Dynamic Relationships and the Persistence of Pairings

Ahmed Amer and Darrell D. E. Long†
Jack Baskin School of Engineering
University of California, Santa Cruz
1156 High Street
Santa Cruz, CA 95064
amer4,darrell@cse.ucsc.edu

Abstract

The ability to automatically hoard data on a computer's local store would go a long way towards freeing the mobile user from dependence on the network and potentially unbounded latencies. An important step in developing a fully automated file hoarding algorithm is the ability to automatically identify strong relationships between files.

We present a mechanism for visualizing the degree of long-term relationships inherent in a file access stream. We do this by comparing the performance of static and dynamic relationship predictors. We demonstrate that even the simplest associations (from a static/first-successor predictor) maintain relatively high accuracy over extended periods of time, closely tracking the performance of an equivalent dynamic (last-successor) predictor. We then introduce rank-difference plots, a visualization technique which allows us to demonstrate how this behavior is caused by stable static pairings of files that are lost by the adaptation of the dynamic predictor for a substantial subset of frequently accessed files. We conclude by demonstrating how a third pairing mechanism can make use of these observations to outperform both the dynamic and static predictors.

1. Introduction and Motivation

Although wireless communication can allow data connections while the user is mobile, wireless bandwidth does not approach the bandwidth available with conventional wired networks and is more susceptible to unforeseen disruptions (*e.g.*, driving through a tunnel, or signal interference). This means that a mobile user is susceptible to significant data access latencies imposed by the network. In the case of disconnected operation this latency is unbounded, and yet with increasing data transfer rates latency actually becomes a more serious penalty.

Reducing latency is one of the more challenging problems in modern data storage. With many of the newer storage technologies on the horizon, there is an ever growing gap between seek performance (latency) and bandwidth (data transfer speed). Increases in capacity and transfer rates seem to be increasing at an impressive rate, and upcoming technologies such as holographic stores promise even greater improvements in capacity and transfer speeds (terabytes of storage accessible at gigabytes per second) [10, 12]. Random access speeds to data on such devices is not showing the same degree of improvement, in large part due to the physical limitations of the mechanical process this involves, *e.g.*, disk arm movement, media replacement in a tertiary library, or basically any mechanical movement in newer high-bandwidth storage devices. This is analogous to the problems with network communication for a user who may be weakly connected or temporarily disconnected from his main data repositories.

Increases in bandwidth to data only help to compound the performance impact of data access latencies. To help illustrate this, Figure 1 shows the percentage contribution of data access latency to overall data access time for different average request sizes. Fixed or relatively large latencies become increasing performance bottlenecks as quickly as data transfer technologies improve.

Mechanical overheads have quickly become the largest component of storage system overhead, and analogously for mobile systems, remote data access rapidly becomes the dominant component of data access cost as data transfer technologies improve in bandwidth. It is therefore important to reduce the number of times we incur avoidable data access latency.

Figure 1 also suggests a possible solution to this problem. It clearly shows the advantages of using more data per operation. Such techniques are useful only if the additional data accessed is likely to be requested in the immediate fu-

†This work was supported in part by the National Science Foundation award CCR-9972212, and the Usenix Association.

0-7695-1080-9/01 $10.00 © 2001 IEEE

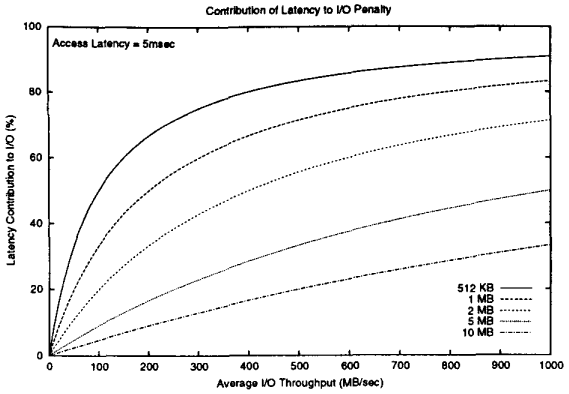


Figure 1. The effect of increasing latency/bandwidth gap on system throughput.

ture, in other words, only if the larger volume of data retrieved has effectively resulted in a prefetch of useful data. This observation implies a substantial gain through the prediction of sequences of data access, and placement of data in such a manner as to increase the likelihood of adjacent data items being useful. In other words it would be very helpful if we could group related data items together, to ensure that moving larger volumes of data results in more effective utilization of bandwidth, while minimizing the likelihood of incurring an avoidable latency cost, *i.e.* we need to minimize the number of requests to retrieve remote data, and not simply reduce the volume of data moved.

