 | 58 | 0.0 | 7503 | 11 | 5 | 2 | 6 | 65267.355 | *39* |
| | 1 | 44 | 1.8 | 13054 | 21 | 4 | 2 | 9 | 83124.820 | *40* |
| G | 3 | 11 | | —— Modified LRU —— | | | | | 18777.038 | *41* |
| | 3 | 11 | 0.0 | 94201 | 20 | 4 | 18 | 19 | 18778.856 | *42* |
| | **1** | **40** | **1.2** | **25000** | **20** | **3** | **12** | **14** | **19014.288** | *43* |
| | 2 | 58 | 0.0 | 7503 | 11 | 5 | 2 | 6 | 19150.942 | *44* |
| | 1 | 44 | 1.8 | 13054 | 21 | 4 | 2 | 9 | 19861.692 | *45* |
| H | 3 | 11 | | —— Modified LRU —— | | | | | 35196.609 | *46* |
| | 3 | 11 | 0.0 | 94201 | 20 | 4 | 18 | 19 | 35351.000 | *47* |
| | **1** | **40** | **1.2** | **25000** | **20** | **3** | **12** | **14** | **36579.696** | *48* |
| | 2 | 58 | 0.0 | 7503 | 11 | 5 | 2 | 6 | 36714.870 | *49* |
| | 1 | 44 | 1.8 | 13054 | 21 | 4 | 2 | 9 | 38438.739 | *50* |
| | 99 | 40 | 1.2 | 25000 | 40 | 0 | 0 | 0 | 285865.960 | *51* |

**Table 1 *(concluded)*: Results of exploring various parameter combinations.**

LRU with additional heuristics, which are explained in more detail in Section 4.2.

We ran many more experiments than are presented in Table 1, but have edited this presentation to highlight only the most interesting results. The table presents more detail for user A (though still only a subset of the parameters we tested) to give the reader a feel for the variations explored in our experiments. For all other users, we report at a minimum the parameters that achieved both the best and the worst performance, the Modified LRU performance, and the performance with SEER's default clustering parameters.

The reader will note from the parameters shown that we did not systematically attempt to precisely identify the individual effect of each parameter on SEER's performance. This decision was due primarily to the fact that our original plan was to first explore the parameter space and then study sensitivities in the areas of greatest interest. However, the results reported in Section 4.2 caused us to abandon the sensitivity analysis because of the high cost of simulations.

## 4.1 Cluster-Based Hoarding

When we began our investigations of parameters, we expected to achieve significant improvement in SEER's behavior compared to that reported in [8]. Although our earlier results were excellent, we had no particular reason to believe that they were the best achievable, especially because the parameters had been chosen using a less accurate performance measure. The results, however, were quite contrary to our expectations: on five of six the traces tested, the best value we achieved for minimum required hoard space was no more than 6% better than that recorded using the default parameter settings reported in [9] and shown in boldface in Table 1; the sixth trace (user A) produced only a 15% improvement.

Nevertheless, Table 1 reveals that the clustering parameters are not irrelevant to the system's performance. Examining the data, we can see that although the clustering algorithm produces similar results under some parameter combinations, other values can have a catastrophic effect on the results.

Some of the more notable observations include:

- Minor changes in the clustering parameters produce little effect when clustering is near-LRU (lines 3-9) due to high clustering thresholds. But an apparently similar near-LRU setting (line 20) produced the worst multiple-cluster results observed.

- Similarly, even without systematic sensitivity analysis, it seems that minor variations in the heuristic parameters don't have much effect. For user A, we experimented with changing the age weighting factor (lines 6-7), aging threshold (lines 11-12), often percent (lines 5 and 8), and peek threshold (lines 14-15); none produced significant changes in behavior.

- It is difficult to predict the consequences of adjusting certain parameters. For example, storing more information ("keepRefs") can (counterintuitively) harm the quality of clustering (line 11 vs. line 19). This sort of behavior is very typical of the Jarvis/Patrick algorithm [5].

- Turning off the heuristic hoarding of often-used files (by setting "oftenPercent" to 99) produced the worst performance of any parameter combination. (lines 21 and 51).

- The combination of parameters that produced the worst performance (other than disabling the frequent-use heuristic) for most users (lines 26, 31, 36, 40, 45, and 50) also caused poor results for user A. It is reasonable to assume that had we not done more extensive experiments for that user, the combination would have been the worst across the board.

- A combination that produced near-optimal performance on most users (lines 23, 28, 35, 38, 42, and 47) produced extremely bad results on one (line 20).

- The parameters that achieved the best clustering results (lines 3, 23, 28, 32, 38, 42, and 47) varied widely from user to user.

- It was nearly always effective to set "sharedTightThreshold" larger than "keepReferences"—a choice that modified the clustering algorithm so that near neighbors joined other clusters but never caused the clusters to merge—(lines 16, 24, 29, 32, 39, 44, and 49). This discovery was accidental, a side effect of our method of choosing random parameters.

In summary, the effects of the parameter choices are extremely complex, making intelligent adjustment very difficult, although it is possible to find settings that work fairly well for all users.