Using immediate-successor predictors, or pairing relationship estimators, we demonstrate a new mechanism for visualizing the degree of relationship inherent in a file access trace. The detection of such accurate pairings, that remain accurate for extended time periods, is a very promising result for automated file hoarding and data grouping.

2. Gauging Inter-Access Relationships

Probabilistic approaches to data placement are often based on assumptions of independence. Data access events are generated by user behavior and deterministic programs, and are therefore rarely independent. The question we consider is how do we gauge the degree of inter-file relatedness inherent in a stream of file access events. We propose that by following the performance of dynamic and static file access predictors which operate on a per-file basis to predict the next-successor for each file (*i.e.* predictors that offer pairings between files) we can visualize this degree of relatedness.

By tracking a time-dependent accuracy metric (which we have called LTscore – the long-term pair score) we can demonstrate that simple first-guesses for file relationships are reasonably accurate, and that this accuracy persists for extended time periods (several months). Using the rank-difference plots we can demonstrate that persistence of this simple static prediction is indeed the cause of this sustained accuracy, and not a constant shift to new files being accessed.

2.1. The Predictors/Pairing Estimators

We use file access predictors that are designed to predict a successor given a single predicate. Given an access to file A our predictors offer a candidate file B as its prediction for the next file to be accessed after A. In this manner these predictors define a one-way relationship between pairs of files. We refer to these predictions as candidate pairings.

The static predictor observes the first file B to follow A, and from that point on the static predictor will always provide B as the successor prediction. The static successor predictor provides a pairing that remains static, effectively pairing a file with its first successor and never changing this first impression. Although this approach would intuitively appear fruitless, we demonstrate below how this incredibly simple scheme achieves an accuracy that is very close to that of the dynamic predictor.

The dynamic predictor is simply the last successor prediction model, where the last file observed to follow file A is offered as the prediction for the next successor of A. This model requires checking and possibly updating the prediction with each and every file access.

Our final predictor is "Noah," which is an extension of the basic last-successor predictor to filter out noise in the observed access stream. This filtered model effectively ignores observations that vary too rapidly, effectively acting as a low-pass filter for observations. As with the last-successor model, this predictor maintains a record of the last file to be a successor to the current file. In addition to this, Noah maintains a current successor prediction, which is updated to become the last-successor, only if the last successor is observed to have remained unchanged from the last access to this file.

The last-successor (dynamic) predictor updates its prediction with every access to a file, and the first successor never updates its initial prediction, but Noah will update a file's successor only if a new successor is observed consistently for two consecutive accesses.

2.2. The LTscore: Long-Term pair score

Our first test is applied to the dynamic and static predictors, and produces what we refer to as long-term pair scores for a given trace of access events. In particular the accuracy of a pairing is evaluated over a trace period of several months. A pairing is considered to be accurate if an access event confirms the existing pairing (in other words, no

update is required for the case of dynamic pairings). All averages are weighted by the frequency of particular access events, an item accessed frequently contributes proportionally to the evaluated average "score."

A higher score for dynamic placement is to be expected. The alternative would imply a workload that exhibited pathological alternation between multiple pairings, a possibility but not very likely to be the dominant case. A rapid divergence between the score for static pairings and dynamic pairings would indicate a strong need to revise pairings, indicating a strong tendency towards change in the inter-file relationships.

Given a sequence of access events $\omega = r_1, r_2, r_3, ...,$ such that $r_i = x$ implies the i^{th} file access was to file x. A pairing is a tuple (x, y) that assumes if $r_i = x$ then $r_{i+1} = y$. A pairing (x, y) is *valid* for an occurrence of $r_i = x$ if r_{i+1} is indeed found to be y, otherwise it is invalid. A pairing (x, y) is formed at the first encounter of $r_i = x$ and $r_{i+1} = y$.

For dynamic pairings, if a pairing (x, y) is found to be invalid and $r_{i+1} = z$, then the pairing (x, y) is replaced with (x, z), but static pairings do not perform this update. If the number of valid pairings observed at time t are $P_{dynamic}(t)$ and $P_{static}(t)$ for dynamic and static pairings respectively. And the total number of events was $T(t)$, the *long-term pair score (LTscore)* for dynamic pairing at time t is calculated as

$$\mathrm{LTscore}_{dynamic}(t) = P_{dynamic}(t)/T(t)$$

and likewise the score for the static pairing is simply

$$\mathrm{LTscore}_{static}(t) = P_{static}(t)/T(t)$$

Long-term pair scores are therefore cumulative over the length of the trace period. This allows us to judge the development of a final performance average as it is calculated over the entire trace period. When referring to the degree of "matching," we are referring to the degree to which the accuracy for static pairing follows the trend of the accuracy for dynamic pairing. Matching is therefore proportional to the correlation between $\mathrm{LTscore}_{dynamic}(t)$ and $\mathrm{LTscore}_{static}(t)$ The second term "divergence" refers to the increase in separation between these two curves. In other words, the divergence d can be defined as:

$$d = \frac{\partial}{\partial t}(\mathrm{LTscore}_{dynamic}(t) - \mathrm{LTscore}_{static}(t))$$

2.3. Rank-Difference Plots

A rank-difference plot represents a performance difference between two successor-predictors, plotted against instances of files ordered according to access frequency (increasing in the direction of the increasing x-axis). The y-axis on these graphs represents the difference between the accuracy (%) of two predictors, for the specific file at that point on the x-axis. In effect this measures the improvement in final LTscore from using a particular dynamic predictor *vs.* leaving the initially observed pairings.

An important feature of the rank-difference plot is that the highest-frequency files are at the right of the graph, meaning that observed differences at higher x ranges are more significant than differences at the lower (left) x ranges. When considering these plots it should be noted that all file traces examined showed a tremendous skew in file access frequencies. These traces included multiple computers ranging from personal workstations to file servers, and were tested for trace periods varying up to several months and over a year.

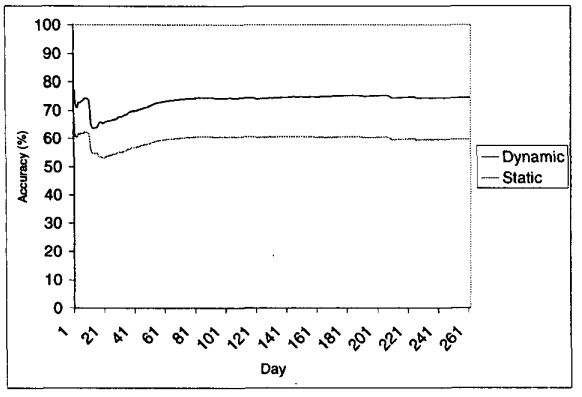
3. Experimental Results

To describe the scoring schemes in real-world scenarios we present results from experiments conducted on two sets of traces collected from Carnegie Mellon University (CMU) [8], and the University of California, Berkeley (UCB) [11]. Both sets of traces were processed using the DFSTrace trace reading library [8]. These traces offered detailed records of all system calls and file accesses performed over a period of several years, and spanning a wide range of systems and workloads. Tests were run for periods extending over several months and up to a year.