## 4.2 LRU Hoarding

In previous work, we compared SEER's clustering-based algorithm to pure LRU hoarding. In the current investigation, we extended that study to discover the effects of a modified LRU method in which heuristics were applied to increase the method's effectiveness. These heuristics are the same as those used by SEER's clustering mechanism, and are discussed in detail in [7, 9]. In brief, they include:

- Identify and ignore so-called "meaningless" programs such as `find`, which access many files in an scanning mode that does not necessarily indicate user interest. Several techniques are necessary to reliably detect such programs; see [7] for details. The "peekThreshold" parameter is part of this heuristic.

- Force the hoarding of critically important files such as shared libraries, regardless of the recency of reference. Such files are detected by frequency of access; if a file represents more than "oftenPercent" of all accesses, it is always hoarded.

- Force the hoarding of files that are rarely accessed, but are necessary for system initialization, such as files in `/etc`. SEER automatically hoards any plain file whose name begins with a period (so-called "dot" files), since these are often used during the login process. Other critical files and directories can be hand-specified; fortunately, doing so is rarely necessary (and was not done in these simulations).

- Always hoard all non-files (directories, symbolic links, and device files). This heuristic is inexpensive, since such objects are relatively small, but it greatly improves the user's experience.

In all cases, files that are handled by these heuristics are omitted from all other hoarding calculations.

Fortunately for our investigation, it was simple to disable SEER's clustering mechanism while retaining the heuristics. We did so by choosing the clustering parameters so that the two thresholds were larger than the number of references retained (keepRefs). This parameter selection results in each file being placed in a separate cluster. Since SEER uses LRU to choose which clusters to hoard, this technique resulted in LRU with heuristics. In Table 1, lines in which LRU hoarding was used are indicated by replacing the relevant four parameters by the string "Modified LRU".

As can be seen in the table, five of the six users achieved their best performance while using the Modified LRU hoarding method, despite extensive experimentation with the clustering parameters in an attempt to find settings that would produce optimum performance. In other words, the best hoarding method for most users involves applying several simple heuristic changes to standard LRU. For the sixth user (E), Modified LRU performed only 0.81% worse than the best parameters we found.

Because the Modified LRU method still depends on the heuristic parameters, we briefly investigated a few additional parameter combinations (lines 1, 2, 10, and 13). Unfortunately, the resources available to us prevented a full exploration of the available options. Nevertheless, we can see that setting the often-referenced file threshold too low (line 10) or raising the peek-detection threshold (line 13) had a small negative effect. The implication is that not only is Modified LRU a better method than the more complex clustering approach, it is relatively insensitive to particular parameter settings.

## 4.3 Conclusions and Future Work

Previous work on hoarding either required substantial user control [1, 2, 3, 6] or involved a relatively complex computation with significant memory requirements [7, 12]. In contrast, the experiments reported here suggest that hoarding can be made to work well by combining an efficient algorithm (LRU) with a number of simple heuristics, and that those heuristics are not very sensitive to their parameters.

At the moment, our investigation should be considered to be preliminary, since it was performed with a relatively small set of traces and we only explored a limited range of the heuristic parameters. Now that we believe the Modified LRU method to be preferable, it would be wise to further examine its parameters, using more extensive traces, to verify our conclusions and to better characterize its behavior.

Finally, we note that the discovery that a simpler approach produces superior results is especially important because recent developments in storage technology have caused the hoarding problem to migrate from laptop to handheld computers. The latter devices are typically extremely limited in the amount of memory that they can dedicate to auxiliary algorithms. Thus, the efficiency of the Modified LRU approach has become essential to the ability to use hoarding on such systems. Our study indicates that automated hoarding still holds great promise in extending the usability of handheld devices.

# 5. ACKNOWLEDGMENTS

The authors would like to thank the members of UCLA's File Mobility Group for allowing traces of their behavior to be used in this study.

# 6. REFERENCES

[1] Rafael Alonso, Daniel Barbará, and Luis L. Cova. Using stashing to increase node autonomy in distributed file systems. In *Proceedings of the Ninth IEEE Symposium on Reliability in Distributed Software and Database Systems*, pages 12–21, October 1990.

[2] Maria R. Ebling, Bonnie E. John, and Mahadev Satyanarayanan. The importance of translucence in mobile computing systems. *ACM Transactions on Computer-Human Interaction*, 9(1):42–67, March 2002.

[3] Peter Honeyman, Larry Huston, Jim Rees, and Dave Bachmann. The Little Work project. In *Proceedings of*

the Third Workshop on Workstation Operating Systems*, pages 11–14. IEEE, April 1992.

[4] 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.

[5] R. A. Jarvis and E. A. Patrick. Clustering using a similarity measure based on shared near neighbors. *IEEE Transactions on Computers*, C-22(11):1025–1034, November 1973.

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

[7] Geoffrey H. Kuenning and Gerald J. Popek. Automated hoarding for mobile computers. In *Proceedings of the 16th Symposium on Operating Systems Principles*, pages 264–275, St. Malo, France, October 1997. ACM.

[8] Geoffrey H. Kuenning, Peter Reiher, and Gerald J. Popek. Experience with an automated hoarding system. *Personal Technologies*, 1(3):145–155, September 1997.

[9] 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.

[10] Mahadev Satyanarayanan. The evolution of Coda. *ACM Transactions on Computer Systems*, 20(2):85–124, May 2002.

[11] 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.

[12] Carl D. Tait, Hui Lei, Swarup Acharya, and Henry Chang. Intelligent file hoarding for mobile computers. In *ACM/IEEE International Conference on Mobile Computing and Networking*, pages 119–125, Berkeley, CA, November 1995.