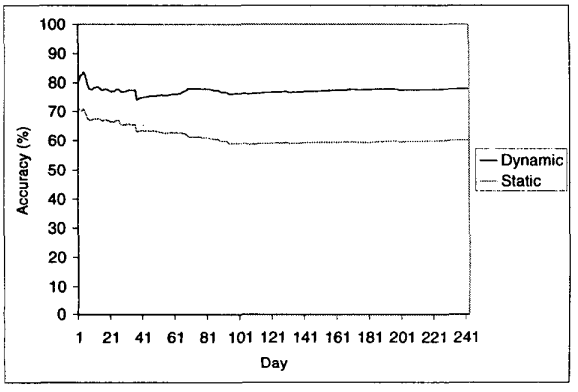
3.1. LTscore Results

Figure 2 shows two results from the CMU traces that are worthy of comparison. Figure 2(a) represents a personal workstation, whereas Figure 2(b) was a server which exhibited the highest rate of file accesses of the traced systems. One would expect the inter-access relationships for a busy system to be heavily interleaved and attempts to discern the "higher-level knowledge" to be masked by more noise than that for a less loaded system with fewer sources of access requests. In fact, you can see that Figure 2(a) shows better matching for static and dynamic pairing. Whereas in the case of the file server (Figure 2(b)), the matching is much poorer, with more pronounced divergence.

It should be noted that the accuracy results remain reasonably high, and can in fact tend to increase for both static and dynamic pairings. This is a result of the most recent files accessed being the most likely to be accessed frequently. As long as new files are being created and used, static pairing accuracy values calculated cumulatively over an extended period (as for the long-term pair scoring), can increase and decrease depending on the degree of inter-access dependencies inherent in the current working set of files.



(a) Personal Workstation

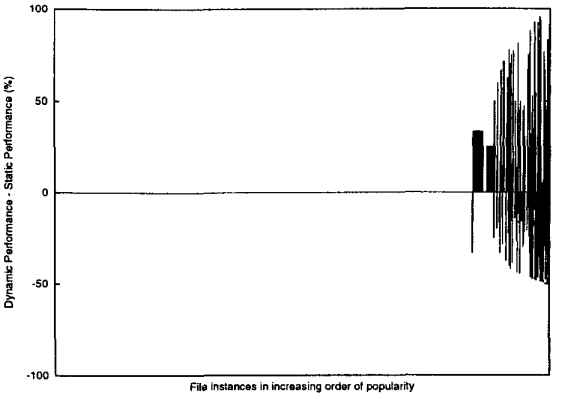


(b) File Server

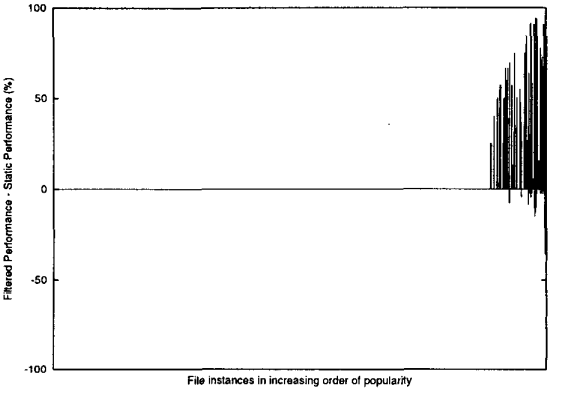
Figure 2. Plots of a long-term pair score for two representative workloads, from the Carnegie-Mellon traces [8].

Another feature of the long-term pair scores involves the divergence of the static and dynamic scores. If the working set of files being accessed varies greatly, and is not well-conserved, then the rate of divergence between static and dynamic pair scores would be expected to increase. If on the other hand the working sets were well-conserved, the rate of divergence should be much lower. This is supported by the increased divergence in the file server results of Figure 2(b).

It is not possible to discern from Figure 2 whether the surprising correlation between static and dynamic pairings is due to new files being accessed, or is in fact a true persistence of static pairings. To answer this question we can use the results of the rank-difference plots, which we present in the following section.



(a) Dynamic *vs.* Static



(b) Noah *vs.* Static

Figure 3. Rank-difference plots.

3.2. Rank-Difference: The persistence of pairings

Figure 3(a) demonstrate the difference between the dynamic (last-successor) model and the static (first-successor) model for each file. Figure 3(b) demonstrates the difference between Noah and the static (first-successor) model. The x-axis is simply an ordering of file instances such that the most frequently accessed file is at the rightmost point on the axis.

The first thing to notice from these plots is that differences tend to occur towards the right of the plot. This is an artifact of the access-frequency distribution. Any improvement in predictions can only happen if a file is accessed more than once. And if a file is accessed very few times, then any improvement in performance is subsequently limited. Algorithms that observe the dynamic access stream to make their decisions would also have no time to learn.

The most important thing to notice about this figure is

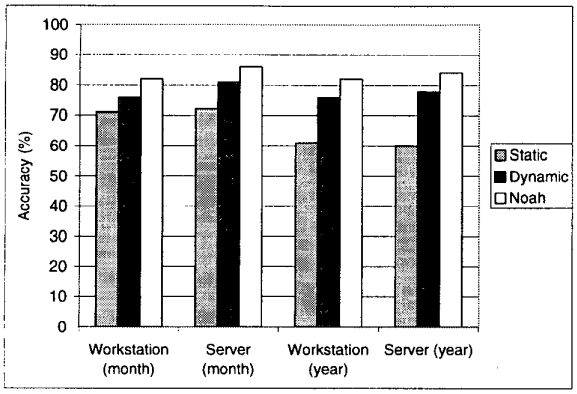


Figure 4. Prediction accuracy.

that for Figure 3(a) there is a very wide variation for what each file finds to be a good pairing estimator (dynamic *vs.* static). Also worthy of note is how these differences are more pronounced for higher-frequency files. This is especially notable when you consider that the distribution of file access frequencies is exponentially skewed (this was repeatedly confirmed for all tested workloads at multiple timescales ranging from a day to a year). With such a large variation among individual files, it is definitely beneficial to use a mechanism that can capture the best of both worlds.

Figure 3(b) shows one thing very clearly, Noah is successful at eliminating noisy data, and filtering out transient successors. By accepting a successor only if it is observed as such two consecutive times, Noah captures pairings that are indeed stable over extended durations, without confusion due to once-off differences in the observed successor. In this manner Noah can be seen to incorporate the static predictor and is unlikely to ever perform worse over an extended period of time.

Figure 4 summarizes the relative predictor performance scores (final LTscore or average accuracy) for the two sets of CMU traces over two different time periods. Not surprisingly Noah outperforms the dynamic predictor, which in turn outperforms the static predictor.

4. Related Work

To overcome latency problems without some form of prefetching would not appear to be possible. This explains the increased interest in prefetching and predictive caching for such applications as web-proxies and file caches. The graph-based model for predictive prefetching was first proposed by Griffioen and Appleton [2], and has been used in a range of applications including web proxies [9]. A model based on data compression technology, the Finite Multi-Order Context (FMOC) model, was evaluated in comparison to Griffioen and Appleton's graph-based model [6]. FMOC requires state exponential in the number of unique files. This comparison showed both FMOC, and its static-space derivatives the Partitioned Context Model (PCM), and Extended PCM (EPCM) [5], could outperform equivalent graph-based models. Our own experiments demonstrated that Noah could improve upon the accuracy of PCM when using similar state space, and could match the performance of PCM models utilizing more than 10 times more state per file. The first to suggest the application of techniques from data compression to predictive caching were Curewitz, Krishnan, and Vitter [1, 14].

Similar concepts of detecting and prefetching "working sets" of data were also used to cache enough data on a mobile system to allow disconnected operation [4, 7]. Among the most successful hoarding algorithms developed to date is that of the SEER system [7]. SEER introduced the concept of "semantic distance" and achieved a high level of automation.

SEER was also based on identifying groups of related files. It used "semantic distance" to evaluate how "near" one file was to another. This was combined with clustering based on a thresholded number of "shared *near* neighbors." This was effectively an efficient version of the Jarvis and Patrick clustering algorithm [3] (The original clustering algorithm was $O(n^2)$ in the number of files, which is obviously impractical for large file systems hoarding), utilizing semantic distance as a nearness metric. Unfortunately, the automation of SEER involved a substantial research effort to find parameters that provided good performance. Whether the actual hoarding process requires extensive user input, or parameter selection was based on extensive experimental tuning, we feel that the major obstacle to general adoption of file hoarding has been an insufficient level of automation.

Predicting based on a choice between two alternatives is a common and well established scenario in the domain of processor branch prediction [13]. In branch prediction you are attempting to predict whether a branch will be taken or not. This is critical for effective pipelining and processor hardware utilization, in a similar manner to data access prediction's usefulness for improving the performance of the storage subsystem. Branch prediction differs from our application in that it is a domain that is limited to only two possibilities: a branch is taken or not. In file access streams the observed successor could potentially be any file in the file system space. The fact that our predictors work so well in spite of this observation is indicative of the inherent relatedness in file access events.

5. Conclusions

Pairings produced by a dynamic predictor (a last-successor guess) are more accurate than static (first-successor) pairings, and yet this is only true on average.

The suitability of different predictors for different files can be seen very clearly using a rank-difference plot. For many cases of files that are accessed very frequently we can see a distinct tendency for static pairings to remain valid over extended periods of time. By using a predictor that attempts to capture the stability of a static predictor, while allowing for the adaptability of the dynamic predictor, we have seen how an improvement over both in terms of prediction accuracy can be achieved.

These results, combined with our argument for the ever-increasing penalties of data access latency make a strong case for mobile data hoarding, or dynamic placement policies, based on the dynamic detection of relationships. The latency argument shows it is useful to do this, and the persistence results show that it is reasonable and feasible to detect and exploit dynamic relationships, as they will often remain valid for extended time periods.

6 Acknowledgements

We are grateful to all the members of the Computer Systems Laboratory at the University of California, Santa Cruz, for their continuous feedback, support and valuable discussions. Our most extensive multi-year traces were kindly made available by M. Satyanaryanan of Carnegie Mellon University, through the greatly appreciated efforts of Tom Kroeger in processing and conversion.

References

[1] K. M. Curewitz, P. Krishnan, and J. S. Vitter, "Practical prefetching via data compression," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD '93)*, (Washington, D. C.), pp. 257–266, May 1993.

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

[3] R. A. Jarvis and E. A. Patrick, "Clustering using a similarity measure based on shared near neighbors," *IEEE Transactions on Computers*, vol. C22, pp. 1025–34, November 1973.

[4] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the Coda file system," *Operating Systems Review*, vol. 25, no. 5, pp. 213–25, 1991. Thirteenth ACM Symposium on Operating Systems Principles, Pacific Grove, CA, USA, 13-16 Oct. 1991.

[5] T. M. Kroeger, *Modeling File Access Patterns to Improve Caching Performance*. PhD thesis, University of California, Santa Cruz, Mar. 2000.

[6] T. M. Kroeger and D. D. E. Long, "The case for efficient file access pattern modeling," in *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems (HotOS-VII)*, (Rio Rico, Arizona), IEEE, Mar. 1999.

[7] G. H. Kuenning and G. J. Popek, "Automated hoarding for mobile computers," in *Sixteenth ACM Symposium on Operating Systems Principles*, (Saint Malo, France), pp. 264–75, Oct. 1997.

[8] L. Mummert and M. Satyanarayanan, "Long term distributed file reference tracing: Implementation and experience," *Software - Practice and Experience (SPE)*, vol. 26, pp. 705–736, June 1996.

[9] V. N. Padmanabhan and J. C. Mogul, "Using predictive prefetching to improve world wide web latency," in *Proccedings of the 1996 SIGCOMM*, ACM, July 1996.

[10] D. Psaltis and G. W. Burr, "Holographic data storage," *IEEE Computer*, pp. 52–60, Feb. 1998.

[11] D. Roselli, "Characteristics of file system workloads," Technical Report CSD-98-1029, University of California, Berkeley, Dec. 23, 1998.

[12] G. T. Sincerbox, ed., *Selected Papers on Holographic Storage*, vol. MS 95 of *SPIE Milestone series*. International Society for Optical Engineering, 1994.

[13] J. E. Smith, "A study of branch prediction strategies," in *Proceedings of the Eighth Annual Symposium on Computer Architecture*, (Minneapolis, MN, USA), pp. 135–48, IEEE, May 1981.

[14] J. S. Vitter and P. Krishnan, "Optimal prefetching via data compression," *Journal of the ACM*, vol. 43, pp. 771–93, Sept. 1996.